Bounded Model Construction for Duration Calculus

Jacob Enslev Anne-Sofie Nielsen

Master's Thesis, 30 ECTS points
Informatics and Mathematical Modelling
Technical University of Denmark
March 2005

IMM-THESIS-2005-5

Abstract

In this thesis, we shall present a tool, the *BMC/DCValidator* for automatic validation of *duration calculus* formulas. Duration calculus (DC for short) is a temporal logic, i.e. a logic for reasoning about time. DC may be used for specifying systems at an abstract level and the purpose of the tool is to verify that the system meets certain requirements, which are also specified using DC. The abstractness and expressiveness of DC makes it well-suited for system description, but hard to verify properties by automatic means.

Our tool provides an efficient implementation of the work of [MF02] whose main idea is to translate DC formulas to propositional clauses or linear constraint systems, both of which may be handed over to an external solver. The timing aspects of the formulas are represented in a discrete way, and the properties of the formulas are validated within a bounded time interval, as the unbounded validation turns out to be undecidable. In some cases, it is possible to determine a bound such that bounded validity implies general validity.

The tool has been implemented with a multitude of options such as polarity optimisation and output format that allow us to experiment with the translation algorithm. Our benchmark results show that the tool has good performance in general, but that the success of the tool is extremely dependent on the implementation details of the solver.

Resumé

I denne opgave vil vi præsentere et værktøj, BMC/DCValidator, til automatisk validering af *duration calculus* formler. Duration calculus (forkortet DC) er en temporallogik, dvs. en logik til at ræsonere omkring tid. DC kan benyttes til at specificere systemer på et abstrakt niveau, og formålet med værktøjet er at verificere, at systemet har visse egenskaber, der også specificeres med DC. DC er abstrakt og udtryksfuldt, hvilket gør det velegnet til systembeskrivelse, hvorimod det er svært at verificere egenskaber automatisk.

Vores værktøj tilbyder en effektiv implementatering af arbejdet i [MF02], hvis hovedidé er at oversætte DC formler til udsagnslogiske klausuler eller systemer af lineære uligheder, som i begge tilfælde kan gives til en ekstern løser. De tidslige aspekter af formlerne repræsenteres diskret, og egenskaberne ved formlerne valideres indenfor et begrænset tidsinterval, da ubegrænset validering viser sig at være uafgørbart. I nogle tilfælde er det muligt at bestemme en tidsgrænse, således at begrænset validitet medfører generel validitet.

Værktøjet er blevet implementeret med et større antal valgmuligheder som f.eks. polaritetsoptimering og uddataformat, hvilket gør os i stand til at eksperimentere med oversættelsesalgoritmen. Vores tidsmålinger viser, at værtøjet generelt har god ydeevne, men at værktøjets succes afhænger utroligt meget af implementeringsmæssige detaljer i løseren.

Preface

This thesis presents a tool for automated validation of a real-time logic, building on the work of [MF02]. The complete problem description may be found in [MF04].

Our background in this field is primarily a course on Real-Time Systems that was given by Martin Fränzle in the Spring 2003. In this course, we solved an assignment where we translated timed automata into linear constraints. We found this area of research very interesting and together with Martin Fränzle we agreed on this project, in which a real-time logic is translated to propositional clauses or linear constraints. We have been very pleased to be able to employ many of the skills learned throughout our studies — both concerning compilers, real-time systems, computability and software engineering in general.

This thesis is intended for a reader who is familiar with software systems in general, but not particularly with real-time systems. A sound knowledge of mathematical logic is necessary to understand the theoretical aspects of our work. To fully understand the chapters regarding design and implementation of the tool, it will be an advantage to be well acquainted with object-oriented methods, including UML and Java.

Acknowledgements

First and foremost, we would like to thank our two supervisors, Martin Fränzle and Michael R. Hansen, for their help and guidance.

Our project builds on the research of Martin Fränzle, who has been very patient in making sure that we have fully understood his work, and has engaged in multiple discussions with us, particularly when we needed to make decisions that required knowledge beyond the realms of this project.

Michael R. Hansen has provided invaluable help with the proof of correctness of the translation algorithm. He has always found the time to answer our questions, and has been a great resource regarding duration calculus and thesis writing in general.

We would also like to thank IMM for providing us an office throughout the entire project period.

Finally, thanks to Torben Gjaldbæk for proof-reading this thesis.

Contents

Chapter 1

Introduction

We will begin by describing the problem area in which we operate: Real-time systems. Then, we shall briefly explain our contributions to this area, and finally lay out the structure of the remainder of this thesis.

1.1 Real-Time Systems

Real-time systems are computer systems that interact with an external environment through *events* while adhering to *timing constraints*. The most common form of a timing constraint is a *deadline*, i.e. an absolute time limit within which some task must be accomplished.

Today, the real-time systems are everywhere! Examples of just a few of the real-time systems we may encounter are:

- The telephone system
- Vending machines
- Computer games
- Cars
- Medical systems for patient monitoring
- Space navigation systems
- Robots
- Burglar alarms
- Power plant control systems

The real-time systems may be categorised on a scale ranging from "soft" to "hard". Hard real-time systems are those whose timing constraints *must* be met, or else the system fails. The soft real-time systems may still accomplish their duties successfully even if some constraints are violated. A control system for an aircraft is an example of a hard real-time system. It can simply have catastrophic consequences if the aircraft does not respond in time to the instructions of the pilot. An example of a soft real-time system is a vending machine. It may be

annoying to wait 30 seconds more for your favourite beverage, but certainly this is not a dramatic consequence. A data network lies somewhere between "hard" and "soft". It is not a problem if it occasionally loses or delays some packets of data, but in order for retransmission to be feasible it must not happen too often.

Generally, there are two forms of requirements for real-time sytems. One is *safety properties*, typically stating that the system should never enter certain undesirable states. E.g. an aircraft should never go below a specified altitude, except during takeoff and landing. The other is *liveness properties*, stating that the system must progress in some sense. Typically, it is possible to meet the safety requirements by doing nothing at all, e.g. by keeping the aircraft on the ground. A liveness property for the aircraft could be that it eventually takes off.

1.1.1 Examples

We will now introduce some examples of real-time systems that we will treat in some detail. These examples will later be formalised and used for benchmarking of our tool. We have taken the gas burner example from [ZH04].

Gas Burner

A gas burner may be either *heating* when the flame is burning, *leaking* when the gas is switched on but there is no flame, or *idling* when the gas is off.

It is clear that the leaking state is undesirable, since it may be dangerous if too much gas leaks out into the room. A safety property for the gas burner would then reasonably be that the time intervals in which the gas is leaking are not too long. For instance, let us say that we have calculated that we do not risk a critical situation if for any time interval at most 30 seconds long, the gas burner leaks no more than n seconds.

The challenge is then to design a control system for the gas burner so that the safety property can be guaranteed to hold. One could for instance decide that leaks should be detected and stopped within one second, and to avoid frequent leaks, it should not be possible to switch on the gas for a period of 30 seconds after a leak.

Before implementation of the control system, one would naturally like to verify that the conjunction of the design decisions imply the safety requirement. There are a multitude of logics for specifying such properties, one of which we shall present in Section 2.1 on page 6.

Scheduler

We shall now introduce a simple scheduler. The scheduler controls access to a processor that may only run one process P_i at a time. This is commonly known as a requirement for *mutual exclusion*. There are n processes in our system, and we shall assume that each processes will run anytime, and for as long time as possible. The task is then to allocate processor time so that each process has run for 2 time units for each full period of $2n$ time units that has elapsed. To achieve this, it shall be a design decision of the scheduler that process P_i will run only when process P_{i-1} has completed its 2 time unit run for this period.

Process P_1 will not run again until process P_n has completed its run. So, in effect, this is a type of round-robin scheduler.

Naturally, one would now like to verify that the design decision is powerful enough to ensure that all processes have run for 2 time units for each $2n$ time period. In the following section, we shall investigate possible methods for such validation.

1.1.2 Validation of Real-Time Systems

As described in Section 1.1 on page 1, many of the applications of real-time systems are highly safety critical. Thus, a need for a thorough validation of such a system arises. We shall consider the method of *formal validation*: Given a formal description of both system and requirements, to verify that the requirements are satisfied by the system.

Fundamentally, there are two different approaches to formal validation: *Theorem proving* and *model checking*, which we shall describe next[1].

Theorem Proving

We shall define theorem proving as proving theorems of a particular logic using the assistance of a computer system, referred to as a *theorem proving system*. The group of theorem proving systems may be subdivided based on automation level, ranging from the *proof checkers* that can only assist in the checking of a proof that has already been done "by hand", to the fully automatic so-called *automated theorem provers*. The latter are, unfortunately, typically intractable, so in practice the user must guide the system somehow, e.g. by choosing appropriate lemmas. Many systems try to combine the best of both worlds.

Model Checking

The model checking approach tries to establish some finite model (e.g. a state transition graph) of the system with the requirements formulated in a temporal logic, and, in principle, performs exhaustive search of the reachable state space.

The model checkers may be categorised as either *homogenous* or *heterogenous* [JK98].

In the homogenous model checkers, both the model and the properties to be validated are specified in the same notation (e.g. as automata), and it is verified that one implies the other, or that one *behaves*[2] at least as the other. In the heterogenous model checkers, the notation differs. For instance, the system could be modeled by an automaton while the requirements may be stated in a temporal logic. Later in this section, we shall look at the UPPAAL tool, which is an example of a heterogenous model checker.

Some *advantages* of model checking are:

- A general approach for both hardware, software and embedded systems.
- Requires a low degree of interaction with the user — providing for model checking to be used routinely during the development process.

[1] The structuring of validation approaches has been taken from [TR02].

[2] The homogenous approach is also known as the *behaviour-based approach*.

- Has shown to be useful in practice, both though case studies and through widespread use in the industry. [JK98]

Some *disadvantages* of model checking are:

- Verifies properties of a *model* of the system rather than of the system itself — as stated by [JK98]: "Any validation using model checking is only as good as the model of the system".
- There may be decidability issues for some representations — and certainly severe time and space complexities. The decidability and complexity results on model construction for the temporal logic *duration calculus* (DC) will be presented in Section 2.2 on page 10.

Clearly, it may be impossible to completely verify the correctness of complex systems, but certainly model checking can increase one's confidence in a system.

The UPPAAL Model Checker

Numerous tools are available for model checking. In this section, we shall take a closer look at just one of them: The popular UPPAAL tool. We have chosen this tool as an introduction to the world of model checkers because we have had some personal experience with it, and it appears to be both useful and widespread.

UPPAAL consists of an entire tool suite for verification of real-time systems. It integrates modeling, simulation and model checking. Compared to model checking, simulation offers a computationally cheap method of debugging models and specifications, and hence these tools supplement each other well.

Normally, UPPAAL models consist of a model of the (continuous) environment and a model of the (discrete) controller program, interacting through sensors and actuators. The models are given as timed automata interpreted over dense time, i.e. time is represented as a real number. The classical timed automata have been extended with, among other features, synchronisation over channels.

The query language is a modified version of Computation Tree Logic (CTL), a temporal logic that enables one to reason about paths (e.g. *for all paths ...* or *there exists a (maximal) path ...*), thus providing for the formulation of reachability constraints. The logic supports temporal operators such as *always*, *eventually* and *leads to*, making it fairly easy to specify both safety and liveness properties of the system. The UPPAAL Tutorial [BDL] also shows how it is possible to specify bounded liveness properties, i.e. that some state must be reached within a fixed time limit.

The model checker works by exploring the state space of the model over the symbolic states represented by the constraints. Hence, it suffers from what is known as *state space explosion*: The exponential burden of enumerating and analyzing the entire reachable state space. However, in practice, performance of the UPPAAL tool appears good, and it has been applied to a number of real-world cases with success, including analysis of LEGO Mindstorms programs and an audio/video protocol from Bang & Olufsen. In [LP00], its creators boast that from 1995-2000, UPPAAL has increased its performance 10-fold every 9 months. One would expect that in the future, UPPAAL will continue to remain one of the strongest modeling and validation tools.

1.2 Contributions

Our main contribution to the area of validation of real-time systems is the implementation of a model checker for DC. The model checker is based the an article [MF02] by Fränzle, who also provided a prototype. Our efforts have been put into making an efficient implementation, while also extending the algorithm given and providing formal and informal arguments for its correctness. In addition, we have given an algorithm for determining a temporal bound for a large class of formulas for which bounded validity implies general validity.

1.3 How to Read This Thesis

The next chapter introduces the key notions of *model construction*, *duration calculus* and *SAT solving*.

Chapter 3 contains the core algorithm that translates DC formulas to propositional or linear constraint satisfiability problems. The algorithm is developed in a stepwise manner, while we argue for the improvement made by each step.

Then, Chapter 4 presents some simplifications — both on the DC formulas and the constraints to which they are translated — that may help decrease the size of the problem.

Chapter 5 states that a particular class of DC formulas has models of elementary length in the size of the formula, and gives an algorithm for determining that model length. This is important because bounded model checking may be used to prove validity of these formulas, not just come up with counter examples if the formula is invalid.

In Chapter 6, we introduce the high-level design of the BMC/DCValidator. Then, in Chapter 7, we go into details with the implementation, which is based on the ideas and algorithms of Chapters 3 to 5.

Chapter 8 will go through the extensive functional tests that have been performed on the tool, while Chapter 9 presents the results of testing the performance of the BMC/DCValidator with different combinations of options.

Chapter 10 discusses the results of the project and compares some related model checking approaches, while Chapter 11 finally offers a conclusion.

Chapter 2

Foundations

This chapter will sketch the foundations on which we build our work: Syntax and semantics for the duration calculus logic, unbounded and bounded model construction, and the mechanics of SAT solving.

2.1 Duration Calculus

Duration calculus (DC) is a logic for reasoning about real-time systems at a high abstraction level, and was introduced in [CHR91] as a result of the ProCoS project. It has been applied in numerous case studies, one of which we shall use as an example in this report, and as a benchmark of the BMC/DCValidator tool.

DC reasons about *durations* of *state assertions*. As described in [ZH04], states are functions over time. We shall only deal with boolean functions, giving us boolean states. Correspondingly, we have chosen a discrete model of time by using the natural numbers as time domain. The reason for this choice is that DC interpreted over continuous time, i.e. over the real numbers, turns out to be undecidable unless the syntax is severely restricted.

First, we will proceed with an example, and then provide a specification of the DC syntax and semantics.

2.1.1 Example

As an example of what DC can be used for, we can state the safety requirement for the gas burner of Section 1.1.1 on page 2, namely that for any period of 30 seconds, it must leak at most 5 seconds. A leak is defined as the gas being switched on (represented by state *gas* having the value *true*) while the flame is not burning (represented by state *flame* having the value *false*).

$$\Box\left(\ell \leq 30 \Rightarrow \int \text{gas} \wedge \neg\text{flame} \leq 5\right)$$

A DC formula is interpreted over a time interval. The $\Box$ symbol should be read as "for all subintervals". ℓ gives the length of the interval in question.

2.1.2 Syntax

In the following, the symbol $s \in State$ will denote a state. The syntax of DC as we shall use it in this report is:

$$
\begin{array}{lll}
\phi & ::= & \int S \geq 1 \mid \\
 & & \texttt{true} \mid \texttt{false} \mid \\
 & & \neg\phi \mid \\
 & & (\phi \wedge \phi) \mid \\
 & & (\phi \vee \phi) \mid \\
 & & (\phi \frown \phi) \\
S & ::= & s \mid \\
 & & \texttt{true} \mid \texttt{false} \mid \\
 & & \neg S \mid \\
 & & (S \wedge S) \mid \\
 & & (S \vee S)
\end{array}
$$

The formulas generated by ϕ are DC formulas. First, and foremost, we have the duration formulas $\int S \geq 1$, stating that *state assertion* S must hold at least one time instant.

The $\frown$ operator is pronounced "chop". $\phi_1 \frown \phi_2$ indicates the splitting of the time interval in which $\phi_1 \frown \phi_2$ must hold into two adjacent subinterval in which ϕ_1 and ϕ_2 must hold, respectively. E.g. the duration formula $\int S_1 \geq 1 \frown \int S_2 \geq 1$ states that first S_1 must hold for at least 1 time instant, and then S_2 must hold for at least 1 time instant. Note that this is different from the formula $\int S_1 \geq 1 \wedge \int S_2 \geq 1$ in which the two subintervals where $\int S_1 \geq 1$ and $\int S_2 \geq 1$ must hold, respectively, may freely overlap.

Formal semantics of the DC formulas shall be given next.

2.1.3 Semantics

We shall use the symbol DC to denote the set of DC formulas. Correspondingly, the symbol SA will denote the set of state assertions.

DC formulas are intepreted over *trajectories*. A trajectory is a function in $Traj \stackrel{\text{def}}{=} Time \rightarrow State \rightarrow \mathbf{B}$, i.e. given a time instant it gives a valuation of the states. We have chosen to use a discrete representation of time, namely $Time = \mathbf{N}$.

DC formulas shall furthermore be interpreted over a specific (closed) time interval. A trajectory in combination with a time interval is often referred to as an *observation*. Now, we will define what it means for an observation $\rho, [i, j]$ with $\rho \in Traj$ and $i, j \in Time$ to *satisfy* a formula $\phi \in DC$, denoted $\rho, [i, j] \models \phi$. We shall use "iff" as short for "if and only if".

$$
\begin{array}{lcll}
\rho,[i,j] & \models & \texttt{true} & \\
\rho,[i,j] & \not\models & \texttt{false} & \\
\rho,[i,j] & \models & \int S \geq 1 & \text{iff} \quad \bigvee_{m=i}^{j-1} \mathcal{I}[\![S]\!](\rho(m)) \\
\rho,[i,j] & \models & \neg\phi & \text{iff} \quad \rho,[i,j] \not\models \phi \\
\rho,[i,j] & \models & \phi_1 \wedge \phi_2 & \text{iff} \quad (\rho,[i,j] \models \phi_1) \wedge (\rho,[i,j] \models \phi_2) \\
\rho,[i,j] & \models & (\phi_1 \vee \phi_2) & \text{iff} \quad (\rho,[i,j] \models \phi_1) \vee (\rho,[i,j] \models \phi_2) \\
\rho,[i,j] & \models & (\phi_1 \frown \phi_2) & \text{iff} \quad \exists m \in \mathit{Time} \cdot m \in [i,j] \Rightarrow \\
 & & & \quad (\rho,[i,m] \models \phi_1) \wedge (\rho,[m,j] \models \phi_2)
\end{array}
$$

where $\mathcal{I}[\![S]\!](\sigma) \in \mathbf{B}$ is an interpretation of the state assertion S given a valuation $\sigma \in \mathit{State} \to \mathbf{B}$ of the states. We can define it as

$$
\begin{array}{lcl}
\mathcal{I}[\![\texttt{true}]\!](\sigma) & \stackrel{\text{def}}{=} & \mathit{true} \\
\mathcal{I}[\![\texttt{false}]\!](\sigma) & \stackrel{\text{def}}{=} & \mathit{false} \\
\mathcal{I}[\![s]\!](\sigma) & \stackrel{\text{def}}{=} & \sigma(s) \text{ for } s \in \mathit{State} \\
\mathcal{I}[\![(\neg S_1)]\!](\sigma) & \stackrel{\text{def}}{=} & \neg \mathcal{I}[\![S_1]\!](\sigma) \\
\mathcal{I}[\![(S_1 \wedge S_2)]\!](\sigma) & \stackrel{\text{def}}{=} & \mathcal{I}[\![S_1]\!](\sigma) \wedge \mathcal{I}[\![S_2]\!](\sigma) \\
\mathcal{I}[\![(S_1 \vee S_2)]\!](\sigma) & \stackrel{\text{def}}{=} & \mathcal{I}[\![S_1]\!](\sigma) \vee \mathcal{I}[\![S_2]\!](\sigma)
\end{array}
$$

We shall say that a trajectory $\rho \in \mathit{Traj}$ *satisfies* ϕ iff all prefix observations of ρ satisfy ϕ, formally

$$\rho \models \phi \text{ iff } \forall\, t \in \mathit{Time} \cdot \rho,[0,t] \models \phi$$

A formula ϕ is said to be *valid* iff $\forall\, \rho \in Traj \cdot \rho \models \phi$.

The trajectories satisfying ϕ are commonly referred to as the *models* of ϕ. Correspondingly, ϕ is valid if all trajectories in $Traj$ are models of ϕ or, in other words, if $\neg\phi$ has no models. This definition of validity of a DC formula does not take us much closer to a decision procedure, as we are dealing with infinite trajectories. Fortunately, we can introduce the related concept of finite traces and show a close correspondance between the two.

A finite *trace* $tr \in (\mathit{State} \to \mathbf{B})^*$ is said to *satisfy* ϕ when there is a corresponding observation that satisfies ϕ. Formally:

$$
\begin{array}{lcl}
tr \models \phi & \text{iff} & \exists \rho \in \mathit{Traj} \cdot \\
 & & (\forall t \in Time \cdot t \in [0, len(tr)-1] \Rightarrow tr(t) = \rho(t)) \wedge \\
 & & \rho,[0,len(tr)-1] \models \phi
\end{array}
$$

The traces satisfying a formula ϕ are referred to as the *finite models* of ϕ. Lemma 1 of [MF02] states that *A DC formula ϕ is valid iff $\neg\phi$ has no finite model.* To see why this is true, recall that by definition, ϕ is invalid if there exists a $\rho \in \mathit{Traj}$ and a $t \in Time$ so that $\rho,[0,t] \models \neg\phi$. This corresponds exactly to the definition of satisfiability of traces, hence proving the lemma.

The lemma allows us to conclude that if we are able to find a finite trace satisfying $\neg\phi$, we know that ϕ is invalid. However, we may search infinitely long for such a trace if ϕ is valid, so the lemma is no help in this respect. Fortunately, Chapter 5 on page 54 allows us to determine a bound on the model length for a large class of DC formulas.

2.1.4 Extended Syntax

In addition to the syntax given in Section 2.1.2 on page 7, we shall sometimes use an extended syntax that contains some useful abbreviations (some of which were used in the example in Section 2.1.1).

$\int S \geq n$	$\stackrel{\text{def}}{=}$	$\underbrace{\int S \geq 1 \frown \ldots \frown \int S \geq 1}_{n \text{ times}}$	meaning that S must hold at least n time instants
$\int S < n$	$\stackrel{\text{def}}{=}$	$\neg \int S \geq n$	
$\int S \leq n$	$\stackrel{\text{def}}{=}$	$\neg \int S \geq n + 1$	
$\int S > n$	$\stackrel{\text{def}}{=}$	$\int S \geq n + 1$	
$\int S = n$	$\stackrel{\text{def}}{=}$	$\int S \geq n \wedge \int S \leq n$	
$\ell \ op \ n$	$\stackrel{\text{def}}{=}$	$\int \texttt{true} \ op \ n$, $op \in \{>, \geq, \leq, <, =\}$	ℓ gives the length of the time interval
$\phi \Rightarrow \psi$	$\stackrel{\text{def}}{=}$	$\neg\phi \vee \psi$	
$\phi \Leftrightarrow \psi$	$\stackrel{\text{def}}{=}$	$((\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi))$	
$\Diamond\phi$	$\stackrel{\text{def}}{=}$	$(\texttt{true} \frown \phi \frown \texttt{true})$	stating that ϕ will hold for some subinterval (pronounced "eventually ϕ")
$\Box\phi$	$\stackrel{\text{def}}{=}$	$\neg\Diamond(\neg\phi)$	stating that ϕ must hold for all subintervals (pronounced "always ϕ")

2.2 Model Construction

In this section, we shall describe unbounded and bounded model construction for DC. In Appendix B on page 99, the reader may find a brief introduction to the complexity classes P, NP, PSPACE and (NON-)ELEMENTARY that are referenced in this section.

2.2.1 Unbounded Model Construction

Unbounded model construction is the task of, for some DC formula ϕ, to construct a finite model of ϕ, or rule out the existence of a such.

We shall assume a discrete interpretation, i.e. $\mathit{Time} = \mathbf{N}$, as the continuous case turns out to be undecidable as referenced in [MF02]. We can show the decidability of discrete DC through a reduction to the emptiness problem of star-free regular languages that has shown to be decidable. The reduction is given below, with R mapping DC formulas ϕ to star-free regular expresssions over $\alpha \stackrel{\text{def}}{=} V \to \mathbf{B}$ where V is a superset of the free variables of ϕ. These variables are selected from *State*, the set of boolean states.

$$\begin{aligned}
R[\![\texttt{true}]\!] &\stackrel{\text{def}}{=} \alpha^* \\
R[\![\texttt{false}]\!] &\stackrel{\text{def}}{=} \emptyset \\
R[\![\textstyle\int S \geq 1]\!] &\stackrel{\text{def}}{=} \alpha^* \left(\cup_{s \in \tilde{S}}\right) \alpha^* \\
R[\![\neg\phi]\!] &\stackrel{\text{def}}{=} \overline{R[\![\phi]\!]} \\
R[\![\phi \wedge \psi]\!] &\stackrel{\text{def}}{=} R[\![\phi]\!] \cap R[\![\psi]\!] \\
R[\![\phi \vee \psi]\!] &\stackrel{\text{def}}{=} R[\![\phi]\!] \cup R[\![\psi]\!] \\
R[\![\phi \frown \psi]\!] &\stackrel{\text{def}}{=} R[\![\phi]\!] \cdot R[\![\psi]\!]
\end{aligned}$$

where $\tilde{S}$ contains the valuations for which S holds.

We can now prove by induction on the structure of the formula ϕ that

$$tr \models \phi \text{ iff } w \in \mathcal{L}_{R[\![\phi]\!]} \text{ for } w \stackrel{\text{def}}{=} \langle tr(0)|_V, tr(1)|_V, \ldots, tr(\text{len}(tr)-1)|_V \rangle$$

Basis:

- Case $\phi = \texttt{true}$
  $\mathcal{L}_{R[\![\texttt{true}]\!]} = \alpha^*$ so for all tr, $w \in \mathcal{L}_{R[\![\texttt{true}]\!]}$ and $tr \models \texttt{true}$.
- Case $\phi = \texttt{false}$
  Trivial, since $\mathcal{L}_{R[\![\texttt{false}]\!]} = \emptyset$ and $\neg\exists tr \cdot tr \models \texttt{false}$.
- Case $\phi = \int S \geq 1$
  Recall that the semantics for $\int S \geq 1$ gives us that $tr \models \int S \geq 1$ iff $\bigvee_{t=0}^{\text{len}(tr)-1} \mathcal{I}[\![S]\!](tr(t))$. This means that that $tr \models \int S \geq 1$ iff $\exists i \cdot \mathcal{I}[\![S]\!](tr(i))$, giving us that $tr \models \int S \geq 1$ iff $w \in \alpha^* \left(\cup_{s \in \tilde{S}} s\right) \alpha^*$ where $\tilde{S}$ is the set of valuations that satisfy S.

Induction:

- Case $\phi = \neg\psi$
  By induction, $tr \models \psi$ iff $w \in \mathcal{L}_{R[\![\psi]\!]}$. The semantics gives us that $tr \models \neg\psi$ iff $w \notin \mathcal{L}_{R[\![\psi]\!]}$ or, equivalently, $tr \models \neg\psi$ iff $w \in \mathcal{L}_{\overline{R[\![\psi]\!]}}$.

- Case $\phi = \phi_1 \wedge \phi_2$
  By induction, $tr \models \phi_1$ iff $w \in \mathcal{L}_{R[\![\phi_1]\!]}$ and $tr \models \phi_2$ iff $w \in \mathcal{L}_{R[\![\phi_2]\!]}$. The semantics gives us that $tr \models \phi_1 \wedge \phi_2$ iff $tr \models \phi_1$ and $tr \models \phi_2$, so conclusively we get that $tr \models \phi_1 \wedge \phi_2$ iff $w \in \left(\mathcal{L}_{R[\![\phi_1]\!]} \cap \mathcal{L}_{R[\![\phi_2]\!]}\right) = \mathcal{L}_{R[\![\phi_1]\!] \cap R[\![\phi_2]\!]}$.

- Case $\phi = \phi_1 \vee \phi_2$
  Equivalent.

- Case $\phi = \phi_1 \frown \phi_2$
  By induction, $tr \models \phi_1$ iff $w \in \mathcal{L}_{R[\![\phi_1]\!]}$ and $tr \models \phi_2$ iff $w \in \mathcal{L}_{R[\![\phi_2]\!]}$.

  The semantics gives us that $tr \models \phi_1 \frown \phi_2$ iff there exist traces tr_1 and tr_2 with tr being the concatenation of tr_1 and tr_2 so that $tr_1 \models \phi_1$ and $tr_2 \models \phi_2$. Now, $tr \models \phi_1 \wedge \phi_2$ iff $w \in \mathcal{L}_{R[\![\phi_1]\!] \cdot R[\![\phi_2]\!]}$.

The reduction from validity of DC formulas to emptiness of regular languages that was given above generates a star-free regular expression of size at most $2^{|free(\phi)|}|\phi$ where $free(\phi)$ denotes the free (state) variables of ϕ. However, since checking emptiness of regular languages is non-elementary, we have hereby shown that the checking validity of a DC formula is *decidable*, and *at most non-elementary*. Now, we shall show that it is also *at least non-elementary*, using a reduction from emptiness of extended regular languages to deciding existence of a finite model to a DC formula, since it is known that the emptiness problem of extended regular languages is non-elementary.

The mapping F from exended regular expressions re to DC formulas is given below, with the alphabet of the regular expressions being $\alpha = V \to \mathbf{B}$, and $V \subset State$ being finite. The letters $a \in \alpha$ are encoded as duration formulas with a state assertion S_a and a bound on the model length by $\ell < 2$. The state assertion S_a is:

$$S_a \stackrel{\text{def}}{=} \bigwedge_{\substack{u \in V, \\ a(u)=\texttt{true}}} u \quad \wedge \quad \bigwedge_{\substack{v \in V, \\ a(v)=\texttt{false}}} \neg v$$

A word $w = \langle w_1, \ldots, w_n \rangle \in \alpha^*$ will be encoded by finite traces tr that first satisfy S_{w_1}, then S_{w_2} etc.

$$\begin{aligned}
F[\![a]\!] &\stackrel{\text{def}}{=} \textstyle\int S_a \geq 1 \wedge \ell < 2 \\
F[\![re_1 \cap re_2]\!] &\stackrel{\text{def}}{=} F[\![re_1]\!] \wedge F[\![re_2]\!] \\
F[\![re_1 \cdot re_2]\!] &\stackrel{\text{def}}{=} F[\![re_1]\!] \frown F[\![re_2]\!] \\
F[\![\overline{re_1}]\!] &\stackrel{\text{def}}{=} \neg F[\![re_1]\!]
\end{aligned}$$

The bound on the model length imposed when translating $a \in \alpha$ is necessary to be able to conclude that $tr \models F[\![re]\!]$ only if $w \in \mathcal{L}_{re}$ for $w \in \langle tr(0)|_V, \ldots, tr(\text{len}(tr)-1)|_V \rangle$.

We shall omit the proof, as the construction is quite similar to the "reverse" reduction, and a proof by induction could be given in the same manner as above.

2.2.2 Bounded Model Construction

It should be apparent from Section 2.2.1 that checking validity of DC formulas using unbounded model construction poses severe problems due to its non-elementary complexity.

Instead, we shall introduce *bounded model construction* (BMC), defined as the problem of, given a DC formula ϕ and a bound on the model length k to assign to ϕ a model of length at most k iff such a model exists. In short, we shall refer to an instance of the BMC problem as BMC(ϕ, k).

In [MF02], Fränzle states that: *For each* $k \geq 1$, BMC(ϕ, k) *is NP-hard in* $|\phi|$ and proves it by a simple reduction of satisfiability of propositional formulas to satisfiability of DC formulas by noting that a propositional formula S is satisfiable if and only if $\int S \geq 1$ is satisfiable for $k \geq 1$.

To show that BMC belongs to the complexity class NP, [MF02] gives a polynomial reduction to propositional logic, as satisfiabiliy of propositional logic is NP-complete. We shall not repeat this reduction here, as a modified version of the encoding into propositional logic is described in Chapter 3 on page 16, forming the core of the BMC/DCValidator tool.

2.3 SAT Solving

We shall later (in Chapter 3 on page 16, to be precise) show how one may validate DC formulas by translating them to constraint systems and solving these. This section shall present the basics of how such systems are solved, because this provides a foundation for understanding the effect that different options available for the translation in Chapter 3 will have on the time required to solve the constraint system that the translation results in.

2.3.1 Some Definitions and Notation

First, we must introduce some terms that will be used in the following. Most definitions has been taken from [FH03]: *A literal* x_i is a signed propositional variable. *A propositional clause* is a disjunction of literals $x_1 \vee x_2 \vee \ldots x_n$. *A linear threshold clause* is an inequality of the form $a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq n$ where the $a_i \in \mathbf{N} \setminus \{0\}$ are the *weights* of the literals and the $n \in \mathbf{N}$ is the *threshold*. Linear threshold clauses are also known as *zero-one linear constraints* and we will often speak of *zero-one linear constraint systems (ZOLCS)* which are conjunctions of such constraints. Generally, we shall use the terms *constraint* and *clause* interchangeably.

A valuation σ is a total function $V \stackrel{total}{\rightarrow} \mathbf{B}$ from the set of propositional variables V to boolean values $\mathbf{B}$. We shall also speak of a *partial valuation* ρ which is a partial function $V \stackrel{part}{\rightarrow} \mathbf{B}$.

SAT solving is the satisfiability checking of a system of clauses P, be that either propositional clauses or linear threshold clauses.

A valuation σ *satisfies* a conjunction of clauses P, denoted $\sigma \models P$, iff σ satisfies each of the clauses of P. A partial valuation ρ is said to be *consistent for* P iff there exists an extension of ρ to a valuation σ that satisfies P, and *inconsistent for* P otherwise. We shall also say that ρ satisfies P, denoted $\rho \models P$ iff all extensions of ρ to a total valuation σ satisfy P.

σ satisfies a linear threshold clause $a_1x_1 + a_2x_2 + \ldots a_nx_n \geq n$ iff $a_1\chi_\sigma(x_1) + a_2\chi_\sigma(x_2) + \ldots + a_n\chi_\sigma(x_n) \geq n$ where $\chi_\sigma \in V \rightarrow \mathbf{B}$ is defined as

$$\begin{array}{lcl} \chi_\sigma(x) & = & 0 \text{ if } \sigma(x) = \texttt{false} \\ \chi_\sigma(x) & = & 1 \text{ if } \sigma(x) = \texttt{true} \\ \chi_\sigma(\neg x) & = & 1 \text{ if } \sigma(x) = \texttt{false} \\ \chi_\sigma(\neg x) & = & 0 \text{ if } \sigma(x) = \texttt{true} \end{array}$$

2.3.2 DPLL Algorithm

The core of most contemporary SAT solvers is the Davis-Putnam-Loveland-Logemann (DPLL) algorithm that was presented in [DLL62], extending [DP60].

When checking the satisfiability of a propositional formula P, the algorithm proceeds by extending a partial valuation ρ until it either becomes a total valuation that satisifies P, or ρ turns out to be inconsistent for P. ρ may be extended through either *unit resolution* or *deduction steps*, the latter being a more or less random[1] assignment of a truth value to one of the unassigned variables. These assignments may later be reversed through *backtracking* if they turn out to be inconsistent.

[1] Naturally, heuristics are available.

The algorithm works as follows:

1. First, the solver performs *unit resolution* in which it searches for *unit clauses* in P. A unit clause in CNF is a clause in which all of the literals have been assigned the value *false*, except for one unassigned literal, referred to as the *unit literal*. The corresponding variable must consequently be assigned the value *true* if the literal has positive sign, and *false* otherwise. The issue of unit clauses in ZOLCS will be addressed in Section 2.3.3.

2. When unit resolution can no longer proceed, the solver searches for the existence of any *conflicting clauses* in P. A conflicting clause is a clause all of whose literals are assigned the value *false*. This means that ρ is inconsistent for P.

3. If a conflicting clause is found, *backtracking* is begun. It reverses the last assignment made in a decision step and the assignments made by unit resolution as a consequence of this choice. Some versions of the algorithm use *non-chronological backtracking*, meaning that it is not necessarily the last assignment that is reversed.

4. If no conflicting clauses were found, a *decision step* is taken. In this step, an unassigned variable is selected and is assigned a truth value. Heuristics are available for determining the variable to select and the value it should be given.

The algorithm terminates when either

a) ρ has been extended to a total valuation σ that satisfies P, or,

b) All decision variables have been assigned both truth values, without leading to a solution.

2.3.3 SAT Solver Input Formats

We shall consider two input formats for the SAT solver: CNF and ZOLCS. Often, we will also refer to the CNF format as DIMACS since this is the name of the ASCII encoding of CNF.

The SAT solver can process an input in CNF in a straightforward manner using the DPLL algorithm described in Section 2.3.2 on the preceding page. The drawback of CNF is that a propositional formula P can grow exponentially in size when rewritten to CNF. Algorithms (e.g. as given in [PG86]) are available that construct a CNF that is $\mathcal{O}(|P|)$ where $|P|$ is the size of P measured as the number of subformulas in P. In turn, such schemes generate $\mathcal{O}(|P|)$ auxiliary variables which can give an exponential increase in the size of the search tree for backtracking.

The ZOLCS format allows for a more concise representation that can lower the number of auxiliary variables and constraints needed to describe a problem. Using this format requires some changes to the DPLL algorithm. These changes were first suggested by Barth in [Ba95].

We have previously described how unit clauses in P force the value of the unit literal to *true*. This is also called *propagating* the assignment of the unit

literal. In ZOLCS, the condition is not that all of the other literals must have the value *false*, but rather that the sum of the coefficients of the literals whose value is *true* or unassigned cannot reach the threshold without addition of the coefficient of the literal in question. Note that a ZOLCS equation may propagate more than one literal, as in $1x_1 + 1\neg x_2 + 3x_3 \geq 4$ that propagates $\neg x_2$ and x_3 when x_1 has been assigned the value *false*.

2.3.4 HySat

Our tool for DC validation shall integrate with HySat, a bounded model checker for hybrid systems by Fränzle and Herde [FH04]. Its core is a pseudo-boolean SAT solver implementing the DPLL algorithm in combination with a linear programming package, allowing it to support a mixture of zero-one linear constraints and guarded linear constraints, i.e. constraints over the real variables.

HySat employs a number of optimisations for bounded model checking (BMC) by letting the SAT solver exploit the symmetry of the SAT problems generated by BMC frontends. The symmetry arises from isomorphic constraints over different time intervals.

We shall not take advantage of HySat's ability to handle hybrid systems but only of its efficiency on the type of SAT problems that we will generate.

Chapter 3

DC → SAT Translation

In the following, we shall describe how we translate a DC formula to a SAT problem that is satisifiable if and only if the DC formula is. First, we present a simple version of the algorithm and prove its correctness. Then, a number of improvements are added in an incremental fashion and argued informally for. The algorithm is shown in full in Appendix C on page 101.

3.1 Initial Remarks

When describing the translation, we shall consider occurrences of formulas rather than the formulas themselves, so that when P occurs multiple times in a larger formula such as in the formula $P \wedge \neg P$, we shall treat the two occurrences of P independently. In the following, when we speak of "a formula P" we actually mean "an occurrence of the formula P".

We shall not deal explicitly with the constructs presented in Section 2.1.4 on page 9 covering the extended DC syntax, as rules have been given for rewriting those to the basic syntax. E.g. the reader should imagine that formulas of type $p \Rightarrow q$ are rewritten into $\neg p \vee q$ and that formulas of type $p \Leftrightarrow q$ have been rewritten to $(\neg p \vee q) \wedge (p \vee \neg q)$, now containing two occurrences of p and q.

3.2 Tseitin Variables

The key idea of the algorithm is to use auxiliary propositional variables to describe the truth value of DC (sub-)formulas on given intervals. Signed variables will be termed *literals*. We will use the symbols Lit_{DC} and Lit_{SA} to denote the set of literals representing DC and SA formulas, respectively. The symbol Lit will denote the union of Lit_{DC} and Lit_{SA}.

Specifically, a literal $[\phi]_{i,j} \in Var_{DC}$ will represent the truth value of DC formula ϕ on interval $[i, j[$. Our implementation must ensure unique literals $[\phi]_{i,j}$ for all DC formulas ϕ and indices $i, j \in Time$ where $i \leq j$. Similarly, literal $[S]_i \in Lit_{SA}$ will represent the truth value of state assertion S on interval $[i, i+1[$. These literals must also be unique and not collide with the variables for the DC formulas.

The idea of introducing such literals stems from Tseitin [Tse68] who showed that the complexity of rewriting propositional satisfiability problems to CNF

can be reduced to linear time by this method.

3.3 Initial Construction of a SAT Problem

In the following, we shall term our translation scheme $t^k_{\mathrm{DC}} \in \overline{Lit}$ where $\overline{Lit}$ denotes the set of boolean combinations of the literals in Lit.

To obtain a SAT problem that is satisfiable if and only if the DC formula is, we must generate constraints in the form of "definitions" of the literals to ensure their correspondance. E.g. the value of $[\phi_1 \wedge \phi_2]_{0,2}$ depends on $[\phi_1]_{0,2}$ and $[\phi_2]_{0,2}$ as follows:

$$[\phi_1 \wedge \phi_2]_{0,2} \Leftrightarrow [\phi_1]_{0,2} \wedge [\phi_2]_{0,2}$$

So in the general case of a conjunction of DC formulas, we must therefore add constraints

$$[\phi_1 \wedge \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}$$

for all $i \in \{0, \ldots, k\}$ and $j \in \{i, \ldots, k\}$, as well as constraints describing ϕ_1 and ϕ_2 by defining the literals of type $[\phi_1]_{i,j}$ and $[\phi_2]_{i,j}$. Hence, our translation scheme t^k_{DC} should be recursive over the DC formulas.

For conjunctions $\phi_1 \wedge \phi_2$, we then get

$$t^k_{\mathrm{DC}}[\![\phi_1 \wedge \phi_2]\!] \quad = \quad t^k_{\mathrm{DC}}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC}}[\![\phi_2]\!] \wedge \bigwedge_{\substack{i \in \{0,\ldots k\} \\ j \in \{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j})$$

The case for disjunctions $\phi_1 \vee \phi_2$ is equivalent. In a similarly straightforward way, we can handle negations $\neg\phi_1$:

$$t^k_{\mathrm{DC}}[\![\neg\phi_1]\!] \quad = \quad t^k_{\mathrm{DC}}[\![\phi_1]\!] \wedge \bigwedge_{\substack{i \in \{0,\ldots k\} \\ j \in \{i,\ldots,k\}}} ([\neg\phi_1]_{i,j} \Leftrightarrow \neg[\phi_1]_{i,j})$$

For the chop modality, the setting is slightly more difficult. $\phi_1 \frown \phi_2$ holds on interval $[i, j[$ if and only if we can find a point $m \in Time$ with $i \leq m \leq j$ so that ϕ_1 holds on interval $[i, m[$ and ϕ_2 holds on interval $[m, j[$. So $[\phi_1 \frown \phi_2]_{i,j}$ must be true if and only if $\exists m \in \{i, \ldots, j\} \cdot [\phi_1]_{i,m} \wedge [\phi_2]_{m,j}$, leading to

$$\begin{aligned} t^k_{\mathrm{DC}}[\![\phi_1 \frown \phi_2]\!] \quad = \quad & t^k_{\mathrm{DC}}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC}}[\![\phi_2]\!] \wedge \\ & \bigwedge_{\substack{i \in \{0,\ldots k\} \\ j \in \{i,\ldots,k\}}} \left([\phi_1 \frown \phi_2]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j} [\phi_1]_{i,m} \wedge [\phi_2]_{m,j} \right) \end{aligned}$$

As defined in Section 3.2 on the page before, the literal $[S]_i$ represents the truth value of state assertion S on observation interval $[i, i+1]$ for $i \in \{0, \ldots, k-1\}$.

So, for $i \in \{0, \ldots, k-1\}$ and $j \in \{i+1, \ldots, k\}$ we must require that

$$[\textstyle\int S \geq 1]_{i,j} \Leftrightarrow [S]_i \vee [S]_{i+1} \vee \ldots \vee [S]_{j-1}$$

If the interval is of length zero, naturally the duration of any state assertion cannot be greater than or equal to one, so

$$\forall i \in \{0, \ldots, k\} \cdot \neg[\textstyle\int S \geq 1]_{i,i}$$

Summing up, we get

$$\begin{aligned} t^k_{\mathrm{DC}}\,[\![\textstyle\int S \geq 1]\!] \quad = \quad & t^k_{\mathrm{SA}}\,[\![S]\!] \wedge \bigwedge_{\substack{i \in \{0,\ldots k-1\} \\ j \in \{i+1,\ldots,k\}}} \left([\textstyle\int S \geq 1]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j-1} [S]_m \right) \wedge \\ & \bigwedge_{i \in \{0,\ldots k\}} \neg[\textstyle\int S \geq 1]_{i,i} \end{aligned}$$

where t^k_{SA} translates state assertions similarly to the translation presented for DC formulas.

The translation t^k_{DC} generates no definitions when encountering a DC formula `true` or `false`. Instead, we once and for all generate a literal [`true`] and define it so that the SAT problem is satisfiable only if the value of this literal is true. Then, when e.g. $[\texttt{true}]_{i,j}$ is referenced, the generic literal [`true`] is used. The value `false` is simply represented as $\neg[\texttt{true}]$.

Finally, we must require that the SAT problem is satisfiable only when the value of one of the literals representing the formula itself on a prefix of the observation interval $[0, k]$ is true. So if we term our main translation function BMC we get that

$$BMC\,[\![\phi, k]\!] = t^k_{\mathrm{DC}}\,[\![\phi]\!] \wedge \left(\bigvee_{i=0}^{k} [\phi]_{0,i} \right) \wedge [\texttt{true}]$$

The algorithm is listed in full in Appendix C.1 on page 102.

3.4 Correctness of Translation Algorithm

3.4.1 Main Theorem

We want to show

Theorem 1 *A DC formula ϕ has a model of length at most k iff* $\mathrm{BMC}(\phi, k)$ *is satisfiable.*

First, we will need to establish and prove a number of lemmas, which will be done in Section 3.4.2. In Section 3.4.3 on page 35 we shall finally prove Theorem 1.

3.4.2 Proving Some Necessary Lemmas

It should be quite obvious that we shall need separate lemmas to express properties of the state assertions, and that these lemmas will be used in the proofs of the lemmas regarding DC formulas.

Our first approach at constructing this proof was naively based on just two lemmas (one for SA and one for DC formulas) of the type

$$\begin{aligned}&\textit{There exists a trajectory } \rho \textit{ such that } \rho, [i,j] \models \phi \textit{ iff}\\ &t^k\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}] \textit{ is satisfiable}\end{aligned}$$

A straightforward proof by induction rather quickly was established for the cases of `true`, `false`, $\int S \geq 1$ (using on the lemma for state assertions), $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ and $\phi_1 \frown \phi_2$. But for $\neg\phi_1$ problems arose: If we assumed that $\rho, [i,j] \models \neg\phi_1$ then we knew that $\rho, [i,j] \not\models \phi_1$ but had no chance of proving that there did not exist some *other* trajectory ρ' for which $\rho', [i,j] \models \phi_1$. The conclusion of this was, unfortunately, that a much more complicated set of lemmas had to be constructed. The heart of these lemmas is the use of functions, building on valuations and trajectories, so that the proofs may be about function values and not require existential quantifiers.

First, we will state and prove the lemmas regarding state assertions. Then, we will state and prove the lemmas regarding DC formulas, with the majority being similar to the lemmas on state assertions. Keep in mind that the goal is still to show the close correspondance between interpretation and satisfiability — the path is just not as straightforward as in the original idea!

Lemmas Regarding State Assertions

We shall begin by defining a function

$$\hat{\sigma} \in \mathit{Lit}_{\mathrm{SA}} \to \mathbf{B}$$

that extends

$$\sigma \in \mathit{State} \to \mathbf{B}$$

to give the value of a literal representing the value of a state assertion at a given time instant. Remember that by definition $[\texttt{true}]_i = [\texttt{true}]$ and $[\texttt{false}]_i = \neg[\texttt{true}]$.

$$\begin{aligned}
\hat{\sigma}([\texttt{true}]) &= \mathit{true} \\
\hat{\sigma}([s]_i) &= \sigma(s) \text{ for } s \in \mathit{State} \\
\hat{\sigma}([\neg S]_i) &= \neg\hat{\sigma}([S]_i) \\
\hat{\sigma}([S_1 \wedge S_2]_i) &= \hat{\sigma}([S_1]_i) \wedge \hat{\sigma}([S_2]_i) \\
\hat{\sigma}([S_1 \vee S_2]_i) &= \hat{\sigma}([S_1]_i) \vee \hat{\sigma}([S_2]_i)
\end{aligned}$$

Furthermore, we shall extend $\hat{\sigma}$ to $\overline{\hat{\sigma}} \in \overline{Lit_{SA}} \to \mathbf{B}$ giving the value of boolean combinations of the propositional variables:

$$\begin{aligned}
\overline{\hat{\sigma}}(\mathit{true}) &= \mathit{true} \\
\overline{\hat{\sigma}}(\mathit{false}) &= \mathit{false} \\
\overline{\hat{\sigma}}([S]_i) &= \hat{\sigma}([S]_i) \\
\overline{\hat{\sigma}}(\neg e) &= \neg\overline{\hat{\sigma}}(e) \\
\overline{\hat{\sigma}}(e_1 \wedge e_2) &= \overline{\hat{\sigma}}(e_1) \wedge \overline{\hat{\sigma}}(e_2) \\
\overline{\hat{\sigma}}(e_1 \vee e_2) &= \overline{\hat{\sigma}}(e_1) \vee \overline{\hat{\sigma}}(e_2)
\end{aligned}$$

The following Lemma indicates that $\overline{\hat{\sigma}}$ is well-defined:

Lemma 1 *$\overline{\hat{\sigma}}(e_1) = \overline{\hat{\sigma}}(e_2)$ implies $\overline{\hat{\sigma}}(e_1 \Leftrightarrow e_2) = \mathit{true}$.*

Proof:
Assume $\overline{\hat{\sigma}}(e_1) = \overline{\hat{\sigma}}(e_2)$.

$$\begin{aligned}
\overline{\hat{\sigma}}(e_1 \Leftrightarrow e_2) &\overset{(1)}{=} \overline{\hat{\sigma}}\left((\neg e_1 \vee e_2) \wedge (e_1 \vee \neg e_2)\right) \\
&\overset{(2)}{=} \left(\neg\overline{\hat{\sigma}}(e_1) \vee \overline{\hat{\sigma}}(e_2)\right) \wedge \left(\overline{\hat{\sigma}}(e_1) \vee \neg\overline{\hat{\sigma}}(e_2)\right) \\
&\overset{(3)}{=} \mathit{true}
\end{aligned}$$

with the following arguments:

(1) By definition of the '⇔' operator

(2) By definition of $\overline{\hat{\sigma}}$

(3) Using the assumption that $\overline{\hat{\sigma}}(e_1) = \overline{\hat{\sigma}}(e_2)$

In the following, we shall use the symbol $t^k_{SA,i}[\![S]\!]$ to denote the constraints of the translation $t^k_{SA}[\![S]\!]$ that regard variables with index i. By definition, $t^k_{SA}[\![S]\!] = \bigwedge_{i=0}^{k.1} t^k_{SA,i}[\![S]\!]$.

The next Lemma states the correctness of the $t^k_{SA,i}$ translation with respect to $\overline{\hat{\sigma}}$:

Lemma 2 $\overline{\hat{\sigma}}(t^k_{SA,i}[\![S]\!]) = \mathit{true}$

Proof:

We shall establish a proof by structural induction on S.

BASIS:

- Case $S = \texttt{true}$:

$$\begin{aligned}\overline{\hat{\sigma}}(t^k_{SA,i}[\![\texttt{true}]\!]) &\overset{(1)}{=} \overline{\hat{\sigma}}(\texttt{true})\\ &\overset{(2)}{=} \mathit{true}\end{aligned}$$

  with the following arguments:

  (1) By definition of $t^k_{SA,i}$

  (2) By definition of $\overline{\hat{\sigma}}$

- Case $S = \texttt{false}$:
  Equivalent.

- Case $S = s$ for $s \in \mathit{State}$:
  Equivalent.

INDUCTION:

- Case $S = \neg S_1$:

$$\begin{aligned}\overline{\hat{\sigma}}(t^k_{SA,i}[\![\neg S_1]\!]) &\overset{(1)}{=} \overline{\hat{\sigma}}\left(t^k_{SA,i}[\![S_1]\!] \wedge ([\neg S_1]_i \Leftrightarrow \neg[S_1]_i)\right)\\ &\overset{(2)}{=} \overline{\hat{\sigma}}\left(t^k_{SA,i}[\![S_1]\!]\right) \wedge \overline{\hat{\sigma}}\left([\neg S_1]_i \Leftrightarrow \neg[S_1]_i\right)\\ &\overset{(3)}{=} \overline{\hat{\sigma}}\left([\neg S_1]_i \Leftrightarrow \neg[S_1]_i\right)\\ &\overset{(4)}{=} \mathit{true}\end{aligned}$$

  with the following arguments:

  (1) By definition of $t^k_{SA,i}$

  (2) By definition of $\overline{\hat{\sigma}}$

  (3) By induction

  (4) Using Lemma 1 since by definition of $\overline{\hat{\sigma}}$ we have that $\overline{\hat{\sigma}}\left([\neg S_1]_i\right) = \neg\overline{\hat{\sigma}}\left([S_1]_i\right) = \overline{\hat{\sigma}}\left(\neg[S_1]_i\right)$

- Case $S = S_1 \wedge S_2$:

$$\begin{aligned}\overline{\hat{\sigma}}(t^k_{SA,i}[\![S_1 \wedge S_2]\!]) &\overset{(1)}{=} \overline{\hat{\sigma}}(t^k_{SA,i}[\![S_1]\!] \wedge t^k_{SA,i}[\![S_2]\!] \wedge\\ &\qquad ([S_1 \wedge S_2]_i \Leftrightarrow [S_1]_i \wedge [S_2]_i))\\ &\overset{(2)}{=} \overline{\hat{\sigma}}(t^k_{SA,i}[\![S_1]\!]) \wedge \overline{\hat{\sigma}}(t^k_{SA,i}[\![S_2]\!]) \wedge\\ &\qquad \overline{\hat{\sigma}}\left([S_1 \wedge S_2]_i \Leftrightarrow [S_1]_i \wedge [S_2]_i\right)\\ &\overset{(3)}{=} \overline{\hat{\sigma}}\left([S_1 \wedge S_2]_i \Leftrightarrow [S_1]_i \wedge [S_2]_i\right)\\ &\overset{(4)}{=} \mathit{true}\end{aligned}$$

  with the following arguments:

(1) By definition of $t^k_{SA,i}$

(2) By definition of $\overline{\hat{\sigma}}$

(3) By induction

(4) Using Lemma 1 since by definition of $\overline{\hat{\sigma}}$ we have that $\overline{\hat{\sigma}}\left([S_1 \wedge S_2]_i\right) = \overline{\hat{\sigma}}\left([S_1]_i\right) \wedge \overline{\hat{\sigma}}\left([S_2]_i\right) = \overline{\hat{\sigma}}\left([S_1]_i \wedge [S_2]_i\right)$

- Case $S = S_1 \vee S_2$: Equivalent.

Then, we must show a connection between the semantic interpretation of S, $\mathcal{I}[\![S]\!](\sigma)$, with valuation σ, and the value of $\overline{\hat{\sigma}}(S)$:

Lemma 3 *$\mathcal{I}[\![S]\!](\sigma) = true$ iff $\overline{\hat{\sigma}}([S]_i) = true$*

Proof:

We shall prove Lemma 3 by structural induction on S.

Basis:

- Case $S = \texttt{true}$:

$$\begin{aligned} \overline{\hat{\sigma}}([\texttt{true}]_i) &\overset{(1)}{=} \overline{\hat{\sigma}}([\texttt{true}]) \\ &\overset{(2)}{=} true \\ &\overset{(3)}{=} \mathcal{I}[\![\texttt{true}]\!](\sigma) \end{aligned}$$

with the following arguments:

(1) By definition of $[\texttt{true}]_i$

(2) By definition of $\overline{\hat{\sigma}}$

(3) By definition of $\mathcal{I}$

- Case $S = \texttt{false}$:
Equivalent.

- Case $S = s$ for $s \in State$:

$$\begin{aligned} \overline{\hat{\sigma}}([s]_i) &\overset{(1)}{=} \sigma(s) \\ &\overset{(2)}{=} \mathcal{I}[\![s]\!](\sigma) \end{aligned}$$

with the following arguments:

(1) By definition of $\overline{\hat{\sigma}}$

(2) By definition of $\mathcal{I}$

Induction:

- Case $S = \neg S_1$:

$$\begin{aligned}\overline{\hat{\sigma}}([\neg S_1]_i) &\overset{(1)}{=} \neg\overline{\hat{\sigma}}([S_1]_i)\\ &\overset{(2)}{=} \neg\mathcal{I}[\![S_1]\!](\sigma)\\ &\overset{(3)}{=} \mathcal{I}[\![\neg S_1]\!](\sigma)\end{aligned}$$

  with the following arguments:

  (1) By definition of $\overline{\hat{\sigma}}$
  (2) By induction
  (3) By definition of $\mathcal{I}$

- Case $S = S_1 \wedge S_2$:

$$\begin{aligned}\overline{\hat{\sigma}}([S_1 \wedge S_2]_i) &\overset{(1)}{=} \overline{\hat{\sigma}}([S_1]_i) \wedge \overline{\hat{\sigma}}([S_2]_i)\\ &\overset{(2)}{=} \mathcal{I}[\![S_1]\!](\sigma) \wedge \mathcal{I}[\![S_2]\!](\sigma)\\ &\overset{(3)}{=} \mathcal{I}[\![S_1 \wedge S_2]\!](\sigma)\end{aligned}$$

  with the following arguments:

  (1) By definition of $\overline{\hat{\sigma}}$
  (2) By induction
  (3) By definition of $\mathcal{I}$

- Case $S = S_1 \vee S_2$:
  Equivalent.

Given a valuation $\xi \in \mathit{Lit} \to \mathbf{B}$ and $i \in \mathit{Time}$, we shall define a function $\overleftarrow{\xi}$, that creates a corresponding trajectory.

$$\overleftarrow{\xi}\ (i)(s) = \xi([s]_i) \text{ for } s \in \mathit{State}$$

Then, we may state the following Lemma:

Lemma 4 *If ξ satisfies $t^k_{SA,i}[\![S]\!] \wedge [\mathtt{true}]$ then $\xi([S]_i) = \overleftarrow{\overleftarrow{\xi}\ (i)}([S]_i)$*

Proof:
We shall prove Lemma 4 by structural induction on S.
Basis:

- Case $S = \mathtt{true}$:

$$\begin{aligned}\xi([\mathtt{true}]_i) &\overset{(1)}{=} \xi([\mathtt{true}])\\ &\overset{(2)}{=} \mathit{true}\\ &\overset{(3)}{=} \overleftarrow{\overleftarrow{\xi}\ (i)}([\mathtt{true}])\end{aligned}$$

  with the following arguments:

(1) By definition of $[\texttt{true}]_i$
(2) Since ξ satisfies $t^k_{SA,i}[\![S]\!] \wedge [\texttt{true}]$
(3) By definition of $\overleftarrow{\overleftarrow{\xi}}\ (i)$

- Case $S = \texttt{false}$:
  Equivalent.
- Case $S = s \in \mathit{State}$:

$$\begin{aligned} \xi([s]_i) &\overset{(1)}{=} \overleftarrow{\xi}\ (i)(s) \\ &\overset{(2)}{=} \overleftarrow{\overleftarrow{\xi}}\ (i)([s]_i) \end{aligned}$$

with the following arguments:

(1) By definition of $\overleftarrow{\xi}\ (i)$
(2) By definition of $\overleftarrow{\overleftarrow{\xi}}\ (i)$

Induction:

- Case $S = \neg S_1$:

$$\begin{aligned} \xi([\neg S_1]_i) &\overset{(1)}{=} \neg\xi([S_1]_i) \\ &\overset{(2)}{=} \neg\overleftarrow{\overleftarrow{\xi}}\ (i)([S_1]_i) \\ &\overset{(3)}{=} \overleftarrow{\overleftarrow{\xi}}\ (i)(\neg[S_1]_i) \\ &\overset{(4)}{=} \overleftarrow{\overleftarrow{\xi}}\ (i)([\neg S_1]_i) \end{aligned}$$

with the following arguments:

(1) Since ξ satisfies $[\neg S_1]_i \Leftrightarrow \neg[S_1]_i$
(2) By induction, since by definition of $t^k_{SA,i}$, ξ satisfies $t^k_{SA,i}[\![S_1]\!] \wedge [\texttt{true}]$
(3) By definition of $\overleftarrow{\overleftarrow{\xi}}\ (i)$
(4) By definition of $\overleftarrow{\overleftarrow{\xi}}\ (i)$

- Case $S = S_1 \wedge S_2$:

$$\begin{aligned} \xi([S_1 \wedge S_2]_i) &\overset{(1)}{=} \xi([S_1]_i) \wedge \xi([S_2]_i) \\ &\overset{(2)}{=} \overleftarrow{\overleftarrow{\xi}}\ (i)([S_1]_i) \wedge \overleftarrow{\overleftarrow{\xi}}\ (i)([S_1]_i) \\ &\overset{(3)}{=} \overleftarrow{\overleftarrow{\xi}}\ (i)([S_1]_i \wedge [S_2]_i) \\ &\overset{(4)}{=} \overleftarrow{\overleftarrow{\xi}}\ (i)([S_1 \wedge S_2]_i) \end{aligned}$$

with the following arguments:

(1) Since ξ satisfies $[S_1 \wedge S_2]_i \Leftrightarrow [S_1]_i \wedge [S_2]_i$

(2) By induction, since by definition of $t^k_{SA,i}$, ξ satisfies $t^k_{SA,i}[\![S_1]\!] \wedge [\mathtt{true}]$ and $t^k_{SA,i}[\![S_2]\!] \wedge [\mathtt{true}]$

(3) By definition of $\overleftarrow{\xi}\ (i)$

(4) By definition of $\overleftarrow{\xi}\ (i)$

- Case $S = S_1 \vee S_2$:
  Equivalent.

Lemmas Regarding DC Formulas

Given a trajectory ρ we will define a function $\tilde{\rho} \in \mathit{Lit} \rightarrow \mathbf{B}$:

$$\begin{aligned}
\tilde{\rho}([\mathtt{true}]) &= \mathit{true} \\
\tilde{\rho}([\textstyle\int S \geq 1]_{i,j}) &= \bigvee_{m=i}^{j-1} \tilde{\rho}([S]_m) \\
\tilde{\rho}([S]_i) &= \widehat{\rho(i)}([S]_i) \\
\tilde{\rho}([\neg\phi_1]_{i,j}) &= \neg\tilde{\rho}([\phi_1]_{i,j}) \\
\tilde{\rho}([\phi_1 \wedge \phi_2]_{i,j}) &= \tilde{\rho}([\phi_1]_{i,j}) \wedge \tilde{\rho}([\phi_2]_{i,j}) \\
\tilde{\rho}([\phi_1 \vee \phi_2]_{i,j}) &= \tilde{\rho}([\phi_1]_{i,j}) \vee \tilde{\rho}([\phi_2]_{i,j}) \\
\tilde{\rho}([\phi_1 \frown \phi_2]_{i,j}) &= \bigvee_{m=i}^{j} \tilde{\rho}([\phi_1]_{i,m}) \wedge \tilde{\rho}([\phi_2]_{m,j})
\end{aligned}$$

We extend that function to $\overline{\tilde{\rho}}$, giving the value of expressions of the propositional variables.

$$\begin{aligned}
\overline{\tilde{\rho}}(\mathit{true}) &= \mathit{true} \\
\overline{\tilde{\rho}}(\mathit{false}) &= \mathit{false} \\
\overline{\tilde{\rho}}([\phi]_{i,j}) &= \tilde{\rho}([\phi]_{i,j}) \\
\overline{\tilde{\rho}}([S]_i) &= \tilde{\rho}([S]_i) \\
\overline{\tilde{\rho}}(\neg e) &= \neg\tilde{\rho}(e) \\
\overline{\tilde{\rho}}(e_1 \wedge e_2) &= \tilde{\rho}(e_1) \wedge \tilde{\rho}(e_2) \\
\overline{\tilde{\rho}}(e_1 \vee e_2) &= \tilde{\rho}(e_1) \vee \tilde{\rho}(e_2)
\end{aligned}$$

We shall need a Lemma describing how we may handle the occurence of biimplications in proofs concerning $\overline{\tilde{\rho}}$:

Lemma 5 *$\overline{\tilde{\sigma}}(e_1) = \overline{\tilde{\sigma}}(e_2)$ implies $\overline{\tilde{\sigma}}(e_1 \Leftrightarrow e_2) = \mathit{true}$.*

Proof:
Assume $\overline{\overline{\sigma}}(e_1) = \overline{\overline{\sigma}}(e_2)$.

$$\begin{aligned}
\overline{\overline{\sigma}}(e_1 \Leftrightarrow e_2) &\overset{(1)}{=} \overline{\overline{\sigma}}\left((\neg e_1 \vee e_2) \wedge (e_1 \vee \neg e_2)\right) \\
&\overset{(2)}{=} \left(\neg\overline{\overline{\sigma}}(e_1) \vee \overline{\overline{\sigma}}(e_2)\right) \wedge \left(\overline{\overline{\sigma}}(e_1) \vee \neg\overline{\overline{\sigma}}(e_2)\right) \\
&\overset{(3)}{=} \mathit{true}
\end{aligned}$$

with the following arguments:

(1) By definition of the '$\Leftrightarrow$' operator

(2) By definition of $\overline{\overline{\sigma}}$

(3) Using the assumption that $\overline{\overline{\sigma}}(e_1) = \overline{\overline{\sigma}}(e_2)$

Then, we must demonstrate the well-formedness of the translation t^k_{DC} with respect to $\overline{\overline{\rho}}$:

Lemma 6 $\overline{\overline{\rho}}(t^k_{DC}\,[\![\phi]\!]) = \mathit{true}$

Proof:
We shall establish a proof by structural induction on ϕ.
Basis:

- Case $\phi = \texttt{true}$:

$$\begin{aligned}
\overline{\overline{\rho}}(t^k_{DC}\,[\![\texttt{true}]\!]) &\overset{(1)}{=} \overline{\overline{\rho}}(\mathit{true}) \\
&\overset{(2)}{=} \mathit{true}
\end{aligned}$$

  with the following arguments:

  (1) By definition of t^k_{DC}
  (2) By definition of $\overline{\overline{\rho}}$

- Case $\phi = \texttt{false}$:
  Equivalent.

- Case $\phi = \int S \geq 1$:

$$\begin{aligned}
\overline{\tilde{\rho}}(t^k_{DC}[\![\textstyle\int S \geq 1]\!]) &\overset{(1)}{=} \overline{\tilde{\rho}}\left(t^k_{SA}[\![S]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i+1,\ldots,k\}}} \left([\textstyle\int S \geq 1]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j-1}[S]_m\right)\right)\\
&\overset{(2)}{=} \overline{\tilde{\rho}}\left(t^k_{SA}[\![S]\!]\right) \wedge \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i+1,\ldots,k\}}} \overline{\tilde{\rho}}\left([\textstyle\int S \geq 1]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j-1}[S]_m\right)\\
&\overset{(3)}{=} \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i+1,\ldots,k\}}} \overline{\tilde{\rho}}\left([\textstyle\int S \geq 1]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j-1}[S]_m\right)\\
&\overset{(4)}{=} \mathit{true}
\end{aligned}$$

with the following arguments:

(1) By definition of t^k_{DC}
(2) By definition of $\overline{\tilde{\rho}}$
(3) Using Lemma 2
(4) Using Lemma 1 since by definition of $\overline{\tilde{\rho}}$ we have that $\overline{\tilde{\rho}}([\int S \geq 1]_{i,j}) = \tilde{\rho}([\int S \geq 1]_{i,j}) = \bigvee_{m=i}^{j-1}\widehat{\rho(m)}([S]_m) = \bigvee_{m=i}^{j}\tilde{\rho}([S]_m) = \overline{\tilde{\rho}}\left(\bigvee_{m=i}^{j}([S]_m)\right)$

INDUCTION:

- Case $\phi = \neg\phi_1$:

$$\begin{aligned}
\overline{\tilde{\rho}}(t^k_{DC}[\![\neg\phi_1]\!]) &\overset{(1)}{=} \overline{\tilde{\rho}}\left(t^k_{DC}[\![\phi_1]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} ([\neg\phi_1]_{i,j} \Leftrightarrow \neg[\phi_1]_{i,j})\right)\\
&\overset{(2)}{=} \overline{\tilde{\rho}}\left(t^k_{DC}[\![\phi_1]\!]\right) \wedge \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \overline{\tilde{\rho}}\left([\neg\phi_1]_{i,j} \Leftrightarrow \neg[\phi_1]_{i,j}\right)\\
&\overset{(3)}{=} \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \overline{\tilde{\rho}}\left([\neg\phi_1]_{i,j} \Leftrightarrow \neg[\phi_1]_{i,j}\right)\\
&\overset{(4)}{=} \mathit{true}
\end{aligned}$$

with the following arguments:

(1) By definition of t^k_{DC}
(2) By definition of $\overline{\tilde{\rho}}$
(3) By induction
(4) Using Lemma 5 since by definition of $\overline{\tilde{\rho}}$ it holds that $\overline{\tilde{\rho}}([\neg\phi_1]_{i,j}) = \neg\overline{\tilde{\rho}}([\phi_1]_{i,j}) = \overline{\tilde{\rho}}(\neg[\phi_1]_{i,j})$

- Case $\phi = \phi_1 \wedge \phi_2$:

$$\begin{aligned}
\overline{\tilde{\rho}}(t^k_{DC}[\![\phi_1 \wedge \phi_2]\!]) &\overset{(1)}{=} \overline{\tilde{\rho}}\left(t^k_{DC}[\![\phi_1]\!] \wedge t^k_{DC}[\![\phi_2]\!] \wedge \right. \\
&\qquad \left. \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}) \right) \\
&\overset{(2)}{=} \overline{\tilde{\rho}}\left(t^k_{DC}[\![\phi_1]\!]\right) \wedge \overline{\tilde{\rho}}\left(t^k_{DC}[\![\phi_2]\!]\right) \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \overline{\tilde{\rho}}\left([\phi_1 \wedge \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}\right) \\
&\overset{(3)}{=} \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \overline{\tilde{\rho}}\left([\phi_1 \wedge \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}\right) \\
&\overset{(4)}{=} \mathit{true}
\end{aligned}$$

  with the following arguments:

  (1) By definition of t^k_{DC}
  (2) By definition of $\overline{\tilde{\rho}}$
  (3) By induction
  (4) Using Lemma 5 since by definition of $\overline{\tilde{\rho}}$ it holds that $\overline{\tilde{\rho}}([\phi_1 \wedge \phi_2]_{i,j}) = \overline{\tilde{\rho}}([\phi_1]_{i,j}) \wedge \overline{\tilde{\rho}}([\phi_1]_{i,j}) = \overline{\tilde{\rho}}([\phi_1]_{i,j} \wedge [\phi_2]_{i,j})$

- Case $\phi = \phi_1 \vee \phi_2$:
  Equivalent.

- Case $\phi = \phi_1 \frown \phi_2$:

$$\begin{aligned}
\overline{\tilde{\rho}}(t^k_{DC}[\![\phi_1 \frown \phi_2]\!]) &\overset{(1)}{=} \overline{\tilde{\rho}}\left(t^k_{DC}[\![\phi_1]\!] \wedge t^k_{DC}[\![\phi_2]\!] \wedge \right. \\
&\qquad \left. \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \left([\phi_1 \frown \phi_2]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j} ([\phi_1]_{i,m} \wedge [\phi_2]_{m,j})\right) \right) \\
&\overset{(2)}{=} \overline{\tilde{\rho}}\left(t^k_{DC}[\![\phi_1]\!]\right) \wedge \overline{\tilde{\rho}}\left(t^k_{DC}[\![\phi_2]\!]\right) \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \overline{\tilde{\rho}}\left([\phi_1 \frown \phi_2]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j} ([\phi_1]_{i,m} \wedge [\phi_2]_{m,j})\right) \\
&\overset{(3)}{=} \bigwedge_{\substack{i\in\{0,\ldots,k\}\\ j\in\{i,\ldots,k\}}} \overline{\tilde{\rho}}\left([\phi_1 \frown \phi_2]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j} ([\phi_1]_{i,m} \wedge [\phi_2]_{m,j})\right) \\
&\overset{(4)}{=} \mathit{true}
\end{aligned}$$

  with the following arguments:

(1) By definition of t^k_{DC}

(2) By definition of $\overline{\overline{\rho}}$

(3) By induction

(4) Using Lemma 5 since by definition of $\overline{\overline{\rho}}$ it holds that $\overline{\overline{\rho}}\left([\phi_1 \frown \phi_2]_{i,j}\right) = \tilde{\rho}([\phi_1 \frown \phi_2]_{i,j}) = \bigvee_{m=i}^{j}\left(\tilde{\rho}([\phi_1]_{i,m}) \wedge \tilde{\rho}([\phi_2]_{m,j})\right) = \overline{\overline{\rho}}\left(\bigvee_{m=i}^{j}([\phi_1]_{i,m} \wedge [\phi_2]_{m,j})\right)$

We must also show a connection between the semantic interpretation of ϕ in observation $\rho, [i,j]$ and the value of $\overline{\overline{\rho}}$:

Lemma 7 $\rho, [i,j] \models \phi$ *iff* $\overline{\overline{\rho}}([\phi]_{i,j}) = \mathit{true}$

Proof:

We shall prove Lemma 7 by structural induction on ϕ.

Basis:

- Case $\phi = \texttt{true}$:
  Trivial, since $\rho, [i,j] \models \texttt{true}$ and $\overline{\overline{\rho}}([\texttt{true}]_{i,j}) = \overline{\overline{\rho}}([\texttt{true}]) = \mathit{true}$.
- Case $\phi = \texttt{false}$:
  Trivial, since $\rho, [i,j] \not\models \texttt{false}$ and $\overline{\overline{\rho}}(\texttt{false}) = \overline{\overline{\rho}}(\neg[\texttt{true}]) = \mathit{false}$.
- Case $\phi = \int S \geq 1$:

$$\begin{aligned}
\rho, [i,j] \models \textstyle\int S \geq 1 \quad & \overset{(1)}{\text{iff}} \quad \bigvee_{m=i}^{j-1} \mathcal{I}\,[\![S]\!](\rho(m)) = \mathit{true} \\
& \overset{(2)}{\text{iff}} \quad \bigvee_{m=i}^{j-1} \widehat{\rho(m)}([S]_m) = \mathit{true} \\
& \overset{(3)}{\text{iff}} \quad \bigvee_{m=i}^{j-1} \widetilde{\rho}([S]_m) = \mathit{true} \\
& \overset{(4)}{\text{iff}} \quad \overline{\overline{\rho}}([\textstyle\int S \geq 1]_{i,j}) = \mathit{true}
\end{aligned}$$

with the following arguments:

(1) By definition of $\rho, [i,j] \models \int S \geq 1$

(2) Using Lemma 3

(3) By definition of $\widetilde{\rho}$

(4) By definition of $\overline{\overline{\rho}}$

Induction:

- Case $\phi = \neg\phi_1$

$$\begin{array}{lll} \rho, [i,j] \models \neg\phi_1 & \overset{(1)}{\text{iff}} & \rho, [i,j] \not\models \phi_1 \\ & \overset{(2)}{\text{iff}} & \overline{\overline{\rho}}([\phi_1]_{i,j}) = \mathit{false} \\ & \overset{(2)}{\text{iff}} & \overline{\overline{\rho}}([\neg\phi_1]_{i,j}) = \mathit{true} \end{array}$$

  with the following arguments:

  (1) By definition of $\rho, [i,j] \models \neg\phi_1$
  (2) By induction
  (3) By definition of $\overline{\overline{\rho}}$

- Case $\phi = \phi_1 \wedge \phi_2$:

$$\begin{array}{lll} \rho, [i,j] \models \phi_1 \wedge \phi_2 & \overset{(1)}{\text{iff}} & (\rho, [i,j] \models \phi_1) \wedge (\rho, [i,j] \models \phi_2) \\ & \overset{(2)}{\text{iff}} & \big(\overline{\overline{\rho}}([\phi_1]_{i,j}) = \mathit{true}\big) \wedge \big(\overline{\overline{\rho}}([\phi_2]_{i,j}) = \mathit{true}\big) \\ & \overset{(3)}{\text{iff}} & \overline{\overline{\rho}}([\phi_1 \wedge \phi_2]_{i,j}) = \mathit{true} \end{array}$$

  with the following arguments:

  (1) By definition of $\rho, [i,j] \models \phi_1 \wedge \phi_2$
  (2) By induction
  (3) By definition of $\overline{\overline{\rho}}$

- Case $\phi = \phi_1 \vee \phi_2$:
  Equivalent.

- Case $\phi = \phi_1 \frown \phi_2$:

$$\begin{array}{lll} \rho, [i,j] \models \phi_1 \frown \phi_2 & \overset{(1)}{\text{iff}} & \exists m \in \{i, \ldots, j\} \cdot (\rho, [i,m] \models \phi_1) \wedge (\rho, [m,j] \models \phi_2) \\ & \overset{(2)}{\text{iff}} & \exists m \in \{i, \ldots, j\} \cdot \big(\overline{\overline{\rho}}([\phi_1]_{i,m}) = \mathit{true}\big) \wedge \big(\overline{\overline{\rho}}([\phi_2]_{m,j}) = \mathit{true}\big) \\ & \overset{(3)}{\text{iff}} & \bigvee_{m=i}^{j} \big(\big(\overline{\overline{\rho}}([\phi_1]_{i,m}) = \mathit{true}\big) \wedge \big(\overline{\overline{\rho}}([\phi_2]_{m,j}) = \mathit{true}\big)\big) \\ & \overset{(4)}{\text{iff}} & \overline{\overline{\rho}}([\phi_1 \frown \phi_2]_{i,j}) = \mathit{true} \end{array}$$

  with the following arguments:

  (1) By definition of $\rho, [i,j] \models \phi_1 \frown \phi_2$
  (2) By induction
  (3) Using the meaning of $\exists m \in \{i, \ldots, j\}$
  (4) By definition of $\overline{\overline{\rho}}$

We shall show a Lemma relating satisfiability of constraints to values of $\overleftarrow{\overleftarrow{\xi}}$:

Lemma 8 *If ξ satisfies $t^k_{DC}[\![\phi]\!] \wedge [\mathtt{true}]$ then*
$\forall i,j \in Time \cdot 0 \leq i \leq j \leq k \Rightarrow \xi([\phi]_{i,j}) = \overleftarrow{\overleftarrow{\xi}}([\phi]_{i,j})$

Proof: We shall prove Lemma 8 by structural induction on ϕ.

Basis:

- Case $\phi = \mathtt{true}$:
  Assume ξ satisfies $t^k_{DC}[\![\mathtt{true}]\!] \wedge [\mathtt{true}]$.

$$\begin{aligned} \xi([\mathtt{true}]_{i,j}) &\overset{(1)}{=} \xi([\mathtt{true}]) \\ &\overset{(2)}{=} true \\ &\overset{(3)}{=} \overleftarrow{\overleftarrow{\xi}}([\mathtt{true}]) \end{aligned}$$

  with the following arguments

  (1) By definition of $[\mathtt{true}]_{i,j}$
  (2) Using the assumption
  (3) By definition of $\overleftarrow{\overleftarrow{\xi}}$

- Case $\phi = \mathtt{false}$:
  Equivalent.

- Case $\phi = \int S \geq 1$:
  We have that

$$\begin{aligned} t^k_{DC}[\![\textstyle\int S \geq 1]\!] \wedge [\mathtt{true}] \quad = \quad & \bigwedge_{i=0}^{k-1} t^k_{SA,i}[\![S]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k-1\} \\ j\in\{i+1,\ldots,k\}}} \left([\textstyle\int S \geq 1]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j-1}[S]_m\right) \wedge \\ & \bigwedge_{i\in\{0,\ldots,k\}} \neg[\textstyle\int S \geq 1]_{i,i} \end{aligned}$$

  For $0 \leq i = j \leq k$:

$$\begin{aligned} \xi([\textstyle\int S \geq 1]_{i,i}) &\overset{(1)}{=} false \\ &\overset{(2)}{=} \bigvee_{m=i}^{i-1}([S]_m) \\ &\overset{(3)}{=} \overleftarrow{\overleftarrow{\xi}}\left([\textstyle\int S \geq 1]_{i,i}\right) \end{aligned}$$

  with the following arguments:

(1) Since ξ satisfies $\neg[\int S \geq 1]_{i,i}$ for $i \in \{0,\ldots,k\}$
(2) By definition of a disjunction with zero elements
(3) By definition of $\overleftarrow{\overleftarrow{\sim}}_{\xi}$

For $0 \leq i < j \leq k$*:*

$$\begin{aligned}\xi([\textstyle\int S \geq 1]_{i,j}) &\overset{(1)}{=} \bigvee_{m=i}^{j-1} \xi([S]_m) \\ &\overset{(2)}{=} \bigvee_{m=i}^{j-1} \overleftarrow{\widehat{\xi}}\,(m)([S]_m) \\ &\overset{(3)}{=} \overleftarrow{\overleftarrow{\sim}}_{\xi}([\textstyle\int S \geq 1]_{i,j})\end{aligned}$$

with the following arguments:

(1) Since ξ satisfies $[\int S \geq 1]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j-1}[S]_m$ for $i \in \{0,\ldots,k-1\}$, $j \in \{i+1,\ldots,k\}$
(2) By Lemma 4 since ξ satisfies $t^k_{SA,m}[\![S]\!] \wedge [\texttt{true}]$ for $m \in \{i,\ldots,j-1\}$
(3) By definition of $\overleftarrow{\overleftarrow{\sim}}_{\xi}$

INDUCTION:

- Case $\phi = \neg\phi_1$:
  We have that

$$t^k_{DC}[\![\neg\phi_1]\!] = t^k_{DC}[\![\phi_1]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\neg\phi_1]_{i,j} \Leftrightarrow \neg[\phi_1]_{i,j}) \wedge [\texttt{true}]$$

  For $0 \leq i \leq j \leq k$*:*

$$\begin{aligned}\xi([\neg\phi_1]_{i,j}) &\overset{(1)}{=} \neg\xi([\phi_1]_{i,j}) \\ &\overset{(2)}{=} \neg\overleftarrow{\overleftarrow{\sim}}_{\xi}([\phi_1]_{i,j}) \\ &\overset{(3)}{=} \overleftarrow{\overleftarrow{\sim}}_{\xi}([\neg\phi_1]_{i,j})\end{aligned}$$

  with the following arguments:

  (1) Since ξ satisfies $[\neg\phi_1]_{i,j} \Leftrightarrow \neg[\phi_1]_{i,j}$ for $i \in \{0,\ldots,k\}$, $j \in \{i,\ldots,k\}$
  (2) By induction
  (3) By definition of $\overleftarrow{\overleftarrow{\sim}}_{\xi}$

- Case $\phi = \phi_1 \wedge \phi_2$:
  We have that

$$\begin{aligned}t^k_{DC}[\![\phi_1 \wedge \phi_2]\!] = {} & t^k_{DC}[\![\phi_1]\!] \wedge t^k_{DC}[\![\phi_2]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}) \wedge [\texttt{true}]\end{aligned}$$

For $0 \leq i \leq j \leq k$*:*

$$\begin{aligned}
\xi([\phi_1 \wedge \phi_2]_{i,j}) &\overset{(1)}{=} \xi([\phi_1]_{i,j}) \wedge \xi([\phi_2]_{i,j}) \\
&\overset{(2)}{=} \overleftarrow{\overleftarrow{\xi}}([\phi_1]_{i,j}) \wedge \overleftarrow{\overleftarrow{\xi}}([\phi_2]_{i,j}) \\
&\overset{(3)}{=} \overleftarrow{\overleftarrow{\xi}}([\phi_1 \wedge \phi_2]_{i,j})
\end{aligned}$$

with the following arguments:

(1) Since ξ satisfies $[\phi_1 \wedge \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}$ for $i \in \{0, \ldots, k\}$, $j \in \{i, \ldots, k\}$

(2) By induction

(3) By definition of $\overleftarrow{\overleftarrow{\xi}}$

- Case $\phi = \phi_1 \vee \phi_2$:
  Equivalent.

- Case $\phi = \phi_1 \frown \phi_2$:
  We have that

$$\begin{aligned}
t^k_{DC}[\![\phi_1 \frown \phi_2]\!] \quad = \quad & t^k_{DC}[\![\phi_1]\!] \wedge t^k_{DC}[\![\phi_2]\!] \wedge \\
& \bigwedge_{\substack{i \in \{0,\ldots k\} \\ j \in \{i,\ldots,k\}}} ([\phi_1 \frown \phi_2]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j} ([\phi_1]_{i,m} \wedge [\phi_2]_{m,j})) \wedge [\texttt{true}]
\end{aligned}$$

For $0 \leq i \leq j \leq k$*:*

$$\begin{aligned}
\xi([\phi_1 \wedge \phi_2]_{i,j}) &\overset{(1)}{=} \bigvee_{m=i}^{j} \xi([\phi_1]_{i,m}) \wedge \xi([\phi_2]_{m,j}) \\
&\overset{(2)}{=} \bigvee_{m=i}^{j} \left(\overleftarrow{\overleftarrow{\xi}}([\phi_1]_{i,m}) \wedge \overleftarrow{\overleftarrow{\xi}}([\phi_2]_{m,j}) \right) \\
&\overset{(3)}{=} \overleftarrow{\overleftarrow{\xi}}([\phi_1 \frown \phi_2]_{i,j})
\end{aligned}$$

with the following arguments:

(1) Since ξ satisfies $[\phi_1 \frown \phi_2]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j}([\phi_1]_{i,m} \wedge [\phi_2]_{m,j})$ for $i \in \{0, \ldots, k\}$, $j \in \{i, \ldots, k\}$

(2) By induction

(3) By definition of $\overleftarrow{\overleftarrow{\xi}}$

And, building on Lemma 8, we have:

Lemma 9 *If ξ satisfies $t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}]$ then $\overline{\overleftarrow{\simeq}}\,\xi\,(t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}]) = true$*

Proof:

$$\begin{aligned}
\overline{\overleftarrow{\simeq}}\,\xi\,(t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}]) &\overset{(1)}{=} \overline{\overleftarrow{\simeq}}\,\xi\,(t^k_{DC}\,[\![\phi]\!]) \wedge \overline{\overleftarrow{\simeq}}\,\xi\,([\phi]_{i,j}) \\
&\overset{(2)}{=} \overline{\overleftarrow{\simeq}}\,\xi\,([\phi]_{i,j}) \\
&\overset{(3)}{=} \xi([\phi]_{i,j}) \\
&\overset{(4)}{=} true
\end{aligned}$$

with the following arguments:

(1) By definition of $\overline{\overleftarrow{\simeq}}\,\xi$

(2) Using Lemma 6

(3) Using Lemma 8

(4) Since ξ satisfies $t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}]$

Finally, we may show the last Lemma:

Lemma 10 *$\exists \rho \in Traj \cdot \rho, [i,j] \models \phi$ iff $t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}]$ is satisfiable.*

Proof:
If:

$$\begin{aligned}
\text{If } \exists \rho \in Traj \cdot \rho, [i,j] \models \phi \quad &\overset{(1)}{\text{then}} \quad \exists \rho \in Traj \cdot \overline{\tilde{\rho}}([\phi]_{i,j}) = true \\
&\overset{(2)}{\text{then}} \quad \exists \rho \in Traj \cdot \big(\overline{\tilde{\rho}}(t^k_{DC}\,[\![\phi]\!]) = true\big) \wedge \\
&\qquad\qquad\qquad \big(\overline{\tilde{\rho}}([\phi]_{i,j}) = true\big) \\
&\overset{(3)}{\text{then}} \quad \exists \rho \in Traj \cdot \big(\overline{\tilde{\rho}}(t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}]) = true\big) \\
&\overset{(4)}{\text{then}} \quad \exists \rho \in Traj \cdot \tilde{\rho} \text{ satisfies } t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}] \\
&\overset{(5)}{\text{then}} \quad t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}] \text{ is satisfiable}
\end{aligned}$$

with the following arguments:

(1) By Lemma 7

(2) By Lemma 6

(3) By definition of $\overline{\tilde{\rho}}$

(4) Since the definition of $\overline{\tilde{\rho}}$ is simply the extension of $\tilde{\rho}$ to boolean combinations of the variables

(5) By definition of satisfiability

Only if:

$$
\begin{array}{lll}
\text{If } t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}] \text{ is satisfiable} & \overset{(1)}{\text{then}} & \exists \xi \in \mathit{Lit} \to \mathbf{B} \cdot \\
 & & \xi \text{ satisfies } t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}] \\
 & \overset{(2)}{\text{then}} & \exists \xi \in \mathit{Lit} \to \mathbf{B} \cdot \\
 & & \overline{\overset{\leftarrow}{\xi}}\,(t^k_{DC}\,[\![\phi]\!] \wedge [\phi]_{i,j} \wedge [\texttt{true}]) = \mathit{true} \\
 & \overset{(3)}{\text{then}} & \exists \xi \in \mathit{Lit} \to \mathbf{B} \cdot \overset{\leftarrow}{\xi}, [i,j] \models \phi \\
 & \overset{(4)}{\text{then}} & \exists \rho \in \mathit{Traj} \cdot \rho, [i,j] \models \phi
\end{array}
$$

with the following arguments:

(1) By definition of satisfiability

(2) By Lemma 9

(3) By Lemma 7

(4) Namely $\rho = \overset{\leftarrow}{\xi}$

3.4.3 Proof of Main Theorem

$$
\begin{array}{lll}
\phi \text{ has a model of length at most } k & \overset{(1)}{\text{iff}} & \exists \rho \in \mathit{Traj}, i \in \{0, \ldots, k\} \cdot (\rho, [0,i]) \models \phi \\
 & \overset{(2)}{\text{iff}} & t^k_{DC}\,[\![\phi]\!] \wedge \bigvee_{i=0}^{k} [\phi]_{0,i} \wedge [\texttt{true}] \text{ is satisfiable} \\
 & \overset{(3)}{\text{iff}} & \mathrm{BMC}(\phi, k) \text{ is satisfiable}
\end{array}
$$

with the following arguments:

(1) By definition of the left-hand side

(2) Using Lemma 10

(3) By definition of BMC(ϕ, k)

3.5 Negation Optimisation

In Section 3.3 on page 17, it was described how definitions are added to ensure the correspondance between literals $[\phi]_{i,j}$ and $[\neg\phi]_{i,j}$ whenever $\neg\phi$ occurs as a subformula.

However, the same variable may be used to represent ϕ and $\neg\phi$, only with different signs, so that

$$[\neg\phi]_{i,j} \stackrel{\text{def}}{=} \neg[\phi]_{i,j}$$

This eliminates the need for the correspondance definitions and reduces the overall number of propositional variables. The translation of $\neg\phi$ is then simply:

$$t^k_{DC}[\![\neg\phi]\!] \quad = \quad t^k_{DC}[\![\phi]\!]$$

3.6 Polarity Optimisation

3.6.1 Definition of Polarity

A formula is said to occur with positive polarity if it occurs under an even number of negations, and negative polarity otherwise.

This definition applies to state assertions as well as DC formulas. The polarity of a top-level state assertion S is that of its duration formula $\int S \geq n$.

3.6.2 Polarity and Satisfiability

If a DC subformula ϕ occurs with positive polarity in a formula ϕ_2, either ϕ_2 is satisfiable only when ϕ is satisfied, or the satisfiability of ϕ_2 is unaffected by ϕ. By the definition of polarity, it clearly cannot be the case that ϕ_2 is satisfiable only when ϕ is unsatisfied.

Similarly, if ϕ occurs with negative polarity in ϕ_2, then ϕ_2 is satisifiable only when ϕ is unsatisfied, or the satisfiability of ϕ_2 is unaffected by ϕ.

3.6.3 Using Polarity for Optimisations

As sketched in [MF02], the relation between polarity and satisfiability leads us to replacing the biimplications of the definitions for the literals $[\phi]_{i,j}$ with one-sided implications. In this section, we shall show the resulting translation algorithm and we shall argue for its correctness in Section 3.6.4 on the following page.

E.g. if $\phi_1 \wedge \phi_2$ occurs with positive polarity, its definition can be changed to

$$\begin{aligned} t^k_{\mathrm{DC},+}[\![\phi_1 \wedge \phi_2]\!] \quad = \quad & t^k_{\mathrm{DC},+}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},+}[\![\phi_2]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k\} \\ j\in\{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} \Rightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}) \end{aligned}$$

where $t^k_{\mathrm{DC},p}[\![\phi]\!]$ represents the translation of DC formula ϕ with polarity p.

Equivalently, if $\phi_1 \wedge \phi_2$ occurs with negative polarity, its definition can be changed to:

$$t^k_{\mathrm{DC},-}[\![\phi_1 \wedge \phi_2]\!] = t^k_{\mathrm{DC},-}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},-}[\![\phi_2]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} \Leftarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j})$$

The cases for $\phi_1 \vee \phi_2$ and $\phi_1 \frown \phi_2$ are similar. When $\neg\phi_1$ is encountered, ϕ_1 is simply translated with the opposite polarity of $\neg\phi_1$:

$$\begin{aligned} t^k_{\mathrm{DC},+}[\![\neg\phi_1]\!] &= t^k_{\mathrm{DC},-}[\![\phi_1]\!] \\ t^k_{\mathrm{DC},-}[\![\neg\phi_1]\!] &= t^k_{\mathrm{DC},+}[\![\phi_1]\!] \end{aligned}$$

For duration formulas $\int S \geq 1$ with positive polarity we get:

$$t^k_{\mathrm{DC},+}[\![\textstyle\int S \geq 1]\!] = t^k_{\mathrm{SA},+}[\![S]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k-1\}\\ j\in\{i+1,\ldots,k\}}} \left([\textstyle\int S \geq 1]_{i,j} \Rightarrow [S]_i \vee [\int S \geq 1]_{i+1,j}\right) \wedge \bigwedge_{i\in\{0,\ldots k\}} \neg[\textstyle\int S \geq 1]_{i,i}$$

And for negative polarity, we need not explicitly state that the literals representing duration formulas over an interval of length zero should have the value false, as we will argue in Section 3.6.4:

$$t^k_{\mathrm{DC},-}[\![\textstyle\int S \geq 1]\!] = t^k_{\mathrm{SA},-}[\![S]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k-1\}\\ j\in\{i+1,\ldots,k\}}} \left([\textstyle\int S \geq 1]_{i,j} \Leftarrow [S]_i \vee [\int S \geq 1]_{i+1,j}\right)$$

3.6.4 Correctness of Polarity Optimisation

It is obvious that polarity optimisation only *removes* constraints from the SAT problem. Hence, if it was satisifiable without this optimisation, it will surely be satisfiable with the optimisation as well.

Now we need to argue why a SAT problem (representing the satisfiability of a DC formula) that was unsatisfiable *without* the optimisation cannot become satisfiable *with* the optimisation.

Let us look at the case of $\phi_1 \wedge \phi_2$ under positive polarity. We have dropped all constraints of the form $[\phi_1 \wedge \phi_2]_{i,j} \Leftarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}$. This allows the value of $[\phi_1]_{i,j} \wedge [\phi_2]_{i,j}$ to be *true* even when the value of $[\phi_1 \wedge \phi_2]_{i,j}$ is *false*, which appears problematic.

We have three cases:

1. The context in which $[\phi_1 \wedge \phi_2]_{i,j}$ occurs requires that its value is *true* in order for the problem to be satisfiable. So if $[\phi_1 \wedge \phi_2]_{i,j}$ were to take on the value *false*, the problem must surely still be unsatisfiable.

2. The context requires that the value of $[\phi_1 \wedge \phi_2]_{i,j}$ is *false* for the problem to be satisfiable. This contradicts the assumption of $[\phi_1 \wedge \phi_2]_{i,j}$ occurring under positive polarity, and hence this case can be disregarded.

3. The value of $[\phi_1 \wedge \phi_2]_{i,j}$ has no impact on the satisfiability of the problem, e.g. if it occurs as $\texttt{true} \vee \phi_1 \wedge \phi_2$. In this case, it clearly should not make a difference for the satisfiability of the problem if $[\phi_1 \wedge \phi_2]_{i,j}$ was given the value *false*.

Also, since $[\phi_1 \wedge \phi_2]_{i,j}$ is merely an auxiliary literal we do not use its value for e.g. visualising state trajectories, so the fact that the SAT solver may allocate an "undesired" value to it will make no difference.

The cases for $\phi_1 \vee \phi_2$, $\phi_1 \frown \phi_2$, $\neg\phi_1$ and $\int S \geq 1$ are similar under positive polarity.

For the case of $\phi_1 \wedge \phi_2$ under negative polarity, we have dropped constraints of the form $[\phi_1 \wedge \phi_2]_{i,j} \Rightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}$, opening up for the possibility that the value of $[\phi_1 \wedge \phi_2]_{i,j}$ could be *true* while the value of $[\phi_1]_{i,j} \wedge [\phi_2]_{i,j}$ is *false*. The argument for its correctness is symmetric to that of positive polarity, and similar cases can be established for $\phi_1 \vee \phi_2$, $\phi_1 \frown \phi_2$, $\neg\phi_1$.

For duration formulas $\int S \geq 1$, we must aditionally argue why the constraint of

$$\bigwedge_{i \in \{0,\ldots k\}} \neg[\textstyle\int S \geq 1]_{i,i}$$

can be dropped under negative polarity: Since the negative polarity implies that either the context requires that the value of $[\int S \geq 1]_{i,j}$ is false in order for the problem to be satisfiable, or the value of $[\int S \geq 1]_{i,j}$ makes no difference for the satisifiability of the problem, we need not explicitly force its value to false.

3.6.5 Effect on Validation Time

Obviously, polarity optimisation will improve the frontend translation time, since the frontend need not generate as many constraints. But will polarity optimisation also improve the SAT solving time, or could it even make it worse?

We shall consider the SAT solver algorithm described in [FH03] that builds on the Davis-Putnam-Loveland-Logemann (DPLL) procedure and employs backtracking, as described in Section 2.3 on page 13.

In the following, we shall list the arguments why polarity optimisation may be an advantage (+) or a disadvantage (÷):

\+ The fewer constraints to be checked during unit resolution and conflict detection, the faster each step of the procedure will be.

\+ In the case where the SAT problem is satisfiable, with fewer constraints the chance increases that the partial valuation "guessed" during the decision steps will be a solution for the problem.

÷ In the case where the SAT problem is unsatisfiable, all of the "guesses" made by the SAT solver during the decision steps will lead to conflicting clauses, and hence require backtracking. Here, it will be an advantage if the problem constrains as many of the variables as possible, so that the values of variables are assigned in unit resolution rather than through decision steps.

We therefore expect that the SAT solving time will improve with polarity optimisation for satisfiable problems, while there are contradictory effects on unsatisfiable problems. It is a rather unpleasant property that we should somehow predict whether or not the formula of interest is valid, when selecting if we should apply this optimisation. Benchmarking (see Chapter 9 on page 81) will hopefully provide some more useful insights in this area.

3.7 Common Subexpression Elimination and Literal Reuse

When a DC subformula ϕ occurs more than once under identical polarities, the same literals $[\phi]_{i,j}$ should be used whenever the subformula is referenced. Furthermore, the constraints given by $t^k_{\mathrm{DC},p}[\![\phi]\!]$ should be generated only once. Similar recurrence optimisation may be employed for state assertions.

The BMC/DCValidator shall offer three different ways of subformula recurrence recognition (time and space complexities for recognising if a subformula has occurred before are given based on the size n of the subformula):

1. Recognises solely identical formulas if they stem from the same occurrence of a formula. This implies that only if a duration $\int S \geq n$ has been chopped into n durations $\int S \geq 1$ will these duration formulas be determined to be identical.

   Time and space complexity: $\mathcal{O}(1)$

2. Recognises syntactically identical formulas.

   Time and space complexity: $\mathcal{O}(n)$

3. Recognises syntactically identical formulas, and some semantically identical formulas. The DC formulas as well as their state assertions are rewritten using CNF. Then, sorting is performed in clauses and of clauses before comparison. The rewritten formulas are used for comparison only, and the translation process continues with the original formulas (since the size of the formula will often increase with the rewriting).

   Time and space complexity: $\mathcal{O}\left(2^n\right)$

3.8 Optimisation for Output Format

In the previous chapters, we have presented the translation algorithm where literal definitions used a mixture of implications, negations, conjunctions and disjunctions. With the DIMACS output format, the output must be in CNF. With the ZOLCS output format, the output must be a conjunction of linear inequalities.

Naturally, we could employ standard schemes for rewriting the output of the previous translation algorithm to the appropriate format, but it is more efficient to perform the rewriting once and for all in our algorithm. Also, the ZOLCS format provides more compact representation of some of the definitions.

By exploiting this, we may lower the number of constraints in the final SAT problem.

Before proceeding with the presentation of the two versions of the translation algorithm, we note that the previous version of the algorithm already generates an output that is very close to CNF, since the individual literal definitions are all joined by conjunctions. So we only need to rewrite the definitions themselves.

3.8.1 Output Format: DIMACS

We may use a recursive definition of the $[\int S \geq 1]_{i,j}$ literals. This will not change the number of constraints, but each constraint will contain fewer literals. When writing these constraints in CNF form, we get (for positive polarity):

$$\begin{aligned} t^k_{\mathrm{DC},+}[\![\textstyle\int S \geq 1]\!] \quad = \quad & t^k_{\mathrm{SA},+}[\![S]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k-1\}\\ j\in\{i+1,\ldots,k\}}} \left(\neg[\textstyle\int S \geq 1]_{i,j} \vee [S]_i \vee [\int S \geq 1]_{i+1,j}\right) \wedge \\ & \bigwedge_{i\in\{0,\ldots k\}} \neg[\textstyle\int S \geq 1]_{i,i} \end{aligned}$$

and for negative polarity:

$$\begin{aligned} t^k_{\mathrm{DC},-}[\![\textstyle\int S \geq 1]\!] \quad = \quad & t^k_{\mathrm{SA},-}[\![S]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k-1\}\\ j\in\{i+1,\ldots,k\}}} \left([\textstyle\int S \geq 1]_{i,j} \vee \neg[S]_i\right) \wedge \left([\textstyle\int S \geq 1]_{i,j} \vee \neg[\int S \geq 1]_{i+1,j}\right) \end{aligned}$$

We rewrite the definition of $[\phi_1 \wedge \phi_2]_{i,j}$ into CNF. For positive polarity, this immediately gives us:

$$\begin{aligned} t^k_{\mathrm{DC},+}[\![\phi_1 \wedge \phi_2]\!] \quad = \quad & t^k_{\mathrm{DC},+}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},+}[\![\phi_2]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} (\neg[\phi_1 \wedge \phi_2]_{i,j} \vee [\phi_1]_{i,j}) \wedge (\neg[\phi_1 \wedge \phi_2]_{i,j} \vee [\phi_2]_{i,j}) \end{aligned}$$

and for negative polarity:

$$\begin{aligned} t^k_{\mathrm{DC},-}[\![\phi_1 \wedge \phi_2]\!] \quad = \quad & t^k_{\mathrm{DC},-}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},-}[\![\phi_2]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} \vee \neg[\phi_1]_{i,j} \vee \neg[\phi_2]_{i,j}) \end{aligned}$$

The change to $t^k_{\mathrm{DC},p}[\![\phi_1 \vee \phi_2]\!]$ and $t^k_{\mathrm{DC},p}[\![\int S \geq 1]\!]$ is equivalent. Obviously, no change is needed for the trivial $t^k_{\mathrm{DC},p}[\![\neg\phi_1]\!]$. The translation of $\phi_1 \frown \phi_2$ under negative polarity is also straightforward, yielding

$$t^k_{\mathrm{DC},-}[\![\phi_1 \frown \phi_2]\!] \quad = \quad t^k_{\mathrm{DC},-}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},-}[\![\phi_2]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}\\ m\in\{i,\ldots,j\}}} ([\phi_1 \frown \phi_2]_{i,j} \vee \neg[\phi_1]_{i,m} \vee \neg[\phi_2]_{m,j})$$

With $\phi_1 \frown \phi_2$ under positive polarity, we have to rewrite a disjunction of conjunctions $[\phi_1]_{i,m} \wedge [\phi_2]_{m,j}$ to CNF, which would cause an exponential increase in the size of the formula. More precisely, a CNF for $[\phi_1 \frown \phi_2]_{i,j} \Rightarrow \bigvee_{i=m}^{j} [\phi_1]_{i,m} \wedge [\phi_2]_{m,j}$ will contain 2^{j-i+1} clauses.

Instead, we create new literals $[\phi_1 \frown \phi_2]_{i,m,j}$ (note the triple index) that represent the value of $[\phi_1]_{i,m} \wedge [\phi_2]_{m,j}$, i.e. the value of $[\phi_1 \frown \phi_2]_{i,j}$ if the chop between ϕ_1 and ϕ_2 were made at point m. Then, appropriate definitions of $[\phi_1 \frown \phi_2]_{i,m,j}$ must be added, leading to

$$\begin{aligned} t^k_{\mathrm{DC},+}[\![\phi_1 \frown \phi_2]\!] \quad = \quad & t^k_{\mathrm{DC},+}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},+}[\![\phi_2]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} \left(\left(\neg[\phi_1 \frown \phi_2]_{i,j} \vee \bigvee_{m=i}^{j} [\phi_1 \frown \phi_2]_{i,m,j} \right) \wedge \right. \\ & \qquad \bigwedge_{m=i}^{j} ((\neg[\phi_1 \frown \phi_2]_{i,m,j} \vee [\phi_1]_{i,m}) \wedge \\ & \qquad\quad \left. (\neg[\phi_1 \frown \phi_2]_{i,m,j} \vee [\phi_2]_{m,j})) \right) \end{aligned}$$

We note that each definition of $[\phi_1 \frown \phi_2]_{i,j}$ now only generates $2(j-i+1)+1$ clauses in CNF.

3.8.2 Output Format: ZOLCS

The individual constraints of the ZOLCS format consists of linear inequalities which provides for a more compact representation than CNF in some cases. Specifically, the literal definitions shall use the following two rewriting rules:

- A disjunction of literals $x_1 \vee x_2 \vee \ldots x_n$ can be rewritten to an inequality as $x_1 + x_2 + \ldots + x_n \geq 1$.
- An implication $x \Rightarrow y_1 \wedge y_2 \wedge \ldots y_n$, where x and y_i for $i \in \{1, \ldots, n\}$ are literals, can be rewritten to an inequality as $n \cdot \neg x + y_1 + y_2 + \ldots + y_n \geq n$.

where we define that the value of literal x_i in a sum is 0 if x_i is false, and 1 if x_i is true. Similarly, the value of $\neg x_i$ is 0 if x_i is true and 1 if x_i is false.

For example, the translation of $\phi_1 \wedge \phi_2$ with positive polarity is now

$$\begin{aligned} t^k_{\mathrm{DC},+}[\![\phi_1 \wedge \phi_2]\!] \quad = \quad & t^k_{\mathrm{DC},+}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},+}[\![\phi_2]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} (2 \cdot \neg[\phi_1 \wedge \phi_2]_{i,j} + [\phi_1]_{i,j} + [\phi_2]_{i,j} \geq 2) \end{aligned}$$

So in this case, we generate only half as many constraints as we did for the DIMACS output format, as described in Section 3.8.1 on the preceding page.

In our initial version of the translation algorithm that was given in Section 3.3 on page 17, we assumed that duration formulas $\int S \geq n$ with $n > 1$ were chopped up into n occurrences of $\int S \geq 1$. With ZOLCS' linear inequalities this is no longer necessary, thus providing for a dramatic decrease in the size of the SAT problem, both in terms of the number of constraints and literals.

Conceptually, the necessary constraint on $[\int S \geq n]_{i,j}$ is simply

$$[\int S \geq n]_{i,j} \Leftrightarrow \textstyle\sum_{m=i}^{j-1} [S]_m \geq n$$

where $\sum_{m=i}^{j-1} [S]_m$ is equal to zero when $i = j$.

When $\int S \geq n$ occurs with positive polarity, we only need the implication from left to right, giving us

$$t^k_{\mathrm{DC},+}[\![\int S \geq n]\!] \quad = \quad t^k_{\mathrm{DC},+}[\![S]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} \left(n \cdot \neg[\int S \geq n]_{i,j} + \textstyle\sum_{m=i}^{j-1} [S]_m \geq n\right)$$

When $\int S \geq 1$ occurs with negative polarity, we must express the constraints

$$[\int S \geq n]_{i,j} \Leftarrow \textstyle\sum_{m=i}^{j-1} [S]_m \geq n$$

in ZOLCS format. To do this, we note that

$$\neg\left(\sum_{m=i}^{j-1} [S]_m \geq n\right)$$

is equivalent to

$$\sum_{m=i}^{j-1} \neg[S]_m \geq j - i - n + 1$$

since the sum contains $j - i$ literals $[S]_m$ whereof at most $n - 1$ may have the value true. We can therefore write each definition as a single inequality:

$$\begin{aligned} t^k_{\mathrm{DC},-}[\![\int S \geq n]\!] \quad = \quad & t^k_{\mathrm{DC},-}[\![S]\!] \wedge \\ & \bigwedge_{\substack{i\in\{0,\ldots k-n\}\\ j\in\{i+n,\ldots,k\}}} \Big((j - i + 1 - n) \cdot [\int S \geq n]_{i,j} + \\ & \qquad \sum_{m=i}^{j-1} \neg[S]_m \geq (j - i + 1 - n)\Big) \end{aligned}$$

Note that the index variables i and j in this case range only over $\{0, \ldots, k-n\}$ and $\{i + n, \ldots, k\}$, respectively. The "missing" constraints are those describing $[\int S \geq n]_{i,j}$ when $j - i < n$, which would have a non-positive threshold, i.e. be trivially true.

Chapter 4

Simplifications

Prior to translation, the DC formulas (including their state assertions) may be simplified. After translation, the resulting constraints — formulated in propositional logic or as inequalities, depending on the output format — may also be simplified.

From each subformula of the DC formula to be validated, up to $\mathcal{O}(k^3)$ constraints will be generated. For this reason, it will be most rational to invest one's effort into simplifying the DC formulas. Thus, the simplifications presented here include extensive and global simplifications on the DC formulas but only local (i.e. within a single inequality or clause) on the constraints.

First, we will present the notion of *Binary Decision Diagrams* that is used for some of the advanced simplifications. Then, the simplifications performed on DC formulas will be given, and finally we will show what simplifications are performed on the constraints.

4.1 Binary Decision Diagrams

In the following, we shall give an introduction to the concept of *reduced ordered binary decision diagrams* (ROBDDs) which we shall use for expression simplification. The contents of this section are based on [HA97], from which we have also taken figures 4.1 and 4.2.

A *binary decision diagram* (BDD) is a directed acyclic graph that represents a boolean function. An example of a BDD may be found in figure 4.1 on the following page.

The nodes of the graph consist of *terminal nodes* and *non-terminal nodes*. The terminal nodes represent the constants *true* and *false*. These nodes have no outgoing edges. Each non-terminal node u is labeled with a variable $var(u)$, and has two outgoing edges. We shall call these edges the *low-edge* and *high-edge*, respectively, given by function $low(u)$ and $high(u)$. When drawing a BDD, we shall illustrate the low-edges by dashed lines and high-edges by solid lines. All BDDs have exactly one *root* node, i.e. a node that has no incoming edges.

To evaluate the boolean function represented by a BDD, one starts at the root. For each non-terminal node encountered, the low-edge is selected if the value of the variable at the node is *false*, and the high-edge is selected if the value is *true*. When a terminal node is reached, this is the result of the evaluation.

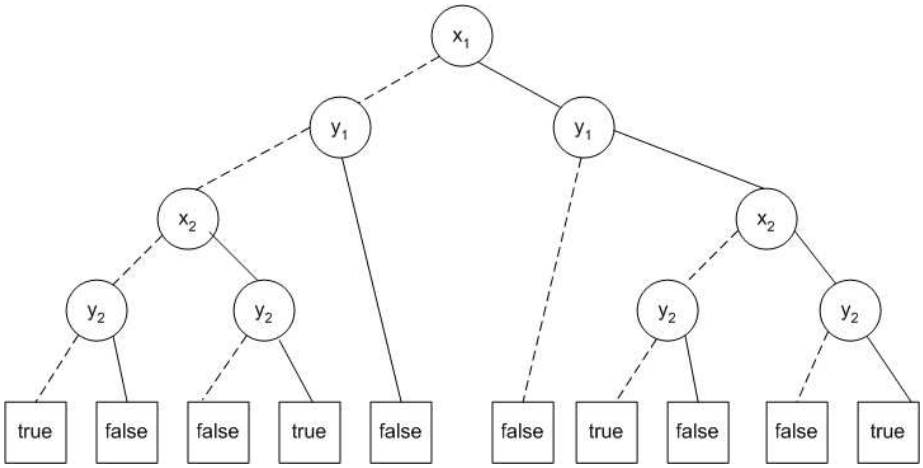

Figure 4.1: BDD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$

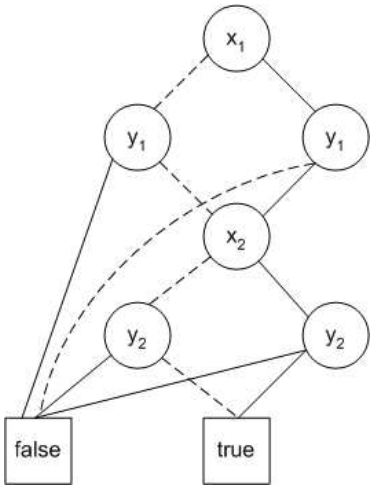

Figure 4.2: Another BDD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$

```
MK[T, H](i, l, h)

    if l = h then return l
    else if member(H, i, l, h) then
        return lookup(H, i, l, h)
    else u ← add(T, i, l, h)
        insert(H, i, l, h, u)
        return u
```

Figure 4.3: Function MK

The order in which the variables are tested can greatly impact the size of the resulting BDD. As an example, the BDD of figure 4.2 on the page before represents the same boolean function as the BDD in figure 4.1, but contains three nodes fewer. If the variables occur in the same order on all paths from the root of a BDD, the BDD is said to be an *Ordered* BDD (or OBDD).

If the low-edge and high-edge of a node u lead to the same node v, node u is clearly redundant. Also, if two nodes u and v have the same variable name and $low(u) = low(v)$ and $high(u) = high(v)$, one of them can be removed from the graph and its incoming edges redirected to the other. A OBDD that has no redundant nodes, and all of whose nodes are *unique*, i.e. $\forall u, v \cdot var(u) = var(v) \wedge low(u) = low(v) \wedge high(u) = high(v) \Rightarrow u = v$, is said to be a *Reduced* OBDD (or ROBDD).

ROBDDs have the nice property that for each boolean function, there is *exactly one* ROBDD representing it. Specifically, the constant functions *true* and *false* are represented by ROBDDs containing exactly one node, namely the terminal node in question. This enables us to test in constant time whether an ROBDD is constantly true or not. We shall not give the proof of this property, but refer the reader to [HA97].

4.1.1 Algorithms

In [HA97], an algorithm BUILD was given for construction of an ROBDD from a boolean expression. It is listed in figure 4.4 on the following page and uses an auxiliary function MK of figure 4.3.

The heart of the algorithms is two tables, T and H. Table T maps nodes u to triples (i, l, h) with $var(u) = i$, $low(u) = l$ and $high(u) = h$. Table H is the equivalent "inverse" mapping. We shall assume the existence of standard *member*, *lookup* and *insert* functions on the two tables with constant running times, a function *init* that clears the table, and table T shall also require a function $u \leftarrow add(T, i, l, h)$ that allocates a new node u with attributes (i, l, h). Futhermore, the algorithms assume an ordering $x_1 < x_2 < \ldots < x_n$ among the variables.

The responsibility of the function MK, as given in figure 4.3, is simply to determine (using table H) if a node already exists with the given attributes (i, l, h) and, if so, return that node, or otherwise create a new node and insert it into the tables. The running time of MK is $\mathcal{O}(1)$.

The BUILD function, listed in figure 4.4, creates an ROBDD from a boolean

```
BUILD[T, H](t)

    function BUILD'(t,i) =
      if i > n
      then if t =false
             then return false
             else return true
      else  v_0 ← BUILD'(t[false/x_i], i + 1)
      v_1 ← BUILD'(t[true/x_i], i + 1)
      return MK(i, v_0, v_1)
    end BUILD'
    return BUILD'(t, 1)
```

Figure 4.4: Function BUILD

expression t by successively going through each variable x_i in order, and building the high- and low-branches by replacing x_i in the boolean expression with true and false, respectively. After having built the high- and low-branches h and l, the MK function is called to create the node with attributes (i, l, h) if a such does not already exist. The running time of BUILD is $\mathcal{O}(2^n)$.

The last algorithm we shall present is the APPLY function, listed in figure 4.5 on the next page. Its purpose is to combine two ROBDDs u_1 and u_2 using a selected boolean operator. The algorithm iterates through the tree, each time selecting the node with the lowest-ordered variable of the two ROBDDs and creating the low- and high-branches of that node by recursive calls on the remaining parts of the ROBDDs. In the case of two terminal nodes, the boolean operator in question is applied to them, yielding a terminal node.

The APPLY function uses dynamic programming by placing the results of applying the operator to a pair of nodes in a table G so that they need not be recomputed if they reoccur. This avoids an exponential worst-case running time, and in fact achieves a running time of $\mathcal{O}(|u_1||u_2|)$ where $|u|$ denotes the number of reachable nodes from u.

```
APPLY[T, H](op, u_1, u_2)

init(G)
        function APP(u_1, u_2) =
          if G(u_1, u_2) ≠ empty
          then return G(u_1, u_2)
          else if u_1 ∈ {true, false} and u_2 ∈ {true, false}
          then u ← op(u_1, u_2)
          else if var(u_1) = var(u_2)
          then u ← MK(var(u_1), APP(low(u_1), low(u_2)), APP(high(u_1), high(u_2)))
          else if var(u_1) < var(u_2)
          then u ← MK(var(u_1), APP(low(u_1), u_2), APP(high(u_1), u_2))
          else (* var(u_1) = var(u_2) *)
                u ← MK(var(u_2), APP(u_1, low(u_2)), APP(u_1, high(u_2)))
        G(u_1, u_2) ← u
        end APP
        return APP(u_1, u_2)
```

Figure 4.5: Function APPLY

4.2 DC Simplifications

In the following, we shall take the notation

$$\frac{b}{\phi \rightarrow \psi}$$

to mean that formula ϕ may be rewritten to ψ if premise b holds. When b is trivially true, we will simply write

$$\phi \rightarrow \psi$$

The simplifications on DC formulas and state assertions may conceptually be divided into two levels:

Level 1 simplifications are simple, local simplification rules that have no premises at all or very simple premises. Consequently, they are relatively inexpensive to run.

An example of a level 1 simplification rule is

$$\texttt{true} \wedge \phi \rightarrow \phi$$

stating that `true` is an identity element for $\wedge$.

Level 2 simplifications are the more complex and expensive simplification rules that will require checking of elaborate premises. They also take the associativity of operators into account, giving them a more global scope.

An example of a level 2 simplification rule is

$$\frac{n_1 \geq n_2, S_1 \Rightarrow S_2}{\int S_1 \geq n_1 \wedge \int S_2 \geq n_2 \rightarrow \int S_1 \geq n_1}$$

stating that under conjunction, one duration formula may absorb another if it implies it (which is determined by comparison of their state assertions and integer bounds).

The complete list of simplifications on DC formulas has been given in Tables 4.1 and 4.2, while the simplifications on state assertions are stated in Tables 4.3 and 4.4.

Both level 1 and level 2 simplifications make use of the commutativity of the operators involved, and hence the reader should imagine the existence of further simplification rules symmetric to those listed in the tables.

To give an example of what it means that the level 2 simplifications make use of associativity and commutativity, a simplification rule such as

$$\frac{n_1 \geq n_2, S_1 \Rightarrow S_2}{\int S_1 \geq n_1 \wedge \int S_2 \geq n_2 \rightarrow \int S_1 \geq n_1}$$

will also be applicable to

$$\left(\int gas \wedge \neg flame \geq 5\right) \wedge \left(\int \neg gas \geq 1\right) \wedge \left(\int \neg flame \geq 4\right)$$

yielding

$$\left(\int gas \wedge \neg flame \geq 5\right) \wedge \left(\int \neg gas \geq 1\right)$$

The implications between state assertions S_1 and S_2 that appear as premises of the level 2 simplifications will be established using Reduced Ordered Binary Decision Diagrams (ROBDDs).

Using the results of Section 4.1 on page 43 we know that this implies that the construction of the two ROBDDs will have the running time of $\mathcal{O}(2^{|S_1|})$ and $\mathcal{O}(2^{|S_2|})$, respectively, where $|S|$ denotes the number of distinct states mentioned in state assertion S. To check the implication, we construct an ROBDD for $S_1 \Rightarrow S_2$ using the ROBDDs for S_1 and S_2. The worst case time complexity of this construction is $\mathcal{O}(2^{|S_1|+|S_2|})$. Finally, we must check whether the resulting ROBDD is constantly true, which takes only constant time $\mathcal{O}(1)$ because we must simply check whether or not the ROBDD consists solely of the terminal *true*.

Table 4.1: Level 1 Simplifications on DC Formulas

$$\neg\texttt{true} \rightarrow \texttt{false} \tag{4.1}$$
$$\neg\texttt{false} \rightarrow \texttt{true} \tag{4.2}$$
$$\neg\neg\phi \rightarrow \phi \tag{4.3}$$
$$\texttt{false} \wedge \phi \rightarrow \texttt{false} \tag{4.4}$$
$$\texttt{true} \wedge \phi \rightarrow \phi \tag{4.5}$$
$$\texttt{false} \vee \phi \rightarrow \phi \tag{4.6}$$
$$\texttt{true} \vee \phi \rightarrow \texttt{true} \tag{4.7}$$
$$\texttt{false} \frown \phi \rightarrow \texttt{false} \tag{4.8}$$
$$\phi \frown \texttt{false} \rightarrow \texttt{false} \tag{4.9}$$
$$\frac{n > 0}{\int \texttt{false} \geq n \rightarrow \texttt{false}} \tag{4.10}$$
$$\frac{n > k}{\int \phi \geq n \rightarrow \texttt{false}} \tag{4.11}$$

Table 4.2: Level 2 Simplifications on DC Formulas

$$\frac{n_1 \geq n_2, S_1 \Rightarrow S_2}{\int S_1 \geq n_1 \wedge \int S_2 \geq n_2 \rightarrow \int S_1 \geq n_1} \tag{4.12}$$
$$\frac{n_1 \geq n_2, S_1 \Rightarrow S_2}{\int S_1 \geq n_1 \wedge \neg \int S_2 \geq n_2 \rightarrow \texttt{false}} \tag{4.13}$$
$$\frac{n_1 \geq n_2, S_1 \Rightarrow S_2}{\neg \int S_1 \geq n_1 \wedge \neg \int S_2 \geq n_2 \rightarrow \neg \int S_2 \geq n_2} \tag{4.14}$$
$$\frac{n_1 \geq n_2, S_1 \Rightarrow S_2}{\int S_1 \geq n_1 \vee \int S_2 \geq n_2 \rightarrow \int S_2 \geq n_2} \tag{4.15}$$
$$\frac{n_1 \geq n_2, S_1 \Rightarrow S_2}{\neg \int S_1 \geq n_1 \vee \int S_2 \geq n_2 \rightarrow \texttt{true}} \tag{4.16}$$
$$\frac{n_1 \geq n_2, S_1 \Rightarrow S_2}{\neg \int S_1 \geq n_1 \vee \neg \int S_2 \geq n_2 \rightarrow \neg \int S_1 \geq n_1} \tag{4.17}$$

Table 4.3: Level 1 Simplification on State Assertions

$$\neg\texttt{true} \rightarrow \texttt{false} \tag{4.18}$$
$$\neg\texttt{false} \rightarrow \texttt{true} \tag{4.19}$$
$$\neg\neg S \rightarrow S \tag{4.20}$$
$$\texttt{false} \wedge S \rightarrow \texttt{false} \tag{4.21}$$
$$\texttt{true} \wedge S \rightarrow S \tag{4.22}$$
$$\texttt{false} \vee S \rightarrow S \tag{4.23}$$
$$\texttt{true} \vee S \rightarrow \texttt{true} \tag{4.24}$$

Table 4.4: Level 2 Simplifications on State Assertions

$$\frac{S_1 \Rightarrow S_2}{S_1 \wedge S_2 \rightarrow S_1} \tag{4.25}$$
$$\frac{S_1 \Rightarrow \neg S_2}{S_1 \wedge S_2 \rightarrow \texttt{false}} \tag{4.26}$$
$$\frac{S_1 \Rightarrow S_2}{S_1 \vee S_2 \rightarrow S_2} \tag{4.27}$$
$$\frac{\neg S_1 \Rightarrow S_2}{S_1 \vee S_2 \rightarrow \texttt{true}} \tag{4.28}$$

4.3 Inequality Simplifications

For the reasons stated in the introduction to this chapter, we shall only perform local simplifications on individual inequalities. As the reader may recall from section 2.3.1 on page 13, the inequations have the form

$$\sum_{i=1}^{m} a_i x_i \geq n$$

where $a_i \in \mathbf{N}\backslash\{0\}$, x_i is a literal and the threshold $n \in \mathbf{Z}$.

The simplifications are listed in Table 4.5 on the following page. They make use of two general rules:

1. If we know that the value of a literal is always *false*, the term in which it occurs will evaluate to zero and can simply be removed.

2. If we know that the value of a part of the sum on the left-hand side of the inequality is constant, we can remove this part and subtract the constant value from the right-hand side threshold.

As an example of the latter rule, we can simplify

$$1x_1 + 2x_2 + 1\neg x_2 + 1x_3 \geq 2$$

to

$$1x_1 + 1x_2 + 1x_3 \geq 1$$

since $2x_2 + 1\neg x_2$ will always evaluate to at least 1 — or more specifically, $1x_2 + 1\neg x_2$ has the constant value 1.

An inequality whose threshold is less than or equal to zero is trivially true and can be deleted from the set of constraints.

4.4 Propositional Clause Simplifications

The simplifications on clauses are equivalent to those made on the linear inequations of Section 4.3, only simpler since a propositional clause corresponds to a linear inequation where all weights are 1 and the threshold is 1. Thus, we shall not go into further detail — the simplifications are listed in Table 4.6 on the following page.

A clause that only contains the literal [`true`] can be deleted from the set of constraints.

Table 4.5: Inequality Simplifications

$$\sum_{i=1}^{m} a_i x_i + a_t[\texttt{true}] \geq n \rightarrow \sum_{i=1}^{m} a_i x_i \geq n - a_t \tag{4.29}$$

$$\sum_{i=1}^{m} a_i x_i + a_t \neg[\texttt{true}] \geq n \rightarrow \sum_{i=1}^{m} a_i x_i \geq n \tag{4.30}$$

$$\sum_{i=1}^{m} a_i x_i + a_{j1} x_j + a_{j2} x_j \geq n \rightarrow \sum_{i=1}^{m} a_i x_i + (a_{j1} + a_{j2}) x_j \geq n \tag{4.31}$$

$$\sum_{i=1}^{m} a_i x_i + a_j x_j + a_j \neg x_j \geq n \rightarrow \sum_{i=1}^{m} a_i x_i \geq n - a_j \tag{4.32}$$

$$\frac{a_{j1} > a_{j2}}{\sum_{i=1}^{m} a_i x_i + a_{j1} x_j + a_{j2} \neg x_j \geq n \rightarrow \sum_{i=1}^{m} a_i x_i + (a_{j1} - a_{j2}) x_j \geq n - a_{j_2}} \tag{4.33}$$

$$\frac{a_{j1} < a_{j2}}{\sum_{i=1}^{m} a_i x_i + a_{j1} x_j + a_{j2} \neg x_j \geq n \rightarrow \sum_{i=1}^{m} a_i x_i + (a_{j2} - a_{j1}) \neg x_j \geq n - a_{j_1}} \tag{4.34}$$

The reader should keep the commutativity and associativity of the + operator in mind when viewing the rules; these are not the complete set.

Table 4.6: Clause Simplifications

$$\bigvee_{i=1}^{m} x_i \vee [\texttt{true}] \rightarrow [\texttt{true}] \tag{4.35}$$

$$\bigvee_{i=1}^{m} x_i \vee \neg[\texttt{true}] \rightarrow \bigvee_{i=1}^{m} x_i \tag{4.36}$$

$$\bigvee_{i=1}^{m} x_i \vee x_j \vee x_j \rightarrow \bigvee_{i=1}^{m} x_i \vee x_j \tag{4.37}$$

$$\bigvee_{i=1}^{m} x_i \vee x_j \vee \neg x_j \rightarrow [\texttt{true}] \tag{4.38}$$

The reader should keep the commutativity and associativity of the $\vee$ operator in mind when viewing the rules; these are not the complete set.

Chapter 5

DC Formulas with Models of Elementary Length

Generally, the shortest models of DC formulas may have non-elementary length, as shown in [MF02]. One may therefore wonder if it really makes sense to use bounded model checking for validating DC formulas; it may be of questionable value to know that a particular formula has no models of some length k if this provides no guarantee that it may not have models of length $> k$.

However, for a class of DC formulas, we can achieve the desired property of a bound on the model length, as these formulas have models of elementary length, if any. In [MF02] it is argued that this is the case for any formula that does not contain unguarded negations. A guarded formula ψ occurs as $\ell < n \wedge \psi$, thus bounding the model length of the formula to at most $n - 1$.

As referenced in [ZH04], Skakkebæk et al. have used a reduction from the problem of determining emptiness of star-free regluar languages to deciding validity of DC formulas to prove the non-elementariness of DC validity checking, due to negations. We note that more specifically, it is chop below negation that causes this blow-up, because in the other cases, we can simply rewrite the formula to a negation normal form[1].

We will show how a bound on the model length can be determined for *any DC formula that does not contain unguarded chop below negation*. In Figure 5.1 on the next page, we have specified an algorithm FINDK for calculating the bound k on the model length of a formula ϕ while checking that it belongs to this class of formulas. Otherwise, an error will be raised. The algorithm shall assume that the formula has already been rewritten to NNF.

The FINDK function, which only takes ϕ as its parameter, uses an auxiliary recursive function FINDK' that returns a triple $(k, reqG, isG)$. k is of course the bound on the model length that we are looking for, while $reqG$ is a boolean value that indicates if ϕ requires a guard, i.e. it contains an unguarded chop below negation. isG is a boolean value that is true if and only if ϕ is (a conjunction containing) a formula $\ell < n$. The FINDK function can only return k if FINDK'$(\phi, false)$ returned $reqG = false$.

Most of the algorithm is quite straightforward. However, a few points must

[1] Recall that with discrete time, negated duration formulas, $\neg \int S \geq n$ are equivalent to $\int S \leq n - 1$

```
FINDK(φ)
    function FINDK'(φ)
        if φ = true or φ = false
        then return (0, false, false)
        else if φ = ∫S ≥ n
        then return (n, false, false)
        else if φ = ¬ψ
        then if ψ = ∫S ≥ n
                then return (n − 1, false, S = true)
                else if ψ = ψ1 ⌢ ψ2
                then return (0, true, false)
                else raise error (* Not in NNF*)
        else if φ = φ1 ∧ φ2
        then let (k1, reqG1, isG1) = FINDK'(φ1)
                 (k2, reqG2, isG2) = FINDK'(φ2)
             in  if isG1 and isG2
                 then return (min(k1, k2), false, true)
                 else if isG1
                 then return (k1, false, true)
                 else if isG2
                 then return (k2, false, true)
                 else return (k1 + k2, reqG1 ∨ reqG2, false)
             end
        else if φ = φ1 ∨ φ2
        then let (k1, reqG1, isG1) = FINDK'(φ1)
                 (k2, reqG2, isG2) = FINDK'(φ2)
             in return (max(k1, k2), reqG1 ∨ reqG2, isG1 ∧ isG2)
             end
        else (* φ = φ1 ⌢ φ2 *)
             let (k1, reqG1, isG1) = FINDK'(φ1)
                 (k2, reqG2, isG2) = FINDK'(φ2)
             in return (k1 + k2, reqG1 ∨ reqG2, false)
             end
    end FINDK'
    let (k, reqG, isG) = FINDK'(φ)
    in  if reqG
        then raise error (* Unguarded chop below negation *)
        else return k
    end
```

Figure 5.1: An algorithm for finding k, the bound on the model length, while checking that ϕ does not contain chop below negation.

be made:

- If FINDK' encounters a negation, and the operand of that negation is a chop, it returns $(0, true, false)$, i.e. it requires a guard, and is not a guard itself. The k value of 0 is arbitrarily chosen, since it will always be the required guard that bounds the model length — if no guard is present, the k value cannot be deduced.
- A conjunction has $isG = true$ if FINDK' on one of its children returned $isG = true$. In this case, the k value of the conjunction is the k value of the child that is a guard, or the minimum of the k values of the children, if both are guards.
- A disjunction has $isG = true$ if both of its children returned $isG = true$. In this case, the k value is the maximum of the k values of the children.
- Disjunctions and chop have $reqG = true$ if FINDK' on one of their children returned $reqG = true$. Similarly for conjunctions, unless FINDK' on one of the children returned $isG = true$, meaning that the child acts as a guard, hereby achieving a bound on the model length of the conjunction.

We shall conclude with a few examples of running the algorithm:

- For $\int x \geq 5 \wedge \int \neg x \geq 2$ we will find $k = 7$.
- For $\int x \geq 5 \wedge \int y \geq 2$ we will find $k = 7$ even though we could have gone as small as $k = 5$, but we have chosen $k = k_1 + k_2$ as a safe approximation, because a further analysis of the DC formulas and state assertions would quickly grow complex for larger formulas.
- For $\texttt{true} \vee \int x \geq 5 \vee \int y \geq 2$ we will find $k = 5$ even though the entire formula could have been simplified to `true` having $k = 0$. This indicates that it is probably a good idea to apply the DC simplifications of Section 4.2 on page 48 before running the FINDK algorithm.
- For $true \frown \int x \geq 5 \frown \int y \geq 2$ we will find $k = 7$.
- For $\neg \left(true \frown \int x \geq 5\right)$ we will get an error, because it contains unguarded chop below negation.
- For $\neg \left(true \frown \int x \geq 5\right) \wedge (\ell < 7)$ we will find $k = 6$ because $\ell < 7$ acts as a guard.

Chapter 6

Design

In this chapter, we shall focus on splitting the functionality into packages and classes (together called the *model elements*). Generally, we are trying to provide an answer to questions beginning with *where* rather than *how*, the latter being deferred to Chapter 7 on implementation (see page 62). In the following, we will describe the high-level design and give UML diagrams to provide an overview and illustrate dependencies between the model elements. First, we start by a brief introduction to the type of UML diagrams used in this chapter.

6.1 Brief UML Diagram Introduction

In a package diagram (such as the one in Figure 6.1 on the next page) the packages are represented by "folder" symbols. In a class diagram (such as the one in Figure 6.2 on page 59), classes are represented by boxes.

Arrows represent one-way associations, indicating a *has-a* relationship between the two elements. This will typically mean that the class from which the arrow originates has a field of the type of the class to which the arrow points.

Dashed lines with arrows represent dependencies, indicating a *uses-a* relationship, which is considered a weaker relationship than an association.

The symbol  indicates a boundary class, i.e. a class with an interface to the outside world. We shall use this symbol in connection with graphical user interface (GUI) classes.

6.2 Package Overview

We have grouped the classes forming the BMC/DCValidator implementation into a number of packages, with each package representing a coherent set of functionalities. The packages and their dependencies, along with short package descriptions, have been given in Figure 6.1 on the next page.

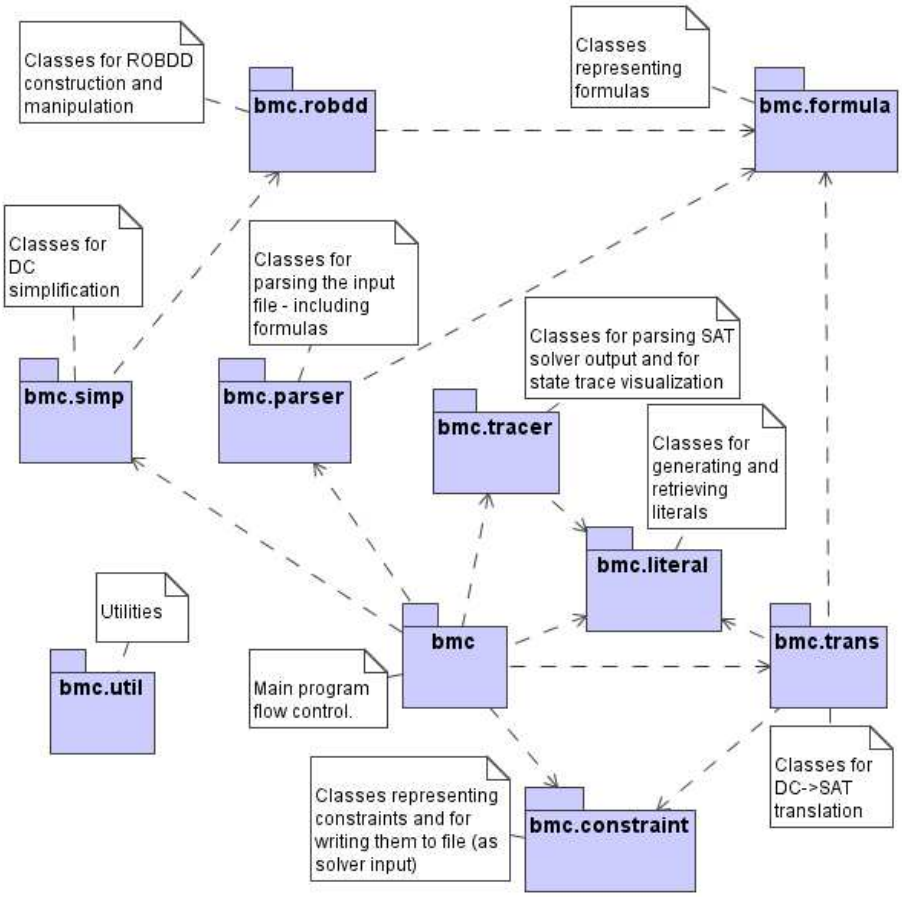

Figure 6.1: Package Diagram. The dependencies on/of package `bmc.util` have been omitted for readability.

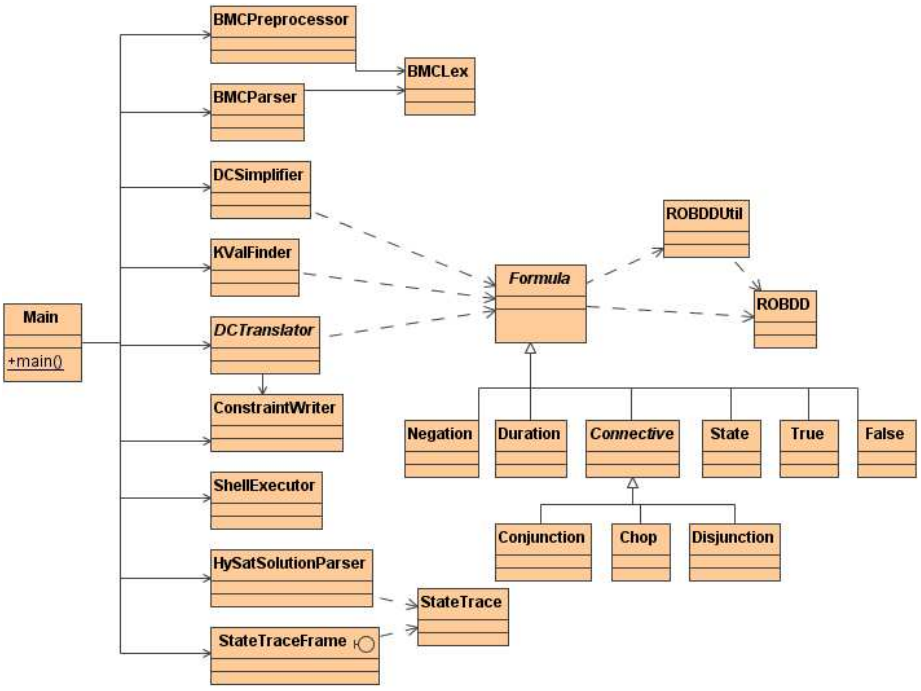

Figure 6.2: Main Class Diagram

6.3 Class Associations and Dependencies

6.3.1 Main

In Figure 6.2, we have illustrated the associations and dependencies between some of the key classes of the BMC/DCValidator. Prominently, the class `bmc.Main` governs the flow of control and instantiates a number of objects that are responsible for subtasks of the validation. The classes corresponding to those objects are:

bmc.parser.BMCPreprocessor Parses the input file and performs inlining of macro definitions.

bmc.parser.BMCParser Parses the result of the preprocessor and creates an internal representation of the formulas to be validated and the settings.

bmc.simp.DCSimplifier Performs simplification of DC formulas.

bmc.util.KValFinder Checks that the DC formula belongs to a class of formulas that have models of elementary length in the formula size, and determines that length.

bmc.trans.DCTranslator Translates the DC formula to an equisatisfiable SAT problem. The design of this class is described further in Section 6.3.3 on the next page.

bmc.constraint.ConstraintWriter Writes individual constraints to a file.

bmc.util.ShellExecutor Spawns a new process, e.g. for invoking the SAT solver.

bmc.trace.HySatSolutionParser Parses the textual output from the SAT solver.

bmc.trace.StateTraceFrame Displays state traces when a counter example has been found, using a graphical user interface.

6.3.2 Formulas

The DC formulas and state assertions (or SA formulas, as we shall refer to them in the context of design and implementation) are represented by an abstract superclass `Formula` with appropriate subclasses `Chop`, `Disjunction` etc. as illustrated in the diagram in Figure 6.2 on the preceding page.

No distinction is made between the DC and SA formulas, since this enables us to reuse code in e.g. the `DCSimplifier` where the DC formulas and SA formulas have common simplification rules for disjunctions and conjunctions. Where appropriate, type checking can be enforced by throwing exceptions when for example a duration formula is encountered where only SA formulas are expected. It is the responsibility of the `BMCParser` to ensure that the formulas to be validated specified by the user are in fact proper DC formulas.

A `Formula` object can save a reference to an ROBDD representing itself. It uses the `ROBDDUtil` class for constructing such an `ROBDD` object.

6.3.3 DC → SAT Translation

The `DCTranslator` is an abstract class whose subclasses are `DCToDIMACSTranslator` and `DCToZOLCSTranslator`. It is the values of the `Settings` object that is carried around with the formula to be validated (based on the user's specification in the input file), that determines which of the two is chosen. They implement the algorithms described in Appendix C on page 101 and produce constraints that are handed to the `ConstraintWriter`.

The `LiteralHandler` generates the literals needed to represent each subformula in different time instants, as described in Section 3.2 on page 16.

It is also responsible for controlling when regeneration of constraints may be avoided in the case of subformula recurrence. Directly corresponding to the three levels of formula recurrence recognition (with different time and space complexities) that were described in Section 3.7, we have three different subclasses of the `LiteralHandler`:

- `IDLookupLiteralHandler`
- `SyntacticLookupLiteralHandler`
- `SemanticLookupLiteralHandler`

As with the output format, it is the user's specification that determines which is selected.

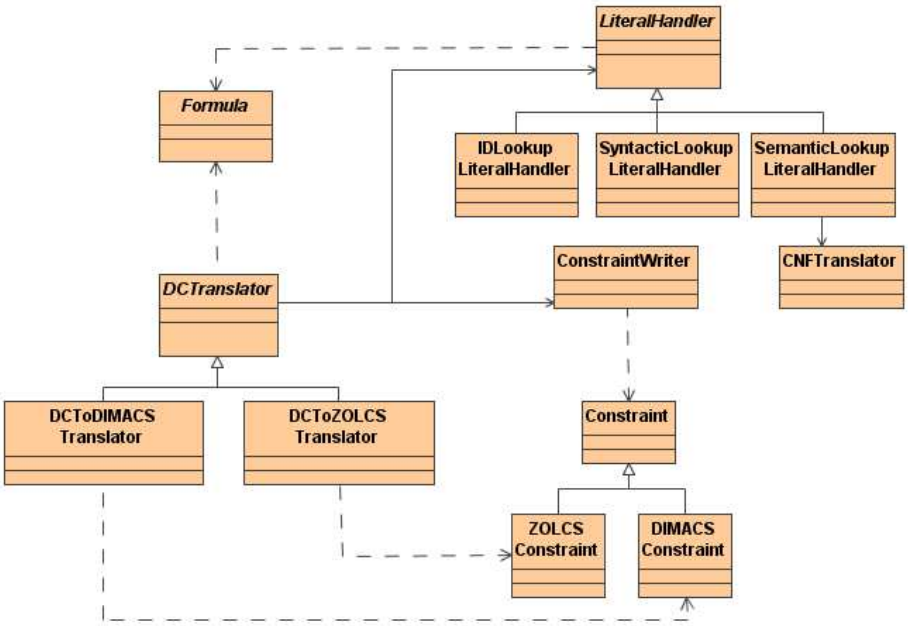

Figure 6.3: Class Diagram for DC → SAT Translation

Chapter 7

Implementation

This chapter shall describe the implementation of the individual classes of the BMC/DCValidator. The segregation of duties among the classes was done in Chapter 6 on Design.

We have chosen not to tire the reader with samples of the source code, but shall merely mention that the entire source code of the program may be found in Appendix I on page 275. The implementation has been written using the Java language.

7.1 Package bmc

7.1.1 Class Main

The `Main` class has a class-scope method `main(String[] args)` that enables it to act as a standalone Java application where the `args` are the command line arguments.

As mandatory arguments, one must specify an input file containing an ASCII representation of the formula(s) to be validated and various settings, and a command specifying where the SAT solver may be found. Optionally, one may specify a command for executing `shell(...)` commands in the input file in between the validations, e.g. for moving or renaming files. Also, one may specify whether counter example traces should be displayed or not. For more details on how to run the program, please refer to the User's Guide in Appendix D on page 107.

If there was an error in one of the command line arguments, the program usage is printed on screen. Otherwise, an instance of `Main` is created after some initial setup and the `run(...)` method is invoked on that instance.

The `run(...)` method will parse the input file using first the `BMCPreprocessor` and then the `BMCParser` with the scanner `BMCLex`. See Section 7.6 on page 68 for details.

In our terminology, the input file consists of various tasks. Each task is either

- a `Goal`, consisting of a `Formula` to be validated, the set of `State`s that may occur in the formula and some `Settings` or,
- a `Command` to be executed

For `Command`s, we simply invoke the `ShellExecutor` of Section 7.10.5 on page 75.

For `Goal`s, we must take a number of steps to determine validity of the formula:

1. A clock is started so we can time the work done by the frontend.
2. The formula is negated, since we are checking whether $\neg\phi$ is satisfiable when validating ϕ.
3. If specified in the `Setting`s, the formula is rewritten to NNF.
4. A `DCSimplifier` is instantiated.
5. The DC formula is simplified by the `DCSimplifier`, see Section 7.7.1 on page 70.
6. If specified in the `Settings`, the `k` value is calculated from the formula using the `KValFinder` as described in Section 7.10.3 on page 74.
7. A `java.io.RandomAccessFile` is instantiated where the SAT solver input file can be written.
8. Based on the `Settings`, a `LiteralHandler` is chosen and instantiated.
9. A `ConstraintWriter` is instantiated.
10. Based on the `Settings`, a `DCTranslator` is chosen and instantiated.
11. The `DCTranslator` (or, to be exact, an instance of one of its subclasses) translates the DC formula to a SAT problem. See Section 7.9 on page 72 for details.
12. The clock value is saved (giving the frontend time).
13. The SAT solver is executed using the `ShellExecutor`.
14. A `HySatSolutionParser` is instantiated and parses the output of the SAT solver as described in Section 7.8.1 on page 71.
15. The validity of the formula is printed on the screen, along with frontend and backend times, the latter being retrieved from the SAT solver's output by the `HySatSolutionParser`.
16. Depending on the command line parameters given, the state traces may be displayed using a `StateTraceFrame` (see Section 7.8.4 on page 72) if the formula turned out to be invalid.

7.2 Package bmc.robdd

7.2.1 ROBDD

The `ROBDD` class represents a Reduced Ordered Binary Decision Diagram, as described in Section 4.1 on page 43. It contains a `java.util.ArrayList` that holds the contents of table T. As the reader should recall, table T maps nodes

u to triples $(var(u), low(u), high(u))$. The index to the `ArrayList` is the ID for node u, while the contents of the list are `NodeDesc`s consisting of variable name and node IDs for $low(u)$ and $high(u)$. Each `ROBDD` object also holds a node ID for the root node.

Additionally, the `ROBDD` class contains a method `negate()` that creates a new `ROBDD` representing the negation of the boolean expression denoted by the `ROBDD` on which the method was invoked.

7.2.2 ROBDDUtil

The `ROBDDUtil` class contains utility methods for ROBDD creation and manipulation, according to the algorithms given in Section 4.1.1 on page 45. The public methods of the class are `makeROBDD(Formula formula)` that returns an `ROBDD` representing `formula`, and `apply(ROBDD left, ROBDD right, int operator)` that returns the `ROBDD` that is the result of applying the operator (given by an integer constant) to the two `ROBDD`s.

For `apply(...)`, the operators implemented are conjunction, disjunction and implication. The method is implemented in a straightforward way from the specification given in Figure 4.5 on page 47, with tables H and G being represented by `java.util.HashMap`s.

The `makeROBDD(...)` method begins by fetching the states mentioned in the formula, using the formula's `getStates()` method and sorting the list of states alphabetically. It then invokes `build(formula, stateList, 0)` with 0 being the index into `stateList` giving the variable name at which `build(...)` should start.

The recursive `build(...)` method will iterate through the state list and act as specified in Figure 4.4 on page 46. It uses an auxiliary `replace(...)` method for replacing the state with `true` and `false`. The `replace(...)` method will also collapse the formula when possible, by performing simplifications of type

$$\texttt{true} \wedge \texttt{true} \rightarrow \texttt{true}$$

This ensures that, when all state names have been replaced by boolean values, the formula will have collapsed to a single `true` or `false`. If the state list was empty, the `build(...)` method will invoke `replace(...)` just once (with a null state name) to uphold this, because the `true` or `false` can then be turned into a terminal node in the ROBDD.

7.3 Package bmc.constraint

7.3.1 Class Constraint

The `Constraint` class is an abstract class that is implemented by `DIMACSConstraint` and `ZOLCSConstraint`. It holds what is common between them, namely an `IntBag` for holding the literals in the constraint, and some abstract methods including `isTriviallyTrue()` which is used by the `DCTranslator` to discard trivially true constraints.

7.3.2 Class DIMACSConstraint

The heart of the `DIMACSConstraint` is the `addTerm(Literal literal)` method. If the special literal representing `true` (recognised by its number) is added to the constraint, an `isTriviallyTrue` flag is raised. The method `isTriviallyTrue()` will only return true when this flag has been raised. If the literal representing `false` is added, it is simply discarded. Otherwise, the list of literals is searched to check if the literal is already present in the constraint. If the literal is present with the same sign, it is discarded with no further ado. If it is present with opposite sign, the `isTriviallyTrue` flag is raised.

7.3.3 Class ZOLCSConstraint

The constructor of the `ZOLCSConstraint` takes a threshold value. A `ZOLCSConstraint` is trivially true if the value of the threshold does not exceed zero. The `addTerm(...)` method of this class takes both a `Literal` and a weight. Only literals with weights different from zero are added to the constraint. When the literal representing `true` is added, it is not put on the list of literals, but instead its weight is subtracted from the threshold. The literal representing `false` is simply discarded.

As with `DIMACSConstraints`, in all other cases the list of literals is searched. If a literal with the same number and sign is found, the weight of the new literal is just added to the weight of the existing one. If one is found with the same number, but opposite signs, the "common" weight is subtracted from the threshold, and the literal with the highest weight stays in the constraint, but has its weight reduced by the common amount. If their weights are equal, none of them will stay in the constraint (since their reduced weights are zero).

7.3.4 Class ConstraintWriter

When a `ConstraintWriter` is instantiated, it begins by writing some comments in the input file for the SAT solver, namely the original DC formula and the settings.

Space in the file is reserved for writing the number of variables and constraints (which has yet to be determined). We have chosen to reserve a total of 20 characters for these figures. The position of the file pointer is stored, so we can later return to this point and write the actual numbers. The choice of representing the output file as a `java.io.RandomAccessFile` enables us to use its `seek(...)` method to achieve this.

In its initialisation, the `ConstraintWriter` also adds the first constraint that forces the value of the special literal for `true` to true.

The `DCTranslator` (see Section 7.9.1 on page 72) uses the method `writeConstraint(Constraint constraint)` to add individual constraints to the SAT problem during the translation process.

When the translation is finished, the `finish()` method must be invoked. It will acquire the number of variables from the literal handler and write this number and the number of constraints into the reserved slots in the file.

7.4 Package bmc.formula

7.4.1 Class Formula

The class `Formula` is an abstract superclass for all DC and SA formulas. Each formula has a type, represented by an integer constant that enables us to use the efficient `switch` statement in Java to determine its type whenever we have a `Formula` at hand.

A `Formula` has an ID. If a `Formula` is cloned, it will retain the same ID as the original. All formulas that are not clones of one another will have different IDs — even if they are syntactically the same. In the event that a formula is changed due to e.g. simplifications, it is assigned a new ID.

For efficiency, a `Formula` saves a reference to its own string representation, once it has been calculated. Similarly, it saves a reference to an ROBDD representing it the first time the `getROBDD()` method is invoked. It uses the `makeROBDD(...)` method of `ROBDDUtil` (see Section 7.2.2 on page 64) to construct the ROBDD. As with IDs, these representations are only stored as long as the `Formula`'s constituents remain unchanged.

All `Formula`s have a reference to their parent, unless they are top-level. Also, methods `updateParent(Formula formula)` and `updateGrandParent(Formula formula)` are available to allow the formula itself or its parent, respectively, to be replaced with another formula. The implementation of these methods is based on Java's reflection mechanism[1].

7.4.2 Subclasses of Formula

The subclasses of `Formula` are:

- `True`
- `False`
- `State`
- `Duration`
- `Negation`
- `Connective` which in turn has subclasses
  - `Conjunction`
  - `Disjunction`
  - `Chop`

Their implementation is quite straightforward, each containing references to children (if any), a `toString()` implementation and a `getType()` method that returns the appropriate type constant.

[1] Java's reflection mechanism allows invocation of a method on an object by passing a reference to that method and object, rather than directly invoking the method on the object. This is necessary because the method to be invoked differs between the subclasses of `Formula`.

7.5 Package bmc.literal

7.5.1 Class Literal

A `Literal` is represented by a number and a sign. The `Literal` class contains some convenience methods for getting the special literals for `true` and `false`, for comparing two literals and for obtaining a literal with the same number but opposite sign of another literal.

7.5.2 Class VariableNoGenerator

The `VariableNoGenerator` is used to assign successive IDs to `Literal`s using the `getNextLiteralNo()` method. It also contains a method `getNumVariables()` that is used by the `ConstraintWriter` because it needs to tell the SAT solver how many variables to expect.

7.5.3 Class IndexedFormula

An `IndexedFormula` is a formula ID with indexes i, (i, j) or (i, m, j) corresponding to literals of type $[S]_i$, $[\phi]_{i,j}$ and $[\phi \frown \psi]_{i,m,j}$, respectively.

7.5.4 Class LiteralHandler

The class `LiteralHandler` is an abstract superclass for retrieving literals and checking formula recurrence. It provides a number of `getLiteral(...)` methods for the acquisition of literals and methods `addDef(Formula formula, boolean polarity)` and `hasDef(Formula formula, boolean polarity)` to set and check formula recurrence, respectively.

For each formula processed, a normalised ID is calculated. It is the responsibility of the subclass (`IDLookupLiteralHandler`, `SyntacticLookupLiteralHandler` or `SemanticLookupLiteralHandler`) to calculate it. In the literal handler, two formulas will be deemed equivalent if they have the same normalised ID.

For efficiency, a map from `Formula` IDs to normalised IDs are stored in the `LiteralHandler` so that the normalised ID need not be re-calculated when the same formula is repeatedly used as an argument to `getLiteral(...)`, which is typically the case when iterating over i and j.

Also, a map is created from `IndexedFormula` (using the normalised ID) to `Literal` so we can make sure to use the same literal whenever a particular normalised ID and index is referenced. If no such mapping exists, a new literal is generated using the `LiteralNoGenerator`.

We have incorporated into the `getLiteral(...)` method that formulas that are equivalent except that one is a negation of the other will use the same literals, only with different signs. This is done by "peeling off" the negations found at the outermost level of a formula and remembering whether we removed an odd or even number when setting the sign of the literal.

With the `addDef(Formula formula, boolean polarity)` method, we calculate a normalised ID for the formula and store that in a `java.util.Set`. If the polarity is false, we store the normalised ID with negative sign, since we should only recognise that constraints defining a subformula have already been generated if it was under the same polarity (because constraints are not the same for the two polarities).

The `hasDef(Formula formula, boolean polarity)` method checks if the normalised ID of the formula is a member of the above-mentioned set.

We also have a special `hasDef(State formula)` method that checks if a `State` has been referenced in any polarity. This is used by the `HySatSolutionParser` (see Section 7.8.1 on page 71) for determining whether or not to acquire state traces from the SAT solver's output for a particular state.

7.5.5 Class IDLookupLiteralHandler

The `IDLookupLiteralHandler` uses the ID of the `Formula` as the normalised ID of the `LiteralHandler`.

7.5.6 Class SyntacticLookupLiteralHandler

The `SyntacticLookupLiteralHandler` creates a string representation of the `Formula` using its `toString()` method. If another formula with the same string representation has previously been encountered, it uses that formula's ID as the normalised ID. Otherwise, the normalised ID is that of the formula itself, and a mapping from its string representation to its ID is stored.

7.5.7 Class SemanticLookupLiteralHandler

The `SemanticLookupLiteralHandler` first translates the formula to CNF using the `CNFTranslator` of Section 7.10.1 on page 74. Then, it sorts each clause and creates a string representation of it. The string representation of the clauses are then sorted, and a string representation of the entire formula is created. It is this representation that is used as a basis for finding the normalised ID in a way similar to that of the `SyntacticLookupLiteralHandler`: Formulas with identical string representations get the same normalised ID.

Note that when a chop is encountered, its children are rewritten to CNF and sorted, but the chop itself remains unchanged. Similarly, for duration formulas, the state assertion is rewritten to CNF and sorted.

7.6 Package bmc.parser

7.6.1 Class BMCLex

The lexical analyzer (or *lexer*, for short) class `BMCLex` has been generated by a lexer generator [JLex]. The lexer specification file that defines keywords, quoted strings etc. may be found in Appendix F.1 on page 117. It is quite straightforward, so we shall not go further into details.

7.6.2 Class BMCPreprocessor

The preprocessor has been handwritten after numerous attempts to produce a parser generator specification for it, when it finally dawned on us that we were not dealing with a context-free grammar! The problem resided in the fact that whether something should be substituted or not depends on the existence of a macro by that name.

The `BMCPreprocessor` uses a `BMCLex` that it acquires tokens from. It searches for macro definitions while keeping track of the state and tokens read since the beginning of the line, whenever it may have seen a macro definition. If it succeeds in finding a macro definition, it parses it and adds it to a map of such definitions. If it turned out not to be a macro definition after all, the tokens read so far can be flushed to the output. While parsing, the `BMCPreprocessor` is continuously checking if what it has read so far could be a macro call that must be inlined. If an ID token is encountered with the name of the macro itself, an exception is thrown as recursive macro definitions are not allowed.

A stack is held containing those tokens that must be processed before more tokens are retrieved from the `BMCLex`. When an identifier has been seen in a context where it could be (the start of) a macro call, the token containing the identifier is pushed before the `checkReplace()` method is invoked. It checks the name against the names of the defined macros, and if a match occurs, parameters are parsed and the macro definition (including substituted parameter values) is pushed onto the stack. If the identifier did not represent a macro name, the identifier token is pushed back onto the stack.

7.6.3 Classes BMCParser and BMCSymbols

The `BMCParser` and `BMCSymbols` classes have been generated with the CUP parser generator [JCup]. CUP generates LALR [2] parsers for Java in a similar way to the C parsers generated by the widely used program YACC.

The parser specification can be found in Appendix F.2 on page 119 and is rather similar to the input file specification given in BNF in Appendix E on page 115.

From the input file, the `BMCParser` creates a goal for each formula to be validated, including the settings as they were at the point of declaration of the formula. Different formulas in the same input file may thus have different settings attached to them. Shell commands to be executed may be interleaved with the goals. See the User's Guide (Appendix D on page 107) for details on the input format.

The parser supports an extended syntax beyond that supported by the `Formula` class and its subclasses, namely as described in Section 2.1.4 on page 9. E.g. the input may contain a biimplication, which will be rewritten to a conjunction of disjunctions by the parser. The only exception is that formulas of type $\int S \geq n$ with $n > 1$ will not be rewritten at this stage, since ZOLCS directly supports this type of construct. Hence, it is a context-sensitive decision whether to rewrite it or not, and it is deferred to the beginning of the actual translation process, see Section 7.9 on page 72.

When a state definition occurs, a `State` object will be created for that state. Each time the particular state is referenced in the formula(s) to be validated, a clone of the state object is used as subformula. This ensures that different variables will not be generated for the same state in the SAT problem, regardless

[2] LALR stands for "Look Ahead Left-to-right Rightmost", see [ASU86]. The "Look Ahead" part means that parser uses *lookahead* sets representing the symbols that may follow after having parsed some item I in parser state S rather than the *follow*-sets of LR parsers because this gives smaller parse tables. "Left-to-right" indicates the scanning direction of the input, and "Rightmost" hints to the fact that the parser constructs the rightmost derivation.

of the choice of literal handler[3], see Section 7.5.4 on page 67.

All binary connectives (conjunction, disjunction and chop) have been given the same precedence because we will require that users explicitly show the structure of the formula by using parenthesis — we believe that this gives the fewest misunderstandings.

7.7 Package bmc.simp

7.7.1 Class DCSimplifier

When initialising, the `DCSimplifier` retrieves the value of the DC simplification level from the `Settings` object. This level determines the effort that will be put into simplification of a formula subsequently fed to the `DCSimplifier`, using rules described in Chapter 4 on page 43.

The `DCSimplifier` provides two public methods: `simplifyDC(Formula formula)` and `simplifySA(Formula formula)` for simplifying DC formulas and state assertions, respectively. If the DC simplification level is `NONE`, both methods will return immediately. Otherwise, they invoke the recursive method `simplifyTree(...)` but with different parameter values.

The `simplifyTree(...)` methods traverses the tree by first reapplying itself to the children of a (sub-)formula and then applying level 1 simplifications to the formula itself. When a conjunction or disjunction is encountered, and its parent (if any) is not of the same type as itself, level 2 simplifications are applied before the level 1 simplifications.

For DC conjunctions and disjunction, the level 2 simplifications begin by populating a so-called `compareList`. This list contains the duration formulas all of whose ancestors are connectives of the same type as the formula to which the level 2 simplification is being applied. In this way, we can exploit the commutativity and associativity of the operators in question. Then, we compare the duration formulas to one another using the rules of Table 4.2 on page 50. If one of the formulas can be removed, we simply remove it from the `compareList` and use the `updateParent(...)` method that was described in Section 7.4.1 on page 66 to replace it with the neutral element (which depends on the operator). Level 1 simplifications will then remove this branch entirely. If we employ one of the rules that replace the two duration formulas with the zero element, we can simply substitute the zero element for the entire subformula that was to be simplified.

For SA conjunctions and disjunctions, the `compareList` is populated with all subformulas until a binary connective of the opposite type is encountered, in which case we add that subformula but do not go deeper into that branch.

For conjunctions, we then use a `SubsequenceROBDDConstructor` (see Section 7.7.2 on the following page) to check for each element of the `compareList`, if all of the other elements combined by conjunction imply that element. If so, it can be replaced by `true`, the neutral element of conjunction. If the conjunction of the other elements implies the negation of that element, the entire subformula can be replaced by `false`.

Symmetrically for disjunctions, we check if one element implies the disjunction of all of the other elements, in which case the element may be replaced by

[3]Since `Formula`s that are clones of one another will have the same ID.

`false`. And if the disjunction of the other elements are implied by the negation of the element, the entire subformula can be replaced by `true`.

7.7.2 Class SubsequenceROBDDConstructor

The `SubsequenceROBDDConstructor` is instantiated with a list of formulas. Its purpose is to create an ROBDD of the combination (using a selected operator) of all elements except one. One should iterate through the list, excluding one element at a time in a strictly increasing order. This class is used only by the `DCSimplifier`.

The only public method of the `SubsequenceROBDDConstructor` class is `getROBDD(int index, int operator)`. It returns a combination of all of the elements in the list except the one at position `index`. It does so by first acquiring an ROBDD for the prefix list and an ROBDD for the postfix list, and combining them using `BDDUtil.apply(...)`.

The postfix list is found by a recursive method, and all ROBDDs for postfix lists are stored in a map `postfixRobdds`. This also implies that it is the first call to `getPostfixROBDD(...)` that in fact creates all of the postfix ROBDDs.

The prefix list is calculated by combining the prefix list of the previous index with one more element. For this reason, prefix lists are stored in a map `prefixRobdds`.

7.8 Package bmc.tracer

7.8.1 Class HySatSolutionParser

The `HySatSolutionParser` parses the ASCII output from the HySat solver to achieve the following:

1. Determine if the SAT problem was satisfiable or not.
2. In case of a satisfiable problem, determine the values of the states of the problem at each clock value until k, forming a set of `StateTrace`s.

The actual parsing takes place in the `parse()` method. It uses a `java.io.LineNumberReader` to read the input (i.e. HySat's output) one line at a time. It goes forth as follows:

1. First, it reads lines until it reaches one that starts with `'STATUS:'`.
2. It then expects to read either `'Satisfiable'` or `'No solution'`, giving the result status.
3. Then, it searches for a line starting with `'cpu time for solving'` and reads that time, giving the backend time for the validation.
4. If graphical display of formula counter examples has been enabled, and the SAT problem was satisfiable, it begins to parse the solution[4]:
   (a) It searches for a line starting with `'Solution:'`.

[4]Note that the HySat source code originally given to us was modified slightly to get the solution printout.

(b) Immediately after this, the variable values are listed, separated by spaces. Variables whose value is true are simply listed by their number. Those whose value is false are preceded by `'-'` signs, and those whose value is immaterial (which we refer to as *wildcard* values) are preceded by `'*'` signs. All of the variable values are stored temporarily in an array, indexed by the number of the variable.

(c) The declared states are fetched from the `Goal`.

(d) For each state, the literal handler is queried for the variable number corresponding to that state. If it has such a number, we use the previously created array to find the value of the state. Otherwise, the state was declared but never referenced in the SAT problem, and we set its value to *wildcard*.

7.8.2 Class StateTrace

The `StateTrace` class represents a trace for a particular state. Its fields hold the state of interest and an array of values, representing the value of that state over a particular time interval. The values may be either *true*, *false* or *wildcard*, the latter meaning that the value is immaterial.

This class implements the `java.util.Comparable` interface to allow for the graphical user interface to display the state traces in alphabetical state order.

7.8.3 Class StateTracePanel

The `StateTracePanel` is a Java Swing GUI component, or more specifically a subclass of `javax.swing.JComponent`. It visualises a set of `StateTrace`s by overriding `JComponent.paint(Graphics g)` to draw axis and graphs corresponding to traces of the given states, using the basic `drawLine(...)`, `drawString(...)` and `fillRect(...)` methods of the `Graphics` object. Wildcard values are drawn as gray areas on the graph.

We shall spare the reader for further details of this class, as these mainly consist of tedious calculations of coordinates for placing the various graphic components.

7.8.4 Class StateTraceFrame

The `StateTraceFrame` is simply a `javax.swing.JFrame`, i.e. a window, that contains a `StateTracePanel`. An example of this GUI component may be seen in Figure 7.1 on the next page.

7.9 Package bmc.trans

7.9.1 DCTranslator

The `DCTranslator` is an abstract class for translating a DC formula to an equisatisfiable SAT problem, represented by `Constraints`. It is implemented by the `DCToDIMACSTranslator` and the `DCToZOLCSTranslator`, depending on the output type.

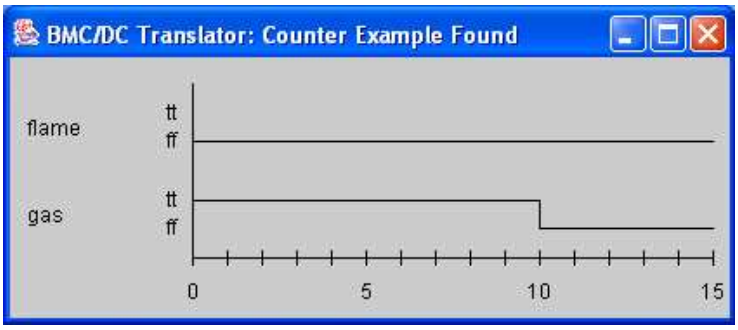

Figure 7.1: Counter Example / State Trace Window

The constructor saves the relevant settings: A `LiteralHandler` and a `ConstraintWriter`. The class contains a utility method `addConstraint(Constraint constraint)` that passes a constraint to the `ConstraintWriter` unless that constraint is trivially true — which is determined by the `Constraint` object itself, see Section 7.3.1 on page 64.

7.9.2 Class DCToDIMACSTranslator

Prominently, the `DCToDIMACSTranslator` class contains a `translate(Formula formula)` method that translates the given DC formula to `Constraint`s that are fed to the `ConstraintWriter` given in the constructor.

It implements the translation specified in Appendix C.2 on page 103 as closely as possible to better be able to argue for the correctness of the program.

The `translate(...)` method first flattens deep durations, i.e. rewrites DC formulas of type $\int S \geq n$ with $n > 1$ to n occurrences of $\int S \geq 1$. This is done by `flattenDeepDurations(...)` which, in turn, invokes the recursive method `chopUp(int n, Duration dur1)` where `n` is the number of occurrences of `dur1` to be created. This method creates a subtree of duration formulas that is as balanced as possible, because this will ultimately generate the fewest constraints due to our subformula recurrence recognition (described later in this section) — we will only need to generate constraints for one half of the tree because the other half will be deemed equivalent[5].

Returning to the `translate(...)` method, it then invokes the method `translateDC(Formula formula, boolean polarity)`, corresponding to the $t^k_{DC,p}$ function of the specification, with the polarity parameter set to `true`. If polarity optimisation is disabled, the `translateDC(...)` method is subsequently invoked with the polarity parameter set to `false`. Finally, a constraint is added that corresponds to the specification's $\bigvee_{i=0}^{k} [\phi]_{0,i}$ where ϕ is the formula to be translated.

The `translateDC(...)` method begins by checking if a definition of the given formula has already been made in this polarity. This is determined by invoking the `hasDef(...)` method of the `LiteralHandler`, see Section 7.5.4 on page 67. If so, the `translateDC(...)` method returns immediately.

[5]The two halves of the tree will be clones of one another and hence have the same ID, so even the simple `IDLookupLiteralHandler` will be able to determine their equivalence.

Otherwise, the type of the formula is determined and specific functions are invoked. The implementations of these are straightforward, using recursive calls to `translateDC(...)` for constituents, "`for`" loops for $\bigwedge_{i,j}$ and $\bigvee_{i,j}$ constructs and creating `DIMACSConstraints` by the addition of `Literal`s as specified.

For state assertions, an equivalent method `translateSA(...)` exists, corresponding to the $t^k_{SA,p}$ function of the specification.

7.9.3 Class DCToZOLCSTranslator

The `DCToZOLCSTranslator` is very similar to the `DCToDIMACSTRanslator`, except that it implements the translation described in Appendix C.3 on page 105 rather than that of Appendix C.2 on page 103. Also, it does not chop up deep durations, since these are directly supported in the translation algorithm for ZOLCS.

7.10 Package bmc.util

7.10.1 Class CNFTranslator

The `CNFTranslator` translates a `Formula` to CNF format in a two-step process in which the formula is first brought to NNF. It is straightforwardly implemented by two recursive methods, `translateToNNF(...)` and `translateToCNF(...)`.

7.10.2 Class IntBag

The `IntBag` class is an extensible bag of `int`s, i.e. an unordered collection that allows duplicate elements. It is used by `Constraint` and subclasses.

7.10.3 Class KValFinder

The class `KValFinder` contains a single public method: `findK(Formula formula)` whose implementation directly corresponds to the algorithm given in Figure 5.1 on page 55. An inner class `KValInfo` represents the triples of $(k, reqG, isG)$ that are return values of the auxiliary method `findK2(Formula formula)`, corresponding to the FINDK' function of the algorithm.

7.10.4 Class Settings

A `Settings` object contains all of the settings that the user may specify in an input file:

- Polarity optimisation: Enabled (default) / disabled
- Rewriting of the formula to NNF prior to translation: Enabled / disabled (default)
- Determining k for formulas with models of linear length: Enabled / disabled (default)
- DC simplification level: 0 / 1 (default) / 2
- Output format: ZOLCS (default) / DIMACS

- Formula recurrence recognition: By ID / syntactic (default) / semantic
- Bound on model length, k (default: 1)
- Output folder name (default: Current folder)

7.10.5 Class ShellExecutor

The `ShellExecutor` class provides functionality for executing a shell, based on the local operating system. It may either be initialised with a command string for the shell, or determine it from the `os.name` system property. In the latter case, it will select

/bin/sh -c for Linux and SunOS

command.com /C for Windows 95

cmd.exe /C for other Windows versions

When the `execCmd(String commandText)` method is invoked, the `ShellExecutor` will execute the command string described above with the `commandText` as the last argument, using `java.lang.Runtime`'s `exec(...)` method. It spawns a new process to execute the command but does not wait for it to finish.

Oppositely, the `execCmdAndFinish(String commandText, boolean output)` method *does* wait for the process to finish. It uses a `StreamGobbler` (see Section 7.10.6) to extract the process' output written on its standard output or error stream — otherwise, the ShellExecutor would deadlock due to the implementation of Java's `Process` class.

7.10.6 Class StreamGobbler

The `StreamGobbler` is a small utility class whose only purpose is to read bytes from an input stream, given as an argument to the constructor, until no more input is available or the thread in which it is running has been interrupted. It is used by the `ShellExecutor` and `Main` (in connection with the reading of input for the `HySatSolutionParser`).

Chapter 8

Test

In this chapter, we shall describe the tests we have performed on the BMC/DC-Validator. First, we will discuss the approach we have chosen. Then, we will explain the implementation of the test framework and test results.

8.1 Approach

We have chosen to perform several different kinds of tests because we believe that each kind of test has both strong and weak points, and by combining different approaches we may hopefully achieve a more covering test.

To the extent possible, we have tried to use automated tests because this allows for repeated runs of a large number of test cases each time the source code has been altered, e.g. to correct an error found in a previous run of the test. A few areas, such as the graphical user interface (GUI), cannot easily be subjected to automated tests so we have also exercised a small number of test cases manually.

We have made use of two main approaches: *White box* and *black box* test.

Some characteristics of the white box test are listed below, with + denoting an advantage and ÷ denoting a disadvantage:

+ Forces a thorough inspection of the code — this may lead to logical errors being detected by the reader

÷ The test itself will not catch logical errors because it is based on the code as is

+ The more complex the code (measured by the number of branches), the more test cases are created

+ Easy to locate and correct errors because the tests are very specific

÷ Encourages small test examples, so larger examples are typically not tested

Some characteristics of the black box test are:

+ Is based on the requirements, not a possibly flawed implementation

÷ Will most likely not cover all branches

+ Forces a specification or at least understanding of the pre- and postconditions of the functional units

÷ May be difficult to apply to units of code if they do not correspond to the functional units of e.g. a specification

÷ Difficult to locate and correct errors

÷ One error may balance another without the test detecting a problem

The white box and black box test supplement each other well, because the strength of one test lies in the areas where the other has its weaknesses.

We find, for the reasons stated above, that the white box test is most suited for unit testing, while the black box test is a good choice for testing the system as a whole.

Also, we have chosen to apply the black box approach to the core translator, which is a function unit for which we have a complete specification (see Appendix C on page 101). Furthermore, it would be difficult to write a white box test for the translator because its results are so complex that we would have to write another program to generate the expected results (in contrast with the hand-written expected results of the other white box unit tests), which would essentially be repeating the code of the translator!

Similarly, for the parser and lexer that have been auto-generated from a specification (see Appendix F on page 117), we have chosen to use a black box approach, since auto-generated code tends to be quite unreadable so it would be difficult to make a sensible white box test — we shall merely consider these classes as "black boxes" that should adhere to our specifications.

8.2 Implementation

8.2.1 Automated White Box Unit Test

The automated white box unit test uses the JUnit framework [JUnit]. This framework provides methods for checking assertions such as `assertTrue(...)` and `assertEquals(...)`, grouping of individual test cases into larger test suites, and a GUI for executing and viewing the results of the test. It is still, however, the responsibility of the programmer to define and implement the appropriate test cases.

Each test case consists of one assertion. The test cases for a single package are placed in a separate class `Test` that is located in that package. All `Test.java` files can be found in Appendix G.1 on page 126.

Now, we shall present an example of how a piece of the unit test for class `ZOLCSConstraint` has been created. Let us look at an exempt from the class' `addTerm(...)` method:

```
32          if (lit1 == -VariableNoGenerator.BOOL_LIT_NO) {
33              return;
34          }
35          if (lit1 == VariableNoGenerator.BOOL_LIT_NO) {
36              threshold -= weight;
37              return;
38          }
```

To test these statements, we want to see that when the literal representing `false` is added to the constraint, the constraint remains unchanged. And when the literal representing `true` is added, the threshold is reduced by the weight of that literal.

The following 5 lines taken from `Test.java` in package `bmc.constraint` should accomplish just that, using the string representation of the constraint:

```
136         /* cstr1 is a ZOLCSConstraint: 3*lit1 >= 3) */
137         cstr1.addTerm(falseLiteral, 2);
138         Assert.assertEquals("False literal added. Makes
                no changes.", "3 "+lit1.getNumber()+" 3 0\n",
                cstr1.toString());
139         cstr1.addTerm(trueLiteral, 3);
140         Assert.assertEquals("True literal added with
                threshold=3. Reduces threshold by 3.", "3 "+
                lit1.getNumber()+" 0 0\n", cstr1.toString());
```

The white box unit test suite consists of 1124 test cases.

8.2.2 Automated Black Box Test

The strategy for the automated black box test is to test the program as a whole using all types of DC and SA formulas (durations, conjunctions, chop etc.) as input, and with several different values of k, producing both valid and invalid examples. Also, we test all simplification rules, finding the bound on model length for each type of DC formula, and recurrences of formulas under different polarities.

The automated black box test runs the BMC/DCValidator using 91 different combinations of formulas and k with 72 different combinations of translation settings, giving a totalt of 6552 test cases. The results (valid/invalid) are checked against the expected results that are specified in a configuration file for each of the 91 combinations of formulas and k. Obviously, the translation settings (e.g. the choice of another output format) should not change the result.

The test cases are listed in Appendix G.2.2 on page 223. The source code for the Java class `BMCBBTest` that performs the automated black box test may be found in Appendix G.2 on page 216.

8.2.3 Manual Black Box Test

The manual black box test consists of four step-by-step tutorials, located in Appendix G.3 on page 228, that test the following aspects:

- Command line arguments and shells
- Larger, valid formulas and the use of "outputFolder"
- Larger, invalid formulas and the trace GUI
- Use of the option "findk"

8.3 Results

The automated white box unit test revealed a series of smaller errors which we have corrected along the way, and not surprisingly these errors were primarily found in the more complex parts of the code (such as the `DCSimplifier` and `CNFTranslator` classes). For instance, we found a copy-and-paste error in the `CNFTranslator` where some variable names had not been changed. The white box unit test has been run on Solaris, Linux and Windows XP (in the Cygwin environment) to ensure that the system-dependent `ShellExecutor` worked correctly on all of these platforms.

The automated black box test revealed errors of varying severity primarily in the `bmc.trans` package and the `bmc.Main` class.

As an example of a relatively simple error, in the `DCSimplifier` class we had forgotten a case label for the `Chop` formula type in a `switch` statement. This caused the program to give an error when a formula containing chop was encountered while running with the highest simplification level.

An example of a more severe error was that due to a wrong method name in a recursive defintion of the `translateSA(...)` method of the `DCTranslator`, a subformula was suddenly translated as a DC formula rather than as a state assertion. This could lead to (undetected) erronous output, such as the BMC/DCValidator reporting to the user that the formula was invalid, even if it was not.

The automated black box test ran our program (including parsing of input, simplification, translation, solving, and parsing of results) more than 6000 times in 5 minutes, implying an overall good performance of the BMC/DCValidator.

The manual black box test revealed one error: When supplying very large constraint systems to the HySat solver, the solver may abort with a "segmentation fault". Obviously, we cannot solve this problem ourselves[1], but it also turned out that the BMC/DCValidator did not handle such an error very well. The HySat solver is run in a separate thread using the `java.lang.Process` class, but our implementation faild to read from the process's error stream. Due to the mechanics of Java's `Process` class, this prevented the thread from terminating and caused the BMC/DCValidator to deadlock. This misbehaviour in the BMC/DCValidator was corrected by adding another thread that reads the process' error stream.

After correction of the above-mentioned errors, all tests have been run successfully, with the exception that HySat still aborts for very large constraint systems — but this is handled neatly by the BMC/DCValidator.

8.4 Evaluation

Our strategy has been to attack the program from several angles in search of potential errors. Specifically, we have performed both white box and black box tests on different levels of the program ranging from the on the smallest units possible (testing the results of individual statements), and on the system as a whole.

Creating the white box unit test cases allowed us to deliberately construct complex and unusual examples that have revealed programming errors in non-

[1] We know that Herde and Fränzle, the authors of the HySat solver, are currently working on a solution for this problem.

trivial parts of the code, e.g. the `DCSimplifier` and `SemanticLookupLiteralHandler` classes. Additionally, it gave us a second chance to have a close look at the code, making it possible for us to clean up, reorganise and even rewrite unnecessarily complex parts of the code.

The automated black box test revealed some errors that were not found in the white box test, for instance an error that consisted of a "missing branch", and errors occurring in the interplay between classes. The small, manual black box test suite was successful in testing the outskirts of the program that were inconvenient to test automatically.

All known errors have been corrected.

We find that the combination of the white box and black box approaches supplemented each other well. Having automated tests made it possible for us to work with a much larger number of test cases, and to run these repeatedly as errors were corrected. The time we have spent in constructing the test framework and the many test cases has been worthwhile because it has significantly increased our confidence in the correctness of the BMC/DCValidator.

Chapter 9

Benchmarks

In this chapter, we shall present some benchmarks for evaluating the performance of the BMC/DCValidator. We have chosen to primarily investigate two aspects:

1. Are there (combinations of) settings (polarity optimisation, output format etc.) that are clearly superior to others?

2. Is there a difference between the optimal settings for valid and invalid problems?

We could also have attempted to benchmark the BMC/DCValidator against other related tools, but have chosen not to. First of all, the description languages and/or expressiveness may be vastly different. Secondly, besides the problems involved in getting an often poorly documented tool (beyond the mere theoretical properties) up and running, it is difficult to create a fair benchmark test when we do not have a strong understanding of the kinds of examples that they are optimised for. Also, the implementation base may be different, so the results of such a comparison would not necessarily teach us much about the usefulness of the various approaches. For these reasons, we have chosen to concentrate on the properties of the BMC/DCValidator.

First, we present and formalise the two cases that will be used for benchmarking. Then, we describe the settings and environment in which the test shall be carried out, and finally we will discuss the results of the benchmarks.

9.1 Cases

9.1.1 Gas Burner

In Section 1.1.1 on page 2 we introduced the gas burner as an example of a real-time system. We shall now try to formulate the design decisions and requirements stated in that section as DC formulas.

First, we will define two states, *gas* and *flame*. The predicate *leak* (stating that the gas is leaking) shall be defined as $gas \wedge \neg flame$.

Design decision 1: *Leaks should be detected and stopped within one second.* This may be formulated as the requirement that if we have a period where *leak*

holds for more than one second, there must be some time within that period where *leak* does not hold, i.e. the leak has been stopped:

$$\Box\left(\int leak > 1 \Rightarrow \int \neg leak \geq 1\right) \tag{des1}$$

Design decision 2: *It should not be possible to switch on the gas for a period of 30 seconds after a leak.* In our subset of DC, this is not completely straightforward to formulate. However, we may state that if we have a period with leak, followed by a period without leak and then again a period with leak, the length of the interval in question must be at least 32:

$$\Box\left(\left(\int leak \geq 1 \frown \int \neg leak \geq 1 \frown \int leak \geq 1\right) \Rightarrow \ell \geq 32\right) \tag{des2}$$

Finally, we must formulate the safety requirement: *For any time interval at most 30 seconds long, the gas burner leaks no more than n seconds.* Using DC, this is formulated as follows:

$$\Box\left(\ell \leq 30 \Rightarrow \int leak \leq n\right) \tag{req1}$$

Now, we must validate that the conjunction of the two design decisions implies the safety requirement (which it does for any $n \geq 1$). Also, we may obtain an invalid problem by leaving out one of the design decisions. We have chosen to leave out design decision 2, stated in equation *des2*, for the invalid test cases.

9.1.2 Scheduler

In Section 1.1.1 on page 2, we reviewed another example of a real-time system, namely a scheduler. We shall now formulate the design decisions and requirement for the scheduler with n processes using DC. The state r_i will be used to indicate that process P_i is occupying the processor.

The design decisions were as follows: *Process P_{i+1} will run only when process P_i has completed its 2 time unit run for this period.*

$$\Box \bigwedge_{\substack{m\in\{1,\ldots,\lfloor k/2\rfloor\}\\ i\in\{1,\ldots,n-1\}}} \left(\int r_i < 2m \Rightarrow \int r_{i+1} \leq 2(m-1)\right) \tag{des1}$$

and: *Process P_1 will not run again until process P_n has completed its run.*

$$\Box \bigwedge_{m\in\{1,\ldots,\lfloor k/2\rfloor\}} \left(\int r_n < 2m \Rightarrow \int r_1 \leq 2m\right) \tag{des2}$$

Then, we will assume that the processor is in constant use. This could have been phrased in several ways, but we shall state the assumption as:

$$\ell = k \Rightarrow \int\left(\bigvee_{i\in\{1,\ldots,n\}} r_i\right) = k \tag{ass1}$$

As for safety requirements for the scheduler, we know that only a single process can be allocated the processor at any given time (mutual exclusion). This requirement could be stated as:

$$\Box \left(\bigwedge_{\substack{i \in \{1,\ldots,n\} \\ j \in \{i+1,\ldots,n\}}} \int r_i \wedge r_j = 0 \right) \tag{req1}$$

We shall also require that it is possible to schedule the processes so that each process has been allocated $2\lfloor \frac{k}{2n} \rfloor$ seconds of processor time after k seconds have elapsed.

$$\ell = k \Rightarrow \bigwedge_{i \in \{1,\ldots,n\}} \int r_i = 2\lfloor \tfrac{k}{2n} \rfloor \tag{req2}$$

It is now possible to validate that the conjunction of the design decisions and assumptions imply the two safety requirements (mutual exclusion and a fair schedule).

An invalid problem may be obtained by dropping one of the design decisions or assumptions. In the benchmarks we shall present next, equation *des2* has been removed in the invalid scheduler case.

9.2 Test Layout and Environment

We shall test each of the four cases (gas burner and scheduler in both valid and invalid versions) with all combinations of settings:

- Output format (zolcs / dimacs)
- Polarity optimisation (true / false)
- NNF (true / false)
- DC simplification level (0 / 1 / 2)
- Formula recurrence recognition level (ID / syntactic / semantic)

The gas burner cases will be tested with $k = 32$ and $k = 50$. The scheduler cases will be tested with $k = 24$. The input files may be found in Appendix H.3 on page 240. For the purpose of this test, we have created a small program that — given a test configuration – iterates through the combinations of settings, and streams the results of the BMC/DCValidator to a file. The source code of this program may be found in Appendix H.1 on page 230.

The test will be carried out in the Cygwin environment on Windows XP Professional on a computer with an AMD Athlon CPU of 1.66 GHz and 640MB RAM. The JVM has been given a maximum heap size of 200 MB.

9.3 Results

As one may understand from Section 9.2, the two test cases concerning the gas burner have been tested with 144 combinations of settings, while the two test cases concerning the scheduler have been tested with 72 combinations of settings. This yields a total of 432 test results for us to analyze. For this reason, we have chosen to place the test results in a database.

To achieve this, we have created a program for parsing the BMC/DCValidator output that was written to file, as well as the input files for HySat since these contain the specification of the settings used for a particular test and the resulting number of variables and constraints. The BMC/DCValidator output itself contains information about the result (valid / invalid / error) and — if an error did not occur — frontend and backend times. The source code of this parser is placed in Appendix H.2 on page 234. Its resulting output is a semicolon-separated file suitable for import into a database. Also, with a particular option, we were able to generate the contents of the LaTeX table that may be found in Appendix H.4 on page 245 and shows our complete test results.

Storing the benchmark results in a database allowed us to query the data to attempt to find patterns, e.g. by having the database count the number of cases in which the ZOLCS output format outperformed the DIMACS format and vice versa. Once the overall picture of the performance of a particular setting had been established, we were able to select the test results of interest to attempt to find reasons for this behavior, e.g. is it primarily for large values of k that ZOLCS outperforms DIMACS?

Our findings will be described in the following.

9.3.1 Output Format

As one would expect from the different translation schemes described in Section 3.8 on page 39, ZOLCS gives lower frontend translation times than DIMACS because formulas of type $\int S \geq n$ give rise to $2n - 1$ DC subformulas when the BMC/DCValidator uses the DIMACS output format, because the $\int S \geq n$ formula is rewritten to a chop tree[1] of depth $\lceil \log_2 n \rceil$. Each subformula gives rise to at least k constraints. Also, less constraints must be generated for ZOLCS than for DIMACS because it may represent implications more compactly. We see from our benchmark results that the frontend takes approximately three times as long in translating the benchmark examples to DIMACS as it does to ZOLCS.

It was, however, not clear what the effect of choosing one output format over the other would be on the total translation time. Generally, one would expect that the fewer constraints and variables, the faster HySat would be able to come up with the result. However, the SAT solving algorithms for ZOLCS and DIMACS differ since the ZOLCS constraints are inequalities with arbitrary thresholds and weights, i.e. $\sum a_i x_i \geq n$, while the DIMACS constraints are much simpler, namely $\sum x_i \geq 1$. The simple structure of the DIMACS constraints enables HySat to perform unit resolution on many of the clauses, while limiting the potentially expensive decision steps. This effect should be particularly visible for valid formulas in which all decision steps must be tried both ways. Our benchmark results show that for all larger formulas, ZOLCS outperforms DIMACS. The only case in which DIMACS has lower total validation time is for the smallest of the valid formulas in our test.

[1] The chop tree is constructed as balanced as possible to exploit formula recurrence recognition, see Section 7.9.2 on page 73

9.3.2 Polarity Optimisation

In the average case, there appears to be a performance gain from enabling polarity optimisation. We have not been able to reveal any distinct patterns (e.g. particularly for valid formulas) where it may or may not be recommendable to enable it. Therefore, we assume that this behaviour is mainly a result of HySat's heuristics for choosing variables and truth values for the decision steps.

9.3.3 NNF

Enabling the NNF option has a positive effect on validation time in about half of the cases and a negative effect in the other half. In particular, it seems that the combination of polarity optimisation and NNF can have a huge impact on the backend time. Let us look at some results for the valid gas burner with $n = 1$ and $k = 50$:

Polarity optimisation	**NNF**	**Backend time**
false	false	25s
false	true	1s
true	false	1s
true	true	8s

Clearly, it is not straightforward to conclude if NNF should be enabled or not. We should note that for other cases, the pattern may be quite different so that the lowest validation time e.g. may be obtained with both NNF and polarity optimisation enabled. As was the case with polarity optimisation, we suspect that the rather confusing results of NNF are caused by the heuristics employed by HySat.

9.3.4 DC Simplification Level

The DC simplification level does not make a significant difference. It is difficult to optimise carefully hand-written formulas, and we believe that the simplifications are probably most useful for machine-generated formulas. On the other hand, the fact that the validation time was unchanged by attempts of even ROBDD-based simplifications imply that the performance on these simplifications must be really good — so by enabling simplifications, one has the potiential of severely reducing validation time without risking any real penalty, should the simplification attempts be unsuccessful.

9.3.5 Formula Recurrence Recognition

In both the valid and invalid scheduler cases, the use of semantic formula recurrence recognition results in an "Out of memory" error by the frontend. This type of behaviour is not seen in the gas burner cases, simply because the number of subformulas in the gas burner example is significantly smaller than in the scheduler example. Recall that when semantic formula recurrence recognition is used, the formula calculates a CNF representation of each DC subformula and stores every unique CNF representation in a map. We have used a Java profiler to verify that it *is* indeed simply the number of CNF representations that grows too large. We will not consider this a severe problem since the user may choose to either:

a) Increase the BMC/DCValidator's maximum heap size by a parameter to the JVM, or,

b) Choose one of the other types of formula recurrence recognition

One may observe than in the cases where the BMC/DCValidator is able to use semantic formula recurrence recognition without experiencing an error, the validation times are the same as those with syntactic formula recurrence recognition. In most cases, the syntactic formula recurrence recognition gives the lowest total validation time but there does not appear to be large differences between the results of the three approaches.

9.3.6 HySat Errors

In some cases, HySat aborts with a severe error (a so-called "segmentation fault"). It appears to be for problems that are "large" in some sense: In most of the cases where HySat fails, `fRecognition = id` which is known to cause more constraints to be generated than with other types of formula recurrence recognition. The only cases in which `fRecognition = semantic / syntactic` leads to segmentation fault, are the valid gas burner examples for $k = 50$, `polarityOpt = false` and `outputType = dimacs`. The scheduler cases fail only with `outputType = dimacs`. We know that the DIMACS constraint systems contain significantly more constraints than ZOLCS. However, it is not merely a question of the number of constraints or variables: The lowest number of constraints and variables at which this behaviour has been observed is approx. 400.000 and 200.000, respectively. But other cases have been run with more than 1.200.000 constraints and 480.000 variables without such problems.

We are therefore not able to precisely pinpoint the nature of the cases that cause HySat to fail, but know that this is currently under investigation by the authors of HySat. Fränzle suggests that this may be the result of a flooding of HySat's clause database, as clauses are added during so-called conflict-driven learning.

9.3.7 Overall Performance Evaluation

To get an idea of the overall performance that a user could experience, the default configuration of `outputFormat = zolcs, polarityOpt = true, nnf = false, dcSimpLevel = 0` gives the following total valiation times:

Case	**Time**
Valid gas burner with $n = 1, k = 32$	5.4s
Valid gas burner with $n = 1, k = 50$	46.5s
Valid gas burner with $n = 6, k = 32$	2.5s
Valid gas burner with $n = 6, k = 50$	24.7s
Invalid gas burner with $n = 1, k = 32$	0.5s
Invalid gas burner with $n = 1, k = 50$	2.4s
Invalid gas burner with $n = 6, k = 32$	0.5s
Invalid gas burner with $n = 6, k = 50$	12.4s
Valid scheduler with $k = 24$	8.8s
Invalid scheduler with $k = 24$	2.0s

We are very pleased that the BMC/DCValidator is able to check the validity of all ten cases within these extremely reasonable time limits.

Finally, we will se how large values of k the BMC/DCValidator can handle for the gas burner case with $n = 1$:

Case	**Time**
Valid gas burner with $n = 1, k = 75$	784.1s
Valid gas burner with $n = 1, k = 100$	HySat error
Invalid gas burner with $n = 1, k = 75$	11.4s
Invalid gas burner with $n = 1, k = 100$	36.7s
Invalid gas burner with $n = 1, k = 200$	Frontend runs out of memory

For the valid gas burner with $k = 100$, HySat aborts with a segmentation fault, as described for a number of other cases in Section 9.3.6 on the preceding page. We have therefore not continued with larger values of k, even though the frontend was successful in generating the constraints for $k = 100$.

Clearly, it is much faster for HySat to find a counter example than to exclude the possibility of a such. Therefore, HySat has no problems in handling the invalid gas burner case with $k = 100$. However, for $k = 200$ the frontend runs out of memory due to an excessive amount of literals (recall that these are continuously stored in a map). This could potentially be remedied by increasing the maximum heap size of the Java Virtual Machine.

All in all, we find that the BMC/DCValidator performs very reasonably even when facing large problems.

Chapter 10

Discussion

In this chapter, we will discuss some key aspects of our work: How we turned our specifications into a running program, also in respect to the development process that it has undergone, and what choices we had to make along the way. We shall also discuss the usefulness of the resulting program, and relate to other approaches.

10.1 Implementation

When reading the report, it may appear as if the structure of the program followed directly from the theoretical work that was presented in Chapters 3 to 5. We must, however, admit that the program has undergone large structural changes during the development process. In our first approach, we generated a large formula of propositional logic (PL), corresponding to the complete translation of the DC formula, and held this in memory. The purpose of this was to apply both local and global simplifications to the PL representation. After these simplifications, the entire PL formula was rewritten to CNF (or a system of linear equations), which was also held entirely in memory for simplifications before it was finally written to file. It quickly turned out that this was not a feasible approach for larger formulas, because particularly the CNF representation grew unwieldy, even when the "smart" CNF algorithm of [PG86] was applied.

One lesson was that the global simplifications on PL/CNF did not really reduce the backend time, but in turn increased the frontend time significantly. This, combined with the memory problems experienced from holding the entire representation in memory caused the decision that the program should only hold parts of the formulas in memory at any one time; we would simply not generate the PL constraints for the next DC subformula before most of the PL constraints for the previous DC subformula had been simplified and rewritten to CNF. The implementation of the program was changed, so that three seperate threads were running concurrently: One thread that translated DC to PL, another thread that translated PL to CNF and a third thread that wrote the CNF to file. The threads exchanged data through two buffers. We employed Java's built-in `wait()` and `notify()` methods to ensure that a thread would not flood its buffer if the next thread could not keep pace (clearly, the thread that translated PL to CNF was the busiest).

This approach worked reasonably well, but generated quite large CNF/-ZOLCS problems. We realised that if we generated the CNF directly from the DC subformulas, it was possible to exploit our knowledge of the structure of the PL to generate much smaller CNFs than if we applied a generic CNF algorithm to the PL. In conjunction with this, we learned that the local simplifications on CNF were much better executed by the backend than by the frontend. The result of this was that we decided to generate CNF/ZOLCS constraints directly from the DC subformulas, and write them to file immediately. This decreased both the memory usage and validation time considerably, leading to the effective implementation that we have today.

10.2 Arguments for Correctness

10.2.1 Formal and Informal Proofs

In Section 3.4 on page 19, we have provided a proof for the correctness of the first, basic version of the translation algorithm. This proof has been augmented with more informal arguments for the correctness of polarity and negation optimisations, and optimisations for the specific output format. Obviously, we would have preferably provided formal proofs for all of these improvements, but found it too time-consuming to fit within the scope of this thesis. Also, these steps are considerably less complex than the translation algorithm itself, so we feel that the informal arguments have provided a sound basis for a belief in the correctness of the final translation algorithm(s). In general, it is also the case that a proof is never better than its authors — we hope that the extensive amount of time spent on constructing and refining the proof was sufficient to convince the reader.

10.2.2 From Specification to Program

The next interesting issue is then to what degree we can be sure that the implementation meets the specification. We could have created a complete formal specification of the program using e.g. the RAISE Specification Language (RSL). Tools are available to translate this type of specification to a running program (provided that the specification is refined to a sufficient level of concreteness). We considered this approach before we started the implementation: In previous projects we found that RSL was very good at expressing the functional requirements of the program, but completely ignored the non-functional requirements (prominently among these: the performance of the program). Since it was clearly stated in the work programme for this project that an efficient implementation was required, we decided to choose an approach in which we were more directly in control of the implementation.

Using the RSL approach we could have proved some properties of the specification using a proof assistant, and used a translator program to create the implementation — whose correctness would then depend on the authors of the translator! In our case, the correctness of the implementation relies on our abilities to transform a specification into program code.

10.2.3 Choice of Implementation Language

We chose to implement the BMC/DCValidator using the Java language. The main reason for this choice of language was our strong knowledge of Java where we could exploit our years of experience with the language and corresponding methodology. This enabled us to concentrate on the algorithmic aspects of the program, rather than the implementation details, because these were done using standard patterns and best practices. Also, we were able to experiment with different approaches without having to spend very much time on the implementation itself.

10.2.4 Tests

Since we chose to implement the program "by hand", it logically followed that a large amount of time should be dedicated to testing the program; invariably, some errors had been introduced. We attacked the program from three different angles: Through automated white box unit tests, automated black box tests and manual black box tests. Where the program code is generated by another program given a specification (as is the case with the parser and lexer), we have put emphasis on the black box tests. The white box unit tests require an understanding of the program code that one, in general, does not want to acquire from a generated program (which is typically quite unreadable). So if we had chosen to have the entire program generated from e.g. an RSL specification, we would only have performed black box tests to verify that the specification was indeed met by the implementation.

10.3 Usefulness of the BMC/DCValidator

During our work with the BMC/DCValidator, we have found the program easy to use, and DC to be well suited for formulating the benchmark examples. However, we can see that model checking in general has a weakness in that it is not possible to check generalisations of formulas. For the scheduler, we wanted to express constraints such as

$$\int r_i < 2m \Rightarrow \int r_{i-1} \leq 2(m-1)$$

We found that it was necessary to explicitly enumerate all values of i and m. If we contrast this with the theorem proving approach, we see that we could have possibly (using perhaps a proof assistant) proven the validity of the safety requirement in the general case, whereas this was not possible by model checking.

The great advantage of model checking is, in our view, the ease at which it is automated. It proved to be very fast to check the specifications even for large time intervals. Also, when we attempted to write the benchmark specifications, our first versions were flawed, but the errors were easy to find and correct using the counter example traces given by the BMC/DCValidator.

10.4 Comparison of Related Approaches

As argued in Section 9 on page 81, we have chosen to concentrate on benchmarking the BMC/DCValidator itself, rather than comparing it to other tools,

and gained some valueable insights from this. However, dispite the apparent good performance of the BMC/DCValidator it remains interesting to find out how it rates compared to other tools: Could it be a preferred way to do model checking in the future?

10.4.1 UPPAAL

We know from Section 2.2.2 on page 12 that the type of model checking performed by the BMC/DCValidator is NP-complete. UPPAAL suffers from state-space explosion, i.e. its complexity is PSPACE-complete. It has been shown that NP $\subseteq$ PSPACE but it is still unknown if NP = PSPACE or NP $\subset$ PSPACE. Since computer scientists expect (but have not shown) the latter, this could imply a general advantage of the approach of the BMC/DCValidator over that of UPPAAL. However, as was argued in Section 1.1.2 on page 4, UPPAAL continues to make performance improvements and algorithmic optimisations which may in practice prove more important than the theoretical differences.

10.4.2 Interval Duration Logic

Interval Duration Logic (IDL) is a variant of DC. As the reader may recall from Section 2.1.3, the models of DC are represented as *trajectories* that are "snapshots" of state values taken at a distance of one time unit. In contrast, the models of IDL connects each of these "snapshots" with a timestamp, so the distance between them may be far greater than a single time unit. These models are referred to as *timed-state sequence*. Potentially, this allows for a much more concise representation if the state values do not change too frequently.

In [SPC04], Pandya et. al. try to compare three different approaches to validity checking of IDL formulas:

1. Translation to discrete DC followed by automata-theoretic analysis using DCValid, a tool previously developed by Pandya
2. Translation to CNF followed by propositional SAT solving
3. Translation to linear constraints followed by the appropriate type of SAT solving (lin-SAT)

The latter two approaches should sound very familiar to the reader; they correspond to what our tool, the BMC/DCValidator, does for DIMACS and ZOLCS output format, respectively. Unfortunately, Pandya's results may not be worth much to us because he encodes the propositional problem as lin-SAT constraints even though a solver dedicated to solving CNF or pseudo-boolean constraint systems (as HySat that we have integrated the BMC/DCValidator with) would have had better performance.

We have not had the opportunity to benchmark our implementation of the work in [MF02] (including some algorithmic improvements for pseudo-boolean constraints) against an implementation of the lin-SAT approach in [SPC04]. However, as Pandya's results also support, the introduction of timestamps that removes the need to deal explicitly with state values at every single instant of time must certainly be expected to outperform the other two approaches, including that of the BMC/DCValidator.

10.5 Future Work

We feel that we have implemented all of the features that were described in the work programme that was set out for os. Additionally, we were able to come up with some improvements for the translation to ZOLCS, as described in Section 3.8.2 on page 41, that allowed us to generate smaller constraint systems. However, we have a few improvment suggestions that will be dealt with in the following.

10.5.1 Performance Improvements

If we look at the benchmark results, the performance of the BMC/DCValidator is generally good. For large values of k, the frontend runs out of memory due to a large number of stored literals. In the current implementation, it is necessary to store a map from (normalised[1]) IDs of formulas to literals so that we can ensure that the same literal is always used to represent a subformula in a given time interval in all of the constraints where it is referenced. This could be avoided by calculating literal numbers rather than looking them up. An algorithm for calculating the literal would have to return unique literal numbers for each subformula and time interval. In itself, this is not difficult if the number of subformulas and k is given. However, the HySat solver assumes that variables are numbered successively and reacts very strongly to an increase in the number of variables. So the algorithm should generate numbers that are "close" without overlapping. This is quite complex due to for instance simplifications in which some subformulas may be removed, and the fact that not all types of subformulas generate the same number of variables: Chops occurring with positive polarity generate more variables than the other types of formulas, ie. a `true` or `false` subformula generate no variables at all. If the authors of HySat could solve the problem of sensitivity to variables not mentioned in the constraints, calculation of literal numbers could be a solution that would enable the BMC/DCValidator to be able to handle even larger problems.

10.5.2 Extension of the Proof

It would perhaps be desirable to extend the proof of correctness of the core algorithm to cover the full, more complex algorithm for which only informal arguments have been given. Also, a formal proof could be given for the algorithm for finding the bound on the model length for a certain class of formulas, as given in Figure 5.1 on page 55.

[1] Depending on the chosen type of formula recurrence recognition, see Section 7.5.4 on page 67.

Chapter 11

Conclusion

We have implemented a tool, the BMC/DCValidator, for automated validation of formulas of the DC logic. The BMC/DCValidator translates DC formulas to propositional logic or linear constraint system that may be solved by an external solver.

The work programme that was set out for us demanded an efficient implementation of the ideas of [MF02], including extensions such as the identification of common subexpressions and semantics-preserving simplifications. We have implemented all of these requirements, including some optimisations for specific output formats, and benchmarks have shown that the BMC/DCValidator performs well on various interesting cases.

In addition to the requirements of the work programme, we have constructed a formal proof for the correctness of the core translation algorithm of the BMC/DCValidator. Informal arguments have been provided for the extension to the full, more complex algorithm. The implementation has undergone thorough automated and manual tests from several different angles to ensure its proper behaviour.

Conclusively, we are therefore very pleased with the results of this project and of the extent to which we have solved the given assignment.

Bibliography

[AS01] A. C. Shaw:
Real-Time Systems and Software
John Wiley & Sons, Inc., 2001.

[ASU86] A. V. Aho, R. Sethi, J. D. Ullman:
Compilers — Principles, Techniques and Tools
Addison-Wesley, 1986.

[Ba95] P. Barth:
A Davis-Putnam Enumeration Algorithm for Linear Pseudo-Boolean Optimization
Technical Report MPI-I-95-2-003, Max-Plack-Institut for Informatik, Saarbrücken, Germany, 1995.

[BDL] G. Behrmann, A. David, K. G. Larsen:
A Tutorial on Uppaal
http://www.uppaal.com

[CHR91] Z. Chaochen, C. A. R. Hoare, A. P. Ravn:
A Calculus of Durations
In *Information Processing Letters*, 40(5):269-276, 991.

[DLL62] M. Davis, G. Logemann, D. Loveland:
A Machine Program for Theorem Proving
In *Communications of the ACM* 5(7), 1962.

[DP60] M. Davis, H. Putnam:
A Computing Procedure for Quantification Theory
In *Journal of the ACM*, 7:210-205, 1960.

[DSE] Dedicated Systems Encyclopedia
http://www.omnio.be/encyc/

[FH03] M. Fränzle, C. Herde:
Efficient SAT Engines for Concise Logics: Accelerating Proof Search for Zero-One Linear Constraint Systems
In A. Voronkov, ed., *Logic for Programming, Artificial Intelligence and Reasoning*, volume 2850 of *LNCS, subseries LNAI*, pages 302-316. Springer Verlag, sep 2003.

[FH04] M. Fränzle, C. Herde:
Efficient Proof Engines for Bounded Model Checking of Hybrid Systems
2004.

[HA97] H. R. Andersen:
An Introduction to Binary Decision Diagrams
1997/98.
http://www.it.dtu.dk/~hra/

[JCup] CUP Parser Generator for Java
http://www.cs.princeton.edu/~appel/modern/java/CUP/

[JK98] J.-P. Katoen: Concepts, Algorithms, and Tools for Model Checking
Friedrich-Alexander Universität Erlangen-Nürnberg, Lecture Notes 1998/99.

[JLex] JLex: A Lexical Analyzer for Java
http://www.cs.princeton.edu/~appel/modern/java/JLex/

[JUnit] JUnit Regression Testing Framework
http://www.junit.org

[JW] M. C. Daconta:
When Runtime.exec() won't
JavaWorld, December 2000.
http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-traps.html

[LP00] K. G. Larsen, P. Pettersson:
Timed and Hybrid Systems in UPPAAL2K
Presentation given at MOVEP'2k, 2000.
http://www.uppaal.com

[MF02] M. Fränzle:
Take it NP-easy: Bounded Model Construction for Duration Calculus
In *Lecture Notes in Computer Science*, vol. 2469. Springer-Verlag, 2002.

[MF04] M. Fränzle:
Exam project "Bounded Model Construction for Duration Calculus"
http://www.imm.dtu.dk/cs/eksamensprojekter/E2004/1110.htm

[PG86] D. A. Plaisted, S. Greenbaum:
A Structure-Preserving Clause Form Translation
In *Journal of Symbolic Computation*, 2:293-304, 1986.

[SPC04] B. Sharma, P. K. Pandya, S. Chakraborty:
Bounded Validity Checking of Interval Duration Logic
2004.

[TR02] T. M. Rasmussen:
Interval Logic — Proof Theory and Theorem Proving
Ph. D. Thesis, Informatics and Mathematical Modelling, DTU, 2002.

[Tse68] G. Tseitin:
On the Complexity of Derivations in Propositional Calculus
In A. Slisenko, ed., *Studies in Constructive Mathematics and Mathematical Logics*, 1968.

[UPP] UPPAAL Homepage
http://www.uppaal.com

[WI] Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/

[ZH04] Z. Chaochen, M. R. Hansen:
Duration Calculus — A Formal Approach to Real-Time Systems
Springer-Verlag, 2004.

Appendix A

Symbol Index

The symbol index references the page where a given symbol was defined.

Symbol	Signature	Description	Page
DC		The set of duration calculus formulas	7
SA		The set of state assertions	7
$State$		The set of states	7
$Time$	$\mathbf{N}$	Time	7
Lit_{SA}		The set of literals representing state assertions at particular time instants	16
Lit_{DC}		The set of literals representing duration calculus formulas at particular time intervals	16
Lit		The union of Lit_{SA} and Lit_{DC}	16
$\overline{Lit_{SA}}$		The extension of Lit_{SA} to boolean combinations of the literals	17
$\overline{Lit_{DC}}$		The extension of Lit_{DC} to boolean combinations of the literals	17
$\overline{Lit}$		The extension of Lit to boolean combinations of the literals	17
ϕ, ϕ_i, ψ	DC	Duration calculus formula	7
S, S_i	SA	State assertion	7
σ	$State \to \mathbf{B}$	Valuation of states	8
$\hat{\sigma}$	$Lit_{SA} \to \mathbf{B}$	Valuation of literals representing state assertions	19
$\overline{\hat{\sigma}}$	$\overline{Lit_{SA}} \to \mathbf{B}$	Valuation of boolean combinations of literals representing state assertions	20

ρ	$Time \to State \to \mathbf{B}$	Trajectory, i.e. time-dependent valuation of states	7
$\tilde{\rho}$	$Lit \to \mathbf{B}$	Valuation of literals	25
$\overline{\tilde{\rho}}$	$\overline{Lit} \to \mathbf{B}$	Valuation of boolean combinations of literals	25
ξ	$Lit \to \mathbf{B}$	Valuation of literals	25
$\overleftarrow{\xi}$	$Time \to State \to \mathbf{B}$	Trajectory, i.e. time-dependent valuation of states	23
$\mathcal{I}[\![S]\!](\sigma)$	$\mathbf{B}$	Interpretation of S in σ	8
$\rho, [i,j] \models \phi$	$\mathbf{B}$	Observation $(\rho, [i,j])$ satisfies ϕ	7
$[S]_i$	Lit_{SA}	Literal	16
$[\phi]_{i,j}$	Lit_{DC}	Literal	16
$[\phi \frown \psi]_{i,m,j}$	Lit_{DC}	Literal	41
$[\mathtt{true}]$	Lit	Literal	18
x_i,y_i	Lit	Literal	13
a_i	$\mathbf{N} \setminus \{0\}$	Weight	13
V	$\mathcal{P}(State)$	Set of state variables	10
$\frown$	$DC \times DC \to DC$	Chop	7
$\int S \geq n$	DC	Duration formula	7
$t^k_{DC}[\![\phi]\!]$	$\overline{Lit}$	Translation of subformula ϕ	17
$t^k_{DC,p}[\![\phi]\!]$	$\overline{Lit}$	Translation of subformula ϕ with polarity p	36
$t^k_{SA}[\![S]\!]$	$\overline{Lit_{SA}}$	Translation of S	18
$t^k_{SA,p}[\![S]\!]$	$\overline{Lit_{SA}}$	Translation of S with polarity p	36
$BMC[\![\phi, k]\!]$	$\overline{Lit}$	Complete translation of ϕ	18
$\Box$	$DC \to DC$	For all subintervals	9
$\Diamond$	$DC \to DC$	For some subinterval	9
$\phi \to \psi$		Simplification rule	48

Appendix B

Complexity Classes

In this appendix, we shall briefly describe the different complexity classes that are referenced in this thesis. The contents of this chapter are based on [WI].

B.1 The P and NP Complexity Classes

A decision problem that can be solved in polynomial time by a deterministic machine is said to be a member of the polynomial (P) complexity class. Equivalently, a decision problem that can be solved in polynomial time by a non-deterministic machine is said to be a member of the non-deterministic polynomial (NP) complexity class.

The relation between the two classes is P $\subseteq$ NP. It is commonly believed that P is in fact a proper subset of NP, but this claim has not been proven.

Decision problems in the complexity class NP-hard are those problems which are at least as hard as any problem in NP.

The NP-complete problems are the hardest problems in NP, i.e. the intersection between the complexity classes NP and NP-hard.

B.2 The PSPACE Complexity Class

The decision problems that are solvable in polynomial space on a deterministic machine are said to reside in the polynomial space (PSPACE) complexity class.

Equivalent to the relation between the NP and NP-complete complexity classes, the PSPACE-complete problems are the hardest problems in the PSPACE complexity class.

It is known that NP $\subseteq$ PSPACE and suspected that NP $\subset$ PSPACE, i.e. that the problems in PSPACE are harder than the problems in NP.

B.3 ELEMENTARY Complexity Class

The problems in the ELEMENTARY complexity class are solvable by a deterministic machine in time $\mathcal{O}(2^{n^k})$, $\mathcal{O}(2^{2^{n^k}})$, $\mathcal{O}(2^{2^{2^{n^k}}})$, ...

B.4 NONELEMENTARY Complexity Class

The NONELEMENTARY problems are simply those that are not ELEMENTARY.

Appendix C

Translation Algorithm

Appendix C.1 contains the basic version of the translation algorithm. It is this version that has been proved correct in Section 3.4 on page 19.

Appendix C.2 contains the translation algorithm that outputs in DIMACS format, while the algorithm in Appendix C.3 translates to ZOLCS format.

C.1 Basic Translation Algorithm

BMC $[\![\phi, k]\!]$ gives the translation of DC formula ϕ with model length at most k.

Note that $[\texttt{true}]_{i,j} \stackrel{\text{def}}{=} [\texttt{true}]$ and $[\texttt{false}]_{i,j} \stackrel{\text{def}}{=} \neg[\texttt{true}]$.

$$
\begin{aligned}
\text{BMC}\,[\![\phi, k]\!] &= t^k_{\text{DC}}\,[\![\phi]\!] \wedge \bigvee_{i=0}^{k} [\phi]_{0,i} \wedge [\texttt{true}] \\
t^k_{\text{DC}}\,[\![\texttt{true}]\!] &= \epsilon \\
t^k_{\text{DC}}\,[\![\texttt{false}]\!] &= \epsilon \\
t^k_{\text{DC}}\,[\![\textstyle\int S \geq 1]\!] &= t^k_{\text{SA}}\,[\![S]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k-1\}\\ j\in\{i+1,\ldots,k\}}} \left([\textstyle\int S \geq 1]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j-1} [S]_m\right) \wedge \\
&\qquad \bigwedge_{i\in\{0,\ldots k\}} \neg[\textstyle\int S \geq 1]_{i,i} \\
t^k_{\text{DC}}\,[\![\neg\phi_1]\!] &= t^k_{\text{DC}}\,[\![\phi_1]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\neg\phi_1]_{i,j} \Leftrightarrow \neg[\phi_1]_{i,j}) \\
t^k_{\text{DC}}\,[\![\phi_1 \wedge \phi_2]\!] &= t^k_{\text{DC}}\,[\![\phi_1]\!] \wedge t^k_{\text{DC}}\,[\![\phi_2]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \wedge [\phi_2]_{i,j}) \\
t^k_{\text{DC}}\,[\![\phi_1 \vee \phi_2]\!] &= t^k_{\text{DC}}\,[\![\phi_1]\!] \wedge t^k_{\text{DC}}\,[\![\phi_2]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\phi_1 \vee \phi_2]_{i,j} \Leftrightarrow [\phi_1]_{i,j} \vee [\phi_2]_{i,j}) \\
t^k_{\text{DC}}\,[\![\phi_1 \frown \phi_2]\!] &= t^k_{\text{DC}}\,[\![\phi_1]\!] \wedge t^k_{\text{DC}}\,[\![\phi_2]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} \left([\phi_1 \frown \phi_2]_{i,j} \Leftrightarrow \bigvee_{m=i}^{j} ([\phi_1]_{i,m} \wedge [\phi_2]_{m,j})\right)
\end{aligned}
$$

$$
\begin{aligned}
t^k_{\text{SA}}\,[\![\texttt{true}]\!] &= \epsilon \\
t^k_{\text{SA}}\,[\![\texttt{false}]\!] &= \epsilon \\
t^k_{\text{SA}}\,[\![s]\!] &= \epsilon \quad \text{for } s \in \mathit{State} \\
t^k_{\text{SA}}\,[\![\neg\phi_1]\!] &= t^k_{\text{SA}}\,[\![\phi_1]\!] \wedge \bigwedge_{i\in\{0,\ldots k-1\}} ([\neg\phi_1]_i \Leftrightarrow \neg[\phi_1]_i) \\
t^k_{\text{SA}}\,[\![S_1 \wedge S_2]\!] &= t^k_{\text{SA}}\,[\![S_1]\!] \wedge t^k_{\text{SA}}\,[\![S_2]\!] \wedge \bigwedge_{i\in\{0,\ldots,k-1\}} ([S_1 \wedge S_2]_i \Leftrightarrow [S_1]_i \wedge [S_2]_i) \\
t^k_{\text{SA}}\,[\![S_1 \vee S_2]\!] &= t^k_{\text{SA}}\,[\![S_1]\!] \wedge t^k_{\text{SA}}\,[\![S_2]\!] \wedge \bigwedge_{i\in\{0,\ldots,k-1\}} ([S_1 \vee S_2]_i \Leftrightarrow [S_1]_i \vee [S_2]_i)
\end{aligned}
$$

C.2 Translation to DIMACS Format

BMC $[\![\phi, k, o]\!]$ gives the translation of DC formula ϕ with model length at most k and `polarityOpt` $= o$.

Note that $[\texttt{true}]_{i,j} \stackrel{\text{def}}{=} [\texttt{true}]$ and $[\texttt{false}]_{i,j} \stackrel{\text{def}}{=} \neg[\texttt{true}]$.

$$
\begin{aligned}
\text{BMC}\,[\![\phi, k, \text{tt}]\!] &= t^k_{\text{DC},+}\,[\![\phi]\!] \wedge \bigvee_{i=0}^{k} [\phi]_{0,i} \wedge [\texttt{true}] \\
\text{BMC}\,[\![\phi, k, \text{ff}]\!] &= t^k_{DC,+}\,[\![\phi]\!] \wedge t^k_{DC,-}\,[\![\phi]\!] \wedge \bigvee_{i=1}^{k} [\phi]_{0,i} \wedge [\texttt{true}] \\
t^k_{\text{DC},p}\,[\![\texttt{true}]\!] &= \epsilon \\
t^k_{\text{DC},p}\,[\![\texttt{false}]\!] &= \epsilon \\
t^k_{\text{DC},+}\,[\![\textstyle\int S \geq 1]\!] &= t^k_{\text{SA},+}\,[\![S]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k-1\}\\ j\in\{i+1,\ldots,k\}}} \left(\neg[\textstyle\int S \geq 1]_{i,j} \vee [S]_i \vee [\textstyle\int S \geq 1]_{i+1,j}\right) \wedge \\
&\qquad \bigwedge_{i\in\{0,\ldots k\}} \neg[\textstyle\int S \geq 1]_{i,i} \\
t^k_{\text{DC},-}\,[\![\textstyle\int S \geq 1]\!] &= t^k_{\text{SA},-}\,[\![S]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k-1\}\\ j\in\{i+1,\ldots,k\}}} \left(([\textstyle\int S \geq 1]_{i,j} \vee \neg[S]_i) \wedge \right. \\
&\qquad\qquad \left.([\textstyle\int S \geq 1]_{i,j} \vee \neg[\textstyle\int S \geq 1]_{i+1,j})\right) \\
t^k_{\text{DC},+}\,[\![\neg\phi_1]\!] &= t^k_{\text{DC},-}\,[\![\phi_1]\!] \\
t^k_{\text{DC},-}\,[\![\neg\phi_1]\!] &= t^k_{\text{DC},+}\,[\![\phi_1]\!] \\
t^k_{\text{DC},+}\,[\![\phi_1 \wedge \phi_2]\!] &= t^k_{\text{DC},+}\,[\![\phi_1]\!] \wedge t^k_{\text{DC},+}\,[\![\phi_2]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ((\neg[\phi_1 \wedge \phi_2]_{i,j} \vee [\phi_1]_{i,j}) \wedge (\neg[\phi_1 \wedge \phi_2]_{i,j} \vee [\phi_2]_{i,j})) \\
t^k_{\text{DC},-}\,[\![\phi_1 \wedge \phi_2]\!] &= t^k_{\text{DC},-}\,[\![\phi_1]\!] \wedge t^k_{\text{DC},-}\,[\![\phi_2]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} \vee \neg[\phi_1]_{i,j} \vee \neg[\phi_2]_{i,j})
\end{aligned}
$$

$$
\begin{aligned}
t^k_{\mathrm{DC},+}[\![\phi_1 \vee \phi_2]\!] &= t^k_{\mathrm{DC},+}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},+}[\![\phi_2]\!] \wedge \\
&\quad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} (\neg[\phi_1 \vee \phi_2]_{i,j} \vee [\phi_1]_{i,j} \vee [\phi_2]_{i,j}) \\
t^k_{\mathrm{DC},-}[\![\phi_1 \vee \phi_2]\!] &= t^k_{\mathrm{DC},-}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},-}[\![\phi_2]\!] \wedge \\
&\quad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} (([\phi_1 \vee \phi_2]_{i,j} \vee \neg[\phi_1]_{i,j}) \wedge ([\phi_1 \vee \phi_2]_{i,j} \vee \neg[\phi_2]_{i,j})) \\
t^k_{\mathrm{DC},+}[\![\phi_1 \frown \phi_2]\!] &= t^k_{\mathrm{DC},+}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},+}[\![\phi_2]\!] \wedge \\
&\quad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} \left(\left(\neg[\phi_1 \frown \phi_2]_{i,j} \vee \bigvee_{m=i}^{j} [\phi_1 \frown \phi_2]_{i,m,j} \right) \wedge \right. \\
&\quad \left. \bigwedge_{m=i}^{j} ((\neg[\phi_1 \frown \phi_2]_{i,m,j} \vee [\phi_1]_{i,m}) \wedge (\neg[\phi_1 \frown \phi_2]_{i,m,j} \vee [\phi_2]_{m,j})) \right) \\
t^k_{\mathrm{DC},-}[\![\phi_1 \frown \phi_2]\!] &= t^k_{\mathrm{DC},-}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},-}[\![\phi_2]\!] \wedge \\
&\quad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}\\ m\in\{i,\ldots,j\}}} ([\phi_1 \frown \phi_2]_{i,j} \vee \neg[\phi_1]_{i,m} \vee \neg[\phi_2]_{m,j})
\end{aligned}
$$

$$
\begin{aligned}
t^k_{\mathrm{SA},p}[\![\texttt{true}]\!] &= \epsilon \\
t^k_{\mathrm{SA},p}[\![\texttt{false}]\!] &= \epsilon \\
t^k_{\mathrm{SA},p}[\![s]\!] &= \epsilon \quad \text{for } s \in \mathit{State} \\
t^k_{\mathrm{SA},+}[\![\neg S_1]\!] &= t^k_{\mathrm{SA},-}[\![S_1]\!] \\
t^k_{\mathrm{SA},-}[\![\neg S_1]\!] &= t^k_{\mathrm{SA},+}[\![S_1]\!] \\
t^k_{\mathrm{SA},+}[\![S_1 \wedge S_2]\!] &= t^k_{\mathrm{SA},+}[\![S_1]\!] \wedge t^k_{\mathrm{SA},+}[\![S_2]\!] \wedge \\
&\quad \bigwedge_{i\in\{0,\ldots,k-1\}} ((\neg[S_1 \wedge S_2]_i \vee [S_1]_i) \wedge (\neg[S_1 \wedge S_2]_i \vee [S_2]_i)) \\
t^k_{\mathrm{SA},-}[\![S_1 \wedge S_2]\!] &= t^k_{\mathrm{SA},-}[\![S_1]\!] \wedge t^k_{\mathrm{SA},-}[\![S_2]\!] \wedge \\
&\quad \bigwedge_{i\in\{0,\ldots,k-1\}} ([S_1 \wedge S_2]_i \vee \neg[S_1]_i \vee \neg[S_2]_i) \\
t^k_{\mathrm{SA},+}[\![S_1 \vee S_2]\!] &= t^k_{\mathrm{SA},+}[\![S_1]\!] \wedge t^k_{\mathrm{SA},+}[\![S_2]\!] \wedge \\
&\quad \bigwedge_{i\in\{0,\ldots,k-1\}} (\neg[S_1 \vee S_2]_i \vee [S_1]_i \vee [S_2]_i) \\
t^k_{\mathrm{SA},-}[\![S_1 \vee S_2]\!] &= t^k_{\mathrm{SA},-}[\![S_1]\!] \wedge t^k_{\mathrm{SA},-}[\![S_2]\!] \wedge \\
&\quad \bigwedge_{i\in\{0,\ldots,k-1\}} (([S_1 \vee S_2]_i \vee [S_1]_i) \wedge ([S_1 \vee S_2]_i \vee [S_2]_i))
\end{aligned}
$$

C.3 Translation to ZOLCS Format

BMC $[\![\phi, k, o]\!]$ gives the translation of DC formula ϕ with model length at most k and `polarityOpt` $= o$.

Note that $[\texttt{true}]_{i,j} \stackrel{\text{def}}{=} [\texttt{true}]$ and $[\texttt{false}]_{i,j} \stackrel{\text{def}}{=} \neg[\texttt{true}]$.

$$
\begin{aligned}
\text{BMC}\,[\![\phi, k, \text{tt}]\!] &= t^k_+\,[\![\phi]\!] \wedge \left(\sum_{i=0}^{k} [\phi]_{0,i} \geq 1\right) \wedge [\texttt{true}] \\
\text{BMC}\,[\![\phi, k, \text{ff}]\!] &= t^k_{DC,+}\,[\![\phi]\!] \wedge t^k_{DC,-}\,[\![\phi]\!] \wedge \left(\sum_{i=1}^{k} [\phi]_{0,i} \geq 1\right) \wedge [\texttt{true}] \\
t^k_{\text{DC},p}\,[\![\texttt{true}]\!] &= \epsilon \\
t^k_{\text{DC},p}\,[\![\texttt{false}]\!] &= \epsilon \\
t^k_{\text{DC},+}\,[\![\textstyle\int S \geq n]\!] &= t^k_{\text{SA},+}\,[\![S]\!] \wedge \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} \left(n \cdot \neg[\textstyle\int S \geq n]_{i,j} + \sum_{m=i}^{j-1} [S]_m \geq n\right) \\
t^k_{\text{DC},-}\,[\![\textstyle\int S \geq n]\!] &= t^k_{\text{SA},-}\,[\![S]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k-n\}\\ j\in\{i+n,\ldots,k\}}} \Big((j-i+1-n) \cdot [\textstyle\int S \geq n]_{i,j} + \\
&\qquad\qquad \sum_{m=i}^{j-1} \neg[S]_m \geq (j-i+1-n)\Big) \\
t^k_{\text{DC},+}\,[\![\neg\phi_1]\!] &= t^k_{\text{DC},-}\,[\![\phi_1]\!] \\
t^k_{\text{DC},-}\,[\![\neg\phi_1]\!] &= t^k_{\text{DC},+}\,[\![\phi_1]\!] \\
t^k_{\text{DC},+}\,[\![\phi_1 \wedge \phi_2]\!] &= t^k_{\text{DC},+}\,[\![\phi_1]\!] \wedge t^k_{\text{DC},+}\,[\![\phi_2]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} (2 \cdot \neg[\phi_1 \wedge \phi_2]_{i,j} + [\phi_1]_{i,j} + [\phi_2]_{i,j} \geq 2) \\
t^k_{\text{DC},-}\,[\![\phi_1 \wedge \phi_2]\!] &= t^k_{\text{DC},-}\,[\![\phi_1]\!] \wedge t^k_{\text{DC},-}\,[\![\phi_2]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} ([\phi_1 \wedge \phi_2]_{i,j} + \neg[\phi_1]_{i,j} + \neg[\phi_2]_{i,j} \geq 1) \\
t^k_{\text{DC},+}\,[\![\phi_1 \vee \phi_2]\!] &= t^k_{\text{DC},+}\,[\![\phi_1]\!] \wedge t^k_{\text{DC},+}\,[\![\phi_2]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} (\neg[\phi_1 \vee \phi_2]_{i,j} + [\phi_1]_{i,j} + [\phi_2]_{i,j} \geq 1) \\
t^k_{\text{DC},-}\,[\![\phi_1 \vee \phi_2]\!] &= t^k_{\text{DC},-}\,[\![\phi_1]\!] \wedge t^k_{\text{DC},-}\,[\![\phi_2]\!] \wedge \\
&\qquad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} (2 \cdot [\phi_1 \vee \phi_2]_{i,j} + \neg[\phi_1]_{i,j} + \neg[\phi_2]_{i,j} \geq 2)
\end{aligned}
$$

$$
\begin{aligned}
t^k_{\mathrm{DC},+}[\![\phi_1 \frown \phi_2]\!] &= t^k_{\mathrm{DC},+}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},+}[\![\phi_2]\!] \wedge \\
&\quad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}}} \left(\left(\neg[\phi_1 \frown \phi_2]_{i,j} + \sum_{m=i}^{j} [\phi_1 \frown \phi_2]_{i,m,j} \geq 1 \right) \wedge \right. \\
&\quad \left. \bigwedge_{m=i}^{j} (2 \cdot \neg[\phi_1 \frown \phi_2]_{i,m,j} + [\phi_1]_{i,m} + [\phi_2]_{m,j} \geq 2) \right) \\
t^k_{\mathrm{DC},-}[\![\phi_1 \frown \phi_2]\!] &= t^k_{\mathrm{DC},-}[\![\phi_1]\!] \wedge t^k_{\mathrm{DC},-}[\![\phi_2]\!] \wedge \\
&\quad \bigwedge_{\substack{i\in\{0,\ldots k\}\\ j\in\{i,\ldots,k\}\\ m\in\{i,\ldots,j\}}} ([\phi_1 \frown \phi_2]_{i,j} + \neg[\phi_1]_{i,m} + \neg[\phi_2]_{m,j} \geq 1)
\end{aligned}
$$

$$
\begin{aligned}
t^k_{\mathrm{SA},p}[\![\mathtt{true}]\!] &= \epsilon \\
t^k_{\mathrm{SA},p}[\![\mathtt{false}]\!] &= \epsilon \\
t^k_{\mathrm{SA},p}[\![s]\!] &= \epsilon \quad \text{for } s \in \mathit{State} \\
t^k_{\mathrm{SA},+}[\![\neg S_1]\!] &= t^k_{\mathrm{SA},-}[\![S_1]\!] \\
t^k_{\mathrm{SA},-}[\![\neg S_1]\!] &= t^k_{\mathrm{SA},+}[\![S_1]\!] \\
t^k_{\mathrm{SA},+}[\![S_1 \wedge S_2]\!] &= t^k_{\mathrm{SA},+}[\![S_1]\!] \wedge t^k_{\mathrm{SA},+}[\![S_2]\!] \wedge \\
&\quad \bigwedge_{i\in\{0,\ldots,k-1\}} (2 \cdot \neg[S_1 \wedge S_2]_i + [S_1]_i + [S_2]_i \geq 2) \\
t^k_{\mathrm{SA},-}[\![S_1 \wedge S_2]\!] &= t^k_{\mathrm{SA},-}[\![S_1]\!] \wedge t^k_{\mathrm{SA},-}[\![S_2]\!] \wedge \\
&\quad \bigwedge_{i\in\{0,\ldots,k-1\}} ([S_1 \wedge S_2]_i + \neg[S_1]_i + \neg[S_2]_i \geq 1) \\
t^k_{\mathrm{SA},+}[\![S_1 \vee S_2]\!] &= t^k_{\mathrm{SA},+}[\![S_1]\!] \wedge t^k_{\mathrm{SA},+}[\![S_2]\!] \wedge \\
&\quad \bigwedge_{i\in\{0,\ldots,k-1\}} (\neg[S_1 \vee S_2]_i + [S_1]_i + [S_2]_i \geq 1) \\
t^k_{\mathrm{SA},+}[\![S_1 \vee S_2]\!] &= t^k_{\mathrm{SA},+}[\![S_1]\!] \wedge t^k_{\mathrm{SA},+}[\![S_2]\!] \wedge \\
&\quad \bigwedge_{i\in\{0,\ldots,k-1\}} (2 \cdot [S_1 \vee S_2]_i + [S_1]_i + [S_2]_i \geq 2)
\end{aligned}
$$

Appendix D

User's Guide

D.1 Introduction

The *BMC/DCValidator* is a tool that validates duration calculus (DC) formulas by translating them into propositional satisfiability (SAT) problems and then feeding the SAT problems to an external solver.

The implementation builds on a prototype described in [MF02]. It interprets DC over a discrete-time domain and performs bounded model construction to achieve the desired worst-case complexity of NP.

In the following, the basics of the BMC/DCValidator shall be described. First, an introduction to the internal structure of the validation process is given, and then we introduce a number of settings that may be used to customise this process. Then, we shall present the input and output formats of the BMC/DC-Validator and show how to the tool is run.

D.2 Validation of a DC Formula

Internally in the BMC/DCValidator, the validation of a DC formula ϕ using bounded model construction takes place in a number of steps:

1. First, we negate ϕ, since we know that ϕ is valid if and only if $\neg\phi$ is unsatisfiable.
2. Optionally, user-selectable simplifications are attempted on $\neg\phi$.
3. Then, $\neg\phi$ is translated to a SAT problem that is satisfiable if and only if $\neg\phi$ is. Options are available that enable you to choose the type of SAT problem to be constructed and control certain aspects of the translation process.
4. The SAT problem is fed to a solver, and the solver determines whether or not the problem is satisfiable.
5. If the SAT problem is unsatisfiable, the BMC/DCValidator will inform you that ϕ is valid. Otherwise, ϕ is invalid and you may optionally have a counter example visualised.

D.3 Settings

The validation process of a DC formula ϕ can be controlled by a number of settings that will be described in the following.

D.3.1 Bound on Model Length

When performing bounded model construction of $\neg\phi$, we limit ourselves to checking models up to a certain length `k`. So when the BMC/DCValidator states that the formula ϕ is valid, it means that $\neg\phi$ has no models of length `k` or less.

▶ Default is `k = 1`.

D.3.2 Finding the Bound on Model Length

A special class of DC formulas, namely those that have no occurrences of chop below negation, have models of elementary length in the formula size. If you enable the `findk` option, the BMC/DCValidator will check that the negation of the formula you wish to validate has this property, determine the model length and use that value for `k` (see Section D.3.1). In this case, bounded validity implies general validity, i.e. if $\neg\phi$ has no models of length at most k, it has no models at all.

When the `findk` option is enabled, it also implies the enabling of the `nnf` option (see Section D.3.5 on the next page).

▶ Default is `findk = false`.

D.3.3 Polarity Optimisation

With the option `polarityOpt` enabled, biimplications occurring in the translation process are rewritten into one-sided implications. Hereby, the size of the SAT problem is greatly reduced, and the frontend translation time decreases, too.

> DETAILS: For each DC formula ϕ, a number of auxiliary variables $[\phi]_{i,j}$ are introduced and definitions of the form $[\phi]_{i,j} \Leftrightarrow e$ are appended to the internal representation of the problem.
>
> However, if the ϕ we are translating occurs in a positive context, we can replace the definition $[\phi]_{i,j} \Leftrightarrow e$ with $[\phi]_{i,j} \Rightarrow e$.
>
> Similarly, we can replace $[\phi]_{i,j} \Leftrightarrow e$ with $[\phi]_{i,j} \Leftarrow e$ when ϕ occurs in a negative context.

▶ Default is `polarityOpt = true`.

D.3.4 DC Simplification Level

In the following, we shall take the notation

$$\frac{b}{\phi \rightarrow \psi}$$

to mean that formula ϕ may be rewritten to ψ if premise b holds. When b is trivially true, we will simply write

$$\phi \rightarrow \psi$$

The `dcSimpLevel` setting indicates how much effort should be put into optimising the DC formula prior to translation.

- A level of 0 will skip simplifications entirely.
- A level of 1 will perform simple rewring such as

$$\begin{array}{rcl} \phi \vee \top & \rightarrow & \top \quad \text{and} \\ \neg\neg\phi & \rightarrow & \phi \end{array}$$

- A level of 2 will additionally perform complex rewriting such as

$$\frac{S_1 \Rightarrow S_1 \wedge n \geq m}{\int S_1 \geq n \wedge \int S_2 \geq m \rightarrow \int S_1 \geq n}$$

  (which employs commutativity and associativity to be able to rewrite even if the case looks like $\int S_1 \geq n \wedge \phi_1 \wedge \ldots \wedge \phi_i \wedge \int S_2 \geq m$)

  Note that the level 2 simplifications are comparatively more expensive.

▶ Default is `dcSimpLevel = 0`, unless `fRecognition = semantic` (see Section D.3.6) in which case `dcSimpLevel = 2`. This cannot be overridden.

D.3.5 NNF

When the `nnf` option is enabled, the DC formula will be rewritten to NNF before it is (optionally) simplified and translated.

▶ Default is `nnf = false`, unless `findk = true` in which case `nnf = true`. This cannot be overridden.

D.3.6 Formula Recognition

If the same DC subformula occurs more than once in the formula to be validated, it is advantagerous to recognise this. The BMC/DCValidator can then avoid re-generation of constraints and reuse variables in the SAT problem.

One of three types of recognition may be selected with the `fRecognition` option:

- **id:** Only DC subformulas that have been internally constructed and duplicated are recognised as the same.

- **syntactic:** DC subformulas that are syntactically identical are recognised.
- **semantic:** Some semantically identical DC subformulas are recognised as well.

  Note that setting the formula recognition type to `semantic` also forces the DC simplification level to 2 (see Section D.3.4 on the page before).

▶ Default is `fRecognition = syntactic`.

D.3.7 Output Format

The BMC/DCValidator offers a choice between two output format encodings:

- **DIMACS** is an encoding of a propositional logic formula in Conjunctive Normal Form (CNF).
- **ZOLCS** stands for Zero-One Linear Constraint System. It offers a more compact encoding than DIMACS. The BMC/DCTranslator's ability to handle large formulas (particularly containing constructs of type $\int S \geq n$ with $n > 1$) is improved when using this format.

The output format will determine the extension (i.e. ".zolcs" or ".dimacs") of the file containing the SAT problem.

It is the `outputFormat` setting that sets the output format to either `ZOLCS` or `DIMACS`.

▶ Default is `outputFormat = zolcs`.

D.3.8 Output Folder

The `outputFolder` setting specifies the folder in which to place the output file(s) corresponding to the generated SAT problem(s). The path is given relatively to the folder that the BMC/DCValidator was started from and may not contain spaces.

▶ Default is the folder that the BMC/DCValidator was started from.

D.4 Input Format

D.4.1 Introduction

The BMC/DCValidator expects an ASCII file as its input. This file contains a straightforward textual representation of the DC formula(s) to be verified, along with a selection of the settings as explained in Section D.3 on page 108.

Macros may be defined to abbreviate complex formulas. Before parsing the input files, a preprocessor will inline the macro definitions.

First, we shall give an example of an input file, and then we will give a detailed explanation of the abstract syntax of the input format. For a formal specification of the input format, please refer to the BNF in Appendix E on page 115.

D.4.2 Example

The following is an input file for checking the validity of

$$\Box\left(l \leq 30 \Rightarrow \int(gas \wedge \neg flame) \leq 5\right)$$

by performing bounded model checking with $k = 31$.

```
:- set k = 31.
:- set outputFormat = zolcs.
:- state gas.
:- state flame.
:- leak =^= (gas /\ ~flame).
:- m =^= 30.
:- gbsafe(n) =^= all(l <= m -> dur leak <= n).
:- goal gbsafeout gbsafe(5).
```

The choice of output format is described in Section D.3.7 on the preceding page. The `goal` is the formula to be verified, preceded by the name (without extension) of the output file for the SAT problem.

In this example, we have defined a macro `gbsafe` that would make it easy to check the validity of additional formulas by adding goals with different values for `n`.

D.4.3 Writing Lines of Input

The input file consists of a number of lines. Each line must start with a ':-' token and end with a '.' (dot) token. The following example shows a line containing start and end tokens and some line content.

```
:- some line content.
```

The input format allows line breaks, spaces and tabs within a line and the following input is thus semantically equivalent to the previous example.

```
:- some     line
 content.
```

D.4.4 Declaration of States

All states that will be referred to in the state assertions of the duration formulas must be declared before they are referenced.

As an example, we use the `state` keyword to declare the state *flame*:

```
:- state flame.
```

D.4.5 Declaration of Goals

A goal represents a DC formula to be verified. The first word after the `goal` keyword is the name (without extension) of the file to which the SAT problem will be written. Multiple goals are allowed in the same file. One SAT problem will be generated for each goal, and written to separate files as specified.

Example:

```
:- goal firstGoal dur flame >= 2.
```

This input file makes the BMC/DCValidator check validity of the DC formula $\int flame \geq 2$. The SAT problem corresponding to $\neg \int flame \geq 2$ is written to file `firstGoal.zolcs` (given that the selected output format is ZOLCS, see Section D.3.7 on page 110).

D.4.6 Translation Settings

In order to change the behaviour of the translation algorithm, one case use a number of settings. Setting values may be of type boolean, keyword, quoted string, and integer. See Appendix E on page 115 for further details on possible settings and types.

The following example shows a line that disables polarity optimisation:

```
:- set polarityOpt = false.
:- goal goalWithoutPolOpt dur x /\ y /\ z >= 2.
```

When a setting has been set, its value is preserved throughout the following lines until it is set again. In the above case, polarity optimisation is enabled in the translation process of goal `goalWithPolOpt` and any goal that may follow.

If, instead, one wants to perform a translation process both with and without polarity optimisation on a formula, one could write the following input lines to the BMC/DCValidator:

```
:- set polarityOpt = true.
:- goal goalWithPolOpt dur x /\ y /\ z >= 2.
:- set polarityOpt = false.
:- goal goalWithoutPolOpt dur x /\ y /\ z >= 2.
```

Actually, one can even leave out the first line since polarity optimisation is enabled by default. Find more information about settings and their default values in Section D.3 on page 108.

Example: Setting the Model Length Bound

In most cases, one wants to specify a bound on the number of steps used in the translation process. The `k` value is a setting just like any other setting and can therefore be set one or more times as needed.

Example:

```
:- set k = 3.
:- goal goal3 dur x /\ y /\ z >= 2.
:- set k = 5.
:- goal goal5 dur x /\ y /\ z >= 2.
```

D.4.7 Shell Commands

Between the validation of goals, you may wish to have a shell command executed, e.g. to update your `PATH` system variable to point to the SAT solver. This can be achieved using the `shell` keyword, as illustrated in the following example:

```
:- shell("export PATH=$PATH:~/hysat/hysat-1.7/bin").
```

D.4.8 Taking Advantage of Pre-Processing

The pre-processing macro expansion makes it possible to avoid repeated use of the same text in an input file.

Simple Macros

The pre-processor works in such a way that it looks for definitions (characterised by '=^=') and for references to those definitions in the input file. Each time a definition is referenced in a piece of text, that reference is substituted with the original definition. Redefinition is not allowed, and neither are recursive definitions.

The main idea is shown in the following example that could have been part of an input file for the BMC/DCValidator:

```
:- a =^= x /\ y
:- goal goalWithPolOpt dur a /\ z >= 2.
:- set polarityOpt = false.
:- goal goalWithoutPolOpt dur a /\ u >= 2.
```

The point is that the macro pre-processing substitutes all occurences of `a` with $x/\backslash y$ and leaves out the definition of a.

```
:- goal goalWithPolOpt dur x /\ y /\ z >= 2.
:- set polarityOpt = false.
:- goal goalWithoutPolOpt dur x /\ y /\ u >= 2.
```

Parameterised Macros

It is also possible to create parameterised references with zero or more parameters. Again, one needs to be aware that redefining or overloading of a reference is not allowed.

```
:- a(n) =^= dur x /\ y /\ z >= n.
:- b() =^= a(1).
:- goal goalWithPolOpt2 a(2).
:- goal goalWithPolOpt4 a(4).
:- set polarityOpt = false.
:- goal goalWithoutOptPol2 a(2).
:- goal goalWithoutOptPol4 a(4).
:- goal goalWithoutOptPol1 b().
:- goal goalWithoutOptPol1Again b.
```

After pre-processing, the input looks like this:

```
:- goal goalWithPolOpt2 dur x /\ y /\ z >= 2.
:- goal goalWithPolOpt4 dur x /\ y /\ z >= 4.
:- set polarityOpt = false.
:- goal goalWithoutOptPol2 dur x /\ y /\ z >= 2.
:- goal goalWithoutOptPol4 dur x /\ y /\ z >= 4.
:- goal goalWithoutOptPol1 dur x /\ y /\ z >= 1.
:- goal goalWithoutOptPol1Again dur x /\ y /\ z >= 1.
```

D.5 Running the BMC/DCValidator

To run the BMC/DCValidator tool, you must have a Java Runtime Environment[1] installed on your computer. The BMC/DCValidator is packaged in a `bmc.jar` file and can be run with the command:

```
java -jar bmc.jar options inputfile
```

The validity of the formulas that have been defined as goals in your input file will printed on the screen.
The options are as follows:

-solverCmd *cmd*
: The command to run your SAT solver. The BMC/DCValidator currently supports only HySat, but you must still supply a path, e.g. `bin/hysat.exe`.

-shellCmd *cmd*
: The command necessary to spawn a new process that executes a given program on your operating system. If you do not specify this command, the BMC/DCValidator will try to determine it based on the name of your operating system:

OS Name	*Shell Command*
Linux	`/bin/sh -c`
SunOS	`/bin/sh -c`
Windows 95	`command.com /C`
Win...	`cmd.com /C`

The shell command will be used to execute both the solver command and any commands specified in your input file using "`shell(...)`" (see Section D.4.7 on page 112).

Typically, you will need to specify the `shellCmd` if you are running a Windows operating system and your solver needs to run in the Cygwin environment. In this case, you should specify the `shellCmd` to "`bash -c`", given that Cygwin's `bash.exe` is in your system `PATH`.

-displayTrace
: If you enable the `displayTrace` option, a trace corresponding to a counter example for each of your formulas to be validated will be displayed (in separate windows) if the formula turns out to be invalid.

[1] The BMC/DCValidator has been tested with a JRE version 1.4.1 only.

Appendix E

BNF Specification of Input Format

For those acquianted with BNF[1], we have supplied a grammar that describes what the input format of the file must look like *after* the macro expansion.

The macros are not included, since they may substitute *any* part of the input text, except the '.' symbol.

```
<input> ::= <line>*;
<line> ::= <line_symbol> {
<set_command> | <shell_command> | <state_definition> | <goal_statement> } <dot>;

%% goal statements
<goal_statement> ::= 'goal' <goal_name> <dc_formula>
<goal_name> ::= <id_string>

%% set commands
<set_command> ::= 'set' { | <bool_option> | <string_option> | <int_option> }
<bool_option> ::= { 'findk' | 'nnf' | 'polarityOpt' } <eq_sign> <boolean>
<string_option> ::= { <output_folder_option> | <output_type_option> |
      <f_recognition_option> }
<output_folder_option> ::=  'outputFolder' <eq_sign> <quoted_string>
<output_type_option> ::= 'outputType' <eq_sign> { 'zolcs' | 'dimacs' }
<f_recognition_option> ::= 'fRecognition' <eq_sign> {'id' | 'syntactic' |
            'semantic'}
<int_option> ::= { <k_option> | <dc_simp_level_option> }
<k_option> ::= 'k' <eq_sign> <pos_integer>
<dc_simp_level_option> ::= 'dcSimpLevel' <eq_sign> { '0' | '1' | '2' }

%% shell commands
<shell_command> ::= 'shell' <l_parenthesis> <quoted_string> <r_parenthesis>

%% state_definitions
<state_definition> ::= 'state' <state_name>
```

[1] Backus Naur Form

```
<state_name> ::= <id_string>

%% formula
<dc_formula> ::= { <state_name> | <binary_dc> | <unary_dc> | <boolean_val> |
<l_duration> | <duration> | <l_parenthesis> <dc_formula> <r_parenthesis>}
<boolean_val> ::= { 'true' | 'false' }
<binary_operator> ::= { <op_and> | <op_or> | <op_implication> | <op_biimplication> | <op_chop
<unary_sa_operator> ::= <op_negation>
<unary_dc_operator> :: { <unary_sa_operator> | <op_evt> | <op_all> }
<unary_dc> ::= <unary_dc_operator> <dc_formula>
<binary_dc> ::= <dc_formula> <binary_operator> <dc_formula>
<duration> ::= <op_duration> { <sa_formula> | <id_string> } <int_comp_sign>
<integer>
<l_duration> ::= 'l' <int_comp_sign> <integer>

<sa_formula> ::= { <state_name> | <binary_sa> | <unary_sa> | <boolean_val> |
<l_parenthesis> } <sa_formula> <r_parenthesis>}
<unary_sa> ::= <unary_sa_operator> <sa_formula>
<binary_sa_connective> ::= <sa_formula> <binary_operator> <sa_formula>

%% simple types/values
<pos_integer> ::= <DIGIT>+
<integer> ::= {'-' | } <pos_integer>
<string> ::= { <LETTER> | <DIGIT> }+
<id_string> ::= <LETTER> {<string> | }
<quoted_string> ::= '"' {<string> | } '"'
<l_parenthesis> ::= '('
<r_parenthesis> ::= ')'
<eq_sign> ::= '='
<lte_sign> ::= '<='
<leq_sign> ::= '<'
<gte_sign> ::= '>='
<geq_sign> ::= '>'
<int_comp_sign> ::= { eq_sign | lte_sign | leq_sign | gte_sign | geq_sign}
<op_negation> ::= '~'
<op_duration> ::= 'dur'
<op_and> ::= '/\'
<op_or> ::= '\/'
<op_chop> ::= ';'
<op_implication> ::= '->'
<op_biimplication> ::= '<->'
<op_evt> ::= 'evt'
<op_all> ::= 'all'
<line_symbol> ::= ':-'
<dot> ::= '.'
```

Appendix F

Specifications for Lexer/Parser Generators

In Appendix F.1 we list the specification for the lexer generator JLex, see [JLex]. Correspondingly, Appendix F.2 contains the specification for the parser generator JCup, see [JCup].

F.1 Lexer Specification

```
1  package bmc.parser;
2
3  import java_cup.runtime.Symbol;
4  import java.text.ParseException;
5
6  %%
7  %public
8  %cup
9  %line
10 %class BMCLex
11 %eofval{
12   return new Symbol(BMCSymbols.EOF);
13 %eofval}
14 %yylexthrow{
15 ParseException
16 %yylexthrow}
17 %{
18   public int getLineNo() {
19     return yyline+1;
20   }
21
22   public String getTokenStr() {
23     return yytext();
24   }
25 %}
```

```
%%
"." { return new Symbol(BMCSymbols.DOT); }
";" { return new Symbol(BMCSymbols.CHOP); }
[/][\\] { return new Symbol(BMCSymbols.AND); }
[\\][/] { return new Symbol(BMCSymbols.OR); }
"~" { return new Symbol(BMCSymbols.NEG); }
"->" { return new Symbol(BMCSymbols.IMPL); }
"<->" { return new Symbol(BMCSymbols.BIIMPL); }
">=" { return new Symbol(BMCSymbols.GEQ); }
"<=" { return new Symbol(BMCSymbols.LEQ); }
">" { return new Symbol(BMCSymbols.GT); }
"<" { return new Symbol(BMCSymbols.LT); }
"=" { return new Symbol(BMCSymbols.EQ); }
"(" { return new Symbol(BMCSymbols.LPAREN); }
")" { return new Symbol(BMCSymbols.RPAREN); }
":-" { return new Symbol(BMCSymbols.LINE); }
"=^=" { return new Symbol(BMCSymbols.DEF); }
"," { return new Symbol(BMCSymbols.COMMA); }
"true" { return new Symbol(BMCSymbols.TRUE); }
"false" { return new Symbol(BMCSymbols.FALSE); }
"dur" { return new Symbol(BMCSymbols.DUR); }
"all" { return new Symbol(BMCSymbols.ALL); }
"evt" { return new Symbol(BMCSymbols.EVT); }
"l" { return new Symbol(BMCSymbols.L); }
"state" { return new Symbol(BMCSymbols.STATE); }
"set" { return new Symbol(BMCSymbols.SET); }
"findk" { return new Symbol(BMCSymbols.FINDK); }
"nnf" { return new Symbol(BMCSymbols.NNF); }
"k" { return new Symbol(BMCSymbols.K); }
"goal" { return new Symbol(BMCSymbols.GOAL) ;}
"shell" { return new Symbol(BMCSymbols.SHELL) ;}
"polarityOpt" { return new Symbol(BMCSymbols.POLOPT); }
"dcSimpLevel" { return new Symbol(BMCSymbols.DCSIMPLEVEL)
    ; }
"outputFormat" { return new Symbol(BMCSymbols.
    OUTPUTFORMAT); }
"outputFolder" { return new Symbol(BMCSymbols.
    OUTPUTFOLDER); }
"fRecognition" { return new Symbol(BMCSymbols.FREC); }
[-]?[0-9]+ { return new Symbol(BMCSymbols.NUMBER, new
    Integer(yytext())); }
[ \t\r\n\f] { /* ignore white space. */ }
[a-zA-Z][0-9a-zA-Z]* { return new Symbol(BMCSymbols.ID,
    yytext()); }
\"[^\"]*\" {
        String str = yytext().substring(1,yytext().
            length() - 1);
        return new Symbol(BMCSymbols.ESCAPEDSTRING, str);
}
```

```
69 . { throw new ParseException("Illegal character: "+yytext
     (), getLineNo()); }
```

F.2 Parser Specification

```
1  package bmc.parser;
2
3  import java_cup.runtime.*;
4  import java.util.*;
5  import java.io.Reader;
6  import java.text.ParseException;
7  import bmc.formula.*;
8  import bmc.util.*;
9  import bmc.*;
10
11 parser code {:
12   private List tasks;
13   private Settings settings;
14   private Map states;
15
16   private String errorMsg;
17   private BMCLex lex;
18
19   void addState(String stateName) {
20     states.put(stateName, new State(stateName));
21   }
22
23   State getState(String stateName) throws ParseException,
         CloneNotSupportedException {
24     if (!states.containsKey(stateName)) {
25       throw new ParseException("Undeclared state "+
             stateName, ((BMCLex) getScanner()).getLineNo());
26     }
27     return (State) ((State) states.get(stateName)).clone
         ();
28   }
29
30   void addCommand(String commandText) {
31     Command command = new Command(commandText);
32     tasks.add(command);
33   }
34
35   void addGoal(String goalName, Formula formula)
36   throws ParseException, CloneNotSupportedException {
37     Goal goal = new Goal(goalName, formula, (Settings)
         settings.clone(), new HashSet(states.values()) );
38     tasks.add(goal);
39   }
40
41   public Settings getSettings() {
```

```
42      return settings;
43    }
44
45    public List parseTasks() throws Exception {
46      parse();
47      if (errorMsg != null) {
48        throw new ParseException(errorMsg,0);
49      }
50      return tasks;
51    }
52
53    public void setScanner(Scanner s) {
54      super.setScanner(s);
55      lex = (BMCLex) s;
56    }
57
58    public void report_error(String message, Object info) {
59      errorMsg = "Error: Encountered \""+lex.getTokenStr()+
60      "\" unexpectedly at line "+lex.getLineNo()+"\n";
61    }
62
63    public void report_fatal_error(String message, Object
        info)
64    throws Exception {
65      report_error(message, info);
66      done_parsing();
67      throw new ParseException(errorMsg, 0);
68    }
69
70
71  :}
72
73  init with {:
74    if (tasks == null) {
75      tasks = new LinkedList();
76    }
77    else {
78      tasks.clear();
79    }
80    if (settings == null) {
81      settings = new Settings();
82    }
83    else {
84      settings.setStdValues();
85    }
86    if (states == null) {
87      states = new HashMap();
88    }
89    else {
90      states.clear();
```

```
91      }
92  :}
93
94  terminal DOT, CHOP, AND, OR, NEG, IMPL, BIIMPL;
95  terminal LEQ, GT, LT, EQ, GEQ, LPAREN, RPAREN;
96  terminal TRUE, FALSE;
97  terminal DUR, STATE, GOAL, LINE;
98  terminal ALL, EVT, L;
99  terminal K, OUTPUTFORMAT, OUTPUTFOLDER, POLOPT,
        DCSIMPLEVEL;
100 terminal SET, SHELL, FINDK, NNF;
101 terminal Integer NUMBER;
102 terminal String ID, ESCAPEDSTRING;
103 terminal COMMA, DEF;
104 terminal FREC;
105
106 non terminal Formula saFormula;
107 non terminal Formula dcFormula;
108 non terminal formulaList, formula;
109 non terminal Boolean bool;
110
111 precedence nonassoc BIIMPL;
112 precedence right IMPL;
113 precedence left OR;
114 precedence left AND;
115 precedence left CHOP;
116 precedence nonassoc ALL, EVT, NEG;
117 precedence nonassoc DUR;
118
119 formulaList ::= formulaList formula | formula;
120 formula     ::= LINE STATE ID:id DOT
121       {: parser.addState(id); :}
122       | LINE SHELL LPAREN ESCAPEDSTRING:comm RPAREN DOT
123       {: parser.addCommand(comm); :}
124       | LINE GOAL ID:name dcFormula:f DOT
125       {: parser.addGoal(name, f); :}
126       | LINE SET K EQ NUMBER:n DOT
127       {: parser.getSettings().setK(n.intValue());  :}
128       | LINE SET FINDK EQ bool:b DOT
129       {: parser.getSettings().setFindK(b.booleanValue());
              :}
130       | LINE SET NNF EQ bool:b DOT
131       {: parser.getSettings().setNNF(b.booleanValue());
              :}
132       | LINE SET POLOPT EQ bool:b DOT
133       {: parser.getSettings().setPolarityOpt(b.
              booleanValue());  :}
134       | LINE SET DCSIMPLEVEL EQ NUMBER:n DOT
135       {: parser.getSettings().setDCSimpLevel(n.intValue()
              );  :}
```

```
136         |  LINE SET OUTPUTFORMAT EQ ID:id DOT
137         {:  if (id.equalsIgnoreCase("zolcs")) {
138               parser.getSettings().setOutputFormat(Settings
                      .OUTPUT_FORMAT_ZOLCS);
139             }
140             else if (id.equalsIgnoreCase("dimacs")) {
141               parser.getSettings().setOutputFormat(Settings
                      .OUTPUT_FORMAT_DIMACS);
142             }
143             else {
144               throw new ParseException(
145               "Unknown output type="+id,
146               ((BMCLex) parser.getScanner()).getLineNo());
147             }
148         :}
149         |  LINE SET FREC EQ ID:id DOT
150         {:  if (id.equalsIgnoreCase("id")) {
151               parser.getSettings().setFRecognition(Settings
                      .F_RECOGNITION_ID);
152             }
153             else if (id.equalsIgnoreCase("syntactic")) {
154               parser.getSettings().setFRecognition(Settings
                      .F_RECOGNITION_SYNTACTIC);
155             }
156             else if (id.equalsIgnoreCase("semantic")) {
157               parser.getSettings().setFRecognition(Settings
                      .F_RECOGNITION_SEMANTIC);
158             }
159             else {
160               throw new ParseException(
161               "Unknown literal handler="+id,
162               ((BMCLex) parser.getScanner()).getLineNo());
163             }
164         :}
165         |  LINE SET OUTPUTFOLDER EQ ESCAPEDSTRING:f DOT
166         {: parser.getSettings().setOutputFolder(f);  :}
167         ;
168
169 bool       ::= TRUE {: RESULT = Boolean.TRUE; :}
170         |  FALSE {: RESULT = Boolean.FALSE; :}
171         ;
172
173 saFormula  ::= ID:id
174         {: RESULT = parser.getState(id); :}
175         | saFormula:l OR saFormula:r
176         {: RESULT = new Disjunction(l,r); :}
177         | saFormula:l AND saFormula:r
178         {: RESULT = new Conjunction(l,r); :}
179         | saFormula:l IMPL saFormula:r
180         {: RESULT = new Disjunction(new Negation(l),r); :}
```

```
181         | saFormula:l BIIMPL saFormula:r
182         {: RESULT = new Conjunction(
183           new Disjunction(
184             new Negation((Formula) l.clone()),
185             (Formula) r.clone()),
186           new Disjunction(l, new Negation(r)));
187         :}
188         | TRUE
189           {: RESULT = new True(); :}
190         | FALSE
191           {: RESULT = new False(); :}
192         | NEG saFormula:f
193           {: RESULT = new Negation(f); :}
194         | LPAREN saFormula:f RPAREN
195           {: RESULT = f; :}
196         ;
197
198 dcFormula  ::= dcFormula:l OR dcFormula:r
199         {: RESULT = new Disjunction(l,r); :}
200         | dcFormula:l AND dcFormula:r
201         {: RESULT = new Conjunction(l,r); :}
202         | dcFormula:l CHOP dcFormula:r
203         {: RESULT = new Chop(l,r); :}
204         | dcFormula:l IMPL dcFormula:r
205         {: RESULT = new Disjunction(new Negation(l),r); :}
206         | dcFormula:l BIIMPL dcFormula:r
207         {: RESULT = new Conjunction(
208           new Disjunction(
209             new Negation((Formula) l.clone()),
210             (Formula) r.clone()),
211           new Disjunction(l,new Negation(r)));
212         :}
213         | ALL dcFormula:f
214         {: RESULT = new Negation(new Chop(new Chop(new True
                ()),
215           new Negation(f)),new True()));
216         :}
217         | EVT dcFormula:f
218         {: RESULT = new Chop(new Chop(new True(), f),new
                True()); :}
219         | DUR saFormula:f GEQ NUMBER:n
220         {:   if (n.intValue() <= 0) {
221               RESULT = new True();
222             }
223             else {
224               RESULT = new Duration(f,n.intValue());
225             }
226         :}
227         | DUR saFormula:f LEQ NUMBER:n
228         {:   if (n.intValue() < 0) {
```

```
229             RESULT = new False();
230           }
231           else {
232             RESULT = new Negation(new Duration(f,n.
                  intValue()+1));
233           }
234         :}
235         | DUR saFormula:f GT NUMBER:n
236         {:  if (n.intValue() < 0) {
237             RESULT = new True();
238           }
239           else {
240             RESULT = new Duration(f,n.intValue()+1);
241           }
242         :}
243         | DUR saFormula:f LT NUMBER:n
244         {:  if (n.intValue() <= 0) {
245             RESULT = new False();
246           }
247           else {
248             RESULT = new Negation(new Duration(f,n.
                  intValue()));
249           }
250         :}
251         | DUR saFormula:f EQ NUMBER:n
252         {:  if (n.intValue() < 0) {
253             RESULT = new False();
254           }
255           else if (n.intValue() == 0) {
256             RESULT = new Negation(new Duration(f,1));
257           }
258           else {
259             Formula tmp1 = new Duration(f,n.intValue());
260             Formula tmp2 = new Negation(new Duration(f,n.
                  intValue()+1));
261             RESULT = new Conjunction(tmp1,tmp2);
262           }
263         :}
264         | TRUE
265           {: RESULT = new True(); :}
266         | FALSE
267           {: RESULT = new False(); :}
268               | NEG dcFormula:f
269           {: RESULT = new Negation(f); :}
270         | L LEQ NUMBER:n
271         {:  if (n.intValue() < 0) {
272             RESULT = new False();
273           }
274           else {
```

```
275             RESULT = new Negation(new Duration(new True()
                    ,n.intValue()+1));
276           }
277         :}
278         | L GEQ NUMBER:n
279         {:  if (n.intValue() <= 0) {
280             RESULT = new True();
281           }
282           else {
283             RESULT = new Duration(new True(),n.intValue()
                    );
284           }
285         :}
286         | L LT NUMBER:n
287         {:  if (n.intValue() <= 0) {
288             RESULT = new False();
289           }
290           else {
291             RESULT = new Negation(new Duration(new True()
                    ,n.intValue()));
292           }
293         :}
294         | L GT NUMBER:n
295         {:  if (n.intValue() < 0) {
296             RESULT = new True();
297           }
298           else {
299             RESULT = new Duration(new True(),n.intValue()
                    +1);
300           }
301         :}
302         | L EQ NUMBER:n
303         {:  if (n.intValue() < 0) {
304             RESULT = new False();
305           }
306           else if (n.intValue() == 0) {
307             RESULT = new Negation(new Duration(new True()
                    , 1));
308           }
309           else {
310             Formula tmp1 = new Duration(new True(),n.
                    intValue());
311             Formula tmp2 = new Negation(new Duration(new
                    True(),n.intValue()+1));
312             RESULT = new Conjunction(tmp1,tmp2);
313           }
314         :}
315         | LPAREN dcFormula:f RPAREN
316           {: RESULT = f; :}
317         ;
```

Appendix G

Test Suites

This appendix contains source code and specifications for tests. First, Appendix G.1 gives the source code of the automated white box unit tests, then, Appendix G.2 on page 216 lists the source code and test cases for the automated black box tests, and finally, Appendix G.3 on page 228 gives a description of the manual black box tests.

G.1 Automated White Box Unit Test Suite

The following source code consists of classes that each perform unit tests on a single Java package. They have been listed below, in alphabetical order by package name. Finally, Section G.1.10 on page 215 contains the source code for the test program that uses all of the unit tests from the individual packages.

G.1.1 bmc.constraint.Test.java

```
1  package bmc.constraint;
2
3  import java.io.*;
4
5  import bmc.util.*;
6  import bmc.literal.*;
7  import bmc.formula.*;
8
9  import junit.framework.*;
10
11 public class Test extends TestCase {
12
13
14     private int boolLitNo;
15     private int currLitNo;
16
17     public Test(String testName) {
18         super(testName);
19     }
```

```
20
21
22      protected void setUp() {
23          currLitNo = VariableNoGenerator.BOOL_LIT_NO + 1;
24      }
25
26      public void testDIMACS() {
27          Constraint cstr1 = new DIMACSConstraint();
28          Constraint cstrTrue = new DIMACSConstraint();
29          Constraint cstrFalse = new DIMACSConstraint();
30          Literal lit1 = getLiteral(true);
31          Literal lit2 = getLiteral(false);
32
33          /*
34           * testing normal constraint behaviour
35           */
36          Assert.assertEquals("No added literals.
                Constraint false by def", "-"+getTrueLit().
                getNumber()+" 0\n", cstr1.toString());
37          Assert.assertFalse("Not trivially true", cstr1.
                isTriviallyTrue());
38
39          cstr1.addTerm(lit1);
40          Assert.assertEquals("One normal lit added", lit1.
                getNumber()+" 0\n", cstr1.toString());
41
42          cstr1.addTerm(lit1);
43          Assert.assertEquals("Same lit added again. No
                difference", lit1.getNumber()+" 0\n", cstr1.
                toString());
44
45          Assert.assertFalse("Constraint isnt trivially
                true", cstr1.isTriviallyTrue());
46
47          cstr1.addTerm(getFalseLit());
48          Assert.assertEquals("Adding false lit doesnt
                change anything", lit1.getNumber()+" 0\n",
                cstr1.toString());
49          Assert.assertFalse("Not trivially true", cstr1.
                isTriviallyTrue());
50
51          cstr1.addTerm(lit1.flipSign());
52          Assert.assertEquals("Opposite sign added", cstr1.
                toString(), lit1.getNumber()+" -"+lit1.
                getNumber()+" 0\n");
53          Assert.assertTrue("Now trivially true", cstr1.
                isTriviallyTrue());
54
55          cstrTrue.addTerm(getTrueLit());
```

```
56          Assert.assertEquals("Constraint containing only
                the true literal", getTrueLit().getNumber()+"
                0\n", cstrTrue.toString());
57          Assert.assertTrue("Trivially true", cstrTrue.
                isTriviallyTrue());
58
59          cstrTrue.addTerm(getFalseLit());
60          Assert.assertEquals("Adding the false lit doesnt
                change anything", getTrueLit().getNumber()+"
                0\n", cstrTrue.toString());
61          Assert.assertTrue("Trivially true doesnt change
                either", cstrTrue.isTriviallyTrue());
62
63          cstrFalse.addTerm(getFalseLit());
64          Assert.assertEquals("Constraint containing only
                the false literal", "-"+getTrueLit().getNumber
                ()+" 0\n", cstrFalse.toString());
65          Assert.assertFalse("Isnt trivially true",
                cstrFalse.isTriviallyTrue());
66      }
67
68      public void testZOLCS() {
69          ZOLCSConstraint cstr1 = new ZOLCSConstraint();
70          ZOLCSConstraint cstrTrue = new ZOLCSConstraint();
71          ZOLCSConstraint cstrFalse = new ZOLCSConstraint()
                ;
72          Literal lit1 = getLiteral(true);
73
74          /*
75           * testing normal constraint behaviour
76           */
77          Assert.assertEquals("No added literals.
                Constraint false by def", "1 -"+getTrueLit().
                getNumber()+" 1 0\n", cstr1.toString());
78          Assert.assertFalse("Not trivially true", cstr1.
                isTriviallyTrue());
79
80          cstr1.addTerm(lit1);
81          Assert.assertEquals("One normal lit added", "1 "+
                lit1.getNumber()+" 1 0\n", cstr1.toString());
82
83          cstr1.addTerm(lit1);
84          Assert.assertEquals("Same lit added again. Weight
                 changed", "2 "+lit1.getNumber()+" 1 0\n",
                cstr1.toString());
85
86          Assert.assertFalse("Constraint isnt trivially
                true", cstr1.isTriviallyTrue());
87
88          cstr1.addTerm(getFalseLit());
```

```
89          Assert.assertEquals("Adding false lit doesnt
                change anything", cstr1.toString(), "2 "+lit1.
                getNumber()+" 1 0\n");
90          Assert.assertFalse("Not trivially true", cstr1.
                isTriviallyTrue());
91
92          cstr1.addTerm(lit1.flipSign());
93          Assert.assertEquals("Opposite sign added.
                Constraint reduced a bit", cstr1.toString(),
                "1 "+lit1.getNumber()+" 0 0\n");
94          Assert.assertTrue("Now trivially true", cstr1.
                isTriviallyTrue());
95
96          cstrTrue.addTerm(getTrueLit());
97          Assert.assertEquals("Trivial constraint
                containing only the false literal", "1 -"+
                getTrueLit().getNumber()+" 0 0\n",  cstrTrue.
                toString());
98          Assert.assertTrue("Trivially true", cstrTrue.
                isTriviallyTrue());
99
100         cstrTrue.addTerm(getFalseLit());
101         Assert.assertEquals("Adding the false lit doesnt
                change anything", "1 -"+getTrueLit().getNumber
                ()+" 0 0\n",  cstrTrue.toString());
102         Assert.assertTrue("Trivially true doesnt change
                either", cstrTrue.isTriviallyTrue());
103
104         cstrFalse.addTerm(getFalseLit());
105         Assert.assertEquals("Constraint containing only
                the false literal", "1 -"+getTrueLit().
                getNumber()+" 1 0\n", cstrFalse.toString());
106         Assert.assertFalse("Isnt trivially true",
                cstrFalse.isTriviallyTrue());
107
108         /*
109          * testing ZOLCS behaviour
110          */
111         cstr1 = new ZOLCSConstraint(10);
112
113         Assert.assertEquals("No added literals.
                Constraint false by def. Threshold 10", "1 -"+
                getTrueLit().getNumber()+" 10 0\n", cstr1.
                toString());
114         Assert.assertFalse("Not trivially true", cstr1.
                isTriviallyTrue());
115
116         cstr1.addTerm(lit1, 5);
117         Assert.assertEquals("One normal lit (weight 5)
                added", "5 "+lit1.getNumber()+" 10 0\n", cstr1
```

```
            .toString());
118
119         cstr1.addTerm(lit1, 3);
120         Assert.assertEquals("Same lit added again. Weight
                changed", "8 "+lit1.getNumber()+" 10 0\n",
            cstr1.toString());
121
122         cstr1.addTerm(lit1, 0);
123         Assert.assertEquals("Same lit added again but
            with weight zero. Nothing changed", "8 "+lit1.
            getNumber()+" 10 0\n", cstr1.toString());
124
125         cstr1.addTerm(lit1.flipSign(), 5);
126         Assert.assertEquals("Same lit added again with
            opposite sign. Weight and threshold reduced",
            "3 "+lit1.getNumber()+" 5 0\n", cstr1.toString
            ());
127
128         cstr1.addTerm(getTrueLit(), 2);
129         Assert.assertEquals("True lit added. Reduces
            threshold only", "3 "+lit1.getNumber()+" 3 0\n
            ", cstr1.toString());
130         Assert.assertFalse("Isnt trivially true just
            because true lit was added",cstr1.
            isTriviallyTrue());
131
132         //
133         Literal trueLiteral = getTrueLit();
134         Literal falseLiteral = getFalseLit();
135
136         /* cstr1 is a ZOLCSConstraint: 3*lit1 >= 3) */
137         cstr1.addTerm(falseLiteral, 2);
138         Assert.assertEquals("False literal added. Makes
            no changes.", "3 "+lit1.getNumber()+" 3 0\n",
            cstr1.toString());
139         cstr1.addTerm(trueLiteral, 3);
140         Assert.assertEquals("True literal added with
            threshold=3. Reduces threshold by 3.", "3 "+
            lit1.getNumber()+" 0 0\n", cstr1.toString());
141
142         Assert.assertTrue("Enough true lits were added.
            Now trivially true",cstr1.isTriviallyTrue());
143
144         cstr1.addTerm(lit1.flipSign(), 6);
145         Assert.assertEquals("Lit added again with
            opposite sign with greater weight", "3 -"+lit1
            .getNumber()+" -3 0\n", cstr1.toString());
146
147         cstr1.addTerm(lit1, 3);
```

```
148         Assert.assertEquals("Lit added again. Now weight
                is zero (and lit is gone)", "1 -"+getTrueLit
                ().getNumber()+" -6 0\n", cstr1.toString());
149         Assert.assertTrue("Now trivially true",cstr1.
                isTriviallyTrue());
150     }
151
152     public void testConstraintWriter() throws IOException
            , CloneNotSupportedException {
153         File outputFile1 = new File("test1.dat");
154         RandomAccessFile output1 = new RandomAccessFile(
                outputFile1, "rw");
155         LiteralHandler litHandler1 = new
                IDLookupLiteralHandler();
156         Formula formula1 = new Conjunction(new State("x1
                "), new True());
157         Settings settings1 = new Settings();
158         settings1.setK(5);
159         settings1.setPolarityOpt(false);
160         settings1.setNNF(true);
161         settings1.setDCSimpLevel(Settings.
                DC_SIMP_LEVEL_WITH_BDD);
162         settings1.setOutputFormat(Settings.
                OUTPUT_FORMAT_DIMACS);
163         settings1.setFRecognition(Settings.
                F_RECOGNITION_SEMANTIC);
164         settings1.setOutputFolder("testFolder");
165         ConstraintWriter cw1 = new ConstraintWriter(
                litHandler1, formula1, settings1, output1);
166
167         Assert.assertEquals("Initiated to same lit no as
                bool_lit_no", VariableNoGenerator.BOOL_LIT_NO,
                 cw1.numClauses);
168         Assert.assertSame("", output1, cw1.output);
169         Assert.assertSame("", litHandler1, cw1.
                literalHandler);
170         Assert.assertSame("", output1, cw1.output);
171
172         ZOLCSConstraint zc1 = new ZOLCSConstraint(4);
173         Literal litzc1 = litHandler1.getLiteral(new False
                (), 1, 1);
174         Literal litzc2 = litHandler1.getLiteral(new
                Conjunction(new Duration(new State("x1"), 5),
                new True()),1,5);
175         zc1.addTerm(litzc1, 1);
176         zc1.addTerm(litzc2, 5);
177         cw1.writeConstraint(zc1);
178
179         DIMACSConstraint dc1 = new DIMACSConstraint();
```

```
180         Literal litdc1 = litHandler1.getLiteral(new False
                ), 1, 1);
181         Literal litdc2 = litHandler1.getLiteral(new
                Conjunction(new Duration(new State("x1"), 5),
                new True()), 1,6);
182         dc1.addTerm(litdc1);
183         dc1.addTerm(litdc2);
184         cw1.writeConstraint(dc1);
185
186         cw1.finish();
187
188         BufferedReader br = new BufferedReader(new
                FileReader(outputFile1));
189         LineNumberReader reader = new LineNumberReader(br
                );
190         String line;
191         Assert.assertEquals("Content is comment", "c k
                =5", reader.readLine());
192         Assert.assertEquals("Content is comment", "c
                formula=(x1 /\\ true)", reader.readLine());
193         Assert.assertEquals("Content is comment", "c
                polarityOpt=false", reader.readLine());
194         Assert.assertEquals("Content is comment", "c
                dcSimpLevel="+ Settings.DC_SIMP_LEVEL_WITH_BDD
                , reader.readLine());
195         Assert.assertEquals("Content is comment", "c nnf=
                true", reader.readLine());
196         Assert.assertEquals("Content is comment", "c
                outputFormat=dimacs", reader.readLine());
197         Assert.assertEquals("Content is comment", "c
                fRecognition=semantic", reader.readLine());
198         Assert.assertEquals("Content is comment", "c
                outputFolder=testFolder", reader.readLine());
199         Assert.assertTrue("Content is declaration",
                reader.readLine().startsWith("p cnf "+Integer.
                toString(VariableNoGenerator.BOOL_LIT_NO+2)+"
                3"));
200         Assert.assertEquals("Content is empty", "",
                reader.readLine());
201         Assert.assertEquals("Content is true lit
                declaration", Integer.toString(
                VariableNoGenerator.BOOL_LIT_NO)+" 0", reader.
                readLine());
202         Assert.assertEquals("Content is zc1", "5 "+litzc2
                .getNumber()+" 4 0", reader.readLine());
203         Assert.assertEquals("Content is dc1", litdc2.
                getNumber()+" 0", reader.readLine());
204
205
206         settings1 = new Settings();
```

```
207         settings1.setOutputFormat(Settings.
                OUTPUT_FORMAT_ZOLCS);
208         settings1.setFRecognition(Settings.
                F_RECOGNITION_ID);
209         settings1.setOutputFolder("");
210         output1 = new RandomAccessFile(outputFile1, "rw")
                ;
211         cw1 = new ConstraintWriter(litHandler1, formula1,
                 settings1, output1);
212         br = new BufferedReader(new FileReader(
                outputFile1));
213         reader = new LineNumberReader(br);
214         reader.readLine();reader.readLine();reader.
                readLine();reader.readLine();reader.readLine()
                ;
215         Assert.assertEquals("Content is comment", "c
                outputFormat=zolcs", reader.readLine());
216         Assert.assertEquals("Content is comment", "c
                fRecognition=id", reader.readLine());
217         Assert.assertEquals("Content is comment", "c
                outputFolder=(current folder)", reader.
                readLine());
218         Assert.assertTrue("Content is declaration",
                reader.readLine().startsWith("p zolcs"));
219         Assert.assertEquals("Content is empty", "",
                reader.readLine());
220         Assert.assertEquals("Content is true lit
                declaration (zolcs)", "1 "+Integer.toString(
                VariableNoGenerator.BOOL_LIT_NO)+" 1 0",
                reader.readLine());
221         cw1.finish();
222
223         settings1 = new Settings();
224         settings1.setOutputFormat(Settings.
                OUTPUT_FORMAT_ZOLCS);
225         settings1.setFRecognition(Settings.
                F_RECOGNITION_SYNTACTIC);
226         settings1.setOutputFolder("");
227         output1 = new RandomAccessFile(outputFile1, "rw")
                ;
228         cw1 = new ConstraintWriter(litHandler1, formula1,
                 settings1, output1);
229         br = new BufferedReader(new FileReader(
                outputFile1));
230         reader = new LineNumberReader(br);
231         reader.readLine();reader.readLine();reader.
                readLine();reader.readLine();reader.readLine()
                ;
232         reader.readLine();
```

```
233         Assert.assertEquals("Content is comment", "c
                fRecognition=syntactic", reader.readLine());
234         cw1.finish();
235     }
236
237
238     private  Literal getLiteral(boolean sign) {
239         return new Literal(currLitNo++, sign);
240     }
241
242     private Literal getTrueLit() {
243         return Literal.getTrueLiteral();
244     }
245
246     private Literal getFalseLit() {
247         return Literal.getFalseLiteral();
248     }
249
250     public static junit.framework.Test suite() {
251         return new TestSuite(Test.class);
252     }
253 }
```

G.1.2 bmc.formula.Test.java

```
1  package bmc.formula;
2
3  import java.util.*;
4  import bmc.robdd.*;
5  import junit.framework.*;
6
7  public class Test extends TestCase {
8
9      public Test(String testName) {
10         super(testName);
11     }
12
13     public void performStandardTest(Formula f1, Formula
           f2, int formulaType) throws Exception {
14         Formula f1Cloned = (Formula) f1.clone();
15
16         /* Type and precedence */
17         Assert.assertTrue("Type is correct", f1.getType()
               == formulaType);
18
19         /* Id has been incremented */
20         Assert.assertFalse("Ids are different", f1.getID
               () == f2.getID());
21
22         /* Equality and cloning */
```

```
23          Assert.assertEquals("f1 equals itself",f1,f1);
24          Assert.assertFalse("The id of f1 is not equal to
                that of f2", f1.getID() == f2.getID());
25          Assert.assertNotSame("Clone of f1 is not the same
                 as f1", f1, f1Cloned);
26          Assert.assertEquals("Cloning does not change id",
                 f1.getID(), f1Cloned.getID());
27          Assert.assertEquals("Cloning does not change
                formula content", f1Cloned.toString(), f1.
                toString());
28
29
30          if (f1.getType() == Formula.TYPE_CONJUNCTION ||
31          f1.getType() == Formula.TYPE_DISJUNCTION ||
32          f1.getType() == Formula.TYPE_CHOP) {
33              Connective cf1 = (Connective) f1;
34              Connective cf1Cloned = (Connective) f1.clone
                    ();
35              Assert.assertNotSame("Left child of f1Cloned
                    is different than that of f1", cf1Cloned.
                    getLeftOperand(), cf1.getLeftOperand());
36              Assert.assertNotSame("Right child of f1Cloned
                     is different than that of f1", cf1Cloned.
                    getRightOperand(), cf1.getRightOperand());
37          }
38          else if (f1.getType() == Formula.TYPE_NEGATION) {
39              Negation nf1 = (Negation) f1;
40              Negation nf1Cloned = (Negation) f1.clone();
41              Assert.assertNotSame("Child of f1Cloned is
                    different than that of f1", nf1Cloned.
                    getOperand(), nf1.getOperand());
42          }
43          else if (f1.getType() == Formula.TYPE_DURATION) {
44              Duration df1 = (Duration) f1;
45              Duration df1Cloned = (Duration) f1.clone();
46              Assert.assertNotSame("Child of f1Cloned is
                    different than that of f1", df1Cloned.
                    getStateAssertion(), df1.getStateAssertion
                    ());
47          }
48
49          /* hasParent(), getParent(), hasGrandParent(),
                getGrandParent() without parent and
50           * w/o grandparent
51           */
52          Assert.assertFalse("f1 has no parent", f1.
                hasParent());
53          Assert.assertFalse("f1 has no grand parent", f1.
                hasGrandParent());
```

```
54          Assert.assertNull("Parent of f1 is null", f1.
                getParent());
55
56          /* Testing parental status after connecting
                formula in a conjunction.
57           * hasParent(), getParent(), hasGrandParent(),
                 getGrandParent()
58           */
59          Connective parent1 = new Conjunction(f1, f1Cloned
                );
60          Assert.assertTrue("f1 has parent", f1.hasParent()
                );
61          Assert.assertFalse("f1 has no grand parent", f1.
                hasGrandParent());
62          Assert.assertNull("Grand parent of f1 is null",
                f1.getGrandParent());
63          Assert.assertSame("f1's parent is parent1", f1.
                getParent(), parent1);
64
65          /* Testing parental status after exchanging
                formula f1 with f2.
66           * hasParent(), getParent(), hasGrandParent(),
                 getGrandParent()
67           */
68          int parentID1 = parent1.getID();
69          f1.updateParent(f2);
70          Assert.assertSame("Left operand of parent is now
                f2", parent1.getLeftOperand(), f2);
71          Assert.assertFalse("parent's id has changed",
                parentID1 == parent1.getID());
72          Assert.assertFalse("f1 no longer has a parent",
                f1.hasParent());
73          Assert.assertFalse("f1 still has no grand parent
                ", f1.hasGrandParent());
74          Assert.assertNull("Parent of f1 is null", f1.
                getParent());
75          Assert.assertSame("f2's parent is parent1", f2.
                getParent(), parent1);
76
77          /* Testing parental status after connecting
                parent formula with another formula
78           * in a conjunction.
79           * hasParent(), getParent(), hasGrandParent(),
                 getGrandParent()
80           */
81          Connective grandParent1 = new Conjunction(parent1
                , new False());
82          Assert.assertSame("Left operand of parent is
                still f2", parent1.getLeftOperand(), f2);
```

```
83          Assert.assertTrue("f2 now has a parent again", f2
                .hasParent());
84          Assert.assertTrue("f2 still has grand parent", f2
                .hasGrandParent());
85          Assert.assertNotNull("Parent of f2 is not null",
                f2.getParent());
86          Assert.assertNotNull("Grand parent of f2 is not
                null", f2.getGrandParent());
87
88          /* Testing parental status after reconnecting f1
                into a conjunction.
89           * hasParent(), getParent(), hasGrandParent(),
                getGrandParent()
90           */
91          Connective parent2 = new Conjunction(f1, new
                False());
92          Assert.assertFalse("f1 has no grand parent", f1.
                hasGrandParent());
93          Assert.assertNotSame("f1's parent is no longer
                parent1", f1.getParent(), parent1);
94
95          /* Testing updates of grand parent. Until now, f2
                was a left child of parent1, parent1 was left
                child of grandparent1.
96           * Now we exchange parent1 with parent2. Thus, f1
                's grand parent is grandParent1
97           */
98          f2.updateGrandParent(parent2);
99          Assert.assertFalse("Now f1 has a grand parent",
                parent2.hasGrandParent());
100         Assert.assertSame("Parent is grandParent1", f1.
                getGrandParent(), grandParent1);
101
102         int id = f1.getID();
103         f1.nullReps();
104         Assert.assertFalse("Id has changed", id == f1.
                getID());
105
106         if (f1.getType() == Formula.TYPE_STATE ||
107         f1.getType() == Formula.TYPE_TRUE ||
108         f1.getType() == Formula.TYPE_FALSE) {
109             /* Test that formulas do not cache string
                    representation.
110              */
111             Assert.assertNull("f1.stringRep is null", f2.
                    stringRep);
112             f2.toString();
113             Assert.assertNull("f1.stringRep is still null
                    ", f2.stringRep);
114         }
```

```
115         else {
116
117
118             /* Test of caching of string representation.
                   No testing that actual string
                   representation
119              * is correct! See individual tests (e.g.
                    testFalse(), testConjunction() etc.)
120              */
121             Assert.assertNull("f2.stringRep is null", f2.
                   stringRep);
122             f2.toString();
123             Assert.assertNotNull("f2.stringRep isnt null
                   ", f2.stringRep);
124             f2.nullReps();
125             Assert.assertNull("f2.stringRep is null again
                   ", f2.stringRep);
126             if (f1.getType() == Formula.TYPE_CONJUNCTION
                   ||
127             f1.getType() == Formula.TYPE_DISJUNCTION ||
128             f1.getType() == Formula.TYPE_CHOP) {
129                 Connective cf1 = (Connective) f1;
130                 Formula leftChild = cf1.getLeftOperand();
131                 cf1.toString();
132                 Assert.assertNotNull("Has been requested
                       ",cf1.stringRep);
133                 cf1.setLeftOperand(new False());
134                 Assert.assertNull("Left child has been
                       changed. Thus, reps have been nulled",
                        parent1.stringRep);
135                 Assert.assertFalse("Left child's parent
                       reference has been removed", leftChild
                       .hasParent());
136
137                 Formula rightChild = cf1.getRightOperand
                       ();
138                 cf1.toString();
139                 Assert.assertNotNull("Has been requested
                       ",cf1.stringRep);
140                 cf1.setRightOperand(new False());
141                 Assert.assertNull("Right child has been
                       changed. Thus, reps have been nulled",
                        cf1.stringRep);
142                 Assert.assertFalse("Right child's parent
                       reference has been removed",
                       rightChild.hasParent());
143
144
145             }
146         }
```

```
147
148         if ( f1.getType() != Formula.TYPE_DURATION &&
149         f1.getType() != Formula.TYPE_CHOP) {
150             /* Test of caching of robdd. No testing of
                   the actual bdd is correct! See test of bdd
151              * instead.
152              */
153             ROBDDUtil robddUtil = new ROBDDUtil();
154             Assert.assertNull("getRobdd() has never been
                   called. Thus, f1.robdd is null", f1.robdd)
                   ;
155             ROBDD robdd = f1.getROBDD(robddUtil);
156             Assert.assertNotNull("getRobdd() has been
                   called. Thus, f1.robdd is not null", f1.
                   robdd );
157             Assert.assertSame("New calls of getRobdd()
                   ought to return the cached robdd",robdd,
                   f1.getROBDD(robddUtil));
158             if (f1.getType() == Formula.TYPE_CONJUNCTION
                   ||
159             f1.getType() == Formula.TYPE_DISJUNCTION) {
160                 Connective cf1 = (Connective) f1;
161                 cf1.setLeftOperand(new False());
162                 Assert.assertNull("parent1 has been
                       changed. Thus, parent has no longer an
                        robdd", parent1.robdd);
163                 Assert.assertNotSame("Calling getRobdd()
                       should now return another new ROBDD (
                       not the cached one)",robdd,parent1.
                       getROBDD(robddUtil));
164             }
165             f1.nullReps();
166             Assert.assertNull("f1.robdd is null", f1.
                   robdd);
167
168         }
169
170     }
171
172
173     public void testTrue() throws Exception {
174         True f1 = new True();
175         True f2 = new True();
176         Set stateSet = new HashSet();
177
178         performStandardTest(f1, f2, Formula.TYPE_TRUE);
179
180         /* getStates() */
181         f1.getStates(stateSet);
```

```
182         Assert.assertTrue("No states added from f1",
                stateSet.isEmpty());
183
184         /* Testing string representation.
185          */
186         Assert.assertEquals("String representation is
                correct", f1.toString(), True.TRUE_STR);
187     }
188
189     public void testFalse() throws Exception {
190         False f1 = new False();
191         False f2 = new False();
192         Set stateSet = new HashSet();
193
194         performStandardTest(f1, f2, Formula.TYPE_FALSE);
195
196         /* getStates() */
197         f1.getStates(stateSet);
198         Assert.assertTrue("No states added from f1",
                stateSet.isEmpty());
199
200         /* Testing string representation.
201          */
202         Assert.assertEquals("String representation is
                correct", False.FALSE_STR, f1.toString());
203     }
204
205     public void testState() throws Exception {
206         State f1 = new State("x111");
207         State f2 = new State("x222");
208         Set stateSet = new HashSet();
209
210         performStandardTest(f1, f2, Formula.TYPE_STATE);
211
212         /* getStates() */
213         f1.getStates(stateSet);
214         Assert.assertTrue("One state in stateSet",
                stateSet.size() == 1);
215         Assert.assertTrue("State in stateSet is f1",
                stateSet.contains(f1));
216
217         /* Testing string representation.
218          */
219         Assert.assertEquals("String representation is
                correct", "x111", f1.toString());
220     }
221
222     public void testNegation() throws Exception {
223         State s1 = new State("1");
224         State s2 = new State("2");
```

```
225         Negation f1 = new Negation(s1);
226         Negation f2 = new Negation(new Conjunction(new
                True(),new False()));
227         Set stateSet = new HashSet();
228
229         performStandardTest(f1, f2, Formula.TYPE_NEGATION
                );
230
231         Assert.assertSame("Operand is s1", s1, f1.
                getOperand());
232         f1.setOperand(s2);
233         Assert.assertSame("SA is sa2", s2, f1.getOperand
                ());
234
235         /* getStates() */
236         f2.getStates(stateSet);
237         Assert.assertTrue("No states in stateSet",
                stateSet.isEmpty());
238         f1.getStates(stateSet);
239         Assert.assertTrue("One state in stateSet",
                stateSet.size() == 1);
240         Assert.assertTrue("State in stateSet is f1",
                stateSet.contains(s2));
241
242         /* Testing string representation.
243          */
244         Assert.assertEquals("String representation is
                correct", "~2", f1.toString());
245     }
246
247     public void testDuration() throws Exception {
248         State s1 = new State("1");
249         Formula sa1 = new Conjunction(new True(),new
                False());
250         Formula sa2 = new Conjunction(new True(),new
                False());
251         Duration f1 = new Duration(new Conjunction(s1,
                new State("abc")),1);
252         Duration f2 = new Duration(sa1,15);
253         Set stateSet = new HashSet();
254
255         performStandardTest(f1, f2, Formula.TYPE_DURATION
                );
256
257         Assert.assertEquals("Bound is 15", 15, f2.
                getBound());
258         Assert.assertSame("SA is sa1", sa1, f2.
                getStateAssertion());
259         f2.setStateAssertion(sa2);
```

```
260         Assert.assertSame("SA is sa2", sa2, f2.
                getStateAssertion());
261
262         /* getStates() */
263         f2.getStates(stateSet);
264         Assert.assertTrue("No states in stateSet",
                stateSet.isEmpty());
265         f1.getStates(stateSet);
266         Assert.assertTrue("One state in stateSet",
                stateSet.size() == 2);
267         Assert.assertTrue("State in stateSet is f1",
                stateSet.contains(s1));
268         try {
269             Duration f3 = new Duration(s1,0);
270             Assert.assertTrue("Throws error because bound
                    is too low", false);
271         }
272         catch (IllegalArgumentException e) {
273             Assert.assertTrue("Throws error because bound
                    is too low", true);
274         }
275         /* Testing string representation.
276          */
277         Assert.assertEquals("String representation is
                correct", "dur (1 /\\ abc) >= 1", f1.toString
                ());
278     }
279
280     public void testDisjunction() throws Exception {
281         State s1 = new State("1");
282         State s2 = new State("2");
283         Disjunction f1 = new Disjunction(s1, s2);
284         Disjunction f2 = new Disjunction(new True(),new
                Conjunction(new True(),new False()));
285         Disjunction f3 = new Disjunction(new State("s1"),
                new State("s2"));
286         Set stateSet = new HashSet();
287
288         /* getStates() */
289         f2.getStates(stateSet);
290         Assert.assertTrue("No states in stateSet",
                stateSet.isEmpty());
291         f1.getStates(stateSet);
292         Assert.assertTrue("Two states in stateSet",
                stateSet.size() == 2);
293         Assert.assertTrue("s1 is in stateSet", stateSet.
                contains(s1));
294         Assert.assertTrue("s2 is in stateSet", stateSet.
                contains(s2));
295
```

```
296         /* Testing string representation.
297          */
298         Assert.assertEquals("String representation is
                correct", "(1 \\/ 2)", f1.toString());
299         performStandardTest(f1, f2, Formula.
                TYPE_DISJUNCTION);
300
301     }
302
303     public void testConjunction() throws Exception {
304         State s1 = new State("1");
305         State s2 = new State("2");
306         Conjunction f1 = new Conjunction(s1, s2);
307         Conjunction f2 = new Conjunction(new True(),new
                Disjunction(new True(),new False()));
308         Conjunction f3 = new Conjunction(new State("s1"),
                 new State("s2"));
309         Set stateSet = new HashSet();
310
311         /* getStates() */
312         f2.getStates(stateSet);
313         Assert.assertTrue("No states in stateSet",
                stateSet.isEmpty());
314         f1.getStates(stateSet);
315         Assert.assertTrue("Two states in stateSet",
                stateSet.size() == 2);
316         Assert.assertTrue("s1 is in stateSet", stateSet.
                contains(s1));
317         Assert.assertTrue("s2 is in stateSet", stateSet.
                contains(s2));
318
319         /* Testing string representation.
320          */
321         Assert.assertEquals("String representation is
                correct", "(1 /\\ 2)", f1.toString());
322
323         performStandardTest(f1, f2, Formula.
                TYPE_CONJUNCTION);
324
325     }
326
327     public void testChop() throws Exception {
328         State s1 = new State("1");
329         State s2 = new State("2");
330         Chop f1 = new Chop(s1, s2);
331         Chop f2 = new Chop(new True(),new Disjunction(new
                 True(),new False()));
332         Chop f3 = new Chop(new State("s1"), new State("s2
                "));
333         Set stateSet = new HashSet();
```

```
334
335         /* getStates() */
336         f2.getStates(stateSet);
337         Assert.assertTrue("No states in stateSet",
                stateSet.isEmpty());
338         f1.getStates(stateSet);
339         Assert.assertTrue("Two states in stateSet",
                stateSet.size() == 2);
340         Assert.assertTrue("s1 is in stateSet", stateSet.
                contains(s1));
341         Assert.assertTrue("s2 is in stateSet", stateSet.
                contains(s2));
342
343         /* Testing string representation.
344          */
345         Assert.assertEquals("String representation is
                correct", "(1 ; 2)", f1.toString());
346         performStandardTest(f1, f2, Formula.TYPE_CHOP);
347     }
348
349     public static junit.framework.Test suite() {
350         return new TestSuite(Test.class);
351     }
352 }
```

G.1.3 bmc.literal.Test.java

```
1  package bmc.literal;
2
3  import bmc.*;
4  import bmc.formula.*;
5  import bmc.util.*;
6  import junit.framework.*;
7  import java.io.*;
8  import java.util.*;
9
10 public class Test extends TestCase {
11
12
13     public void testIndexedFormula() {
14         Formula formula1 = new True();
15         Formula formula2 = new False();
16
17         // testing constructor
18         IndexedFormula iF1 = new IndexedFormula(formula1.
               getID(), 1);
19         Assert.assertEquals("iF1.i is 1", 1, iF1.i);
20         Assert.assertEquals("iF1.m is -1", -1, iF1.m);
21         Assert.assertEquals("iF1.j is -1", -1, iF1.j);
22
```

```
23          IndexedFormula iF2 = new IndexedFormula(formula1.
                getID(), 2, 3);
24          Assert.assertEquals("iF2.i is 2", 2, iF2.i);
25          Assert.assertEquals("iF2.m is -1", -1, iF2.m);
26          Assert.assertEquals("iF2.j is 3", 3, iF2.j);
27
28          IndexedFormula iF3 = new IndexedFormula(formula1.
                getID(), 4, 5, 6);
29          Assert.assertEquals("iF3.i is 4", 4, iF3.i);
30          Assert.assertEquals("iF3.m is 5", 5, iF3.m);
31          Assert.assertEquals("iF3.j is 6", 6, iF3.j);
32
33          // testing equals
34          Assert.assertFalse("iF1 does not equal another
                type of object", iF1.equals(new True()));
35
36          Assert.assertEquals("iF1 equals itself", iF1, iF1
                );
37          Assert.assertFalse("iF1 and iF2 are not equal",
                iF1.equals(iF2));
38          IndexedFormula iF1Equal = new IndexedFormula(
                formula1.getID(), 1);
39          Assert.assertEquals("iF1 equals iF1Equal", iF1,
                iF1Equal);
40          IndexedFormula iF1NotEqual1 = new IndexedFormula(
                formula1.getID(), 2);
41          Assert.assertFalse("iF1 does not equal
                iF1NotEqual1", iF1.equals(iF1NotEqual1));
42          IndexedFormula iF1NotEqual2 = new IndexedFormula(
                formula1.getID(), 1,1);
43          Assert.assertFalse("iF1 does not equal
                iF1NotEqual2", iF1.equals(iF1NotEqual2));
44          IndexedFormula iF1NotEqual3 = new IndexedFormula(
                formula1.getID(), 1,1,1);
45          Assert.assertFalse("iF1 does not equal
                iF1NotEqual3", iF1.equals(iF1NotEqual3));
46
47          IndexedFormula iF2Equal = new IndexedFormula(
                formula1.getID(), 2, 3);
48          Assert.assertEquals("iF2 equals iF2Equal", iF2,
                iF2Equal);
49          IndexedFormula iF2NotEqual1 = new IndexedFormula(
                formula1.getID(), 2);
50          Assert.assertFalse("iF2 does not equal
                iF2NotEqual1", iF2.equals(iF2NotEqual1));
51          IndexedFormula iF2NotEqual2 = new IndexedFormula(
                formula1.getID(), 1,1);
52          Assert.assertFalse("iF2 does not equal
                iF2NotEqual2", iF2.equals(iF2NotEqual2));
```

```
53          IndexedFormula iF2NotEqual3 = new IndexedFormula(
                formula1.getID(), 2,4,3);
54          Assert.assertFalse("iF2 does not equal
                iF2NotEqual3", iF2.equals(iF2NotEqual3));
55
56          IndexedFormula iF3Equal = new IndexedFormula(
                formula1.getID(), 4, 5, 6);
57          Assert.assertEquals("iF3 equals iF3Equal", iF3,
                iF3Equal);
58          IndexedFormula iF3NotEqual1 = new IndexedFormula(
                formula1.getID(), 4);
59          Assert.assertFalse("iF3 does not equal
                iF3NotEqual1", iF3.equals(iF3NotEqual1));
60          IndexedFormula iF3NotEqual2 = new IndexedFormula(
                formula1.getID(), 4,6);
61          Assert.assertFalse("iF3 does not equal
                iF3NotEqual2", iF3.equals(iF3NotEqual2));
62          IndexedFormula iF3NotEqual3 = new IndexedFormula(
                formula1.getID(), 4,2,6);
63          Assert.assertFalse("iF3 does not equal
                iF3NotEqual3", iF3.equals(iF3NotEqual3));
64
65          // hashCode
66          Assert.assertEquals("hashCodes for equal objects
                are equal", iF3.hashCode(), iF3Equal.hashCode
                ());
67      }
68
69      public static void testLiteral() {
70          // testing constructor, getNumber(), getSign()
                and toString()
71
72          // constructor: Literal(bool)
73          Literal trueLit = new Literal(true);
74          Literal falseLit = new Literal(false);
75          Assert.assertEquals("trueLit.getNumber() ==
                BOOL_LIT_NO", VariableNoGenerator.BOOL_LIT_NO,
                 trueLit.getNumber());
76          Assert.assertEquals("trueLit.getSign() == true",
                true, trueLit.getSign());
77          Assert.assertEquals("falseLit.getNumber() ==
                BOOL_LIT_NO", VariableNoGenerator.BOOL_LIT_NO,
                 falseLit.getNumber());
78          Assert.assertEquals("falseLit.getSign() == false
                ", false, falseLit.getSign());
79
80          // constructor: Literal(int, bool)
81          Literal lit1 = new Literal(10, true);
82          Assert.assertEquals("lit1.getNumber() == 10", 10,
                 lit1.getNumber());
```

```
83          Assert.assertEquals("lit1.getSign() == true",
                true, lit1.getSign());
84          Assert.assertEquals("toString() looks like this",
                "x10", lit1.toString());
85          Literal lit2 = new Literal(2, false);
86          Assert.assertEquals("lit2.getNumber() == 2", 2,
                lit2.getNumber());
87          Assert.assertEquals("lit2.getSign() == false",
                false, lit2.getSign());
88          Assert.assertEquals("toString() looks like this",
                "~x2", lit2.toString());
89          try {
90              Literal lit3 = new Literal(
                    VariableNoGenerator.BOOL_LIT_NO, true);
91              Assert.assertFalse("Creating literal with
                    number = BOOL_LIT_NO throws exception",
                    true);
92          }
93          catch (Exception e) {
94              Assert.assertTrue("Creating literal with
                    number = BOOL_LIT_NO throws exception",
                    true);
95          }
96          try {
97              Literal lit3 = new Literal(-1, true);
98              Assert.assertFalse("Creating literal with
                    negative number throws exception", true);
99          }
100         catch (Exception e) {
101             Assert.assertTrue("Creating literal with
                    negative number throws exception", true);
102         }
103
104         // testing getTrueLiteral() / getFalseLiteral()
105         trueLit = Literal.getTrueLiteral();
106         falseLit = Literal.getFalseLiteral();
107         Assert.assertEquals("trueLit.getNumber() ==
                BOOL_LIT_NO", VariableNoGenerator.BOOL_LIT_NO,
                trueLit.getNumber());
108         Assert.assertEquals("trueLit.getSign() == true",
                true, trueLit.getSign());
109         Assert.assertEquals("falseLit.getNumber() ==
                BOOL_LIT_NO", VariableNoGenerator.BOOL_LIT_NO,
                falseLit.getNumber());
110         Assert.assertEquals("falseLit.getSign() == false
                ", false, falseLit.getSign());
111
112
113         // testing flipSign()
114         Literal trueLitFlipped = trueLit.flipSign();
```

```
115         Assert.assertEquals("falseLit.getNumber() ==
                BOOL_LIT_NO", VariableNoGenerator.BOOL_LIT_NO,
                 falseLit.getNumber());
116         Assert.assertEquals("falseLit.getSign() == false
                ", false, falseLit.getSign());
117
118         Literal lit1Flipped = lit1.flipSign();
119         Assert.assertEquals("lit1Flipped.getSign() is
                false", lit1Flipped.getSign(), false);
120         // testing equals()
121         Assert.assertEquals("lit1 equals itself", lit1,
                lit1);
122         Assert.assertFalse("lit1 does not equal lit2",
                lit1.equals(lit2));
123         Assert.assertFalse("lit1 does not equal
                lit1Flipped", lit1.equals(lit1Flipped));
124         Assert.assertEquals("lit1 equals lit1Flipped.
                flipSign()", lit1, lit1Flipped.flipSign());
125         Assert.assertFalse("lit1 does not equal a
                different kind of object", lit1.equals(new
                True()));
126
127         // testing hashCode()
128         Assert.assertEquals("lit1.hashCode() equals
                lit1Flipped.flipSign().hashCode()", lit1.
                hashCode(), lit1Flipped.flipSign().hashCode())
                ;
129
130         // testing compareTo()
131         Assert.assertEquals("lit1 is equal to lit1Flipped
                .flipSign()", 0, lit1.compareTo(lit1Flipped.
                flipSign()));
132         Assert.assertTrue("lit1 is greater than
                lit1Flipped", lit1.compareTo(lit1Flipped) > 0)
                ;
133         Assert.assertTrue("lit1Flipped is smaller than
                lit1Flipped", lit1Flipped.compareTo(lit1) < 0)
                ;
134         Assert.assertTrue("lit2 is smaller than lit1",
                lit2.compareTo(lit1) < 0);
135     }
136
137     public void testVariableNoGenerator() {
138         VariableNoGenerator litNoGen1 = new
                VariableNoGenerator();
139         Assert.assertEquals("litNoGen1.nextLitNo is equal
                 to BOOL_LIT_NO+1", VariableNoGenerator.
                BOOL_LIT_NO+1, litNoGen1.nextLiteralNo);
140         Assert.assertEquals("litNoGen1.getNextLitNo
                returns nextLitNo", litNoGen1.nextLiteralNo,
```

```
            litNoGen1.getNextLiteralNo());
141         Assert.assertEquals("litNoGen1.nextLitNo is now
                equal to BOOL_LIT_NO+2", VariableNoGenerator.
                BOOL_LIT_NO+2, litNoGen1.nextLiteralNo);
142         Assert.assertEquals("litNoGen1.getNextLitNo
                really returns the nextLitNo", litNoGen1.
                nextLiteralNo, litNoGen1.getNextLiteralNo());
143         Assert.assertEquals("litNoGen1.getNumLiterals()
                is equal to BOOL_LIT_NO+1", litNoGen1.
                nextLiteralNo-1, litNoGen1.getNumVariables());
144     }
145
146     public void performStandardLitHandlerTest(
            LiteralHandler litHandler) throws
            CloneNotSupportedException {
147
148         Assert.assertNotNull("Fields have been
                initialized", litHandler.formulaIDToNormID);
149         Assert.assertNotNull("Fields have been
                initialized", litHandler.normIDToLiteral);
150         Assert.assertNotNull("Fields have been
                initialized", litHandler.normIDs);
151         Assert.assertNotNull("Fields have been
                initialized", litHandler.varNoGenerator);
152
153         // test getLiteral(true, i, m, j) returns lit(
                BOOL_LIT_NO, true)
154         // test getFalseLit(false, i, m, j) returns lit(
                BOOL_LIT_NO, false)
155         Assert.assertEquals("getLiteral(true,1) returns a
                 true lit", Literal.getTrueLiteral(),
                litHandler.getLiteral(new True(),1));
156         Assert.assertEquals("getLiteral(true,1,5) returns
                 a true lit", Literal.getTrueLiteral(),
                litHandler.getLiteral(new True(),1,5));
157         Assert.assertEquals("getLiteral(true,1,5,10)
                returns a true lit", Literal.getTrueLiteral(),
                 litHandler.getLiteral(new True(),1,5,10));
158         Assert.assertEquals("getLiteral(false,1) returns
                a false lit", Literal.getFalseLiteral(),
                litHandler.getLiteral(new False(),1));
159         Assert.assertEquals("getLiteral(false,1,5)
                returns a false lit", Literal.getFalseLiteral
                (), litHandler.getLiteral(new False(),1,5));
160         Assert.assertEquals("getLiteral(false,1,5,10)
                returns a false lit", Literal.getFalseLiteral
                (), litHandler.getLiteral(new False(),1,5,10))
                ;
161
```

```
162         Assert.assertEquals("getNumVariables equals 1 by
                now", 1, litHandler.getNumVariables());
163
164         // one formula corresponds to one literal
165         Formula formula1 = new Conjunction(new Duration(
                new State("x1"), 3), new True());
166         Integer normID1 = (Integer) litHandler.
                formulaIDToNormID.get(new Integer(formula1.
                getID()));
167         Assert.assertNull("Before retrieving literal
                cache does not have this normId in table",
                normID1);
168         Assert.assertSame("Retrieving a literal twice for
                 a formula gives the same literal", litHandler
                .getLiteral(formula1, 1), litHandler.
                getLiteral(formula1, 1));
169         normID1 = (Integer) litHandler.formulaIDToNormID.
                get(new Integer(formula1.getID()));
170         Assert.assertNotNull("Retrieving literal caches
                normId in table", normID1);
171         IndexedFormula iF1 = new IndexedFormula(normID1.
                intValue(), 1, -1, -1);
172         Literal lit1 = (Literal) litHandler.
                normIDToLiteral.get(iF1);
173         Assert.assertNotNull("Retrieving literal caches
                indexed formula in table", iF1);
174         Assert.assertSame("Retrieving a literal twice for
                 a formula gives the same literal", litHandler
                .getLiteral(formula1, 1, 5), litHandler.
                getLiteral(formula1, 1,5));
175         Assert.assertSame("Retrieving a literal twice for
                 a formula gives the same literal", litHandler
                .getLiteral(formula1, 1, 5, 10), litHandler.
                getLiteral(formula1, 1,5,10));
176         Assert.assertEquals("getNumLiterals equals 4 by
                now", 4, litHandler.getNumVariables());
177
178         // negations are not significant for anything but
                 signs
179         Formula nF1 = new Negation(formula1);
180         Formula nnF1 = new Negation(new Negation(formula1
                ));
181         Formula nFalse = new Negation(new False());
182         Formula nnFalse = new Negation(new Negation(new
                False()));
183         Formula nTrue = new Negation(new True());
184
185         Assert.assertEquals("Literal signs [nF1] and [
                formula1] are opposite", !litHandler.
                getLiteral(formula1,1).getSign(), litHandler.
```

```
            getLiteral(nF1,1).getSign());
186         Assert.assertEquals("Literal numbers are the same
                for [nF1] and [formula1]", litHandler.
            getLiteral(formula1,1).getNumber(), litHandler
            .getLiteral(nF1,1).getNumber());
187         Assert.assertSame("[~~formula] = [formula] (aka
            nnF1, formula1)", litHandler.getLiteral(
            formula1, 1), litHandler.getLiteral(nnF1, 1));
188         Assert.assertEquals("getLiteral(nTrue,1,5)
            returns a true lit", Literal.getFalseLiteral()
            , litHandler.getLiteral(nTrue,1,5));
189         Assert.assertEquals("getLiteral(nFalse,1,5)
            returns a true lit", Literal.getTrueLiteral(),
             litHandler.getLiteral(nFalse,1,5));
190         Assert.assertEquals("getLiteral(nnFalse,1,5)
            returns a false lit", Literal.getFalseLiteral
            (), litHandler.getLiteral(nnFalse,1,5));
191         Assert.assertEquals("getNumLiterals still equals
            4", 4, litHandler.getNumVariables());
192
193         // different indexes give different literals
194         Assert.assertNotSame("Retrieving a literal for
            the same formula in different intervals give
            different literals", litHandler.getLiteral(
            formula1, 1), litHandler.getLiteral(formula1,
            2));
195         Assert.assertNotSame("Retrieving a literal for
            the same formula in different intervals give
            different literals", litHandler.getLiteral(
            formula1, 1, 5), litHandler.getLiteral(
            formula1, 2,5));
196         Assert.assertNotSame("Retrieving a literal for
            the same formula in different intervals give
            different literals", litHandler.getLiteral(
            formula1, 1, 5, 10), litHandler.getLiteral(
            formula1, 2,5,10));
197
198         // different formulas give different literals
199         Formula formula2 = new Disjunction(new Duration(
            new State("x1"),5), new True());
200         Assert.assertNotSame("Retrieving a literal for
            different formulas give different literals",
            litHandler.getLiteral(formula1, 1), litHandler
            .getLiteral(formula2, 1));
201         Assert.assertNotSame("Retrieving a literal for
            different formulas give different literals",
            litHandler.getLiteral(formula1, 1, 5),
            litHandler.getLiteral(formula2, 1,5));
202         Assert.assertNotSame("Retrieving a literal for
            different formulas give different literals",
```

```
            litHandler.getLiteral(formula1, 1, 5, 10),
            litHandler.getLiteral(formula2, 1,5,10));
203
204
205         // Adding a formula f, makes have hasDef(f) =
                hasDef(~~f) = true
206         formula1 = new Conjunction(new True(), new False
                ());
207         nF1 = new Negation(formula1);
208         nnF1 = new Negation(nF1);
209         Assert.assertFalse("Does not have formula1 under
                true pol", litHandler.hasDef(formula1, true));
210         normID1 = (Integer) litHandler.formulaIDToNormID.
                get(new Integer(formula1.getID()));
211         Assert.assertNotNull("normID1 has been added",
                normID1);
212         Assert.assertFalse("Does not have formula1 under
                false pol", litHandler.hasDef(formula1, false)
                );
213
214         litHandler.addDef(formula1, true);
215
216         Assert.assertTrue("Has formula1 under true pol",
                litHandler.hasDef(formula1, true));
217         Assert.assertFalse("Does not have formula1 under
                false pol", litHandler.hasDef(formula1, false)
                );
218
219         Assert.assertTrue("Has nF1 under false pol",
                litHandler.hasDef(nF1, false));
220         Assert.assertFalse("Does not have nF1 under true
                pol", litHandler.hasDef(nF1, true));
221
222         Assert.assertTrue("Has nnF1 under true pol",
                litHandler.hasDef(nnF1, true));
223         Assert.assertFalse("Does not have nnF1 under
                false pol", litHandler.hasDef(nnF1, false));
224
225         litHandler.addDef(formula1, false);
226         Assert.assertTrue("Now has formula1 under false
                pol", litHandler.hasDef(formula1, false));
227
228         Formula f2 = new State("a");
229         litHandler.addDef(f2, true);
230         int normIDsSize = litHandler.normIDs.size();
231         litHandler.addDef(new Negation(f2), false);
232         Assert.assertEquals("nF1 was already added in
                that polarity", normIDsSize, litHandler.
                normIDs.size());
233         // a totally different formula
```

```
234         Assert.assertFalse("Does not have nFalse under
                true pol", litHandler.hasDef(nFalse, true));
235         Assert.assertFalse("Does not have nFalse under
                false pol", litHandler.hasDef(nFalse, false));
236
237         // Identical formulas (id, syntactic, or semantic
                ) gives
238         Formula f1Clone = (Formula) formula1.clone();
239         Assert.assertTrue("Clones are always identical =>
                 f1Clone is in set", litHandler.hasDef(f1Clone
                , false));
240         litHandler.addDef(f1Clone, false);
241
242         // hasDef(State)
243         State s1 = new State("x1");
244         State s2 = new State("x2");
245
246         Assert.assertFalse("Does not have state1",
                litHandler.hasDef(s1));
247         litHandler.addDef(s1, true);
248         Assert.assertTrue("Has s1", litHandler.hasDef(s1)
                );
249         Assert.assertFalse("Does not have state2",
                litHandler.hasDef(s2));
250         litHandler.addDef(s2, false);
251         Assert.assertTrue("Has s2", litHandler.hasDef(s2)
                );
252     }
253
254     public void testIDLookupLiteralHandler() throws
            CloneNotSupportedException {
255         // test constructor
256         IDLookupLiteralHandler litHandler = new
                IDLookupLiteralHandler();
257
258         performStandardLitHandlerTest(litHandler);
259
260         // a formula is equal to it's clone with respect
                to this literal handler
261         Formula formula1 = new Conjunction(new Duration(
                new State("x1"), 5), new True());
262
263         Assert.assertEquals("NormalizedID is formulaId",
                formula1.getID(), litHandler.getNormalizedID(
                formula1));
264     }
265
266     public void testSyntacticLookupLiteralHandler()
            throws CloneNotSupportedException {
267         // test constructor
```

```
268         SyntacticLookupLiteralHandler litHandler = new
                SyntacticLookupLiteralHandler();
269
270         Assert.assertNotNull("Fields have been
                initialized", litHandler.strRepToNormID);
271
272         performStandardLitHandlerTest(litHandler);
273
274         // a formula is equal to it's clone with respect
                to this literal handler
275         Formula formula1 = new Conjunction(new Duration(
                new State("x1"), 5), new True());
276         Formula formula2 = new Conjunction(new Duration(
                new State("x1"), 5), new True());
277
278         int normID1 = litHandler.getNormalizedID(formula1
                );
279         Assert.assertEquals("normids are equal", normID1,
                 litHandler.getNormalizedID(formula1));
280         Assert.assertEquals("normids are equal", normID1,
                 litHandler.getNormalizedID(formula2));
281
282     }
283
284     public void testSemanticLookupLiteralHandler() throws
            CloneNotSupportedException {
285         // test constructor
286         SemanticLookupLiteralHandler litHandler = new
                SemanticLookupLiteralHandler();
287
288         Assert.assertNotNull("Fields have been
                initialized", litHandler.strRepToNormID);
289         Assert.assertNotNull("Fields have been
                initialized", litHandler.cnfTrans);
290
291
292         performStandardLitHandlerTest(litHandler);
293
294         /* getNormalizedClauseRep() */
295
296         SortedSet reps = new TreeSet();
297         litHandler.getNormalizedClauseRep(new Chop(new
                True(), new False()), reps);
298         Assert.assertTrue("Rep added", reps.contains("(
                true ; false)"));
299
300         reps = new TreeSet();
301         litHandler.getNormalizedClauseRep(new Disjunction
                (new True(), new False()), reps);
```

```
302         Assert.assertTrue("Rep added", reps.contains("
                true"));
303         Assert.assertTrue("Rep added", reps.contains("
                false"));
304
305         reps = new TreeSet();
306         litHandler.getNormalizedClauseRep(new Duration(
                new True(),2), reps);
307         Assert.assertTrue("Rep added", reps.contains("dur
                (true) >=2"));
308
309         reps = new TreeSet();
310         litHandler.getNormalizedClauseRep(new False(),
                reps);
311         Assert.assertTrue("Rep added", reps.contains("
                false"));
312
313         reps = new TreeSet();
314         litHandler.getNormalizedClauseRep(new State("abc
                "), reps);
315         Assert.assertTrue("Rep added", reps.contains("abc
                "));
316
317         reps = new TreeSet();
318         litHandler.getNormalizedClauseRep(new Negation(
                new True()), reps);
319         Assert.assertTrue("Rep added", reps.contains("~
                true"));
320
321         reps = new TreeSet();
322         try {
323             litHandler.getNormalizedClauseRep(new
                    Conjunction(new True(), new False()), reps
                    );
324             Assert.assertTrue("Throws exception", false);
325         }
326         catch (Exception e) {
327             Assert.assertTrue("Throws exception", true);
328         }
329
330         /* getNormalizedRep(formula, set) */
331
332         reps = new TreeSet();
333         litHandler.getNormalizedRep(new Disjunction(new
                True(), new False()), reps);
334         Assert.assertTrue("Rep added", reps.contains("
                false | true"));
335
336         reps = new TreeSet();
```

```
337         litHandler.getNormalizedRep(new Conjunction(new
                True(), new False()), reps);
338         Assert.assertTrue("Rep added", reps.contains("
                true"));
339         Assert.assertTrue("Rep added", reps.contains("
                false"));
340
341         // getNormalizedRep(formula)
342         Formula f1 = new Negation(new Disjunction(new
                True(), new Duration(new State("a1"),2)));
343         String normRep = litHandler.getNormalizedRep(f1);
344         Assert.assertEquals("Rep is", "false & ~dur(a1)
                >=2", normRep);
345
346         // getNormalizedID()
347         int normID1 = litHandler.getNormalizedID(f1);
348         Assert.assertEquals("normIDs are equal", normID1,
                 litHandler.getNormalizedID(new Negation(new
                Disjunction(new True(), new Duration(new State
                ("a1"),2)))));
349
350         reps = new TreeSet();
351         // Some functional tests
352         Formula formula3 = new Disjunction(
353         new Conjunction(
354         new Duration(new State("x1"),3),
355         new True()),
356         new Duration(new State("x2"),4));
357         Formula formula3Equiv = new Conjunction(
358         new Disjunction(new Duration(new State("x1"),3),
                new Duration(new State("x2"),4)),
359         new Disjunction(new Duration(new State("x2"),4),
                new True()));
360         Assert.assertSame("Retrieving a literal for some
                sem. equiv. formulas gives the same literal",
                litHandler.getLiteral(formula3, 1), litHandler
                .getLiteral(formula3Equiv, 1));
361
362         Formula formula3Equiv2 = new Disjunction(
363         new Duration(new State("x1"),3),
364         new Conjunction(new Duration(new State("x1"),3),
                new Duration(new State("x2"),4)));
365         Assert.assertNotSame("Retrieving a literal for
                semantically equiv. formulas does not always
                give the same literal", litHandler.getLiteral(
                formula3, 1), litHandler.getLiteral(
                formula3Equiv2, 1));
366
367
368         Formula clause1 = new Disjunction(
```

```
369         new Negation(
370         new Duration(
371         new Conjunction(new True(), new State("a")),3)),
372         new Chop(new True(), new Conjunction(new True(),
                new False())));
373
374         /* getNormalizedClauseRep(clause) -> sorting of
                subformula
375          * Connectives -> sortering of subformulas on
                both sides
376          */
377         litHandler.getNormalizedRep(clause1, reps);
378         Assert.assertEquals("Only one item in reps (only
                one clause)", 1, reps.size());
379         /* Comments:
380          * Disjunctions perform ordering of child nodes
381          * Chop does not reorder, brackets, ordering
                performed of child nodes
382          * Durations use brackets, state ass. is sorted
383          * Disjunctions split by |
384          * Conjunctions perform ordering of child nodes,
                split by &
385          */
386         String expectedRep = "(true ; false & true) | ~
                dur(a & true) >=3";
387         Assert.assertEquals("Normalized clause rep looks
                like this", expectedRep, reps.first());
388
389         litHandler.addDef(formula3,true);
390         Assert.assertTrue("formula3 and formula3Equiv are
                semantically the same -> true", litHandler.
                hasDef(formula3Equiv,true));
391         Assert.assertFalse("formula3 and formula3Equiv2
                are semantically but the litHandler cannot see
                this. -> false", litHandler.hasDef(
                formula3Equiv2,true));
392
393
394     }
395
396
397     public static junit.framework.Test suite() {
398         return new TestSuite(Test.class);
399     }
400 }
```

G.1.4 bmc.parser.Test.java

```
1  package bmc.parser;
2
```

```
3  import bmc.literal.*;
4  import bmc.util.*;
5  import bmc.parser.*;
6  import bmc.formula.*;
7
8  import junit.framework.*;
9
10 import java.io.*;
11 import java.util.*;
12
13 public class Test extends TestCase {
14
15     public Test(String testName) {
16         super(testName);
17     }
18
19     private List parse(String input) throws Exception {
20         StringReader reader = new StringReader(input);
21         BMCParser parser = new BMCParser();
22         parser.setScanner(new BMCLex(reader));
23         return parser.parseTasks();
24     }
25
26     public void testParser() throws Exception {
27         String inputStr = ":- shell(\"some shell command
               in here\").";
28         List tasks = parse(inputStr);
29         Command command = (Command) tasks.get(0);
30
31         // third one is goal2 and has different values
32         Assert.assertEquals("Command text is correct", "
               some shell command in here", command.
               getCommandText());
33         // not setting any settings -> default values
34         Settings stdSettings = new Settings();
35         inputStr +=
36         ":- state x1."+
37         ":- goal goal1 dur(x1) >= 1.";
38         tasks = parse(inputStr);
39         Assert.assertEquals("Two tasks in task list", 2,
               tasks.size());
40         Goal goal = (Goal) tasks.get(1);
41         Set states = goal.getStates();
42         Assert.assertEquals("Only one state in states",
               1, states.size());
43         Assert.assertEquals("That state is x1", "x1", ((
               State) states.iterator().next()).getName());
44         Settings settings = goal.getSettings();
45         Assert.assertEquals("Goal content is correct", "
               goal1", goal.getName());
```

```
46          Assert.assertEquals("Goal content is correct", "
               dur x1 >= 1", goal.getFormula().toString());
47          Assert.assertEquals("Goal content is correct",
               stdSettings.getK(), settings.getK());
48          Assert.assertEquals("Goal content is correct",
               stdSettings.getOutputFormat(), settings.
               getOutputFormat());
49          Assert.assertEquals("Goal content is correct",
               stdSettings.getOutputFolder(), settings.
               getOutputFolder());
50          Assert.assertEquals("Goal content is correct",
               stdSettings.getDCSimpLevel(), settings.
               getDCSimpLevel());
51          Assert.assertEquals("Goal content is correct",
               stdSettings.getFindK(), settings.getFindK());
52          Assert.assertEquals("Goal content is correct",
               stdSettings.getNNF(), settings.getNNF());
53          Assert.assertEquals("Goal content is correct",
               stdSettings.getPolarityOpt(), settings.
               getPolarityOpt());
54          Assert.assertEquals("Goal content is correct",
               stdSettings.getFRecognition(), settings.
               getFRecognition());
55
56
57          // setting all values -> changes
58          inputStr +=
59          ":- set k = 11."+
60          ":- set outputFormat = zolcs."+
61          ":- set outputFolder = \"c:\\\\t\"."+
62          ":- set dcSimpLevel = 0."+
63          ":- set findk = false."+
64          ":- set nnf = false."+
65          ":- set polarityOpt = false."+
66          ":- set fRecognition = id."+
67          ":- state x2."+
68          ":- goal goal2 dur(x2) >= 2.";
69
70          tasks = parse(inputStr);
71          Assert.assertEquals("Two tasks in task list", 3,
               tasks.size());
72          /* Setting values after a goal -> different
               values in different goals:
73           * First one is still the goal1
74           */
75          goal = (Goal) tasks.get(1);
76          Assert.assertEquals("Goal content is correct", "
               goal1", goal.getName());
77          Assert.assertEquals("Goal content is correct", "
               dur x1 >= 1", goal.getFormula().toString());
```

```
78          Assert.assertEquals("Goal content is correct",
                stdSettings.getK(), settings.getK());
79          // second one is goal2 and has different values
80          goal = (Goal) tasks.get(2);
81          settings = goal.getSettings();
82          Assert.assertEquals("Goal content is correct", "
                goal2", goal.getName());
83          Assert.assertEquals("Goal content is correct", "
                dur x2 >= 2", goal.getFormula().toString());
84          Assert.assertEquals("Goal content is correct",
                11, settings.getK());
85          Assert.assertEquals("Goal content is correct",
                Settings.OUTPUT_FORMAT_ZOLCS, settings.
                getOutputFormat());
86          Assert.assertEquals("Goal content is correct", "c
                :\\\\t", settings.getOutputFolder());
87          Assert.assertEquals("Goal content is correct",
                Settings.DC_SIMP_LEVEL_NONE, settings.
                getDCSimpLevel());
88          Assert.assertEquals("Goal content is correct",
                false, settings.getFindK());
89          Assert.assertEquals("Goal content is correct",
                false, settings.getNNF());
90          Assert.assertEquals("Goal content is correct",
                false, settings.getPolarityOpt());
91          Assert.assertEquals("Goal content is correct",
                Settings.F_RECOGNITION_ID, settings.
                getFRecognition());
92
93          inputStr +=
94          ":- set outputFormat = dimacs."+
95          ":- set dcSimpLevel = 1."+
96          ":- set findk = true."+
97          ":- set nnf = true."+
98          ":- set polarityOpt = true."+
99          ":- set fRecognition = syntactic."+
100         ":- state x3."+
101         ":- goal goal3 dur(x3) >= 3.";
102         tasks = parse(inputStr);
103         // third one is goal2 and has different values
104         goal = (Goal) tasks.get(3);
105         settings = goal.getSettings();
106         Assert.assertEquals("Goal content is correct", "
                goal3", goal.getName());
107         Assert.assertEquals("Goal content is correct", "
                dur x3 >= 3", goal.getFormula().toString());
108         Assert.assertEquals("Goal content is correct",
                11, settings.getK());
109         Assert.assertEquals("Goal content is correct",
                Settings.OUTPUT_FORMAT_DIMACS , settings.
```

```
            getOutputFormat());
110         Assert.assertEquals("Goal content is correct", "c
                :\\\\t", settings.getOutputFolder());
111         Assert.assertEquals("Goal content is correct",
                Settings.DC_SIMP_LEVEL_BASIC, settings.
                getDCSimpLevel());
112         Assert.assertEquals("Goal content is correct",
                true, settings.getFindK());
113         Assert.assertEquals("Goal content is correct",
                true, settings.getNNF());
114         Assert.assertEquals("Goal content is correct",
                true, settings.getPolarityOpt());
115         Assert.assertEquals("Goal content is correct",
                Settings.F_RECOGNITION_SYNTACTIC, settings.
                getFRecognition());
116         states = goal.getStates();
117         Assert.assertEquals("Three state in states", 3,
                states.size());
118         Iterator it = states.iterator();
119         String concatStateStr = ":"+((State) it.next()).
                getName()+":"+
120         ((State) it.next()).getName()+":"+
121         ((State) it.next()).getName()+":";
122         Assert.assertTrue("x1 is in string",
                concatStateStr.indexOf(":x1:") >= -1);
123         Assert.assertTrue("x2 is in string",
                concatStateStr.indexOf(":x2:") >= -1);
124         Assert.assertTrue("x3 is in string",
                concatStateStr.indexOf(":x3:") >= -1);
125         inputStr +=
126         ":- set dcSimpLevel = 2."+
127         ":- set fRecognition = semantic."+
128         ":- state x4."+
129         ":- goal goal4 dur(x4) >= 4.";
130         // third one is goal2 and has different values
131         tasks = parse(inputStr);
132         goal = (Goal) tasks.get(4);
133         settings = goal.getSettings();
134         Assert.assertEquals("Goal content is correct", "
                goal4", goal.getName());
135         Assert.assertEquals("Goal content is correct", "
                dur x4 >= 4", goal.getFormula().toString());
136         Assert.assertEquals("Goal content is correct",
                11, settings.getK());
137         Assert.assertEquals("Goal content is correct",
                Settings.OUTPUT_FORMAT_DIMACS , settings.
                getOutputFormat());
138         Assert.assertEquals("Goal content is correct", "c
                :\\\\t", settings.getOutputFolder());
```

```
139         Assert.assertEquals("Goal content is correct",
                Settings.DC_SIMP_LEVEL_WITH_BDD, settings.
                getDCSimpLevel());
140         Assert.assertEquals("Goal content is correct",
                true, settings.getFindK());
141         Assert.assertEquals("Goal content is correct",
                true, settings.getNNF());
142         Assert.assertEquals("Goal content is correct",
                true, settings.getPolarityOpt());
143         Assert.assertEquals("Goal content is correct",
                Settings.F_RECOGNITION_SEMANTIC, settings.
                getFRecognition());
144
145         /* testing
146          * - durations
147          * - state assertion formulas
148          * - dc formulas
149          */
150         inputStr +=
151         ":- state x5.\n"+
152         ":- goal goal5 dur(x1)>=1 ; dur(x2)>2 ; dur(x3)<3
                ; dur(x4)<=4 ; dur(x5)=5. \n"+
153         ":- goal goal6 dur(x1/\\true)>=1 ; dur(x2\\/false
                )>=2 ; dur(x2<->x3)>=3 ; dur(x3->x4)>=4.\n"+
154         ":- goal goal7 dur(x1\\/x2/\\x3)>=1 ; dur(~x1) >=
                1 ; dur((x1\\/x2)/\\x3) >=1.\n"+
155         ":- goal goal8 true \\/ false /\\ ~dur(x2)>=2 /\\
                (true \\/ false).";
156
157         tasks = parse(inputStr);
158         goal = (Goal) tasks.get(5);
159         Assert.assertEquals("Formula is parsed correctly
                ", "((((dur x1 >= 1 ; dur x2 >= 3) ; ~dur x3
                >= 3) ; ~dur x4 >= 5) ; (dur x5 >= 5 /\\ ~dur
                x5 >= 6))", goal.getFormula().toString());
160         goal = (Goal) tasks.get(6);
161         Assert.assertEquals("Formula is parsed correctly
                ", "(((dur (x1 /\\ true) >= 1 ; dur (x2 \\/
                false) >= 2) ; dur ((~x2 \\/ x3) /\\ (x2 \\/ ~
                x3)) >= 3) ; dur (~x3 \\/ x4) >= 4)", goal.
                getFormula().toString());
162         goal = (Goal) tasks.get(7);
163         Assert.assertEquals("Formula is parsed correctly
                ", "((dur (x1 \\/ (x2 /\\ x3)) >= 1 ; dur ~x1
                >= 1) ; dur ((x1 \\/ x2) /\\ x3) >= 1)", goal.
                getFormula().toString());
164         goal = (Goal) tasks.get(8);
165         Assert.assertEquals("Formula is parsed correctly
                ", "(true \\/ ((false /\\ ~dur x2 >= 2) /\\ (
                true \\/ false)))", goal.getFormula().toString
```

```
166
167         // l </<=/=/>=/> int
168         inputStr +=
169         ":- goal goal9 l <= 5.\n"+
170         ":- goal goal10 l < 5.\n"+
171         ":- goal goal11 l = 5.\n"+
172         ":- goal goal12 l >= 5.\n"+
173         ":- goal goal13 l > 5.\n";
174
175         tasks = parse(inputStr);
176         goal = (Goal) tasks.get(9);
177         Assert.assertEquals("Formula is parsed correctly
                ", "~dur true >= 6", goal.getFormula().
                toString());
178         goal = (Goal) tasks.get(10);
179         Assert.assertEquals("Formula is parsed correctly
                ", "~dur true >= 5", goal.getFormula().
                toString());
180         goal = (Goal) tasks.get(11);
181         Assert.assertEquals("Formula is parsed correctly
                ", "(dur true >= 5 /\\ ~dur true >= 6)", goal.
                getFormula().toString());
182         goal = (Goal) tasks.get(12);
183         Assert.assertEquals("Formula is parsed correctly
                ", "dur true >= 5", goal.getFormula().toString
                ());
184         goal = (Goal) tasks.get(13);
185         Assert.assertEquals("Formula is parsed correctly
                ", "dur true >= 6", goal.getFormula().toString
                ());
186
187         // all, evt
188         inputStr +=
189         ":- goal goal14 all dur x1 >= 1."+
190         ":- goal goal15 evt dur x2 >= 2.";
191
192         tasks = parse(inputStr);
193         goal = (Goal) tasks.get(14);
194         Assert.assertEquals("Formula is parsed correctly
                ", "~((true ; ~dur x1 >= 1) ; true)", goal.
                getFormula().toString());
195         goal = (Goal) tasks.get(15);
196         Assert.assertEquals("Formula is parsed correctly
                ", "((true ; dur x2 >= 2) ; true)", goal.
                getFormula().toString());
197
198     }
199
```

```
200     private String preprocess(String input) throws
            Exception {
201         StringReader reader = new StringReader(input);
202         BMCPreprocessor preprocessor = new
                BMCPreprocessor();
203         preprocessor.setScanner(new BMCLex(reader));
204         return preprocessor.preprocess();
205     }
206
207     public void testPreprocessor() throws Exception {
208         /* macros without parameters:
209          * - macro calls are replaced
210          * - macro definitions are left out from result
211          * - other statements are not changed
212          * - can be called using 'empty' parenthesis
213          * - doesnt work using more than zero parameters
214          * - \n is added to end of line
215          */
216         String input = ":- macro1 =^= is replaced."+
217         ":- goal macro macro1."+
218         ":- goal is not changed."+
219         ":- goal macro macro1().";
220         Assert.assertEquals(":- goal macro is replaced .\
                n:- goal is not changed .\n:- goal macro is
                replaced .\n",preprocess(input));
221         try {
222             input = ":- macro1 =^= is replaced."+
223             ":- goal macro macro1(test).";
224             preprocess(input);
225             Assert.assertTrue("Exception thrown", false);
226         }
227         catch (Exception e) {
228             Assert.assertTrue("Exception thrown", true);
229         }
230
231         /* macros with parenthesis but no parameters
232          * - macro calls are replaced
233          * - macro definitions are left out from result
234          * - other statements are not changed
235          * - can be called without parenthesis
236          * - doesnt work using more than zero parameters
237          * - \n is added to end of line
238          */
239         input = ":- macro1() =^= is replaced."+
240         ":- goal macro macro1."+
241         ":- goal is not changed."+
242         ":- goal macro macro1().";
243         Assert.assertEquals(":- goal macro is replaced .\
                n:- goal is not changed .\n:- goal macro is
                replaced .\n",preprocess(input));
```

```
244         try {
245             input = ":- macro1() =^= is replaced."+
246             ":- goal macro macro1(test).";
247             preprocess(input);
248             Assert.assertTrue("Exception thrown", false);
249         }
250         catch (Exception e) {
251             Assert.assertTrue("Exception thrown", true);
252         }
253 
254          /* macros with parenthesis and parameters
255           * - macro calls are replaced
256           * - macro definitions are left out from result
257           * - other statements are not changed
258           * - cannot be called without parameters
259           * - \n is added to end of line
260           */
261         input = ":- macro1(replaceVal) =^= is replaced
                with replaceVal."+
262         ":- goal macro macro1(this value)."+
263         ":- goal is not changed.";
264         Assert.assertEquals(":- goal macro is replaced
                with this value .\n:- goal is not changed .\n
                ",preprocess(input));
265         input = ":- macro1(replaceVal1,replaceVal2) =^=
                is replaced with replaceVal1 and replaceVal2
                ."+
266         ":- goal macro macro1(this value, another value)
                .";
267         Assert.assertEquals(":- goal macro is replaced
                with this value and another value .\n",
                preprocess(input));
268         try {
269             input = ":- macro1(replaceVal) =^= is not
                    replaced."+
270             ":- goal macro macro1().";
271             preprocess(input);
272             Assert.assertTrue("Exception thrown", false);
273         }
274         catch (Exception e) {
275             Assert.assertTrue("Exception thrown", true);
276         }
277         try {
278             input = ":- macro1(replaceVal) =^= is not
                    replaced."+
279             ":- goal macro macro1.";
280             preprocess(input);
281             Assert.assertTrue("Exception thrown", false);
282         }
283         catch (Exception e) {
```

```
284             Assert.assertTrue("Exception thrown", true);
285         }
286         try {
287             input = ":- macro1(replaceVal) =^= is not
                    replaced."+
288             ":- goal macro macro1(val1, val2).";
289             preprocess(input);
290             Assert.assertTrue("Exception thrown", false);
291         }
292         catch (Exception e) {
293             Assert.assertTrue("Exception thrown", true);
294         }
295 
296         /*  macros in macrodefs */
297         input = ":- innermacro =^= inner macro."+
298         ":- outermacro =^= outer macro uses innermacro."+
299         ":- goal goal1 Both macros are changed innermacro
                and outermacro.";
300         Assert.assertEquals(":- goal goal1 Both macros
                are changed inner macro and outer macro uses
                inner macro .\n",preprocess(input));
301 
302         /* keywords are not used as macros */
303         input = ":- set =^= cannot be a macro.";
304         Assert.assertEquals(":- set =^= cannot be a macro
                .\n",preprocess(input));
305 
306         /* macros can be used in set statements */
307         input = ":- val =^= true."+
308         ":- set option1 = val."+
309         ":- set option2 = val().";
310         Assert.assertEquals(":- set option1 = true .\n:-
                set option2 = true .\n",preprocess(input));
311 
312         // use of \" is
313         input = ":- macro1 =^= uses \" some value \"."+
314         ":- goal macro1 and it is all replaced.";
315         Assert.assertEquals(":- goal uses \" some value
                \" and it is all replaced .\n",preprocess(
                input));
316 
317         // testing cyclic definitions
318         try {
319             input = ":- macro1 =^= macro1 is not allowed
                    .";
320             preprocess(input);
321             Assert.assertTrue("Exception thrown", false);
322         }
323         catch (Exception e) {
324             Assert.assertTrue("Exception thrown", true);
```

```
325         }
326         try {
327             input = ":- macro1 =^= macro2 is okay."+
328             ":- macro2 =^= macro1 is not allowed because
                    of cyclic definition;";
329             preprocess(input);
330             Assert.assertTrue("Exception thrown", false);
331         }
332         catch (Exception e) {
333             Assert.assertTrue("Exception thrown", true);
334         }
335     }
336
337     public static junit.framework.Test suite() {
338         return new TestSuite(Test.class);
339     }
340 }
```

G.1.5 bmc.robdd.Test.java

```
1  package bmc.robdd;
2
3
4  import bmc.*;
5  import bmc.util.*;
6  import bmc.formula.*;
7  import junit.framework.*;
8  import java.util.*;
9
10 public class Test extends TestCase {
11
12     public Test(String name) {
13         super(name);
14     }
15
16     public void testROBDDUtil() throws
           CloneNotSupportedException {
17
18         // Test constructor
19         ROBDDUtil robddUtil = new ROBDDUtil();
20         Assert.assertNotNull(robddUtil.tableH);
21         Assert.assertNotNull(robddUtil.tableG);
22
23         /* mk() */
24         robddUtil.init();
25         int tableSize = robddUtil.tableT.size();
26         Assert.assertEquals("Returns h (or l, whatever)
               if h = l", 10, robddUtil.mk("a", 10, 10));
27         Assert.assertEquals("Returns id for new entry (=
               tableSize)", tableSize, robddUtil.mk("a", 10,
```

```
            20));
28          Assert.assertEquals("Returns that value once
               again", tableSize, robddUtil.mk("a", 10, 20));
29
30
31          /* replace() */
32          Assert.assertEquals("true",
33          robddUtil.replace(new Negation(new State("a")), "
               a", false).toString());
34          Assert.assertEquals("false",
35          robddUtil.replace(new Negation(new State("a")), "
               a", true).toString());
36          Assert.assertEquals("~b",
37          robddUtil.replace(new Negation(new State("b")), "
               a", true).toString());
38
39          Assert.assertEquals("false",
40          robddUtil.replace(new Conjunction(new State("a"),
                new State("b")), "a", false).toString());
41          Assert.assertEquals("true",
42          robddUtil.replace(new Conjunction(new State("a"),
                new True()), "a", true).toString());
43          Assert.assertEquals("b",
44          robddUtil.replace(new Conjunction(new State("a"),
                new State("b")), "a", true).toString());
45          Assert.assertEquals("a",
46          robddUtil.replace(new Conjunction(new State("a"),
                new State("b")), "b", true).toString());
47          Assert.assertEquals("(a /\\ b)",
48          robddUtil.replace(new Conjunction(new State("a"),
                new State("b")), "c", true).toString());
49
50          Assert.assertEquals("true",
51          robddUtil.replace(new Disjunction(new State("a"),
                new State("b")), "a", true).toString());
52          Assert.assertEquals("true",
53          robddUtil.replace(new Disjunction(new State("a"),
                new State("b")), "b", true).toString());
54          Assert.assertEquals("false",
55          robddUtil.replace(new Disjunction(new State("a"),
                new False()), "a", false).toString());
56          Assert.assertEquals("b",
57          robddUtil.replace(new Disjunction(new State("a"),
                new State("b")), "a", false).toString());
58          Assert.assertEquals("a",
59          robddUtil.replace(new Disjunction(new State("a"),
                new State("b")), "b", false).toString());
60          Assert.assertEquals("(b \\/ b)",
61          robddUtil.replace(new Disjunction(new Disjunction
               (new State("b"), new False()), new State("b"))
```

```
        , "a", true).toString());
62
63          try {
64              robddUtil.replace(new Chop(new True(), new
                    False()),"a", true);
65              Assert.assertTrue(false);
66          }
67          catch (Exception e) {
68              Assert.assertTrue(true);
69          }
70
71          // init()
72          NodeDesc falseDesc = new NodeDesc(False.FALSE_STR
                , -1, -1);
73          NodeDesc trueDesc = new NodeDesc(True.TRUE_STR,
                -1, -1);
74          robddUtil.init();
75          Assert.assertNotNull(robddUtil.tableT);
76          Assert.assertTrue(robddUtil.tableH.isEmpty());
77          Assert.assertTrue(robddUtil.tableG.isEmpty());
78          Assert.assertTrue(robddUtil.tableT.size() == 2);
79          Assert.assertEquals(robddUtil.tableT.get(0),
                falseDesc);
80          Assert.assertEquals(robddUtil.tableT.get(1),
                trueDesc);
81
82          // build()
83          LinkedList states = new LinkedList();
84          Formula f1 = new Conjunction(new True(), new
                False());
85          Assert.assertEquals("", 0, robddUtil.build(f1,
                states, 0));
86          robddUtil.init();
87          f1 = new True();
88          Assert.assertEquals("", 1, robddUtil.build(f1,
                states, 0));
89          // the rest of build() is tested in the following
                 tests of makeROBDD (where noStates > 0)
90
91
92
93          ROBDD robdd1a = robddUtil.makeROBDD(new
                Conjunction(new State("x1"), new State("x2")))
                ;
94          ArrayList tableT = new ArrayList();
95          tableT.add(falseDesc);
96          tableT.add(trueDesc);
97          tableT.add(new NodeDesc("x2", 0, 1));
98          tableT.add(new NodeDesc("x1", 0, 2));
99          ROBDD robdd1b = new ROBDD(tableT, 3);
```

```
100         Assert.assertEquals("ROBDD construction",robdd1a,
                robdd1b);
101
102         ROBDD robdd2a = robddUtil.makeROBDD(new
                Disjunction(new State("x1"), new State("x2")))
                ;
103         tableT = new ArrayList();
104         tableT.add(falseDesc);
105         tableT.add(trueDesc);
106         tableT.add(new NodeDesc("x2", 0, 1));
107         tableT.add(new NodeDesc("x1", 2, 1));
108         ROBDD robdd2b = new ROBDD(tableT, 3);
109         Assert.assertEquals("ROBDD construction",robdd2a,
                robdd2b);
110
111         ROBDD robdd3a = robddUtil.makeROBDD(new Negation(
                new State("x1")));
112         tableT = new ArrayList();
113         tableT.add(falseDesc);
114         tableT.add(trueDesc);
115         tableT.add(new NodeDesc("x1", 1, 0));
116         ROBDD robdd3b = new ROBDD(tableT, 2);
117         Assert.assertEquals("ROBDD construction",robdd3a,
                robdd3b);
118
119         ROBDD robdd4a = robddUtil.makeROBDD(new State("x1
                "));
120         tableT = new ArrayList();
121         tableT.add(falseDesc);
122         tableT.add(trueDesc);
123         tableT.add(new NodeDesc("x1", 0, 1));
124         ROBDD robdd4b = new ROBDD(tableT, 2);
125         Assert.assertEquals("ROBDD construction",robdd4a,
                robdd4b);
126
127         ROBDD robdd5a = robddUtil.makeROBDD(new True());
128         tableT = new ArrayList();
129         tableT.add(falseDesc);
130         tableT.add(trueDesc);
131         ROBDD robdd5b = new ROBDD(tableT, 1);
132         Assert.assertEquals("ROBDD construction",robdd5a,
                robdd5b);
133
134         ROBDD robdd6a = robddUtil.makeROBDD(new False());
135         tableT = new ArrayList();
136         tableT.add(falseDesc);
137         tableT.add(trueDesc);
138         ROBDD robdd6b = new ROBDD(tableT, 0);
139         Assert.assertEquals("ROBDD construction",robdd6a,
                robdd6b);
```

```
140
141
142         Formula formula1 = new Conjunction(new
                Conjunction(new State("x2"), new True()), new
                State("x1"));
143         Formula formula2 = new Disjunction(new Negation(
                new State("x1")), new False());
144
145         // testing makeROBDD
146         ROBDD robdd1 = robddUtil.makeROBDD(formula1);
147         ROBDD robdd2 = robddUtil.makeROBDD(new
                Conjunction(new State("x1"), new State("x2"))
                );
148         Assert.assertEquals("ROBDDs that represent
                equivalent formulas are the same", robdd1,
                robdd2);
149
150         ROBDD robdd3 = robddUtil.makeROBDD(formula2);
151         ROBDD robdd4 = robddUtil.makeROBDD(new Negation(
                new Conjunction(new State("x1"), new True()))
                );
152         Assert.assertEquals("ROBDDs that represent
                equivalent formulas are the same", robdd3,
                robdd4);
153         Assert.assertFalse("ROBDDs that represent
                different formulas are not the same", robdd3.
                equals(robdd1));
154
155         Formula formula3 = new Disjunction(new State("b")
                , new False());
156         Formula formula4 = new State("c");
157         ROBDD robdd6 = robddUtil.makeROBDD(formula3);
158         ROBDD robdd7 = robddUtil.makeROBDD(formula4);
159
160         // apply()
161         ROBDD robdd8 = robddUtil.makeROBDD(new
                Conjunction(formula3, formula4));
162         ROBDD robdd9 = robddUtil.apply(robdd6, robdd7,
                robddUtil.OP_CONJUNCTION);
163         Assert.assertEquals("apply with conjunction",
                robdd8, robdd9);
164
165         ROBDD robdd10 = robddUtil.makeROBDD(new
                Disjunction(formula3, formula4));
166         ROBDD robdd11 = robddUtil.apply(robdd6, robdd7,
                robddUtil.OP_DISJUNCTION);
167         Assert.assertEquals("apply with disjunction",
                robdd10, robdd11);
168
```

```
169         ROBDD robdd12 = robddUtil.makeROBDD(new
                Disjunction(new Negation(formula3), formula4))
                ;
170         ROBDD robdd13 = robddUtil.apply(robdd6, robdd7,
                robddUtil.OP_IMPLICATION);
171         Assert.assertEquals("apply with implication",
                robdd12, robdd13);
172
173         Assert.assertEquals("apply of two constant robbds
                ", robddUtil.makeROBDD(new Conjunction(new
                True(), new False())),
174         robddUtil.apply(robddUtil.makeROBDD(new True()),
175         robddUtil.makeROBDD(new False()),
176         ROBDDUtil.OP_CONJUNCTION));
177
178         Assert.assertEquals("left robbd is constant",
                robddUtil.makeROBDD(new Conjunction(new State
                ("x1"), new False())),
179         robddUtil.apply(robddUtil.makeROBDD(new State("x1
                ")),
180         robddUtil.makeROBDD(new False()),
181         ROBDDUtil.OP_CONJUNCTION));
182
183         Assert.assertEquals("right robbd is constant",
                robddUtil.makeROBDD(new Conjunction(new State
                ("x1"), new False())),
184         robddUtil.apply(robddUtil.makeROBDD(new False()),
185         robddUtil.makeROBDD(new State("x1")),
186         ROBDDUtil.OP_CONJUNCTION));
187
188         Assert.assertEquals("neither of the robbds are
                constant. Left robdd is ordered below right
                one", robddUtil.makeROBDD(new Conjunction(new
                State("x1"), new State("x2"))),
189         robddUtil.apply(robddUtil.makeROBDD(new State("x1
                ")),
190         robddUtil.makeROBDD(new State("x2")),
191         ROBDDUtil.OP_CONJUNCTION));
192         Assert.assertEquals("neither of the robbds are
                constant. Right robdd is ordered below left
                one", robddUtil.makeROBDD(new Conjunction(new
                State("x1"), new State("x2"))),
193         robddUtil.apply(robddUtil.makeROBDD(new State("x2
                ")),
194         robddUtil.makeROBDD(new State("x1")),
195         ROBDDUtil.OP_CONJUNCTION));
196         Assert.assertEquals("neither of the robbds are
                constant. Both are ordered equally", robddUtil
                .makeROBDD(new Conjunction(new State("x1"),
                new State("x1"))),
```

```
197         robddUtil.apply(robddUtil.makeROBDD(new State("x1
                ")),
198         robddUtil.makeROBDD(new State("x1")),
199         ROBDDUtil.OP_CONJUNCTION));
200
201     }
202
203     public void testROBDD() {
204         ArrayList tableT1 = new ArrayList();
205         tableT1.add(new NodeDesc(False.FALSE_STR, -1, -1)
                );
206         tableT1.add(new NodeDesc(True.TRUE_STR, -1, -1));
207         // robdd true/false
208         ROBDD robddFalse = new ROBDD((ArrayList) tableT1.
                clone(), 0);
209         ROBDD robddTrue = new ROBDD((ArrayList) tableT1.
                clone(), 1);
210
211         // robdd for x1
212         tableT1.add(new NodeDesc("x1", 1, 0));
213         ROBDD robdd1 = new ROBDD(tableT1, 2);
214
215
216
217         // test constructor
218         Assert.assertSame("Fields set correctly", tableT1
                , robdd1.tableT);
219         Assert.assertEquals("Fields set correctly", 2,
                robdd1.root);
220
221         // test getRoot(), getTableT()
222         Assert.assertEquals("robdd1.getRoot() is 2", 2,
                robdd1.getRoot());
223         Assert.assertSame("getTableT()", tableT1, robdd1.
                tableT);
224
225         // test isTrue(), isFalse()
226         Assert.assertFalse("robdd1 is not always true",
                robdd1.isTrue());
227         Assert.assertFalse("robdd1 is not always false",
                robdd1.isFalse());
228         Assert.assertTrue("robddFalse is always false",
                robddFalse.isFalse());
229         Assert.assertTrue("robddTrue is always true",
                robddTrue.isTrue());
230
231         // copy of robdd1
232         ArrayList tableT1Copy = new ArrayList();
233         tableT1Copy.add(new NodeDesc(False.FALSE_STR, -1,
                -1));
```

```
234         tableT1Copy.add(new NodeDesc(True.TRUE_STR, -1,
                -1));
235         tableT1Copy.add(new NodeDesc("x1", 1, 0));
236         ROBDD robdd1Copy = new ROBDD(tableT1Copy, 2);
237
238         // equals()
239         Assert.assertFalse("robdd isnt equal to some
                other object", robdd1.equals(new True()));
240         Assert.assertEquals("robdd1 equals a similar
                robdd", robdd1, robdd1Copy);
241         Assert.assertFalse("robddTrue does not equal
                robddFalse, tables equal, root not", robddTrue
                .equals(robddFalse));
242
243         tableT1 = new ArrayList();
244         tableT1.add(new NodeDesc(False.FALSE_STR, -1, -1)
                );
245         tableT1.add(new NodeDesc(True.TRUE_STR, -1, -1));
246         tableT1.add(new NodeDesc("x1", 0, 1));
247         ROBDD robdd2 = new ROBDD(tableT1, 2);
248         Assert.assertFalse("robdd isnt equal to robdd2 (
                same root, but not same tableT)", robdd1.
                equals(robdd2));
249
250         // test negate (and that the original robdd hasnt
                 been changed)
251         Assert.assertEquals("root is 0", 0, robddTrue.
                negate().getRoot());
252         Assert.assertEquals("root is 1", 1, robddFalse.
                negate().getRoot());
253
254         Assert.assertEquals("robdd1.negate() is equal to
                robdd2", robdd2, robdd1.negate());
255         Assert.assertEquals("robdd2.negate() is equal to
                robdd1", robdd1, robdd2.negate());
256
257         tableT1 = new ArrayList();
258         tableT1.add(new NodeDesc(False.FALSE_STR, -1, -1)
                );
259         tableT1.add(new NodeDesc(True.TRUE_STR, -1, -1));
260         tableT1.add(new NodeDesc("x1", 0, 1));
261         NodeDesc x2ND = new NodeDesc("x2", 2, 2);
262         tableT1.add(x2ND);
263         ROBDD robdd3 = new ROBDD(tableT1, 3);
264         NodeDesc negatedX2ND = (NodeDesc) robdd3.negate()
                .getTableT().get(3);
265         Assert.assertEquals("Even though robdd3 has been
                negated, node desc content has not changed",
                x2ND, negatedX2ND);
```

```
266         Assert.assertNotSame("But node desc is not the
                same object", x2ND, negatedX2ND);
267
268         // toString()
269         Assert.assertEquals("root: 3 tableT: "+tableT1.
                toString(),robdd3.toString());
270     }
271
272     public void testNodePair() {
273         NodePair n1 = new NodePair(3, 5);
274         NodePair n2 = new NodePair(1, 5);
275         NodePair n3 = new NodePair(3, 6);
276         NodePair n11 = new NodePair(3, 5);
277
278         // test constructor
279         Assert.assertEquals("Constructor sets fields
                correctly", 3, n1.u1);
280         Assert.assertEquals("Constructor sets fields
                correctly", 5, n1.u2);
281
282         // test equals()
283         Assert.assertEquals("A node desc equals itself",
                n1, n1);
284         Assert.assertFalse("n1 does not equal n2 -- u1
                wrong", n1.equals(n2));
285         Assert.assertFalse("n1 does not equal n3 -- u2
                wrong", n1.equals(n3));
286         Assert.assertEquals("n1 equals n11", n1, n11);
287         Assert.assertFalse("Other kind of object does not
                 equal node pair", n1.equals(new False()));
288
289         // test get methods
290         Assert.assertEquals("Get methods return field
                values correctly", 3, n1.getU1());
291         Assert.assertEquals("Get methods return field
                values correctly", 5, n1.getU2());
292
293         // hash code
294         Assert.assertEquals("Equals objects return equal
                hash codes", n1.hashCode(), n11.hashCode());
295     }
296
297     public void testNodeDesc() {
298         NodeDesc n1 = new NodeDesc("desc1", 3, 5);
299         NodeDesc n2 = new NodeDesc("desc1", 1, 5);
300         NodeDesc n3 = new NodeDesc("desc1", 3, 6);
301         NodeDesc n4 = new NodeDesc("desc2", 3, 5);
302         NodeDesc n11 = new NodeDesc("desc1", 3, 5);
303
304         // test constructor
```

```
305         Assert.assertEquals("Constructor sets fields
                correctly", "desc1", n1.varID);
306         Assert.assertEquals("Constructor sets fields
                correctly", 3, n1.low);
307         Assert.assertEquals("Constructor sets fields
                correctly", 5, n1.high);
308
309         // test equals()
310         Assert.assertEquals("A node desc equals itself",
                n1, n1);
311         Assert.assertFalse("n1 does not equal n2 -- low
                wrong", n1.equals(n2));
312         Assert.assertFalse("n1 does not equal n3 -- high
                wrong", n1.equals(n3));
313         Assert.assertFalse("n1 does not equal n4 -- varID
                 wrong", n1.equals(n4));
314         Assert.assertEquals("n1 equals n11", n1, n11);
315         Assert.assertFalse("Other kind of object does not
                 equal node pair", n1.equals(new False()));
316
317         // test get methods
318         Assert.assertEquals("Get methods return field
                values correctly", "desc1", n1.getVarID());
319         Assert.assertEquals("Get methods return field
                values correctly", 3, n1.getLow());
320         Assert.assertEquals("Get methods return field
                values correctly", 5, n1.getHigh());
321     }
322
323
324
325     public static junit.framework.Test suite() {
326         return new TestSuite(Test.class);
327     }
328
329 }
```

G.1.6 bmc.simp.Test.java

```
1  package bmc.simp;
2
3  import bmc.*;
4  import bmc.util.*;
5  import bmc.formula.*;
6  import bmc.robdd.*;
7  import junit.framework.*;
8  import java.util.*;
9
10 public class Test extends TestCase {
11
```

```
12      private static int TEST_K_VALUE = 10;
13
14      public Test(String name) {
15          super(name);
16      }
17
18      public void performLevelNoneTest(Settings settings)
            throws Exception {
19          settings.setDCSimpLevel(Settings.
                DC_SIMP_LEVEL_NONE);
20          DCSimplifier dcSimplifier = new DCSimplifier(
                settings);
21          /* testing that simplifyDC() and simplifySA()
                doesnt apply testing when LEVEL_NONE is
                applied */
22          Negation fNeg = new Negation(new False());
23          testNonSimplifiedDCFormula(dcSimplifier, fNeg);
24          testNonSimplifiedDCFormula(dcSimplifier, fNeg);
25      }
26
27      public void performSimplifyTreeTest(Settings settings
            ) throws Exception {
28          settings.setDCSimpLevel(Settings.
                DC_SIMP_LEVEL_WITH_BDD);
29          DCSimplifier dcSimplifierWithBdd = new
                DCSimplifier(settings);
30          settings.setDCSimpLevel(Settings.
                DC_SIMP_LEVEL_BASIC);
31          DCSimplifier dcSimplifierBasic = new DCSimplifier
                (settings);
32
33          // duration
34          testUnsupportedSAFormula(dcSimplifierBasic, new
                Duration(new True(), TEST_K_VALUE-1));
35          testNonSimplifiedDCFormula(dcSimplifierBasic, new
                 Duration(new State("x1"), TEST_K_VALUE-1));
36          testSimplifiedDCFormula(dcSimplifierBasic, new
                Duration(new True(), TEST_K_VALUE+1), "false")
                ;
37
38          // state
39          testNonSimplifiedSAFormula(dcSimplifierBasic, new
                 State("x1"));
40          testUnsupportedDCFormula(dcSimplifierBasic, new
                State("x1"));
41
42          // chop
43          testNonSimplifiedDCFormula(dcSimplifierBasic, new
                 Chop(new Duration(new State("x1"),
               TEST_K_VALUE-1), new Duration(new State("x1"),
```

```
        TEST_K_VALUE-1)));
44      testSimplifiedDCFormula(dcSimplifierBasic, new
        Chop(new Conjunction(new False(), new False())
        ,new True()), "false");
45      testSimplifiedDCFormula(dcSimplifierBasic, new
        Chop(new True(),new Conjunction(new False(),
        new False())),"false");
46      testUnsupportedSAFormula(dcSimplifierBasic, new
        Chop(new True(), new False()));
47
48      // conjunction
49      testNonSimplifiedDCFormula(dcSimplifierBasic,
50      new Conjunction(
51      new Duration(new State("x1"), TEST_K_VALUE-1),
52      new Duration(new State("x1"), TEST_K_VALUE-1)
53      ));
54      testSimplifiedDCFormula(dcSimplifierBasic,
55      new Conjunction(new Duration(new State("x1"),
        TEST_K_VALUE-1), new Duration(new State("x1"),
        TEST_K_VALUE+1)),"false");
56      testSimplifiedDCFormula(dcSimplifierBasic,
57      new Conjunction(new Duration(new State("x1"),
        TEST_K_VALUE+1), new Duration(new State("x1"),
        TEST_K_VALUE-1)),"false");
58      testSimplifiedDCFormula(dcSimplifierBasic,
59      new Conjunction(
60      new Conjunction(
61      new Duration(new State("x1"), TEST_K_VALUE-1),
62      new Duration(new State("x1"), TEST_K_VALUE+1)
63      ),
64      new True()
65      ),"false");
66
67      Formula a = new Conjunction(
68      new Duration(new State("x1"), TEST_K_VALUE-1),
69      new Duration(new State("x1"), TEST_K_VALUE-1)
70      );
71      testSimplifiedDCFormula(dcSimplifierWithBdd,
72      new Conjunction(
73      new Duration(new State("x1"), TEST_K_VALUE-1),
74      new Duration(new State("x1"), TEST_K_VALUE-1)
75      ),"dur x1 >= "+(TEST_K_VALUE-1));
76
77      // disjunction
78      testSimplifiedDCFormula(dcSimplifierBasic,
79      new Disjunction(
80      new Duration(new State("x1"), TEST_K_VALUE+1),
81      new Duration(new State("x1"), TEST_K_VALUE-1)),
82      "dur x1 >= "+(TEST_K_VALUE-1));
83      testSimplifiedDCFormula(dcSimplifierBasic,
```

```
84          new Disjunction(new Duration(new State("x1"),
                TEST_K_VALUE-1),
85          new Duration(new State("x1"), TEST_K_VALUE+1)),
86          "dur x1 >= "+(TEST_K_VALUE-1));
87          testNonSimplifiedDCFormula(dcSimplifierBasic,
88          new Disjunction(
89          new Duration(new State("x1"), TEST_K_VALUE-1),
90          new Duration(new State("x1"), TEST_K_VALUE-1)));
91          testSimplifiedDCFormula(dcSimplifierBasic,
92          new Disjunction(
93          new Disjunction(
94          new Duration(new State("x1"), TEST_K_VALUE-1),
95          new Duration(new State("x1"), TEST_K_VALUE+1)
96          ),
97          new False()
98          ),"dur x1 >= "+(TEST_K_VALUE-1));
99          testSimplifiedDCFormula(dcSimplifierWithBdd,
100         new Disjunction(
101         new Duration(new State("x1"), TEST_K_VALUE-1),
102         new Duration(new State("x1"), TEST_K_VALUE-1)
103         ),"dur x1 >= "+(TEST_K_VALUE-1));
104
105         // negation
106         testNonSimplifiedDCFormula(dcSimplifierBasic,
107         new Negation(new Duration(new State("x1"),
                TEST_K_VALUE-1)));
108         testSimplifiedDCFormula(dcSimplifierBasic, new
                Negation(new Duration(new State("x1"),
                TEST_K_VALUE+1)), "true");
109
110         // true
111         True fTrue = new True();
112         testNonSimplifiedDCFormula(dcSimplifierBasic,
                fTrue);
113         testNonSimplifiedSAFormula(dcSimplifierBasic,
                fTrue);
114
115         // false
116         False fFalse = new False();
117         testNonSimplifiedDCFormula(dcSimplifierBasic,
                fFalse);
118         testNonSimplifiedSAFormula(dcSimplifierBasic,
                fFalse);
119     }
120
121
122     public void performBasicTest(Settings settings)
            throws Exception {
123         settings.setDCSimpLevel(Settings.
                DC_SIMP_LEVEL_BASIC);
```

```
124         DCSimplifier dcSimplifier = new DCSimplifier(
                settings);
125         Formula fTrue = new True();
126         Formula fFalse = new False();
127         Formula fState = new State("x1");
128
129         // DURATION:
130         Duration fDuration = new Duration(new False(), 5)
                ;
131         // dur false >= n -> false, dur phi > k -> false,
                 [recursive on children]
132         testSimplifiedDCFormula(dcSimplifier,
133         new Duration(new Conjunction(fTrue, fFalse),
                TEST_K_VALUE-1), fFalse.toString());
134         testSimplifiedDCFormula(dcSimplifier, new
                Duration(fTrue, TEST_K_VALUE+1), fFalse.
                toString());
135
136         fDuration = new Duration(fTrue, 5);
137         testSimplifiedDCFormula(dcSimplifier,
138         new Duration(new Disjunction(fState, fTrue), 5),
                fDuration.toString());
139
140
141         // CONJUNCTION: left and right children is true/
                false;
142         // true /\ phi -> phi, phi /\ true -> phi
143         testSimplifiedDCFormula(dcSimplifier, new
                Conjunction(fDuration, fTrue), fDuration.
                toString());
144         testSimplifiedDCFormula(dcSimplifier, new
                Conjunction(fTrue, fDuration), fDuration.
                toString());
145
146         // false /\ phi -> false, phi /\ false -> false
147         testSimplifiedDCFormula(dcSimplifier, new
                Conjunction(fDuration, fFalse), fFalse.
                toString());
148         testSimplifiedDCFormula(dcSimplifier, new
                Conjunction(fFalse, fDuration), fFalse.
                toString());
149
150         // phi /\ psi -> simp(phi) /\ simp(psi)
151         testSimplifiedDCFormula(dcSimplifier,
152         new Conjunction(
153         new Duration(new Conjunction(new True(), new True
                ()), 2),
154         new Duration(new Conjunction(new True(), new True
                ()), 2)
155         ),
```

```
156         new Conjunction(
157         new Duration(new True(), 2),
158         new Duration(new True(), 2)).toString());
159
160         testSimplifiedDCFormula(dcSimplifier,
161         new Conjunction(
162         new Duration(new True(), 2),
163         new Duration(new True(), 2)
164         ),
165         new Conjunction(
166         new Duration(new True(), 2),
167         new Duration(new True(), 2)).toString());
168
169
170         /* phi \/ ((true /\ false) /\ false) ->
171          *       phi \/ (false /\ false) ->
172          *       phi \/ false -> phi
173          * [advanced parent-updates: crossing different
                operators]
174          */
175         testSimplifiedDCFormula(dcSimplifier,
176         new Disjunction(
177         fDuration,
178         new Conjunction(
179         new Conjunction(new True(), new False()),
180         fFalse)
181         ),
182         fDuration.toString());
183
184
185         // DISJUNCTION: unsupported operands, left and
               right children is true/false;
186         Disjunction fDisjunction = new Disjunction(fTrue,
               fState);
187         // true \/ phi -> true, phi \/ true -> true
188         testSimplifiedDCFormula(dcSimplifier, new
               Disjunction(fDuration, fTrue), fTrue.toString
               ());
189         testSimplifiedDCFormula(dcSimplifier, new
               Disjunction(fTrue, fDuration), fTrue.toString
               ());
190
191         // false \/ phi -> phi, phi \/ false -> phi
192         testSimplifiedDCFormula(dcSimplifier, new
               Disjunction(fDuration, fFalse), fDuration.
               toString());
193         testSimplifiedDCFormula(dcSimplifier, new
               Disjunction(fFalse, fDuration), fDuration.
               toString());
194
```

```
195         // phi /\ psi -> simp(phi) /\ simp(psi)
196         testSimplifiedDCFormula(dcSimplifier,
197         new Disjunction(
198         new Duration(new Conjunction(new True(), new True
                ()), 2),
199         new Duration(new Conjunction(new True(), new True
                ()), 2)
200         ),
201         new Disjunction(
202         new Duration(new True(), 2),
203         new Duration(new True(), 2)).toString());
204
205         testSimplifiedDCFormula(dcSimplifier,
206         new Disjunction(
207         new Duration(new True(), 2),
208         new Duration(new True(), 2)
209         ),
210         new Disjunction(
211         new Duration(new True(), 2),
212         new Duration(new True(), 2)).toString());
213
214         // 2) phi /\ (true /\ true \/ false) -> phi /\
                true -> phi  [advanced parent-updates:
                crossing different operators]
215         testSimplifiedDCFormula(dcSimplifier,
216         new Conjunction(
217         fDuration,
218         new Disjunction(
219         new Conjunction(new True(), new True()),
220         fFalse)
221         ),
222         fDuration.toString());
223
224         // CHOP: false;phi -> false, phi;false -> false,
225         Chop fChop = new Chop(fTrue, fFalse);
226
227         testSimplifiedDCFormula(dcSimplifier, new Chop(
                fDuration, fFalse), "false");
228         testSimplifiedDCFormula(dcSimplifier, new Chop(
                fFalse, fDuration), "false");
229         // phi ; psi -> simp(phi) ; simp(psi) [recursive
                on children]
230         testSimplifiedDCFormula(dcSimplifier, new Chop(
                new Negation(new False()), new Negation(new
                False())), "(true ; true)");
231         testSimplifiedDCFormula(dcSimplifier, new Chop(
                new True(), new True()), "(true ; true)");
232
233
```

```
234         // NEGATION: unsupported operands, false -> true,
                true -> false,
235         testSimplifiedDCFormula(dcSimplifier, new
                Negation(fFalse), "true");
236
237         testSimplifiedDCFormula(dcSimplifier, new
                Negation(fTrue), "false");
238
239         // ~~phi -> phi
240         testSimplifiedDCFormula(dcSimplifier,
241         new Negation(new Negation(fDuration)),
242         fDuration.toString());
243
244         // ~phi -> ~(simp(phi)) [recursive on children]
245         testSimplifiedDCFormula(dcSimplifier,
246         new Negation(
247         new Duration(
248         new Conjunction(new True(), new True()),
249         2)),
250         "~dur true >= 2");
251
252         testSimplifiedDCFormula(dcSimplifier,
253         new Negation(new Duration(new Conjunction(new
                True(), new True()),2)),
254         "~dur true >= 2");
255     }
256
257     private void performWithBDDTest(Settings settings)
            throws Exception {
258         settings.setDCSimpLevel(Settings.
                DC_SIMP_LEVEL_WITH_BDD);
259         DCSimplifier dcSimplifier = new DCSimplifier(
                settings);
260         Formula fState = new State("x1");
261         Duration fDuration = new Duration(fState, 7);
262         Conjunction fConjunction = new Conjunction(new
                Duration(fState, 5), fDuration);
263
264         // test SA
265
266         Formula fState1 = new State("x1");
267         Formula fState2 = new State("x2");
268         Formula fS1 = new Conjunction((State) fState1.
                clone(), (State) fState2.clone());
269         Formula fFalse = new False();
270         Formula fTrue = new True();
271
272         // SA: only one state. Handles the fact that
                sequence comparison makes no sense with only
                one state
```

```
273         testSimplifiedSAFormula(dcSimplifier, new
                Conjunction((State) fState1.clone(), new True
                ()), fState1.toString());
274
275         // SA: s1 /\ s2 -> s1, if s1 => s2
276         testSimplifiedSAFormula(dcSimplifier,
277         new Conjunction((Conjunction) fS1.clone(), (State
                ) fState2.clone()),
278         fS1.toString());
279         testSimplifiedSAFormula(dcSimplifier,
280         new Conjunction((State) fState2.clone(), (
                Conjunction) fS1.clone()),
281         fS1.toString());
282
283         // SA: s1 /\ ~s2 -> false, if s1 => s2
284         testSimplifiedSAFormula(dcSimplifier,
285         new Conjunction((Conjunction) fS1.clone(), new
                Negation((State) fState2.clone())),
286         fFalse.toString());
287
288         testSimplifiedSAFormula(dcSimplifier,
289         new Conjunction(new Negation((State) fState2.
                clone()), (Conjunction) fS1.clone()),
290         fFalse.toString());
291
292         // SA: s1 \/ s2 -> s2, if s1 => s2
293         testSimplifiedSAFormula(dcSimplifier,
294         new Disjunction((Conjunction) fS1.clone(), (State
                ) fState2.clone()),
295         fState2.toString());
296         testSimplifiedSAFormula(dcSimplifier,
297         new Disjunction((State) fState2.clone(), (
                Conjunction) fS1.clone()),
298         fState2.toString());
299
300         // SA: s1 \/ s2 -> true, if ~s1 => s2
301         testSimplifiedSAFormula(dcSimplifier,
302         new Disjunction((State) fState2.clone(), new
                Negation((Conjunction) fS1.clone())),
303         fTrue.toString());
304         testSimplifiedSAFormula(dcSimplifier,
305         new Disjunction(new Negation((Conjunction) fS1.
                clone()), (State) fState2.clone()),
306         fTrue.toString());
307         // SA: s1 /\ s2 -> false, if ~s1 => s2
308         testSimplifiedSAFormula(dcSimplifier,
309         new Conjunction(new Negation((State) fState2.
                clone()), (Conjunction) fS1.clone()),
310         fFalse.toString());
311         testSimplifiedSAFormula(dcSimplifier,
```

```
312         new Conjunction((Conjunction) fS1.clone(), new
                Negation((State) fState2.clone())),
313         fFalse.toString());
314
315
316         Duration dur1 = new Duration(fS1, 10);
317         Duration dur2 = new Duration(fState2, 5);
318         Duration dur3 = new Duration(fState1, 5);
319
320         // DC: ~dur(s1)>=n1 /\ ~dur(s2)>=n2 -> ~dur(s2)>=
                n2, s1=>s2
321         testSimplifiedDCFormula(dcSimplifier,
322         new Conjunction(
323         new Negation((Duration) dur1.clone()),
324         new Negation((Duration) dur2.clone())),
325         new Negation((Duration) dur2.clone()).toString())
                ;
326         testSimplifiedDCFormula(dcSimplifier,
327         new Conjunction(
328         new Negation((Duration) dur2.clone()),
329         new Negation((Duration) dur1.clone())),
330         new Negation((Duration) dur2.clone()).toString())
                ;
331
332         // DC: ~dur(s1)>=n1 /\ ~dur(s2)>=n2, no change if
                 s1=/>s2
333         testSimplifiedDCFormula(dcSimplifier,
334         new Conjunction(
335         new Negation((Duration) dur2.clone()),
336         new Negation((Duration) dur3.clone())),
337         new Conjunction(
338         new Negation((Duration) dur2.clone()),
339         new Negation((Duration) dur3.clone())).toString()
                );
340         testSimplifiedDCFormula(dcSimplifier,
341         new Conjunction(
342         new Negation((Duration) dur3.clone()),
343         new Negation((Duration) dur2.clone())),
344         new Conjunction(
345         new Negation((Duration) dur3.clone()),
346         new Negation((Duration) dur2.clone())).toString()
                );
347
348         // DC: dur(s1)>=n1 /\ dur(s2)>=n2 -> dur(s1)>=n1,
                 s1=>s2
349         testSimplifiedDCFormula(dcSimplifier,
350         new Conjunction((Duration) dur1.clone(), (
                Duration) dur2.clone()),
351         dur1.toString());
352         testSimplifiedDCFormula(dcSimplifier,
```

```
353         new Conjunction((Duration) dur2.clone(), (
                Duration) dur1.clone()),
354         dur1.toString());
355         // DC: dur(s1)>=n1 /\ dur(s2)>=n2 -> no change,
                s1=/>s2
356         testSimplifiedDCFormula(dcSimplifier,
357         new Conjunction((Duration) dur2.clone(), (
                Duration) dur3.clone()),
358         new Conjunction((Duration) dur2.clone(), (
                Duration) dur3.clone()).toString());
359         testSimplifiedDCFormula(dcSimplifier,
360         new Conjunction((Duration) dur2.clone(), (
                Duration) dur3.clone()),
361         new Conjunction((Duration) dur2.clone(), (
                Duration) dur3.clone()).toString());
362
363         // DC: dur(s1)>=n1 /\ ~dur(s2)>=n2 -> false, if
                s1=>s2
364         testSimplifiedDCFormula(dcSimplifier,
365         new Conjunction((Duration) dur1.clone(), new
                Negation((Duration) dur2.clone())),
366         fFalse.toString());
367         testSimplifiedDCFormula(dcSimplifier,
368         new Conjunction(new Negation((Duration) dur2.
                clone()), (Duration) dur1.clone()),
369         fFalse.toString());
370         // DC: dur(s1)>=n1 /\ ~dur(s2)>=n2 -> no change,
                if s1=/>s2
371         testSimplifiedDCFormula(dcSimplifier,
372         new Conjunction((Duration) dur2.clone(), new
                Negation((Duration) dur3.clone())),
373         new Conjunction((Duration) dur2.clone(), new
                Negation((Duration) dur3.clone())).toString())
                ;
374         testSimplifiedDCFormula(dcSimplifier,
375         new Conjunction(new Negation((Duration) dur3.
                clone()), (Duration) dur2.clone()),
376         new Conjunction(new Negation((Duration) dur3.
                clone()), (Duration) dur2.clone()).toString())
                ;
377
378
379         // DC: dur(s1)>=n1 \/ dur(s2)>=n2 -> dur(n2)>=n2,
                 s1=>s2
380         testSimplifiedDCFormula(dcSimplifier,
381         new Disjunction(
382         (Duration) dur1.clone(),
383         (Duration) dur2.clone()),
384         ((Duration) dur2.clone()).toString());
385         testSimplifiedDCFormula(dcSimplifier,
```

```
386         new Disjunction(
387         (Duration) dur2.clone(),
388         (Duration) dur1.clone()),
389         ((Duration) dur2.clone()).toString());
390         // DC: dur(s1)>=n1 \/ dur(s2)>=n2 -> no change,
                s1=/>s2
391         testSimplifiedDCFormula(dcSimplifier,
392         new Disjunction(
393         (Duration) dur2.clone(),
394         (Duration) dur3.clone()),
395         new Disjunction(
396         (Duration) dur2.clone(),
397         (Duration) dur3.clone()).toString());
398         testSimplifiedDCFormula(dcSimplifier,
399         new Disjunction(
400         (Duration) dur2.clone(),
401         (Duration) dur3.clone()),
402         new Disjunction(
403         (Duration) dur2.clone(),
404         (Duration) dur3.clone()).toString());
405
406         // DC: ~dur(s1)>=n1 \/ ~dur(s2)>=n2 -> ~dur(n1)>=
                n1, s1=>s2
407         testSimplifiedDCFormula(dcSimplifier,
408         new Disjunction(
409         new Negation((Duration) dur1.clone()),
410         new Negation((Duration) dur2.clone())),
411         new Negation((Duration) dur1.clone()).toString())
                ;
412         testSimplifiedDCFormula(dcSimplifier,
413         new Disjunction(
414         new Negation((Duration) dur2.clone()),
415         new Negation((Duration) dur1.clone())),
416         new Negation((Duration) dur1.clone()).toString())
                ;
417         // DC: ~dur(s1)>=n1 \/ ~dur(s2)>=n2 -> no change,
                s1=/>s2
418         testSimplifiedDCFormula(dcSimplifier,
419         new Disjunction(
420         new Negation((Duration) dur2.clone()),
421         new Negation((Duration) dur3.clone())),
422         new Disjunction(
423         new Negation((Duration) dur2.clone()),
424         new Negation((Duration) dur3.clone())).toString()
                );
425         testSimplifiedDCFormula(dcSimplifier,
426         new Disjunction(
427         new Negation((Duration) dur3.clone()),
428         new Negation((Duration) dur2.clone())),
429         new Disjunction(
```

```
430             new Negation((Duration) dur3.clone()),
431             new Negation((Duration) dur2.clone())).toString()
                    );
432
433             // DC: Duration
434             // DC: Negation
435             // DC: True
436             // DC: False
437             testNonSimplifiedDCFormula(dcSimplifier, dur1);
438         }
439
440         public void performPopulateDCTest(Settings settings)
                throws Exception {
441             DCSimplifier dcSimplifier = new DCSimplifier(
                    settings);
442             LinkedList compareToList = new LinkedList();
443
444             Duration dur1 = new Duration(new True(),1);
445             Duration dur2 = new Duration(new True(),2);
446             Duration dur3 = new Duration(new State("x1"), 3);
447             Negation ndur3 = new Negation(dur3);
448
449             Formula f1 = new Conjunction(
450             new Disjunction(dur1, new True()),
451             new Conjunction(
452             new Conjunction(new True(), new Conjunction(new
                    False(), dur2)),
453             new Conjunction(new Chop((Formula) dur1.clone(),
                    new True()), ndur3)));
454
455             dcSimplifier.populateDCCompareList(f1,
                    compareToList, Formula.TYPE_CONJUNCTION);
456             Assert.assertTrue(compareToList.contains(dur2));
457             Assert.assertTrue(compareToList.contains(ndur3));
458             Assert.assertEquals(2, compareToList.size());
459
460             compareToList = new LinkedList();
461
462             f1 = new Disjunction(
463             new Conjunction(dur1, new True()),
464             new Disjunction(
465             new Disjunction(new True(), new Disjunction(new
                    False(), dur2)),
466             new Disjunction(new Chop((Formula) dur1.clone(),
                    new True()), ndur3)));
467
468             dcSimplifier.populateDCCompareList(f1,
                    compareToList, Formula.TYPE_DISJUNCTION);
469             Assert.assertTrue(compareToList.contains(dur2));
470             Assert.assertTrue(compareToList.contains(ndur3));
```

```
471         Assert.assertEquals(2, compareToList.size());
472
473     }
474
475     public void performPopulateSATest(Settings settings)
            {
476         DCSimplifier dcSimplifier = new DCSimplifier(
                settings);
477         LinkedList compareToList = new LinkedList();
478
479         State s1 = new State("x1");
480         State s2 = new State("x2");
481         State s3 = new State("x3");
482         Negation ns3 = new Negation(s3);
483         Disjunction dis1 = new Disjunction(s1, new True()
                );
484         Conjunction con1 = new Conjunction(s1, new True()
                );
485
486         Formula f1 = new Conjunction(
487         dis1,
488         new Conjunction(
489         new Conjunction(new True(), new Conjunction(new
                False(), s2)),
490         new Conjunction(new True(), ns3)));
491
492         dcSimplifier.populateSACompareList(f1,
                compareToList, Formula.TYPE_CONJUNCTION);
493         Assert.assertTrue(compareToList.contains(s2));
494         Assert.assertTrue(compareToList.contains(ns3));
495         Assert.assertTrue(compareToList.contains(dis1));
496         Assert.assertEquals(3, compareToList.size());
497
498         compareToList = new LinkedList();
499
500         f1 = new Disjunction(
501         con1,
502         new Disjunction(
503         new Disjunction(new True(), new Disjunction(new
                False(), s2)),
504         new Disjunction(new True(), ns3)));
505
506
507         dcSimplifier.populateSACompareList(f1,
                compareToList, Formula.TYPE_DISJUNCTION);
508         Assert.assertTrue(compareToList.contains(s2));
509         Assert.assertTrue(compareToList.contains(ns3));
510         Assert.assertTrue(compareToList.contains(con1));
511         Assert.assertEquals(3, compareToList.size());
512     }
```

```
513
514     public void testSimplifier() throws Exception {
515
516         Settings settings = new Settings();
517         settings.setK(TEST_K_VALUE);
518
519         performSimplifyTreeTest(settings);
520
521         performLevelNoneTest(settings);
522
523         performBasicTest(settings);
524
525         performWithBDDTest(settings);
526
527         performPopulateDCTest(settings);
528
529         performPopulateSATest(settings);
530     }
531
532     public static junit.framework.Test suite() {
533         TestSuite testSuite = new TestSuite();
534         testSuite.addTest(new Test("testSimplifier"));
535         testSuite.addTest(new Test("testROBDDConstructor
                "));
536         return testSuite;
537     }
538
539     private void testUnsupportedDCFormula(DCSimplifier
            dcSimplifier, Formula formula) throws Exception {
540         try {
541             dcSimplifier.simplifyDC(formula);
542             Assert.assertTrue("Formula ["+formula+"]'s
                    type is not allowed in DC-formulas. Should
                     throw exception but did not", false);
543         }
544         catch (IllegalArgumentException e) {
545             Assert.assertTrue("Formula ["+formula+"]'s
                    type is not allowed in DC-formulas. Throws
                     exception as expected", true);
546         }
547     }
548
549     private void testUnsupportedSAFormula(DCSimplifier
            dcSimplifier, Formula formula) throws Exception {
550         try {
551             dcSimplifier.simplifySA(formula);
552             Assert.assertTrue("Formula ["+formula+"]'s
                    type is not allowed in DC-formulas. Should
                     throw exception", false);
553         }
```

```
554             catch (IllegalArgumentException e) {
555                 Assert.assertTrue("Formula ["+formula+"]'s
                        type is not allowed in DC-formulas. Should
                        throw exception", true);
556             }
557         }
558
559         private void testNonSimplifiedDCFormula(DCSimplifier
                dcSimplifier, Formula formula) throws Exception {
560             Assert.assertEquals("Formula has not been changed
                    : ", formula.toString(), dcSimplifier.
                    simplifyDC(formula).toString());
561         }
562
563
564         private void testNonSimplifiedSAFormula(DCSimplifier
                dcSimplifier, Formula formula) throws Exception {
565             Assert.assertEquals("Formula has not been changed
                    : ", formula.toString(), dcSimplifier.
                    simplifySA(formula).toString());
566         }
567
568         private void testSimplifiedDCFormula(DCSimplifier
                dcSimplifier, Formula formula, String
                expectedFormulaString) throws Exception {
569             Assert.assertEquals("Simplified DCformula should
                    look like: ", expectedFormulaString,
                    dcSimplifier.simplifyDC(formula).toString());
570         }
571
572
573         private void testSimplifiedSAFormula(DCSimplifier
                dcSimplifier, Formula formula, String
                expectedFormulaString) throws Exception {
574             Assert.assertEquals("Simplified SAformula should
                    look like: ", expectedFormulaString,
                    dcSimplifier.simplifySA(formula).toString());
575         }
576
577         public void testROBDDConstructor() throws Exception {
578             LinkedList list = new LinkedList();
579             list.add(new State("a"));
580             list.add(new Disjunction(new State("b"), new True
                    ()));
581             list.add(new False());
582
583             // Test constructor
584             SubsequenceROBDDConstructor src = new
                    SubsequenceROBDDConstructor(list);
```

```
585         Assert.assertSame("compareList is set properly in
                SROBDDC constr", src.compareList, list);
586         Assert.assertNotNull("robddUtil is not null in
               SROBDDC constr",src.robddUtil);
587         Assert.assertNotNull("postfixRobdds is not null
               in SROBDDC constr",src.postfixRobdds);
588         Assert.assertNotNull("prefixRobdds is not null in
                SROBDDC constr",src.prefixRobdds);
589
590         LinkedList list2 = new LinkedList();
591         list2.add(new State("a"));
592         try {
593             SubsequenceROBDDConstructor src2 = new
                    SubsequenceROBDDConstructor(list2);
594             Assert.assertFalse("It doesnt make sense to
                    create subsequence constructor with only
                    one item in list", true);
595         }
596         catch (IllegalArgumentException e) {
597             Assert.assertFalse("It doesnt make sense to
                    create subsequence constructor with only
                    one item in list", false);
598         }
599
600         // Test getROBDD -- conjunction
601         ROBDDUtil util = new ROBDDUtil();
602         ROBDD res1 = src.getROBDD(0, ROBDDUtil.
               OP_CONJUNCTION);
603         Assert.assertEquals("SROBDDC getROBDD(0) ok",util
               .makeROBDD(new Conjunction((Formula) list.get
               (1),
604         (Formula) list.get(2))), res1);
605         ROBDD res2 = src.getROBDD(1, ROBDDUtil.
               OP_CONJUNCTION);
606         Assert.assertEquals("SROBDDC getROBDD(1) ok",res2
               , util.makeROBDD(new Conjunction((Formula)
               list.get(0),
607         (Formula) list.get(2))));
608         ROBDD res3 = src.getROBDD(2, ROBDDUtil.
               OP_CONJUNCTION);
609         Assert.assertEquals("SROBDDC getROBDD(2) ok",res3
               , util.makeROBDD(new Conjunction((Formula)
               list.get(0),
610         (Formula) list.get(1))));
611
612         try {
613             ROBDD res4 = src.getROBDD(1, ROBDDUtil.
                    OP_CONJUNCTION);
614             Assert.assertFalse("It is not allowed to call
                     getROBDD with a decreased excludeId value
```

```
                ", true);
615         }
616         catch (IllegalArgumentException e) {
617             Assert.assertTrue("It is not allowed to call
                    getROBDD with a decreased excludeId value
                    ", true);
618         }
619
620         // Test getROBDD --- disjunction
621         list = new LinkedList();
622         list.add(new State("a"));
623         list.add(new Conjunction(new State("b"), new True
                ()));
624         list.add(new False());
625         src = new SubsequenceROBDDConstructor(list);
626
627         res1 = src.getROBDD(0, ROBDDUtil.OP_DISJUNCTION);
628         Assert.assertEquals("SROBDDC getROBDD(0) ok",util
                .makeROBDD(new Disjunction((Formula) list.get
                (1),
629         (Formula) list.get(2))), res1);
630         res2 = src.getROBDD(1, ROBDDUtil.OP_DISJUNCTION);
631         Assert.assertEquals("SROBDDC getROBDD(1) ok",res2
                , util.makeROBDD(new Disjunction((Formula)
                list.get(0),
632         (Formula) list.get(2))));
633         res3 = src.getROBDD(2, ROBDDUtil.OP_DISJUNCTION);
634         Assert.assertEquals("SROBDDC getROBDD(2) ok",res3
                , util.makeROBDD(new Disjunction((Formula)
                list.get(0),
635         (Formula) list.get(1))));
636
637         try {
638             ROBDD res4 = src.getROBDD(1, ROBDDUtil.
                    OP_DISJUNCTION);
639             Assert.assertFalse("It is not allowed to call
                     getROBDD with a decreased excludeId value
                    ", true);
640         }
641         catch (IllegalArgumentException e) {
642             Assert.assertTrue("It is not allowed to call
                    getROBDD with a decreased excludeId value
                    ", true);
643         }
644
645     }
646 }
```

G.1.7 bmc.tracer.Test.java

```
1  package bmc.tracer;
2
3  import java.util.*;
4  import java.io.*;
5
6  import bmc.formula.*;
7  import bmc.util.*;
8  import bmc.literal.*;
9
10 import junit.framework.*;
11
12 public class Test extends TestCase {
13
14     public Test(String testName) {
15         super(testName);
16     }
17
18     public void testStateTrace() {
19         State s1 = new State("s1");
20         State s2 = new State("s2");
21         int[] values1 = new int[]
22         {HySatSolutionParser.VALUE_TRUE,
               HySatSolutionParser.VALUE_FALSE,
               HySatSolutionParser.VALUE_WILDCARD};
23         int[] values2 = new int[] {};
24         StateTrace st1 = new StateTrace(s1, values1);
25         StateTrace st2 = new StateTrace(s2, values2);
26
27
28         Assert.assertEquals("st1 equals itself", st1, st1
               );
29         Assert.assertFalse("st1 doesnt equal st2", st1.
               equals(st2));
30
31
32         Assert.assertSame("st1's state is s1", st1.
               getState(), s1);
33         Assert.assertSame("st1's values is the same as
               values1", st1.getValues(), values1);
34         Assert.assertSame("st2's values is the same as
               values2", st2.getValues(), values2);
35         Assert.assertTrue("st1 is less than st2 when
               compared (alphabetically)", st1.compareTo(st2)
                < 0);
36
37         Assert.assertEquals("st1.toString() looks like
               this",
38         "s1 ["+HySatSolutionParser.VALUE_TRUE+","+
39         HySatSolutionParser.VALUE_FALSE+","+
40         HySatSolutionParser.VALUE_WILDCARD+"]",
```

```
41          st1.toString());
42          Assert.assertEquals("st2.toString() looks like
                this", "s2 []", st2.toString());
43      }
44
45
46      private HySatSolutionParser parse(String input,
            boolean displayTrace, LiteralHandler litHandler)
            throws Exception {
47          StringReader reader = new StringReader(input);
48          Goal goal = new Goal("goal1", new Conjunction(new
                True(), new False()), new Settings(), new
                HashSet());
49          HySatSolutionParser parser = new
                HySatSolutionParser(reader, litHandler, goal,
                displayTrace);
50          parser.parse();
51          return parser;
52      }
53
54      public void testHySatSolutionParser() throws
            Exception {
55          LiteralHandler litHandler = new
                IDLookupLiteralHandler();
56          String input = "";
57          StringReader reader = new StringReader(input);
58          Goal goal = new Goal("goal1", new Conjunction(new
                True(), new False()), new Settings(), new
                HashSet());
59          HySatSolutionParser parser = new
                HySatSolutionParser(reader, litHandler, goal,
                false);
60
61          // test constructor
62          Assert.assertSame("Constructor fields match",
                goal, parser.goal);
63          Assert.assertSame("Constructor fields match",
                reader, parser.input);
64          Assert.assertSame("Constructor fields match",
                litHandler, parser.literalHandler);
65          Assert.assertEquals("Constructor fields match",
                false, parser.displayTrace);
66
67          // has status string, sat, cpu string, cpu time
                -> no errors
68          input = "\nSTATUS: Satisfiable\n\ncpu time for
                solving : 01.990\n";
69          parser = parse(input, false, litHandler);
70          Assert.assertEquals("CPU time correct", 1.99,
                parser.getCPUTime(), 0.00);
```

```
71          Assert.assertEquals("Status correct",
                HySatSolutionParser.STATUS_SAT, parser.
                getStatus());
72          // has status string, unsat
73          input = "\nSTATUS: No solution\n\ncpu time for
                solving : 01.990\n";
74          parser = parse(input, false, litHandler);
75          Assert.assertEquals("Status correct",
                HySatSolutionParser.STATUS_UNSAT, parser.
                getStatus());
76
77          // no sat string -> exception
78          try {
79              input = "\nSTATUS:\n\n\n";
80              parser = parse(input, false, litHandler);
81              Assert.assertTrue("Exception is thrown",
                    false);
82          }
83          catch (Exception e) {
84              Assert.assertTrue("Exception is thrown", true
                    );
85          }
86
87          // has no status string -> exception
88          try {
89              input = "\n Satisfiable:\n";
90              parser = parse(input, false, litHandler);
91              Assert.assertTrue("Exception is thrown",
                    false);
92          }
93          catch (Exception e) {
94              Assert.assertTrue("Exception is thrown", true
                    );
95          }
96
97          // has no cpu string
98          try {
99              input = "\nSTATUS: Satisfiable:\n\n";
100             parser = parse(input, false, litHandler);
101             Assert.assertTrue("Exception is thrown",
                    false);
102         }
103         catch (Exception e) {
104             Assert.assertTrue("Exception is thrown", true
                    );
105         }
106
107         // has no cpu time -> exception
108         try {
```

```
109             input = "\nSTATUS: Satisfiable:\n\ncpu time
                    for solving: \n";
110             parser = parse(input, false, litHandler);
111             Assert.assertTrue("Exception is thrown",
                    false);
112         }
113         catch (Exception e) {
114             Assert.assertTrue("Exception is thrown", true
                    );
115         }
116 
117         // has solution string (with / without
                displayTrace):
118         // has datalines ( displayTrace ? stateTrace :
                nothing )
119         // literals with ('*', '-', '+')
120         // unused states (!hasDef(state, i)) -> '*'
121         Formula x1 = new State("x1");
122         Literal x10 = litHandler.getLiteral(x1, 0);
123         Literal x11 = litHandler.getLiteral(x1, 1);
124         Formula y1 = new State("y1");
125         Literal y10 = litHandler.getLiteral(y1, 0);
126         Literal y11 = litHandler.getLiteral(y1, 1);
127         Formula z = new State("z");
128         input = "\nSTATUS: Satisfiable\n\ncpu time for
                solving : 01.990\n\n"+
129         "Solution:\n"+
130         ""+x11.getNumber()+" "+
131         "-"+x10.getNumber()+" "+
132         "*"+y11.getNumber()+" "+
133         ""+y10.getNumber()+" ";
134         litHandler.addDef(x1,true);
135         litHandler.addDef(y1,true);
136         reader = new StringReader(input);
137         Settings settings = new Settings();
138         settings.setK(2);
139         HashSet stateSet = new HashSet();
140         stateSet.add(x1);
141         stateSet.add(y1);
142         stateSet.add(z);
143         goal = new Goal("goal1", new Conjunction(new
                Conjunction(y1, x1), z), settings, stateSet);
144         parser = new HySatSolutionParser(reader,
                litHandler, goal, true);
145         parser.parse();
146 
147         Assert.assertEquals("3 elems in sts", 3, parser.
                getStateTraceSet().size());
148         Iterator it = parser.getStateTraceSet().iterator
                ();
```

```
149         StateTrace stx1 = (StateTrace) it.next();
150         Assert.assertEquals("", "x1", stx1.getState().
                getName());
151         Assert.assertEquals("2 values inside", 2, stx1.
                getValues().length);
152         Assert.assertEquals("Value for x are",
                HySatSolutionParser.VALUE_FALSE, stx1.
                getValues()[0]);
153         Assert.assertEquals("Values are",
                HySatSolutionParser.VALUE_TRUE, stx1.getValues
                ()[1]);
154         StateTrace sty1 = (StateTrace) it.next();
155         Assert.assertEquals("", "y1", sty1.getState().
                getName());
156         Assert.assertEquals("Values are",
                HySatSolutionParser.VALUE_TRUE, sty1.getValues
                ()[0]);
157         Assert.assertEquals("Values are",
                HySatSolutionParser.VALUE_WILDCARD, sty1.
                getValues()[1]);
158         StateTrace stz = (StateTrace) it.next();
159         Assert.assertEquals("", "z", stz.getState().
                getName());
160         Assert.assertEquals("Values are",
                HySatSolutionParser.VALUE_WILDCARD, stz.
                getValues()[0]);
161         Assert.assertEquals("Values are",
                HySatSolutionParser.VALUE_WILDCARD, stz.
                getValues()[1]);
162
163         // has solution string ( displayTrace ?
                stateTrace : nothing )
164         reader = new StringReader(input);
165         parser = new HySatSolutionParser(reader,
                litHandler, goal, false);
166         parser.parse();
167         Assert.assertTrue("Sts empty", parser.
                getStateTraceSet().isEmpty());
168
169         // has no solution string ( displayTrace ?
                exception : nothing )
170         try {
171             input = "\nSTATUS: Satisfiable\n\ncpu time
                    for solving : 01.990\n\n"+
172             "So_lution:\n"+
173             ""+x11.getNumber()+" "+
174             "-"+x10.getNumber()+" "+
175             "*"+y11.getNumber()+" "+
176             ""+y10.getNumber()+" ";
177             parser = parse(input, true, litHandler);
```

```
178             Assert.assertTrue("Exception is thrown",
                    false);
179         }
180         catch (Exception e) {
181             Assert.assertTrue("Exception is thrown", true
                    );
182         }
183         // has no datalines ( displayTrace -> exception )
184         try {
185             input = "\nSTATUS: Satisfiable\n\ncpu time
                    for solving : 01.990\n\n"+
186             "Solution:\n";
187             parser = parse(input, true, litHandler);
188             Assert.assertTrue("Exception is thrown",
                    false);
189         }
190         catch (Exception e) {
191             Assert.assertTrue("Exception is thrown", true
                    );
192         }
193         // has no datalines ( !displayTrace -> nothing )
194         input = "\nSTATUS: Satisfiable\n\ncpu time for
                solving : 01.990\n\n"+
195         "Solution:\n";
196         parser = parse(input, false, litHandler);
197         Assert.assertTrue("Exception is not thrown", true
                );
198     }
199
200
201     public static junit.framework.Test suite() {
202         return new TestSuite(Test.class);
203     }
204 }
```

G.1.8 bmc.util.Test.java

```
1  package bmc.util;
2
3  import bmc.*;
4  import bmc.formula.*;
5  import junit.framework.*;
6  import java.io.*;
7  import java.util.*;
8
9  public class Test extends TestCase {
10
11     public void testCommand() {
12         String commandText = "someCommandText";
13         Command command1 = new Command(commandText);
```

```
14          Command command2 = new Command("someCommandText")
                ;
15
16          Assert.assertEquals("commandText inside command1
                is commandText", commandText, command1.
                getCommandText());
17          Assert.assertEquals("command1.toString() returns
                commandText", commandText, command1.toString()
                );
18      }
19
20      public void testGoal() {
21          String name1 = "name1";
22          Formula formula1 = new Conjunction(new True(),
                new False());
23          Settings settings1 = new Settings();
24          HashSet states1 = new HashSet();
25          Goal goal1 = new Goal(name1, formula1, settings1,
                 states1);
26          Goal goal2 = new Goal("name3", new True(), new
                Settings(), new HashSet());
27
28          Assert.assertEquals("goal1.getName() returns
                name1", name1, goal1.getName());
29          Assert.assertSame("goal1.getFormula() returns
                formula1", formula1, goal1.getFormula());
30          Assert.assertSame("goal1.getStates() returns
                states1", states1, goal1.getStates() );
31          Assert.assertSame("goal1.getSettings() returns
                settings1", settings1, goal1.getSettings());
32          Assert.assertEquals("goal1.toString() returns ",
                "name1: (true /\\ false)\n"+settings1.toString
                (), goal1.toString());
33      }
34
35      public void testIntList() {
36          IntBag intBag1 = new IntBag();
37
38          Assert.assertTrue("intBag1 size() is 0", intBag1.
                size() == 0);
39          Assert.assertTrue("intBag1.nums.length is 8",
                intBag1.nums.length == 8);
40          for (int i=0 ; i<9 ; i++) {
41              intBag1.add(i);
42          }
43          Assert.assertTrue("intBag1 size() is 9", intBag1.
                size() == 9);
44          Assert.assertTrue("intBag1.nums.length increases
                to the double: 16", intBag1.nums.length == 16)
                ;
```

```
45          for (int i=0 ; i<9 ; i++) {
46              Assert.assertEquals("Value at index ["+i+"]",
                    i, intBag1.get(i));
47          }
48          for (int i=0 ; i<9 ; i++) {
49              intBag1.set(i, 9-i);
50          }
51          for (int i=0 ; i<9 ; i++) {
52              Assert.assertEquals("Value at index ["+i+"]",
                    9-i, intBag1.get(i));
53          }
54          for (int i=9 ; i<17 ; i++) {
55              intBag1.add(i);
56          }
57          Assert.assertTrue("intList1.nums.length increases
                with 16 this time (not only 8)", intBag1.nums
               .length == 32);
58          try {
59              intBag1.set(17,17);
60              Assert.assertFalse("intList1 should throw an
                    IndexOutOfBoundsException when index 17 is
                     set", true);
61          }
62          catch (ArrayIndexOutOfBoundsException ignored) {
63              Assert.assertTrue("intList1 throws exception
                    ", true);
64          }
65          try {
66              intBag1.set(-1,-1);
67              Assert.assertFalse("intList1 should throw an
                    IndexOutOfBoundsException when index -1 is
                     set", true);
68          }
69          catch (ArrayIndexOutOfBoundsException ignored) {
70              Assert.assertTrue("intList1 throws exception
                    ", true);
71          }
72          try {
73              intBag1.get(17);
74              Assert.assertFalse("intList1 should throw an
                    IndexOutOfBoundsException when index 17 is
                     requested", true);
75          }
76          catch (ArrayIndexOutOfBoundsException ignored) {
77              Assert.assertTrue("intList1 throws exception
                    ", true);
78          }
79          try {
80              intBag1.get(-1);
```

```
81              Assert.assertFalse("intList1 should throw an
                    IndexOutOfBoundsException when index -1 is
                     requested", true);
82          }
83          catch (ArrayIndexOutOfBoundsException ignored) {
84              Assert.assertTrue("intList1 throws exception
                    ", true);
85          }
86
87          try {
88              intBag1.remove(-1);
89              Assert.assertFalse("intList1 should throw an
                    IndexOutOfBoundsException when index -1 is
                     removed", true);
90          }
91          catch (ArrayIndexOutOfBoundsException ignored) {
92              Assert.assertTrue("intList1 throws exception
                    ", true);
93          }
94          try {
95              intBag1.remove(17);
96              Assert.assertFalse("intList1 should throw an
                    IndexOutOfBoundsException when index -1 is
                     removed", true);
97          }
98          catch (ArrayIndexOutOfBoundsException ignored) {
99              Assert.assertTrue("intList1 throws exception
                    ", true);
100         }
101         intBag1.remove(16);
102         intBag1.remove(0);
103         Assert.assertTrue("intList1 size() is 15",
                intBag1.size() == 15);
104         Assert.assertEquals("toString() is
                ","[15,8,7,6,5,4,3,2,1,9,10,11,12,13,14]",
                intBag1.toString());
105
106         for (int i=0 ; i<15 ; i++) {
107             intBag1.remove(15-i-1);
108         }
109         Assert.assertTrue("intList1 is now empty",
                intBag1.size() == 0);
110         Assert.assertTrue("intList1.nums.length does not
                decrease", intBag1.nums.length == 32);
111     }
112
113     public void testSettings() throws Exception {
114         Settings settings1 = new Settings();
115         Settings settings2 = new Settings();
116
```

```
117         Assert.assertEquals("stdValue findK", false,
                settings1.getFindK());
118         Assert.assertEquals("stdValue nnf", false,
                settings1.getNNF());
119         Assert.assertEquals("stdValue polarityOpt", true,
                 settings1.getPolarityOpt());
120         Assert.assertEquals("stdValue outputFormat",
                Settings.OUTPUT_FORMAT_ZOLCS, settings1.
                getOutputFormat());
121         Assert.assertEquals("stdValue outputFolder", "",
                settings1.getOutputFolder());
122         Assert.assertEquals("stdValue fRecognition",
                Settings.F_RECOGNITION_SYNTACTIC, settings1.
                getFRecognition());
123         Assert.assertEquals("stdValue k", 1, settings1.
                getK());
124         Assert.assertEquals("stdValue DCSimpLevel",
                Settings.DC_SIMP_LEVEL_BASIC, settings1.
                getDCSimpLevel());
125         Assert.assertEquals("toString() is:", "k=1\
                npolarityOpt=true\ndcSimpLevel=1\noutputFormat
                =1\nfRecognition=2\nfindK=false\noutputFolder
                =(current folder)\n", settings1.toString());
126
127         settings1.setDCSimpLevel(Settings.
                DC_SIMP_LEVEL_WITH_BDD);
128         Assert.assertEquals("SimpLevel has been changed",
                 Settings.DC_SIMP_LEVEL_WITH_BDD, settings1.
                getDCSimpLevel());
129         try {
130             settings1.setDCSimpLevel(Settings.
                    DC_SIMP_LEVEL_WITH_BDD+1);
131             Assert.assertFalse("Should have raised an
                    exception", true);
132         }
133         catch (IllegalArgumentException ignored) {
134             Assert.assertTrue("Should have raised an
                    exception", true);
135         }
136         try {
137             settings1.setDCSimpLevel(Settings.
                    DC_SIMP_LEVEL_NONE-1);
138             Assert.assertFalse("Should have raised an
                    exception", true);
139         }
140         catch (IllegalArgumentException ignored) {
141             Assert.assertTrue("Should have raised an
                    exception", true);
142         }
```

```
143         settings1.setDCSimpLevel(Settings.
                DC_SIMP_LEVEL_BASIC);
144         Assert.assertEquals("SimpLevel has been changed",
                 Settings.DC_SIMP_LEVEL_BASIC, settings1.
                getDCSimpLevel());
145
146         settings1.setK(10);
147         Assert.assertEquals("k has been changed", 10,
                settings1.getK());
148         try {
149             settings1.setK(-1);
150             Assert.assertFalse("Should have raised an
                    exception", true);
151         }
152         catch (IllegalArgumentException ignored) {
153             Assert.assertTrue("Should have raised an
                    exception", true);
154         }
155         settings1.setFRecognition(Settings.
                F_RECOGNITION_ID);
156         Assert.assertEquals("fRecognition has been
                changed", Settings.F_RECOGNITION_ID, settings1
                .getFRecognition());
157         settings1.setFRecognition(Settings.
                F_RECOGNITION_SEMANTIC);
158         Assert.assertEquals("fRecognition has been
                changed", Settings.F_RECOGNITION_SEMANTIC,
                settings1.getFRecognition());
159         Assert.assertEquals("SimpLevel is forced to
                WITH_BDD because of semantic fRecognition",
                Settings.DC_SIMP_LEVEL_WITH_BDD, settings1.
                getDCSimpLevel());
160         try {
161             settings1.setFRecognition(Settings.
                    F_RECOGNITION_ID-1);
162             Assert.assertFalse("Should have raised an
                    exception", true);
163         }
164         catch (IllegalArgumentException ignored) {
165             Assert.assertTrue("Should have raised an
                    exception", true);
166         }
167         try {
168             settings1.setFRecognition(Settings.
                    F_RECOGNITION_SEMANTIC+1);
169             Assert.assertFalse("Should have raised an
                    exception", true);
170         }
171         catch (IllegalArgumentException ignored) {
```

```
172             Assert.assertTrue("Should have raised an
                    exception", true);
173         }
174
175         settings1.setFindK(true);
176         Assert.assertEquals("findK has been changed",
                true, settings1.getFindK());
177         settings1.setFindK(false);
178         Assert.assertEquals("findK has been changed",
                false, settings1.getFindK());
179
180         settings1.setNNF(true);
181         Assert.assertEquals("nnf has been changed", true,
                 settings1.getNNF());
182         settings1.setNNF(false);
183         Assert.assertEquals("nnf has been changed", false
                , settings1.getNNF());
184         settings1.setFindK(true);
185         Assert.assertEquals("nnf is forced to true
                because findK = true", true, settings1.getNNF
                ());
186         settings1.setNNF(true);
187         settings1.setFindK(false);
188
189
190         settings1.setPolarityOpt(false);
191         Assert.assertEquals("polarityOpt has been changed
                ", false, settings1.getPolarityOpt());
192         settings1.setPolarityOpt(true);
193         Assert.assertEquals("polarityOpt has been changed
                ", true, settings1.getPolarityOpt());
194
195         settings1.setOutputFormat(Settings.
                OUTPUT_FORMAT_DIMACS);
196         Assert.assertEquals("outputFormat has been
                changed", Settings.OUTPUT_FORMAT_DIMACS,
                settings1.getOutputFormat());
197         try {
198             settings1.setOutputFormat(Settings.
                    OUTPUT_FORMAT_ZOLCS-1);
199             Assert.assertFalse("Should have raised an
                    exception", true);
200         }
201         catch (IllegalArgumentException ignored) {
202             Assert.assertTrue("Should have raised an
                    exception", true);
203         }
204         try {
205             settings1.setOutputFormat(Settings.
                    OUTPUT_FORMAT_DIMACS+1);
```

```
206             Assert.assertFalse("Should have raised an
                    exception", true);
207         }
208         catch (IllegalArgumentException ignored) {
209             Assert.assertTrue("Should have raised an
                    exception", true);
210         }
211
212         settings1.setOutputFolder("\\someFolder\\
                someSubfolder");
213         Assert.assertEquals("outputFolder has changed",
                "\\someFolder\\someSubfolder", settings1.
                getOutputFolder());
214         settings1.setOutputFolder("");
215         Assert.assertEquals("outputFolder has changed",
                "", settings1.getOutputFolder());
216
217         Settings settings1Clone = (Settings) settings1.
                clone();
218         Assert.assertNotSame(settings1, settings1Clone);
219         Assert.assertEquals("Content is the same, though
                ", settings1.toString(), settings1Clone.
                toString());
220
221     }
222
223     public void testCannotFindKException() {
224         Formula f1 = new Conjunction(new True(), new
                State("x1"));
225         Exception exc1 = new CannotFindKException(f1);
226
227         Assert.assertEquals("exc1.message is concat(
                prefix,excExplanation)", "Cannot find k from
                formula: "+f1.toString(), exc1.getMessage());
228     }
229
230     public void testShellExecutor() throws Exception {
231         String[] cmd = new String[3];
232         cmd[0] = "cmd.exe" ;
233         cmd[1] = "/C" ;
234         ShellExecutor shellE1 = new ShellExecutor();
235
236         ShellExecutor shellE2 = new ShellExecutor(cmd);
237         Assert.assertEquals("shellE2.cmd[0] is cmd[0]",
                cmd[0], shellE2.cmd[0]);
238         Assert.assertEquals("shellE2.cmd[1] is cmd[1]",
                cmd[1], shellE2.cmd[1]);
239         // Testing that a command is actually performed
                and that all parameters are sent to the shell
240         File tmpFile = new File("");
```

```
241         tmpFile = new File(tmpFile.getAbsolutePath(), "
                tmpfile.text");
242         String tmpFilePath = tmpFile.getCanonicalPath();
243         String testCommandCreate = "java -cp dev bmc.test
                .ShellExecutorTest \""+tmpFilePath+"\"";
244         String testCommandIllegal = "java -cp dev bmc.
                test.ShellExecutorTest \""+tmpFilePath +"\"
                and another five nonsense parameters";
245
246         if (tmpFile.exists() && !tmpFile.delete()) {
247             throw new IllegalStateException("File delete
                    wasnt allowed. Test cannot proceed ...");
248         }
249         shellE1.execCmdAndFinish(testCommandCreate);
250         Assert.assertTrue("Tmp file exists", tmpFile.
                exists());
251         if (!tmpFile.delete()) {
252             throw new IllegalStateException("File delete
                    wasnt allowed. Test cannot proceed ...");
253         }
254         shellE1.execCmdAndFinish(testCommandIllegal);
255         Assert.assertFalse("More than one parameter were
                sent to program, so the test program didnt
                create file again", tmpFile.exists());
256     }
257
258     public void testCNFTranslator() throws Exception {
259         CNFTranslator cnfT = new CNFTranslator();
260         // test translateToNNF (bringing negations to the
                "bottom" of a formula)
261         // test, that all operators act correctly when
                placed under zero, one or more negations
262         checkNNFTranslation(cnfT, new Negation(new True()
                ), "false", "true");
263
264         checkNNFTranslation(cnfT,
265         new Chop(
266         new Negation(
267         new Conjunction(
268         new State("x1"),
269         new True())),
270         new True()), "((~x1 \\/ false) ; true)", "~((~x1
                \\/ false) ; true)");
271
272         checkNNFTranslation(cnfT, new Conjunction(new
                True(), new False()), "(true /\\ false)", "(
                false \\/ true)");
273
274         checkNNFTranslation(cnfT, new Disjunction(new
                True(), new False()), "(true \\/ false)", "(
```

```
            false /\\ true)");
275
276         checkNNFTranslation(cnfT, new Duration(new True()
                ,4), "dur true >= 4", "~dur true >= 4");
277         checkNNFTranslation(cnfT, new Duration(new
                Negation(new State("x1")),4), "dur ~x1 >= 4",
                "~dur ~x1 >= 4");
278
279         checkNNFTranslation(cnfT, new False(), "false", "
                true");
280
281         checkNNFTranslation(cnfT, new State("x1"), "x1",
                "~x1");
282
283         checkNNFTranslation(cnfT, new True(), "true", "
                false");
284
285         // test translateToCNF (bringing an nnf-formula
                to cnf)
286         // True, False, State, Negation, Chop (recursive
                on children)
287         checkCNFTranslation(cnfT,
288         new Chop(new Disjunction(new Conjunction(new True
                (), new False()), new True()), new Negation(
                new State("x1"))),
289         "(((true \\/ true) /\\ (false \\/ true)) ; ~x1)")
                ;
290         checkCNFTranslation(cnfT,
291         new Chop(new State("x1"), new Disjunction(new
                Conjunction(new True(), new False()), new True
                ())),
292         "(x1 ; ((true \\/ true) /\\ (false \\/ true)))");
293
294         // Duration (recursion on changed child and not-
                changed child)
295         checkCNFTranslation(cnfT,
296         new Duration(
297         new Disjunction(
298         new Conjunction(
299         new True(),
300         new False()),
301         new True()),
302         5),
303         "dur ((true \\/ true) /\\ (false \\/ true)) >=
                5");
304         checkCNFTranslation(cnfT, new Duration(new State
                ("x1"), 5), "dur x1 >= 5");
305
306         // Conjunction:
307         checkCNFTranslation(cnfT,
```

```
308         new Conjunction(
309         new Negation(new True()),
310         new State("x1")), "(false /\\ x1)");
311
312
313         // Disjunction: + special cases + where generated
                subtrees need cnf'ication as well
314         checkCNFTranslation(cnfT,
315         new Disjunction(new State("a"),new State("b")),
316         "(a \\/ b)");
317
318         checkCNFTranslation(cnfT,
319         new Disjunction(
320         new Conjunction(
321         new State("a"),
322         new State("b")),
323         new Conjunction(
324         new State("c"),
325         new State("d"))
326         ),
327         "(((a \\/ c) /\\ (a \\/ d)) /\\ ((b \\/ c) /\\ (b
                \\/ d)))");
328         checkCNFTranslation(cnfT,
329         new Disjunction(
330         new Conjunction(
331         new State("a"),
332         new State("b")),
333         new Duration(
334         new State("cs"),
335         6)
336         ),
337         "((a \\/ dur cs >= 6) /\\ (b \\/ dur cs >= 6))");
338
339
340         checkCNFTranslation(cnfT,
341         new Disjunction(new Duration(new State("as"), 6),
                new Conjunction(new State("b"), new State("c
               "))),
342         "((dur as >= 6 \\/ b) /\\ (dur as >= 6 \\/ c))");
343         checkCNFTranslation(cnfT,
344         new Disjunction(new Conjunction(new Conjunction(
               new State("a1"),new State("a2")), new State("b
               ")), new Conjunction(new State("c"), new State
               ("d"))),
345         "((((a1 \\/ c) /\\ (a2 \\/ c)) /\\ ((a1 \\/ d)
               /\\ (a2 \\/ d))) /\\ ((b \\/ c) /\\ (b \\/ d))
               )");
346
347         // Simple check of translate()
348         Formula formula1 =
```

```
349         new Negation(
350         new Disjunction(
351         new Conjunction(
352         new State("a"),
353         new State("b")),
354         new Conjunction(
355         new State("c"),
356         new State("d"))
357         )
358         );
359         Formula cnfFormula1 = cnfT.translate(formula1);
360         Assert.assertEquals("", "((~a \\/ ~b) /\\ (~c \\/
                ~d))", cnfFormula1.toString());
361
362         formula1 =
363         new Negation(
364         new Conjunction(
365         new Disjunction(
366         new State("a"),
367         new State("b")),
368         new Disjunction(
369         new State("c"),
370         new State("d"))
371         )
372         );
373         Assert.assertEquals("",
374         "(((~a \\/ ~c) /\\ (~a \\/ ~d)) /\\ ((~b \\/ ~c)
               /\\ (~b \\/ ~d)))",
375         cnfT.translate(formula1).toString());
376     }
377
378     private void checkNNFTranslation(CNFTranslator cnfT,
           Formula formula, String expectedREven, String
           expectedROdd)
379     throws CloneNotSupportedException {
380         Formula formula1 = (Formula) formula.clone();
381         Assert.assertEquals("NNF of formula1 "+formula1+"
               is:", expectedREven, cnfT.translateToNNF(
              formula1, true).toString());
382         Assert.assertEquals("NNF of nFormula1 "+formula1
              +" is:", expectedROdd, cnfT.translateToNNF(
              formula1, false).toString());
383     }
384
385     private void checkCNFTranslation(CNFTranslator cnfT,
           Formula formula, String expectedR)
386     throws CloneNotSupportedException {
387         Formula formula1 = (Formula) formula.clone();
388         Formula cnfFormula1 = cnfT.translate(formula1);
```

```
389         Assert.assertEquals("CNF of formula1 "+formula1+"
                is:", expectedR, cnfFormula1.toString());
390     }
391
392     public void testKValFinder() throws
            CannotFindKException {
393         KValFinder kValFinder = new KValFinder();
394
395         /* First, checking kvals from different formulas
396          */
397         Formula formula = new True();
398         Assert.assertEquals("KValFinder returns", 0,
                kValFinder.findK(formula));
399
400         formula = new False();
401         Assert.assertEquals("KValFinder returns", 0,
                kValFinder.findK(formula));
402
403         formula = new Duration(new State("x1"), 5);
404         Assert.assertEquals("KValFinder returns", 5,
                kValFinder.findK(formula));
405
406         formula = new Negation(formula);
407         Assert.assertEquals("KValFinder returns", 4,
                kValFinder.findK(formula));
408
409         formula = new Conjunction(new Duration(new State
                ("x"), 5),
410         new Duration(new Negation(new State("x")),2));
411         Assert.assertEquals("KValFinder returns", 7,
                kValFinder.findK(formula));
412
413         formula = new Conjunction(new Negation(new
                Duration(new True(),5)),
414         new Duration(new State("x"),2));
415         Assert.assertEquals("KValFinder returns", 4,
                kValFinder.findK(formula));
416
417         formula = new Conjunction(new Duration(new State
                ("x"),2),
418         new Negation(new Duration(new True(),5)));
419         Assert.assertEquals("KValFinder returns", 4,
                kValFinder.findK(formula));
420
421         formula = new Conjunction(new Negation(new
                Duration(new True(),5)),
422         new Negation(new Duration(new True(),2)));
423         Assert.assertEquals("KValFinder returns", 1,
                kValFinder.findK(formula));
424
```

```
425         formula = new Disjunction(new Negation(new
                Duration(new True(),2)),
426         new Negation(new Duration(new State("x"),5)));
427         Assert.assertEquals("KValFinder returns", 4,
                kValFinder.findK(formula));
428
429         /* Now, checking whether isGuard and reqGuard
                work properly
430          */
431
432         try {
433             kValFinder.findK(
434             new Disjunction(
435             new True(),
436             new Negation(
437             new Chop(new True(), new False())
438             )
439             ));
440             Assert.assertTrue("KValFinder throws
                    Exception", false);
441         }
442         catch (CannotFindKException e) {
443             Assert.assertTrue("KValFinder throws
                    Exception", true);
444         }
445
446         try {
447             kValFinder.findK(new Chop(new True(), new
                    Negation(new Chop(new True(), new False())
                    )));
448             Assert.assertTrue("KValFinder throws
                    Exception", false);
449         }
450         catch (CannotFindKException e) {
451             Assert.assertTrue("KValFinder throws
                    Exception", true);
452         }
453
454
455         try {
456             kValFinder.findK(
457             new Conjunction(
458             new Disjunction(
459             new False(),
460             new Negation(
461             new Chop(new True(), new False())
462             )
463             ),
464             new Disjunction(
465             new Negation(new Duration(new True(), 3)),
```

```
466             new True()
467             )
468             ));
469             Assert.assertTrue("KValFinder throws
                    Exception. One side of disjunction
                    requires guard. This is bad. Only one side
                     of disjunction is guard. Even more bad",
                    false);
470         }
471         catch (CannotFindKException e) {
472             Assert.assertTrue("KValFinder throws
                    Exception", true);
473         }
474
475         Assert.assertEquals("In disjunctions, both
                operands must be guards to count as a guard",
476         2,
477         kValFinder.findK(
478         new Conjunction(
479         new Negation(new Chop(new True(), new False())),
480         new Disjunction(
481         new Negation(new Duration(new True(), 3)),
482         new Negation(new Duration(new True(), 3))))));
483
484
485
486         Formula unguardedF = new Negation(new Chop(new
                True(), // equiv to nnf'ed version of example
                in report
487         new Duration(new State("x"), 5)));
488         //- show that negated chops require guards
489         try {
490             formula = unguardedF;
491             kValFinder.findK(formula);
492             Assert.assertTrue("KValFinder throws
                    Exception", false);
493         }
494         catch (CannotFindKException e) {
495             Assert.assertTrue("KValFinder throws
                    Exception", true);
496         }
497         formula = new Conjunction(
498         unguardedF,
499         new Negation(new Duration(new True(), 7)));
500         // - show that dur true < n acts as guard and
                saves negated chops
501         Assert.assertEquals("Same formula, now with guard
                ", 6, kValFinder.findK(formula));
502
503
```

```
504         // next two asserts: show that non-negated outer
                chops only require guards if a subformula does
505         try {
506             formula = new Chop(
507             new Negation(
508             new Chop(
509             new Duration(new State("x"), 5),
510             new Duration(new State("y"), 2))),
511             new True());
512             kValFinder.findK(formula);
513             Assert.assertTrue("KValFinder throws
                    Exception", false);
514         }
515         catch (CannotFindKException e) {
516             Assert.assertTrue("KValFinder throws
                    Exception", true);
517         }
518
519         formula = new Chop(
520         new Conjunction(
521         new Negation(new Duration(new State("x"), 5)),
522         new Duration(new State("y"), 2)),
523         new True());
524         Assert.assertEquals("KValFinder returns", 6,
                kValFinder.findK(formula));
525
526
527         try {
528             kValFinder.findK(new Negation(new True()));
529             Assert.assertTrue("KValFinder throws
                    Exception. Formula not in NNF", false);
530         }
531         catch (IllegalStateException e) {
532             Assert.assertTrue("KValFinder throws
                    Exception. Formula not in NNF", true);
533         }
534     }
535
536     public void testStreamGobbler() throws Exception {
537         // in testShellExecutor();
538     }
539
540     public static junit.framework.Test suite() {
541         return new TestSuite(Test.class);
542     }
543 }
```

G.1.9 bmc.Test.ShellExecutorTest.java

```
1 package bmc.test;
```

```
2
3  import java.io.*;
4
5  public class ShellExecutorTest {
6
7      /* Small program to test number of arguments.
8       * If one argument arrives, a file is created on that
            location
9       * If zero or more than one argument arrive, nothing
            happens.
10      */
11     public static void main(String[] args) throws
           Exception {
12         if (args.length == 1) {
13             File testFile = new File(args[0]);
14             testFile.createNewFile();
15         }
16         else {
17             throw new IllegalArgumentException("Only one
                   parameter allowed");
18         }
19     }
20 }
```

G.1.10 bmc.Test.BMCUnitTest.java

```
1  package bmc.test;
2
3  import bmc.*;
4
5  public class BMCUnitTest {
6
7      public static junit.framework.Test suite() {
8          Main.setDebugging(false);
9          junit.framework.TestSuite allTests = new junit.
               framework.TestSuite();
10         allTests.addTest(bmc.robdd.Test.suite());
11         allTests.addTest(bmc.constraint.Test.suite());
12         allTests.addTest(bmc.formula.Test.suite());
13         allTests.addTest(bmc.literal.Test.suite());
14         allTests.addTest(bmc.parser.Test.suite());
15         allTests.addTest(bmc.simp.Test.suite());
16         allTests.addTest(bmc.tracer.Test.suite());
17         allTests.addTest(bmc.util.Test.suite());
18         return allTests;
19     }
20 }
```

G.2 Automated Black Box Test Suite

First, the source code for the program that runs the automated black box test is shown. Then in Section G.2.2 on page 223, all of the black box test cases are listed in a table.

G.2.1 BMCBBTest.java

```
1  package bmc.test;
2
3  import java.io.*;
4  import java.util.*;
5  import java.util.regex.*;
6  import java.text.*;
7  import bmc.*;
8  import bmc.tracer.*;
9  import bmc.util.*;
10
11 public class BMCBBTest {
12
13     private static final String RESULT_VALID_STR = "valid
           ";
14     private static final String RESULT_INVALID_STR = "
           invalid";
15
16     File[] caseFiles;
17     String[] cases;
18     int[] expectedResults;
19     String[] options;
20     String[][] optionValues; // values optionValues[i]
           correspond to options[i]
21     File tmpFile1;
22     File outputFolder;
23     Main main;
24
25     public BMCBBTest(LineNumberReader config, String[]
           shellCmd, String solverCmd) throws IOException,
           ParseException {
26         this.main = new Main(shellCmd, solverCmd,
               StateTraceFrame.DISPLAY_MODE_NONE);
27         Main.setDebugging(false);
28         Main.setPrintOut(false);
29         // Parse configuration
30         String line;
31         ArrayList caseFileList = new ArrayList();
32         ArrayList expectedResultList = new ArrayList();
33         while ((line = config.readLine()) != null) {
34             if (line.length() == 0) {
35                 continue;
36             }
```

```
37              StringTokenizer stok = new StringTokenizer(
                    line);
38              if (stok.countTokens() != 2) {
39                  throw new ParseException("Malformed
                        configuration file", config.
                        getLineNumber());
40              }
41              File caseFile = new File(stok.nextToken());
42              if (!caseFile.exists()) {
43                  throw new FileNotFoundException(caseFile.
                        toString());
44              }
45              caseFileList.add(caseFile);
46              String expectedResult = stok.nextToken();
47              if (RESULT_VALID_STR.equalsIgnoreCase(
                    expectedResult)) {
48                  expectedResultList.add(new Integer(
                        HySatSolutionParser.STATUS_UNSAT));
49              }
50              else if (RESULT_INVALID_STR.equalsIgnoreCase(
                    expectedResult)) {
51                  expectedResultList.add(new Integer(
                        HySatSolutionParser.STATUS_SAT));
52              }
53              else {
54                  throw new ParseException("Unknown
                        expected result type: "+expectedResult
                        ,
55                  config.getLineNumber());
56              }
57          }
58          this.caseFiles = (File[]) caseFileList.toArray(
                new File[0]);
59          this.expectedResults = new int[expectedResultList
                .size()];
60          for (int i=0; i<expectedResults.length; i++) {
61              this.expectedResults[i] = ((Integer)
                    expectedResultList.get(i)).intValue();
62          }
63
64          // Read cases from file
65          this.cases = new String[caseFiles.length];
66          for(int i=0; i<cases.length; i++) {
67              FileReader reader = new FileReader(caseFiles[
                    i]);
68              StringBuffer buffer1 = new StringBuffer();
69              char[] buffer2 = new char[256];
70              int r;
71              while ((r = reader.read(buffer2)) >= 0) {
72                  buffer1.append(buffer2, 0, r);
```

```
73              }
74              cases[i] = buffer1.toString();
75              reader.close();
76          }
77          this.options = new String[]{ "outputFormat", "
                polarityOpt", "nnf",
78          "dcSimpLevel", "fRecognition"};
79          optionValues = new String[][] {
80              {"zolcs", "dimacs"}, // outputFormat
81              {"true", "false"}, // polarityOpt
82              {"true", "false"}, // nnf
83              {"0", "1", "2"}, // dcSimpLevel
84              {"id", "syntactic", "semantic"} //
                    fRecognition
85          };
86          this.tmpFile1 = new File("bmcTmp1.txt");
87          this.tmpFile1.createNewFile();
88          File currFolder = new File("");
89          Calendar currDate = new GregorianCalendar();
90          this.outputFolder = new File(currFolder.
                getCanonicalPath(),
91          "output_"+System.currentTimeMillis());
92          this.outputFolder.mkdir();
93          System.out.println("Using outputFolder=["+
                outputFolder+"]");
94      }
95
96      public void execTest() throws Throwable {
97          int testCount = 0;
98          int testOkCount = 0;
99          int[] optionIdx = new int[options.length];
100         for (int i=0; i<caseFiles.length; i++) {
101             System.out.println();
102             System.out.println
                    ("----------------------------------------");
103             System.out.println("-------- NEW CASE FILE
                    #"+(i+1)+" -------");
104             boolean hasMoreOptCombinations = true;
105             while (hasMoreOptCombinations) {
106                 System.out.println
                        ("----------------------------------------")
                        ;
107                 PrintWriter writer = new PrintWriter(new
                        FileWriter(tmpFile1));
108
109                 StringBuffer errors = new StringBuffer();
110                 StringBuffer trace = new StringBuffer();
111
112                 // Write this combination of options
113                 for (int j=0; j<options.length; j++) {
```

```
114                     String val = ":- set "+options[j]+" =
                            "+optionValues[j][optionIdx[j
                            ]]+".\n";
115                     writer.print(val);
116                     writer.flush();
117                     trace.append(val);
118                 }
119                 //writer.println();
120                 trace.append("\n");
121
122                 // Write case file content
123                 writer.print(cases[i]);
124                 writer.flush();
125                 trace.append(cases[i]);
126
127                 writer.close();
128
129                 int actualResult;
130
131                 try {
132                     // Translate case
133                     List tasks = main.parse(new
                            BufferedReader(new FileReader(
                            tmpFile1)));
134                     if (tasks.size() != 1) {
135                         throw new
                                IllegalArgumentException("One
                                task expected in file "+
                                caseFiles[i].getName());
136                     }
137                     if (!(tasks.get(0) instanceof Goal))
                            {
138                         throw new
                                IllegalArgumentException("No
                                goals in file");
139                     }
140                     actualResult = main.runGoal((Goal)
                            tasks.get(0));
141                 }
142                 catch (Exception e) {
143                     errors.append("Error: ").append(e).
                            append("\n");
144                     actualResult = HySatSolutionParser.
                            STATUS_ERROR;
145                 }
146
147                 testCount++;
148                 System.out.print("Completed test case "+
                        testCount);
149                 if (expectedResults[i] == actualResult) {
```

```
150                     testOkCount++;
151                     System.out.println(" successfully.");
152                 }
153                 else {
154                     System.out.println(" unsuccessfully
                        .");
155                     errors.append("Received: ").append(
                        resultToString(actualResult)).
                        append("\n");
156                     errors.append("Expected: ").append(
                        resultToString(expectedResults[i])
                        ).append("\n");
157                     errors.append("Case: \n\n");
158                     System.out.println(errors);
159                 }
160                 File outputFile = new File(outputFolder,
                    "case_"+testCount+".pl");
161                 tmpFile1.renameTo(outputFile);
162                 System.out.println((trace));
163 
164                 // Advance to new combination of options
165                 int j=0;
166                 while (true) {
167                     optionIdx[j]++;
168                     if (optionIdx[j] >= optionValues[j].
                        length) {
169                         optionIdx[j] = 0;
170                         j++;
171                         if (j>=optionIdx.length) {
172                             hasMoreOptCombinations =
                                false;
173                             break;
174                         }
175                     }
176                     else {
177                         break;
178                     }
179                 }
180             }
181         }
182         System.out.println("\n
            ****************************************");
183         System.out.println("* Test performed, succes rate
            : "+testOkCount+"/"+testCount);
184         System.out.println
            ("****************************************");
185     }
186 
187     public static void printUsage() {
```

```
188         System.out.println("Usage: bmc.BMCBBTest [-
                shellCmd CMD] -solverCmd CMD CONFIG_FILE");
189     }
190
191     public static void main(String[] args) throws
            Throwable {
192         String[] shellCmd = null;
193         String solverCmd = null;
194         String configFileName = null;
195         Pattern pattern = Pattern.compile(" ");
196         for (int i=0; i<args.length; i++) {
197             if (args[i].equals("-shellCmd")) {
198                 if ((++i) == args.length) {
199                     printUsage();
200                     return;
201                 }
202                 shellCmd = pattern.split(args[i]);
203             }
204             else if (args[i].equals("-solverCmd")) {
205                 if ((++i) == args.length) {
206                     printUsage();
207                     return;
208                 }
209                 solverCmd = args[i];
210             }
211             else {
212                 if (configFileName != null) {
213                     printUsage();
214                     return;
215                 }
216                 configFileName = args[i];
217             }
218         }
219         if (configFileName == null || solverCmd == null)
                {
220             printUsage();
221             return;
222         }
223
224         LineNumberReader input = new LineNumberReader(new
                FileReader(configFileName));
225         BMCBBTest test = new BMCBBTest(input, shellCmd,
                solverCmd);
226         input.close();
227         test.execTest();
228     }
229
230     public static String resultToString(int result) {
231         switch (result) {
232             case HySatSolutionParser.STATUS_ERROR:
```

```
233                 return "error";
234             case HySatSolutionParser.STATUS_SAT:
235                 return "invalid";
236             case HySatSolutionParser.STATUS_UNSAT:
237                 return "valid";
238             default:
239                 return "???";
240         }
241     }
242 }
```

G.2.2 Test Cases

Case	Formula	k	Valid	Description
1.1	$\texttt{true}$	1	Yes	DCTrue
2.1	$\texttt{false}$	1	No	DCFalse
3.1	$\int x1 \geq 2$	1	No	DCDuration, SAState
3.2	$\int x1 \geq 2$	2	No	DCDuration, SAState
3.3	$\int \texttt{true} \geq 2$	1	No	DCDuration, SATrue
3.4	$\int \texttt{true} \geq 2$	2	No	DCDuration, SATrue
3.5	$\int \texttt{false} \leq 2$	1	Yes	DCDuration, SAFalse
3.6	$\int x1 \vee \neg x1 \leq 1$	1	Yes	DCDuration, SADisjunction
3.7	$\int x1 \vee \neg x1 \leq 1$	2	No	DCDuration, SADisjunction
3.8	$\int x1 \vee \texttt{false} \geq 1$	1	No	DCDuration, SADisjunction
3.9	$\int x1 \wedge \neg x1 \leq 1$	1	Yes	DCDuration, SAConjunction
3.10	$\int x1 \wedge \neg x1 \geq 1$	1	No	DCDuration, SAConjunction
3.11	$\int x1 \wedge \texttt{true} \geq 1$	1	No	DCDuration, SAConjunction
3.12	$\int \neg x1 \geq 2$	1	No	DCDuration, SANegation
3.13	$\int \neg(x1 \wedge \neg x1) \leq 1$	1	Yes	DCDuration, SANegation
3.14	$\int \neg\texttt{false} \geq 1$	1	No	DCDuration, SAConjunction
3.15	$\int (x1 \Leftrightarrow \texttt{true}) \vee\ x1 \leq 1$	1	Yes	DCDuration, SABiimplication
3.16	$\int (x1 \Rightarrow \texttt{false}) \vee x1 \leq 1$	2	No	DCDuration, SAImplication
3.17	$\int x1 \vee \neg x1 < 2$	1	Yes	DCDuration, $<$
3.18	$\int x1 \vee \neg x1 < 2$	2	No	DCDuration, $<$
3.19	*Case 3.6*	—	—	—
3.20	$\int \texttt{true} > 1$	1	Yes	DCDuration, $>$
3.21	$\int \texttt{true} > 1$	2	No	DCDuration, $>$
3.22	$\neg \int \texttt{true} \geq 1 \vee \int \texttt{true} = 1$	1	Yes	DCDuration, $=$

Case	Formula	k	Valid	Description
3.23	$\neg \int \texttt{true} \geq 1 \vee \int \texttt{true} = 1$	2	No	DCDuration, =
4.1	$\neg \texttt{true}$	1	No	DCNegation, DCTrue
4.2	$\neg \texttt{true}$	1	Yes	DCNegation, DCFalse
4.3	$\neg \int x1 \wedge \neg x1 \geq 2$	1	Yes	DCNegation, DCDuration
4.4	$\neg(\texttt{true} \wedge \texttt{false})$	1	Yes	DCNegation, DCConjunction
4.5	$\neg(\texttt{true} \vee \texttt{false})$	1	No	DCNegation, DCDisjunction
4.6	$\neg(\int \texttt{true} \geq 1 \frown \int \texttt{true} \geq 1)$	1	Yes	DCNegation, DCChop
4.7	$\neg(\int \texttt{true} \geq 1 \frown \int \texttt{true} \geq 1)$	2	False	DCNegation, DCChop
5.1	$\texttt{true} \wedge \texttt{false}$	1	False	DCConjunction, DCTrue, DCFalse
5.2	$\int x1 \leq 1 \wedge \int \neg x1 \leq 1$	1	Yes	DCConjunction, DCDuration
5.3	$\int x1 \leq 1 \wedge \int \neg x1 \leq 1$	3	No	DCConjunction, DCDuration
5.4	$\texttt{true} \wedge \neg \texttt{true}$	1	No	DCConjunction, DCNegation
5.5	$\texttt{true} \wedge (\texttt{false} \vee \texttt{true})$	1	Yes	DCConjunction, DCDisjunction
5.6	$\texttt{true} \wedge (\int \texttt{true} < 1 \frown \int x1 < 1)$	2	No	DCConjunction, DCChop
6.1	$\texttt{true} \vee \texttt{false}$	1	Yes	DCDisjunction, DCTrue, DCFalse
6.2	$\int x1 \leq 1 \vee \int \neg\ x1 \leq 11$	1	Yes	DCDisjunction, DCDuration
6.3	$\int x1 \geq 1 \vee \int \neg x1 \leq 1 \wedge \texttt{true}$	1	Yes	DCDisjunction, DCConjunction
6.4	$\int x1 \geq 1 \vee \neg \int x1 \geq 1$	1	Yes	DCDisjunction, DCNegation
6.5	$\int x1 \geq 1 \vee (\int \neg x1 \geq 1 \frown \int \neg x1 \geq 1)$	2	No	DCDisjunction, DCChop
7.1	$\texttt{true} \frown \texttt{false}$	1	No	DCChop, DCTrue, DCFalse
7.2	$\texttt{true} \frown \texttt{true}$	1	Yes	DCChop, DCTrue
7.3	$\int \texttt{true} < 1 \frown \texttt{true}$	1	Yes	DCChop, DCDuration
7.4	*Case 4.6, 4.7*	—	—	DCChop, DCNegation, DCDuration
7.5	$\int \texttt{true} < 1 \frown (\texttt{true} \wedge \texttt{false})$	1	No	DCChop, DCConjunction
7.6	$\int \texttt{true} < 1 \frown (\texttt{true} \wedge \texttt{true})$	1	Yes	DCChop, DCConjunction

Case	Formula	k	Valid	Description
7.7	$\int true \geq 1 \frown (\texttt{false} \vee \texttt{true})$	1	Yes	DCChop, DCDisjunction
7.8	$\texttt{true} \frown \neg\texttt{true}$	1	No	DCChop, DCNegation
8.1	*Case 4.6, 4.7*	—	—	Reoccurence of DCFormulas, same polarity
8.2	$\neg(\int x1 \geq 1 \frown \neg \int x1 \geq 1)$	1	No	Reoccurence of DCFormulas, opposite polarity
8.3	$\int x1 \wedge x2 \wedge \neg\neg(x1 \wedge x2) \geq 1$	2	No	Reoccurence of SAFormulas, same polarity
8.4	$\neg \int x1 \wedge x2 \wedge \neg(x1 \wedge x2) \geq 1$	2	Yes	Reoccurence of DCFormulas, opposite polarity
8.5	$\Box(\int \texttt{true} \geq 1 \frown \int \texttt{true} \geq 1)$	2	No	$\Box$
8.6	$\int x1 < 2$	2	No	(Without $\Diamond$)
8.7	$\Diamond \int x1 < 2$	2	No	$\Diamond$
8.8	$\neg \int x1 \geq 2$	1	Yes	Deep durations
8.9	$\neg \int x1 \geq 2$	2	No	Deep durations
8.10	$\neg \int x1 \geq 2$	1	Yes	FindK disabled
8.11	$\neg \int x1 \geq 2$	1	No	FindK enabled
8.12	$\neg(\int \texttt{true} \geq 1 \frown \int \texttt{true} \geq 1)$	1	No	FindK enabled
8.13	$\neg(\int x1 \geq 1 \wedge \int \neg x1 \geq 1)$	1	No	FindK enabled
8.14	$\neg(\int \texttt{false} \geq 1 \vee \int \texttt{true} \geq 2)$	1	No	FindK enabled
8.15	$l < 3 \Rightarrow \int \texttt{true} < 2$	1	No	FindK enabled
8.16	$l < 3 \Rightarrow \int \texttt{true} < 2$	1	Yes	FindK enabled
9.1	$l > 1 \Rightarrow \int \texttt{true} \geq 1$	2	Yes	DCImplication
9.2	$l > 1 \Rightarrow \int \texttt{true} \geq 3$	3	No	DCImplication
9.3	$l > 1 \Leftrightarrow \int \texttt{true} \geq 2$	2	Yes	DCBiimplication
9.4	$l > 1 \Leftrightarrow \int \texttt{true} \geq 1$	2	No	DCBiimplication
9.5	*Case 9.1 - 9.4*	—	—	Restriction on length
10.1	$\neg(\int x1 \wedge x2 \geq 2 \wedge \int x1 \geq 1)$	1	Yes	DCSimp 4.12
10.2	$\neg(\int x1 \wedge x2 \geq 2 \wedge \int x1 \geq 1)$	2	No	DCSimp 4.12

Case	Formula	k	Valid	Description
10.3	$\int x1 \wedge x2 \geq 2 \wedge \neg \int x1 \geq 1$	2	No	DCSimp 4.13
10.4	$\neg \int x1 \wedge x2 \geq 3 \wedge \neg \int x1 \geq 1)$	1	Yes	DCSimp 4.14
10.5	$\neg \int x1 \wedge x2 \geq 3 \wedge \neg \int x1 \geq 1)$	2	No	DCSimp 4.14
10.6	$\neg(\int x1 \wedge x2 \geq 2 \vee \int x1 \geq 1)$	1	No	DCSimp 4.15
10.7	$\neg(\int x1 \wedge x2 \geq 2 \vee \neg \int x1 \geq 1)$	1	No	DCSimp 4.16
10.8	$\neg \int x1 \wedge x2 \geq 3 \vee \neg \int x1 \geq 2$	2	Yes	DCSimp 4.17
10.9	*Case 4.1*	—	—	DCSimp 4.1
10.10	*Case 4.2*	—	—	DCSimp 4.2
10.11	$\neg\neg(l > 1 \Rightarrow \int \texttt{true} \geq 1)$	2	Yes	DCSimp 4.3
10.12	*Case 5.1*	—	—	DCSimp 4.4
10.13	*Case 5.6*	—	—	DCSimp 4.5
10.14	*Case 6.1*	—	—	DCSimp 4.6
10.15	$\texttt{true} \vee (\texttt{true} \Leftrightarrow \texttt{false})$	1	Yes	DCSimp 4.7
10.16	*Case 7.1*	—	—	DCSimp 4.8
10.17	$\texttt{false} \frown \texttt{true}$	1	No	DCSimp 4.9
10.18	*Case 3.5*	—	—	DCSimp 4.10
10.19	$\neg \int \texttt{true} \geq 5$	1	Yes	DCSimp 4.11
10.20	$l > 1 \Rightarrow \int \neg \texttt{true} \geq 1$	2	No	SASimp 4.18
10.21	$l > 1 \Rightarrow \int \neg \texttt{false} \geq 1$	2	Yes	SASimp 4.19
10.22	$l > 1 \Rightarrow \int \neg\neg \texttt{true} \geq 1$	2	Yes	SASimp 4.20
10.23	$l > 1 \Rightarrow \int \texttt{false} \wedge \texttt{true} \geq 1$	2	No	SASimp 4.21
10.24	$l > 1 \Rightarrow \int \texttt{true} \wedge \texttt{true} \geq 1$	2	Yes	SASimp 4.22
10.25	$l > 1 \Rightarrow \int \texttt{false} \vee (x1 \vee \neg x1) \geq 1$	2	Yes	SASimp 4.23
10.26	$l > 1 \Rightarrow \int \texttt{true} \wedge x1 \geq 1$	2	Yes	SASimp 4.24
10.27	$l > 1 \Rightarrow \int (x1 \wedge x2) \wedge x1 \geq 1$	2	No	SASimp 4.25

Case	Formula	k	Valid	Description
10.28	$l > 1 \Rightarrow \int(x1 \wedge x2) \wedge \neg x1 \leq 1$	2	Yes	SASimp 4.26
10.29	$l > 1 \Rightarrow \int(x1 \wedge x2) \vee x1 \leq 1$	2	No	SASimp 4.27
10.30	$l > 1 \Rightarrow \int\neg(x1 \wedge x2) \vee x1 \geq 1$	2	Yes	SASimp 4.28

G.3 Manual Black Box Test Suite

This appendix describes how to carry out the manual black box tests. It consists of four sections, each specifying the test of one aspect of the BMC/DCValidator.

G.3.1 Command Line Interface

No arguments

Run the BMC/DCValidator without any command line arguments. The program is expected to print out *usage* information and do nothing further.

With or Without Command Arguments, Input File

Run the BMC/DCValidator on the input file from Section G.3.2, first without specifying command parameters. (This test cannot be performed using Cygwin, since Cygwin is exactly the reason for the possibility of supplying command parameters.)

Next, run the BMC/DCValidator on the same input file, now specifying appropriate command parameters.

DisplayTrace

This is tested in Section G.3.4 on the next page.

G.3.2 Shells

Remove any file called "2shell.pl" from the current directory. Run the following input file through the BMC/DCValidator. After the program has finished, a new file "2shell.pl" should have been created in the current directory.

```
1 :- shell("cat shell.pl shell.pl > 2shell.pl").
```

G.3.3 Large, Valid Formulas and Use of "outputFolder"

Run the following input file through the BMC/DCValidator. Expect the BMC/DCValidator to print out results for two validations:

1. Formula is "Valid", constraint system file is streamed into current folder "output".
2. Formula is "Valid", constraint system file is streamed into folder "output" in the current folder.

Additionally, the BMC/DCValidator should print out progress statements during the validation process.

Finally, the BMC/DCValidator should print out running times for frontend and backend. Frontend times are expected to be in approximately 1 second, backend times to be 3-5 seconds.

```
1  :- state gas.
2  :- state flame.
3  :- set k = 30.
4  :- leak =^= gas /\ ~flame.
5  :- des1 =^= all( dur leak > 1 -> dur ~(leak) >= 1 ).
6  :- des2 =^= all((dur leak >= 1 ; dur ~(leak) >= 1 ; dur
       leak >= 1) -> l >= 32).
7  :- gbreq(n) =^= all(l <= 30 -> dur leak <= n).
8  :- gbsafe(n) =^= (des1 /\ des2) -> gbreq(n).
9
10 :- set outputFolder = "".
11 :- goal gbsafe1 gbsafe(1).
12 :- set outputFolder = "output".
13 :- goal gbsafe1 gbsafe(1).
```

G.3.4 Large, Invalid Formulas and Trace GUI

Run the following input file through the BMC/DCValidator, using command line parameter "displayTrace". Expect the BMC/DCValidator to state the formula as being invalid, and to show a trace for the the two state variables `x1` and `x2`.

The trace should show, that `x1` is `true` in an interval of at least length = 20, eventually followed by an interval of at least length = 20 where `x1` is `false`.

`x2` though, has no influence on validity of the formula and should be marked using a grey colour, symbolising a *wildcard*.

```
1  :- set k = 50.
2  :- state x1.
3  :- goal invalid ~(dur x1 >= 20 ; dur ~x1 >= 20).
```

G.3.5 Use of "findk"

Run the program with the following input file. Expect the BMC/DCValidator to state the it is impossible to find a value for k, since the formula is not in class of formulas mentioned in Chapter 5 on page 54.

```
1  :- set k = 2.
2  :- set findk = true.
3  :- goal invalid dur true >= 1 ; dur true >= 1.
```

Appendix H

Benchmarks

H.1 BMCBenchmark.java

This is the source code for running the benchmark test.

```
1  package bmc.benchmark;
2
3  import java.io.*;
4  import java.util.*;
5  import java.util.regex.*;
6  import java.text.*;
7  import bmc.*;
8  import bmc.tracer.*;
9  import bmc.util.*;
10
11 public class BMCBenchmark {
12
13     File[] caseFiles;
14     String[] cases;
15     int[] expectedResults;
16     String[] options;
17     String[][] optionValues; // values optionValues[i]
           correspond to options[i]
18     File tmpFile1;
19     File outputFolder;
20     Main main;
21
22     public BMCBenchmark(LineNumberReader config, String[]
           shellCmd, String solverCmd) throws IOException,
           ParseException {
23         this.main = new Main(shellCmd, solverCmd,
               StateTraceFrame.DISPLAY_MODE_NONE);
24         Main.setDebugging(false);
25         // Parse configuration
26         String line;
27         ArrayList caseFileList = new ArrayList();
```

```
28          ArrayList expectedResultList = new ArrayList();
29          while ((line = config.readLine()) != null) {
30              if (line.length() == 0) {
31                  continue;
32              }
33              StringTokenizer stok = new StringTokenizer(
                    line);
34              if (stok.countTokens() != 1) {
35                  throw new ParseException("Malformed
                        configuration file", config.
                        getLineNumber());
36              }
37              File caseFile = new File(stok.nextToken());
38              if (!caseFile.exists()) {
39                  throw new FileNotFoundException(caseFile.
                        toString());
40              }
41              caseFileList.add(caseFile);
42          }
43          this.caseFiles = (File[]) caseFileList.toArray(
                new File[0]);
44
45          // Read cases from file
46          this.cases = new String[caseFiles.length];
47          for(int i=0; i<cases.length; i++) {
48              FileReader reader = new FileReader(caseFiles[
                    i]);
49              StringBuffer buffer1 = new StringBuffer();
50              char[] buffer2 = new char[256];
51              int r;
52              while ((r = reader.read(buffer2)) >= 0) {
53                  buffer1.append(buffer2, 0, r);
54              }
55              cases[i] = buffer1.toString();
56              reader.close();
57          }
58          this.options = new String[]{ "outputFormat", "
                polarityOpt", "nnf",
59          "dcSimpLevel", "fRecognition"};
60          optionValues = new String[][] {
61              {"zolcs", "dimacs"}, // outputFormat
62              {"true", "false"}, // polarityOpt
63              {"true", "false"}, // nnf
64              {"0", "1", "2"}, // dcSimpLevel
65              {"id", "syntactic", "semantic"}, //
                    fRecognition
66              {"32", "50"} // k
67          };
68          this.tmpFile1 = new File("bmcTmp1.txt");
69          this.tmpFile1.createNewFile();
```

```
70          File currFolder = new File("");
71      }
72
73      public void execTest() throws Throwable {
74          int testCount = 0;
75          int testOkCount = 0;
76          int[] optionIdx = new int[options.length];
77          int goalNo = 1;
78          for (int i=0; i<caseFiles.length; i++) {
79              boolean hasMoreOptCombinations = true;
80              while (hasMoreOptCombinations) {
81                  PrintWriter writer = new PrintWriter(new
                        FileWriter(tmpFile1));
82
83                  // Write this combination of options
84                  for (int j=0; j<options.length; j++) {
85                      String val = ":- set "+options[j]+" =
                            "+optionValues[j][optionIdx[j
                            ]]+".\n";
86                      writer.print(val);
87                      writer.flush();
88                  }
89
90                  // Write case file content
91                  String newGoalName = "g"+(goalNo++);
92                  Pattern pattern = Pattern.compile("
                        GOALNAME");
93                  Matcher matcher = pattern.matcher(cases[i
                        ]);
94                  writer.print(matcher.replaceAll(
                        newGoalName));
95                  System.out.println("Goal: "+newGoalName);
96                  writer.flush();
97
98                  writer.close();
99
100                 // Translate case
101                 List tasks = main.parse(new
                        BufferedReader(new FileReader(tmpFile1
                        )));
102                 main.runGoal((Goal) tasks.get(0));
103
104                 // Advance to new combination of options
105                 int j=0;
106                 while (true) {
107                     optionIdx[j]++;
108                     if (optionIdx[j] >= optionValues[j].
                            length) {
109                         optionIdx[j] = 0;
110                         j++;
```

```
111                         if (j>=optionIdx.length) {
112                             hasMoreOptCombinations =
                                    false;
113                             break;
114                         }
115                     }
116                     else {
117                         break;
118                     }
119                 }
120             }
121         }
122     }
123
124     public static void printUsage() {
125         System.out.println("Usage: bmc.BMCBenchmark [-
                shellCmd CMD] -solverCmd CMD CONFIG_FILE");
126     }
127
128     public static void main(String[] args) throws
            Throwable {
129         String[] shellCmd = null;
130         String solverCmd = null;
131         String configFileName = null;
132         Pattern pattern = Pattern.compile(" ");
133         for (int i=0; i<args.length; i++) {
134             if (args[i].equals("-shellCmd")) {
135                 if ((++i) == args.length) {
136                     printUsage();
137                     return;
138                 }
139                 shellCmd = pattern.split(args[i]);
140             }
141             else if (args[i].equals("-solverCmd")) {
142                 if ((++i) == args.length) {
143                     printUsage();
144                     return;
145                 }
146                 solverCmd = args[i];
147             }
148             else {
149                 if (configFileName != null) {
150                     printUsage();
151                     return;
152                 }
153                 configFileName = args[i];
154             }
155         }
156         if (configFileName == null || solverCmd == null)
                {
```

```
157                 printUsage();
158                 return;
159             }
160 
161             LineNumberReader input = new LineNumberReader(new
                    FileReader(configFileName));
162             BMCBenchmark test = new BMCBenchmark(input,
                    shellCmd, solverCmd);
163             input.close();
164             test.execTest();
165         }
166 
167 }
```

H.2 BenchmarkParser.java

This is the source code for extracting benchmark results to a CSV file or a LaTeXtable.

```
1  package bmc.benchmark;
2 
3  import java.io.*;
4  import java.util.regex.*;
5 
6  public class BenchmarkParser {
7      // Input files
8      private File configFile;
9      private File resultFolder;
10 
11     // Output file
12     private File outputFile;
13 
14     private String problemName;
15 
16     private String separator;
17     private String lineTerminator;
18 
19     private static final Pattern RESULT_PATTERN = Pattern
           .compile("Goal: (\\w+?)\\s*#+\\s*Formula is (valid
           |invalid)\\s*Frontend time: \\s*(\\d+.\\d+) s\\s*
           Backend time :\\s*(\\d+.\\d+) s.*?#+\\s*", Pattern
           .DOTALL);
20     private static final Pattern HYSAT_ERROR_PATTERN =
           Pattern.compile("Goal: (\\w+)\\s*Syntax error: No
           status line at line \\d+\\s*");
21     private static final Pattern MEMORY_ERROR_PATTERN =
           Pattern.compile("Goal: (\\w+)\n.*java.lang.
           OutOfMemoryError.*Goal: .*",Pattern.DOTALL);
```

```
22      private static final Pattern MEMORY_ERROR_PATTERN_END
            = Pattern.compile("Goal: (\\w+)\n.*java.lang.
        OutOfMemoryError.*",Pattern.DOTALL);
23      private static final Pattern OTHER_ERROR_PATTERN =
            Pattern.compile("Goal: (\\w+)\n.*Goal: .*",Pattern
            .DOTALL);
24      private static final Pattern OTHER_ERROR_PATTERN_END
            = Pattern.compile("Goal: (\\w+)\n.*",Pattern.
        DOTALL);
25      private static final Pattern KEYVAL_PATTERN = Pattern
            .compile("c (\\w*)=(.*)");
26      private static final Pattern PROBLEM_PATTERN =
            Pattern.compile("p (?:zolcs|cnf) (\\d+) (\\d+)\\s
            *");
27
28
29      public BenchmarkParser(File configFile, File
            resultFolder, File outputFile, String problemName,
30      String separator, String lineTerminator) {
31          this.configFile = configFile;
32          this.resultFolder = resultFolder;
33          this.outputFile = outputFile;
34          this.problemName = problemName;
35          this.separator = separator;
36          this.lineTerminator = lineTerminator;
37      }
38
39      public void parse() throws IOException {
40          BufferedReader configReader = new BufferedReader(
                new FileReader(configFile));
41          PrintWriter output = new PrintWriter(new
                BufferedWriter(new FileWriter(outputFile)),
                true);
42          StringBuffer buffer = new StringBuffer();
43          String line;
44          int state;
45          while ((line = configReader.readLine()) != null)
                {
46              buffer.append(line).append("\n");
47
48              String goalId = null;
49              String result = null;
50              String frontendTime = "";
51              String backendTime = "";
52
53              Matcher matcher = RESULT_PATTERN.matcher(
                    buffer);
54              if (matcher.matches()) {
55                  goalId = matcher.group(1);
56                  result = matcher.group(2);
```

```
57                  frontendTime = matcher.group(3);
58                  backendTime = matcher.group(4);
59                  buffer = new StringBuffer();
60              }
61              else {
62                  matcher = HYSAT_ERROR_PATTERN.matcher(
                        buffer);
63                  if (matcher.matches()) {
64                      goalId = matcher.group(1);
65                      result = "HySat error";
66                      buffer = new StringBuffer();
67                  }
68                  else {
69                      matcher = MEMORY_ERROR_PATTERN.
                            matcher(buffer);
70                      if (matcher.matches()) {
71                          goalId = matcher.group(1);
72                          result = "Out of memory";
73                          // Give line starting with "Goal"
                                back to buffer
74                          buffer = new StringBuffer();
75                          buffer.append(line).append("\n");
76                      }
77                      else {
78                          matcher = OTHER_ERROR_PATTERN.
                                matcher(buffer);
79                          if (matcher.matches()) {
80                              goalId = matcher.group(1);
81                              result = "Other error";
82                              // Give line starting with "
                                    Goal" back to buffer
83                              buffer = new StringBuffer();
84                              buffer.append(line).append("\
                                    n");
85                          }
86                      }
87                  }
88              }
89
90              if (goalId != null) {
91                  parseResults(goalId, result, frontendTime
                        , backendTime, output);
92              }
93          }
94          if (buffer.length() > 0) {
95              Matcher matcher = MEMORY_ERROR_PATTERN_END.
                    matcher(buffer);
96              if (matcher.matches()) {
97                  String goalId = matcher.group(1);
98                  String result = "Out of memory";
```

```
99                  parseResults(goalId, result, "", "",
                        output);
100             }
101             else {
102                 matcher = OTHER_ERROR_PATTERN_END.matcher
                        (buffer);
103                 if (matcher.matches()) {
104                     String goalId = matcher.group(1);
105                     String result = "Other error";
106                     parseResults(goalId, result, "", "",
                            output);
107                 }
108             }
109         }
110         configReader.close();
111         output.close();
112     }
113
114     private void parseResults(String goalId, String
            result, String frontendTime, String backendTime,
            PrintWriter output)
115     throws IOException {
116         String resultLine = null;
117         String k = "";
118         String formula = "";
119         String polarityOpt = "";
120         String dcSimpLevel = "";
121         String nnf = "";
122         String outputFormat = "";
123         String fRecognition = "";
124         String noVars = "";
125         String noConstraints = "";
126
127         // Result file should be either .zolcs or .dimacs
                but we don't know which
128         File resultFile = new File(resultFolder, goalId
                +".zolcs");
129         if (!resultFile.exists()) {
130             resultFile = new File(resultFolder, goalId+".
                    dimacs");
131         }
132         if (resultFile.exists()) {
133             BufferedReader resultReader = new
                    BufferedReader(new FileReader(resultFile))
                    ;
134             while ((resultLine = resultReader.readLine())
                        != null) {
135                 Matcher matcher = KEYVAL_PATTERN.matcher(
                            resultLine);
136                 if (matcher.matches()) {
```

```
137                     String key = matcher.group(1);
138                     if ("k".equals(key)) {
139                         k = matcher.group(2);
140                     }
141                     else if ("polarityOpt".equals(key)) {
142                         polarityOpt = matcher.group(2);
143                     }
144                     else if ("dcSimpLevel".equals(key)) {
145                         dcSimpLevel = matcher.group(2);
146                     }
147                     else if ("nnf".equals(key)) {
148                         nnf = matcher.group(2);
149                     }
150                     else if ("outputFormat".equals(key))
                            {
151                         outputFormat = matcher.group(2);
152                     }
153                     else if ("fRecognition".equals(key))
                            {
154                         fRecognition = matcher.group(2);
155                     }
156                 }
157                 else {
158                     matcher = PROBLEM_PATTERN.matcher(
                            resultLine);
159                     if (matcher.matches()) {
160                         noVars = matcher.group(1);
161                         noConstraints = matcher.group(2);
162                     }
163                     break;
164                 }
165             }
166             resultReader.close();
167         }
168         else {
169             System.out.println("WARNING: File "+
                    resultFile+" does not exist.");
170         }
171         output.print(k); output.print(separator);
172         output.print(polarityOpt); output.print(separator
                );
173         output.print(outputFormat); output.print(
                separator);
174         output.print(fRecognition); output.print(
                separator);
175         output.print(nnf); output.print(separator);
176         output.print(dcSimpLevel); output.print(separator
                );
177         output.print(problemName); output.print(separator
                );
```

```
178         output.print(noVars); output.print(separator);
179         output.print(noConstraints); output.print(
                separator);
180         output.print(frontendTime); output.print(
                separator);
181         output.print(backendTime); output.print(separator
                );
182         output.print(result); output.print(lineTerminator
                );
183         output.flush();
184     }
185
186
187     public static void printUsage() {
188         System.out.println("Usage: BenchmarkParser -
                config <configFilename> -problem <problemName>
                 -output <outputFilename> [-folder <
                resultFolder>] [-separator <sep>] [-
                lineTerminator <term>]");
189     }
190
191     public static void main(String[] args) throws
            Exception {
192         if (args.length % 2 != 0) {
193             printUsage();
194             return;
195         }
196         File configFile = null;
197         String problemName = null;
198         File outputFilename = null;
199         File resultFolder = null;
200         String separator = ";";
201         String lineTerminator = "\n";
202         for (int i=0; i<args.length; i++) {
203             if ("-config".equals(args[i])) {
204                 configFile = new File(args[++i]);
205             }
206             else if ("-problem".equals(args[i])) {
207                 problemName = args[++i];
208             }
209             else if ("-output".equals(args[i])) {
210                 outputFilename = new File(args[++i]);
211             }
212             else if ("-folder".equals(args[i])) {
213                 resultFolder = new File(args[++i]);
214             }
215             else if ("-separator".equals(args[i])) {
216                 separator = args[++i];
217             }
218             else if ("-lineTerminator".equals(args[i])) {
```

```
219                     lineTerminator = args[++i]+"\n";
220                 }
221                 else {
222                     System.out.println("Unrecognized option:
                        "+args[i]);
223                     printUsage();
224                     return;
225                 }
226             }
227             if (configFile == null) {
228                 System.out.println("Configuration file
                    unspecified.");
229                 printUsage();
230                 return;
231             }
232             if (outputFilename == null) {
233                 System.out.println("Output filename
                    unspecified.");
234                 printUsage();
235                 return;
236             }
237             if (problemName == null) {
238                 System.out.println("Problem name unspecified
                    .");
239                 printUsage();
240                 return;
241             }
242
243             BenchmarkParser parser = new BenchmarkParser(
                    configFile, resultFolder, outputFilename,
                    problemName, separator, lineTerminator);
244             parser.parse();
245         }
246     }
```

H.3 Cases

These are the case files that correspond to the four cases described in Section 9.1 on page 81.

H.3.1 Valid Gas Burner, $n = 1$

This case is run with $k = 32$ and $k = 50$.

```
1 :- set outputFolder = "gbsafe1".
2 :- state gas.
3 :- state flame.
4 :- leak =^= gas /\ ~flame.
5 :- des1 =^= all( dur leak > 1 -> dur ~(leak) >= 1 ).
```

```
:- des2 =^= all((dur leak >= 1 ; dur ~(leak) >= 1 ; dur
    leak >= 1) -> l >= 32).
:- gbreq(n) =^= all(l <= 30 -> dur leak <= n).
:- gbsafe(n) =^= (des1 /\ des2) -> gbreq(n).
:- goal GOALNAME gbsafe(1).
```

H.3.2 Valid Gas Burner, $n = 6$

This case is run with $k = 32$ and $k = 50$.

```
:- set outputFolder = "gbsafe6".
:- state gas.
:- state flame.
:- leak =^= gas /\ ~flame.
:- des1 =^= all( dur leak > 1 -> dur ~(leak) >= 1 ).
:- des2 =^= all((dur leak >= 1 ; dur ~(leak) >= 1 ; dur
    leak >= 1) -> l >= 32).
:- gbreq(n) =^= all(l <= 30 -> dur leak <= n).
:- gbsafe(n) =^= (des1 /\ des2) -> gbreq(n).
:- goal GOALNAME gbsafe(6).
```

H.3.3 Invalid Gas Burner, $n = 1$

This case is run with $k = 32$ and $k = 50$.

```
:- set outputFolder = "gbsafe1".
:- state gas.
:- state flame.
:- leak =^= gas /\ ~flame.
:- des1 =^= all( dur leak > 1 -> dur ~(leak) >= 1 ).
:- gbreq(n) =^= all(l <= 30 -> dur leak <= n).
:- gbsafe(n) =^= des1 -> gbreq(n).
:- goal GOALNAME gbsafe(1).
```

H.3.4 Invalid Gas Burner, $n = 6$

This case is run with $k = 32$ and $k = 50$.

```
:- set outputFolder = "gbsafe6".
:- state gas.
:- state flame.
:- leak =^= gas /\ ~flame.
:- des1 =^= all( dur leak > 1 -> dur ~(leak) >= 1 ).
:- gbreq(n) =^= all(l <= 30 -> dur leak <= n).
:- gbsafe(n) =^= des1 -> gbreq(n).
:- goal GOALNAME gbsafe(6).
```

H.3.5 Valid Scheduler

```
:- set outputFolder = "sched".
:- set k = 24.
:- state r1.
:- state r2.
:- state r3.
:- mutex =^= all(dur r1 /\ r2 = 0 /\ dur r1 /\ r3 = 0 /\
    dur r2 /\ r3 = 0).

:- alwaysrun(len) =^= (l = len -> dur r1 \/ r2 \/ r3 =
    len).

:- priority24 =^= all((dur r1 < 2 -> dur r2 <= 0) /\
                  (dur r2 < 2 -> dur r3 <= 0) /\
                  (dur r3 < 2 -> dur r1 <= 2) /\
                  (dur r1 < 4 -> dur r2 <= 2) /\
                  (dur r2 < 4 -> dur r3 <= 2) /\
                  (dur r3 < 4 -> dur r1 <= 4) /\
                  (dur r1 < 6 -> dur r2 <= 4) /\
                  (dur r2 < 6 -> dur r3 <= 4) /\
                  (dur r3 < 6 -> dur r1 <= 6) /\
                  (dur r1 < 8 -> dur r2 <= 6) /\
                  (dur r2 < 8 -> dur r3 <= 6) /\
                  (dur r3 < 8 -> dur r1 <= 8) /\
                  (dur r1 < 10 -> dur r2 <= 8) /\
                  (dur r2 < 10 -> dur r3 <= 8) /\
                  (dur r3 < 10 -> dur r1 <= 10) /\
                  (dur r1 < 12 -> dur r2 <= 10) /\
                  (dur r2 < 12 -> dur r3 <= 10) /\
                  (dur r3 < 12 -> dur r1 <= 12) /\
                  (dur r1 < 14 -> dur r2 <= 12) /\
                  (dur r2 < 14 -> dur r3 <= 12) /\
                  (dur r3 < 14 -> dur r1 <= 14) /\
                  (dur r1 < 16 -> dur r2 <= 14) /\
                  (dur r2 < 16 -> dur r3 <= 14) /\
                  (dur r3 < 16 -> dur r1 <= 16) /\
                  (dur r1 < 18 -> dur r2 <= 16) /\
                  (dur r2 < 18 -> dur r3 <= 16) /\
                  (dur r3 < 18 -> dur r1 <= 18) /\
                  (dur r1 < 20 -> dur r2 <= 18) /\
                  (dur r2 < 20 -> dur r3 <= 18) /\
                  (dur r3 < 20 -> dur r1 <= 20) /\
                  (dur r1 < 22 -> dur r2 <= 20) /\
                  (dur r2 < 22 -> dur r3 <= 20) /\
                  (dur r3 < 22 -> dur r1 <= 22) /\
                  (dur r1 < 24 -> dur r2 <= 22) /\
                  (dur r2 < 24 -> dur r3 <= 22) /\
                  (dur r3 < 24 -> dur r1 <= 24)).

```

```
47 :- scheduler24 =^= (priority24 /\ alwaysrun(24)).
48
49 :- wasched24 =^= (l = 24 -> (dur r1 = 8 /\ dur r2 = 8 /\
       dur r3 = 8)).
50
51 :- goal GOALNAME scheduler24 -> (mutex /\ wasched24).
```

H.3.6 Invalid Scheduler

```
1  :- set outputFolder = "insched".
2  :- set k = 24.
3  :- state r1.
4  :- state r2.
5  :- state r3.
6  :- mutex =^= all(dur r1 /\ r2 = 0 /\ dur r1 /\ r3 = 0 /\
       dur r2 /\ r3 = 0).
7
8  :- alwaysrun(len) =^= (l = len -> dur r1 \/ r2 \/ r3 =
       len).
9
10 :- priority24 =^= all((dur r1 < 2 -> dur r2 <= 0) /\
11                       (dur r2 < 2 -> dur r3 <= 0) /\
12                       (dur r1 < 4 -> dur r2 <= 2) /\
13                       (dur r2 < 4 -> dur r3 <= 2) /\
14                       (dur r1 < 6 -> dur r2 <= 4) /\
15                       (dur r2 < 6 -> dur r3 <= 4) /\
16                       (dur r1 < 8 -> dur r2 <= 6) /\
17                       (dur r2 < 8 -> dur r3 <= 6) /\
18                       (dur r1 < 10 -> dur r2 <= 8) /\
19                       (dur r2 < 10 -> dur r3 <= 8) /\
20                       (dur r1 < 12 -> dur r2 <= 10) /\
21                       (dur r2 < 12 -> dur r3 <= 10) /\
22                       (dur r1 < 14 -> dur r2 <= 12) /\
23                       (dur r2 < 14 -> dur r3 <= 12) /\
24                       (dur r1 < 16 -> dur r2 <= 14) /\
25                       (dur r2 < 16 -> dur r3 <= 14) /\
26                       (dur r1 < 18 -> dur r2 <= 16) /\
27                       (dur r2 < 18 -> dur r3 <= 16) /\
28                       (dur r1 < 20 -> dur r2 <= 18) /\
29                       (dur r2 < 20 -> dur r3 <= 18) /\
30                       (dur r1 < 22 -> dur r2 <= 20) /\
31                       (dur r2 < 22 -> dur r3 <= 20) /\
32                       (dur r1 < 24 -> dur r2 <= 22) /\
33                       (dur r2 < 24 -> dur r3 <= 22)).
34
35 :- scheduler24 =^= (priority24 /\ alwaysrun(24)).
36
37 :- wasched24 =^= (l = 24 -> (dur r1 = 8 /\ dur r2 = 8 /\
       dur r3 = 8)).
38
```

```
39 :- goal GOALNAME scheduler24 -> (mutex /\ wasched24).
```

H.4 Results

This section lists the result of the benchmark test that was described in Chapter 9 on page 81. The "Problem" column refers to the name of the test case:

- *gbsafe1* is the valid gas burner example described in Section 9.1.1 on page 81 with $n = 1$.
- *gbsafe6* is the valid gas burner example described in Section 9.1.1 on page 81 with $n = 6$.
- *gbunsafe1* and *gbunsafe6* are the invalid gas burner examples created from *gbsafe1* and *gbsafe6*, respectively by dropping a design decision.
- *sched* is the valid scheduler example described in Section 9.1.2 on page 82.
- *insched* is the invalid scheduler example, created from *sched* by dropping a design decision.

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	true	zolcs	id	true	0	gbsafe1	25132	60248	0.901	9.333	valid
32	true	dimacs	id	true	0	gbsafe1	72813	205474	2.163	21.161	valid
32	false	zolcs	id	true	0	gbsafe1	64402	122800	1.793	15.662	valid
32	false	dimacs	id	true	0	gbsafe1	170988	446111	4.807	11.156	valid
32	true	zolcs	id	false	0	gbsafe1	25128	60246	0.701	9.313	valid
32	true	dimacs	id	false	0	gbsafe1	72813	205474	1.843	21.191	valid
32	false	zolcs	id	false	0	gbsafe1	64398	122796	1.562	18.306	valid
32	false	dimacs	id	false	0	gbsafe1	170988	446111	4.347	11.156	valid
32	true	zolcs	id	true	1	gbsafe1	25132	60248	0.761	9.303	valid
32	true	dimacs	id	true	1	gbsafe1	72813	205474	1.862	21.191	valid
32	false	zolcs	id	true	1	gbsafe1	64402	122800	1.772	15.672	valid
32	false	dimacs	id	true	1	gbsafe1	170988	446111	5.007	11.156	valid
32	true	zolcs	id	false	1	gbsafe1	25128	60246	0.721	9.534	valid
32	true	dimacs	id	false	1	gbsafe1	72813	205474	1.963	21.251	valid
32	false	zolcs	id	false	1	gbsafe1	64398	122796	1.583	18.286	valid
32	false	dimacs	id	false	1	gbsafe1	170988	446111	4.396	11.166	valid
32	true	zolcs	id	true	2	gbsafe1	25132	60248	0.711	9.323	valid
32	true	dimacs	id	true	2	gbsafe1	72813	205474	1.933	21.230	valid
32	false	zolcs	id	true	2	gbsafe1	64402	122800	1.562	15.722	valid
32	false	dimacs	id	true	2	gbsafe1	170988	446111	4.356	11.156	valid
32	true	zolcs	id	false	2	gbsafe1	25128	60246	0.701	9.313	valid
32	true	dimacs	id	false	2	gbsafe1	72813	205474	1.963	21.201	valid
32	false	zolcs	id	false	2	gbsafe1	64398	122796	1.572	18.316	valid
32	false	dimacs	id	false	2	gbsafe1	170988	446111	4.477	11.166	valid
32	true	zolcs	syntactic	true	0	gbsafe1	23320	59656	0.671	5.408	valid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	true	dimacs	syntactic	true	0	gbsafe1	68758	203298	1.762	8.593	valid
32	false	zolcs	syntactic	true	0	gbsafe1	62590	119307	1.482	24.145	valid
32	false	dimacs	syntactic	true	0	gbsafe1	153843	397778	3.685	4.557	valid
32	true	zolcs	syntactic	false	0	gbsafe1	23285	59590	0.681	4.707	valid
32	true	dimacs	syntactic	false	0	gbsafe1	68726	203202	1.823	7.581	valid
32	false	zolcs	syntactic	false	0	gbsafe1	62555	119241	1.473	25.036	valid
32	false	dimacs	syntactic	false	0	gbsafe1	153811	397682	3.935	3.155	valid
32	true	zolcs	syntactic	true	1	gbsafe1	23320	59656	0.681	5.428	valid
32	true	dimacs	syntactic	true	1	gbsafe1	68758	203298	1.923	8.573	valid
32	false	zolcs	syntactic	true	1	gbsafe1	62590	119307	1.462	24.185	valid
32	false	dimacs	syntactic	true	1	gbsafe1	153843	397778	3.936	4.577	valid
32	true	zolcs	syntactic	false	1	gbsafe1	23285	59590	0.681	4.717	valid
32	true	dimacs	syntactic	false	1	gbsafe1	68726	203202	1.942	7.591	valid
32	false	zolcs	syntactic	false	1	gbsafe1	62555	119241	1.452	25.056	valid
32	false	dimacs	syntactic	false	1	gbsafe1	153811	397682	3.925	3.165	valid
32	true	zolcs	syntactic	true	2	gbsafe1	23320	59656	0.681	5.398	valid
32	true	dimacs	syntactic	true	2	gbsafe1	68758	203298	1.933	8.603	valid
32	false	zolcs	syntactic	true	2	gbsafe1	62590	119307	1.452	24.175	valid
32	false	dimacs	syntactic	true	2	gbsafe1	153843	397778	3.896	4.567	valid
32	true	zolcs	syntactic	false	2	gbsafe1	23285	59590	0.671	4.697	valid
32	true	dimacs	syntactic	false	2	gbsafe1	68726	203202	1.943	7.581	valid
32	false	zolcs	syntactic	false	2	gbsafe1	62555	119241	1.472	25.096	valid
32	false	dimacs	syntactic	false	2	gbsafe1	153811	397682	3.906	3.155	valid
32	true	zolcs	semantic	true	2	gbsafe1	23320	59656	0.701	5.408	valid
32	true	dimacs	semantic	true	2	gbsafe1	68758	203298	1.973	8.603	valid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	false	zolcs	semantic	true	2	gbsafe1	62590	119307	1.452	24.165	valid
32	false	dimacs	semantic	true	2	gbsafe1	153843	397778	3.956	4.567	valid
32	true	zolcs	semantic	false	2	gbsafe1	23285	59590	0.681	4.717	valid
32	true	dimacs	semantic	false	2	gbsafe1	68726	203202	1.953	7.591	valid
32	false	zolcs	semantic	false	2	gbsafe1	62555	119241	1.462	25.116	valid
32	false	dimacs	semantic	false	2	gbsafe1	153811	397682	3.995	3.155	valid
32	true	zolcs	semantic	true	2	gbsafe1	23320	59656	0.681	5.418	valid
32	true	dimacs	semantic	true	2	gbsafe1	68758	203298	1.943	8.603	valid
32	false	zolcs	semantic	true	2	gbsafe1	62590	119307	1.452	24.195	valid
32	false	dimacs	semantic	true	2	gbsafe1	153843	397778	3.966	4.557	valid
32	true	zolcs	semantic	false	2	gbsafe1	23285	59590	0.681	4.717	valid
32	true	dimacs	semantic	false	2	gbsafe1	68726	203202	1.942	7.591	valid
32	false	zolcs	semantic	false	2	gbsafe1	62555	119241	1.463	25.076	valid
32	false	dimacs	semantic	false	2	gbsafe1	153811	397682	3.945	3.155	valid
32	true	zolcs	semantic	true	2	gbsafe1	23320	59656	0.691	5.408	valid
32	true	dimacs	semantic	true	2	gbsafe1	68758	203298	1.933	8.603	valid
32	false	zolcs	semantic	true	2	gbsafe1	62590	119307	1.462	24.185	valid
32	false	dimacs	semantic	true	2	gbsafe1	153843	397778	3.996	4.557	valid
32	true	zolcs	semantic	false	2	gbsafe1	23285	59590	0.671	4.727	valid
32	true	dimacs	semantic	false	2	gbsafe1	68726	203202	1.942	7.591	valid
32	false	zolcs	semantic	false	2	gbsafe1	62555	119241	1.473	25.076	valid
32	false	dimacs	semantic	false	2	gbsafe1	153811	397682	3.935	3.165	valid
50	true	zolcs	id	true	0	gbsafe1	75103	206042	2.533	203.922	valid
50	true	dimacs	id	true	0	gbsafe1	235545	716425	7.691	498.035	valid
50	false	zolcs	id	true	0	gbsafe1	215659	417448			HySat error

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	false	dimacs	id	true	0	gbsafe1	586935	1555586			HySat error
50	true	zolcs	id	false	0	gbsafe1	75099	206040	2.594	203.722	valid
50	true	dimacs	id	false	0	gbsafe1	235545	716425	6.589	496.434	valid
50	false	zolcs	id	false	0	gbsafe1	215655	417444			HySat error
50	false	dimacs	id	false	0	gbsafe1	586935	1555586			HySat error
50	true	zolcs	id	true	1	gbsafe1	75103	206042	2.504	204.133	valid
50	true	dimacs	id	true	1	gbsafe1	235545	716425	7.020	496.474	valid
50	false	zolcs	id	true	1	gbsafe1	215659	417448			HySat error
50	false	dimacs	id	true	1	gbsafe1	586935	1555586			HySat error
50	true	zolcs	id	false	1	gbsafe1	75099	206040	2.474	203.812	valid
50	true	dimacs	id	false	1	gbsafe1	235545	716425	6.960	496.454	valid
50	false	zolcs	id	false	1	gbsafe1	215655	417444			HySat error
50	false	dimacs	id	false	1	gbsafe1	586935	1555586			HySat error
50	true	zolcs	id	true	2	gbsafe1	75103	206042	2.463	203.572	valid
50	true	dimacs	id	true	2	gbsafe1	235545	716425	7.031	496.593	valid
50	false	zolcs	id	true	2	gbsafe1	215659	417448			HySat error
50	false	dimacs	id	true	2	gbsafe1	586935	1555586			HySat error
50	true	zolcs	id	false	2	gbsafe1	75099	206040	2.473	203.973	valid
50	true	dimacs	id	false	2	gbsafe1	235545	716425	7.000	496.273	valid
50	false	zolcs	id	false	2	gbsafe1	215655	417444			HySat error
50	false	dimacs	id	false	2	gbsafe1	586935	1555586			HySat error
50	true	zolcs	syntactic	true	0	gbsafe1	70924	204667	2.403	85.012	valid
50	true	dimacs	syntactic	true	0	gbsafe1	226063	711225	6.879	200.609	valid
50	false	zolcs	syntactic	true	0	gbsafe1	211480	409293	5.207	722.168	valid
50	false	dimacs	syntactic	true	0	gbsafe1	530601	1393673			HySat error

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	true	zolcs	syntactic	false	0	gbsafe1	70871	204565	2.373	44.154	valid
50	true	dimacs	syntactic	false	0	gbsafe1	226013	711075	6.409	62.750	valid
50	false	zolcs	syntactic	false	0	gbsafe1	211427	409191	5.438	401.907	valid
50	false	dimacs	syntactic	false	0	gbsafe1	530551	1393523			HySat error
50	true	zolcs	syntactic	true	1	gbsafe1	70924	204667	2.373	84.982	valid
50	true	dimacs	syntactic	true	1	gbsafe1	226063	711225	6.499	200.639	valid
50	false	zolcs	syntactic	true	1	gbsafe1	211480	409293	5.237	721.487	valid
50	false	dimacs	syntactic	true	1	gbsafe1	530601	1393673			HySat error
50	true	zolcs	syntactic	false	1	gbsafe1	70871	204565	2.433	44.213	valid
50	true	dimacs	syntactic	false	1	gbsafe1	226013	711075	6.880	62.740	valid
50	false	zolcs	syntactic	false	1	gbsafe1	211427	409191	5.307	401.998	valid
50	false	dimacs	syntactic	false	1	gbsafe1	530551	1393523			HySat error
50	true	zolcs	syntactic	true	2	gbsafe1	70924	204667	2.374	84.982	valid
50	true	dimacs	syntactic	true	2	gbsafe1	226063	711225	6.780	200.609	valid
50	false	zolcs	syntactic	true	2	gbsafe1	211480	409293	5.218	722.059	valid
50	false	dimacs	syntactic	true	2	gbsafe1	530601	1393673			HySat error
50	true	zolcs	syntactic	false	2	gbsafe1	70871	204565	2.424	44.194	valid
50	true	dimacs	syntactic	false	2	gbsafe1	226013	711075	6.830	62.760	valid
50	false	zolcs	syntactic	false	2	gbsafe1	211427	409191	5.318	402.749	valid
50	false	dimacs	syntactic	false	2	gbsafe1	530551	1393523			HySat error
50	true	zolcs	semantic	true	2	gbsafe1	70924	204667	2.393	84.962	valid
50	true	dimacs	semantic	true	2	gbsafe1	226063	711225	6.890	200.589	valid
50	false	zolcs	semantic	true	2	gbsafe1	211480	409293	5.217	721.377	valid
50	false	dimacs	semantic	true	2	gbsafe1	530601	1393673			HySat error
50	true	zolcs	semantic	false	2	gbsafe1	70871	204565	2.403	44.194	valid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	true	dimacs	semantic	false	2	gbsafe1	226013	711075	6.860	62.750	valid
50	false	zolcs	semantic	false	2	gbsafe1	211427	409191	5.218	402.178	valid
50	false	dimacs	semantic	false	2	gbsafe1	530551	1393523			HySat error
50	true	zolcs	semantic	true	2	gbsafe1	70924	204667	2.383	84.932	valid
50	true	dimacs	semantic	true	2	gbsafe1	226063	711225	6.790	200.589	valid
50	false	zolcs	semantic	true	2	gbsafe1	211480	409293	5.238	721.557	valid
50	false	dimacs	semantic	true	2	gbsafe1	530601	1393673			HySat error
50	true	zolcs	semantic	false	2	gbsafe1	70871	204565	2.393	44.224	valid
50	true	dimacs	semantic	false	2	gbsafe1	226013	711075	6.790	62.740	valid
50	false	zolcs	semantic	false	2	gbsafe1	211427	409191	5.228	402.309	valid
50	false	dimacs	semantic	false	2	gbsafe1	530551	1393523			HySat error
50	true	zolcs	semantic	true	2	gbsafe1	70924	204667	2.384	84.902	valid
50	true	dimacs	semantic	true	2	gbsafe1	226063	711225	6.790	200.648	valid
50	false	zolcs	semantic	true	2	gbsafe1	211480	409293	5.218	721.466	valid
50	false	dimacs	semantic	true	2	gbsafe1	530601	1393673			HySat error
50	true	zolcs	semantic	false	2	gbsafe1	70871	204565	2.384	44.194	valid
50	true	dimacs	semantic	false	2	gbsafe1	226013	711075	6.810	62.730	valid
50	false	zolcs	semantic	false	2	gbsafe1	211427	409191	5.268	402.868	valid
50	false	dimacs	semantic	false	2	gbsafe1	530551	1393523			HySat error
32	true	zolcs	id	true	0	gbsafe6	25132	60248	0.751	3.124	valid
32	true	dimacs	id	true	0	gbsafe6	94131	246427	2.053	47.107	valid
32	false	zolcs	id	true	0	gbsafe6	64402	122655	1.502	14.540	valid
32	false	dimacs	id	true	0	gbsafe6	192306	506699	4.567	33.488	valid
32	true	zolcs	id	false	0	gbsafe6	25128	60246	0.691	3.134	valid
32	true	dimacs	id	false	0	gbsafe6	94131	246427	2.052	47.097	valid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	false	zolcs	id	false	0	gbsafe6	64398	122651	1.502	18.015	valid
32	false	dimacs	id	false	0	gbsafe6	192306	506699	4.366	33.498	valid
32	true	zolcs	id	true	1	gbsafe6	25132	60248	0.671	3.124	valid
32	true	dimacs	id	true	1	gbsafe6	94131	246427	2.063	47.077	valid
32	false	zolcs	id	true	1	gbsafe6	64402	122655	1.503	14.560	valid
32	false	dimacs	id	true	1	gbsafe6	192306	506699	4.586	33.498	valid
32	true	zolcs	id	false	1	gbsafe6	25128	60246	0.691	3.124	valid
32	true	dimacs	id	false	1	gbsafe6	94131	246427	2.053	47.097	valid
32	false	zolcs	id	false	1	gbsafe6	64398	122651	1.523	18.025	valid
32	false	dimacs	id	false	1	gbsafe6	192306	506699	4.426	33.498	valid
32	true	zolcs	id	true	2	gbsafe6	25132	60248	0.701	3.124	valid
32	true	dimacs	id	true	2	gbsafe6	94131	246427	2.073	47.087	valid
32	false	zolcs	id	true	2	gbsafe6	64402	122655	1.713	14.560	valid
32	false	dimacs	id	true	2	gbsafe6	192306	506699	4.476	33.508	valid
32	true	zolcs	id	false	2	gbsafe6	25128	60246	0.701	3.134	valid
32	true	dimacs	id	false	2	gbsafe6	94131	246427	2.073	47.118	valid
32	false	zolcs	id	false	2	gbsafe6	64398	122651	1.513	18.005	valid
32	false	dimacs	id	false	2	gbsafe6	192306	506699	5.267	33.488	valid
32	true	zolcs	syntactic	true	0	gbsafe6	23881	59656	0.721	2.624	valid
32	true	dimacs	syntactic	true	0	gbsafe6	90076	244251	2.113	28.621	valid
32	false	zolcs	syntactic	true	0	gbsafe6	63151	120219	1.462	12.568	valid
32	false	dimacs	syntactic	true	0	gbsafe6	175161	458366	4.637	22.192	valid
32	true	zolcs	syntactic	false	0	gbsafe6	23846	59590	0.671	1.823	valid
32	true	dimacs	syntactic	false	0	gbsafe6	90044	244155	2.093	21.010	valid
32	false	zolcs	syntactic	false	0	gbsafe6	63116	120153	1.493	9.294	valid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	false	dimacs	syntactic	false	0	gbsafe6	175129	458270	4.106	24.155	valid
32	true	zolcs	syntactic	true	1	gbsafe6	23881	59656	0.661	2.624	valid
32	true	dimacs	syntactic	true	1	gbsafe6	90076	244251	2.173	28.621	valid
32	false	zolcs	syntactic	true	1	gbsafe6	63151	120219	1.452	12.568	valid
32	false	dimacs	syntactic	true	1	gbsafe6	175161	458366	4.447	22.172	valid
32	true	zolcs	syntactic	false	1	gbsafe6	23846	59590	0.671	1.823	valid
32	true	dimacs	syntactic	false	1	gbsafe6	90044	244155	2.273	20.980	valid
32	false	zolcs	syntactic	false	1	gbsafe6	63116	120153	1.442	9.294	valid
32	false	dimacs	syntactic	false	1	gbsafe6	175129	458270	4.416	24.165	valid
32	true	zolcs	syntactic	true	2	gbsafe6	23881	59656	0.651	2.624	valid
32	true	dimacs	syntactic	true	2	gbsafe6	90076	244251	2.243	28.621	valid
32	false	zolcs	syntactic	true	2	gbsafe6	63151	120219	1.452	12.568	valid
32	false	dimacs	syntactic	true	2	gbsafe6	175161	458366	4.447	22.182	valid
32	true	zolcs	syntactic	false	2	gbsafe6	23846	59590	0.681	1.812	valid
32	true	dimacs	syntactic	false	2	gbsafe6	90044	244155	2.264	20.980	valid
32	false	zolcs	syntactic	false	2	gbsafe6	63116	120153	1.452	9.284	valid
32	false	dimacs	syntactic	false	2	gbsafe6	175129	458270	4.426	24.155	valid
32	true	zolcs	semantic	true	2	gbsafe6	23881	59656	0.651	2.614	valid
32	true	dimacs	semantic	true	2	gbsafe6	90076	244251	2.283	28.621	valid
32	false	zolcs	semantic	true	2	gbsafe6	63151	120219	1.442	12.548	valid
32	false	dimacs	semantic	true	2	gbsafe6	175161	458366	4.456	22.192	valid
32	true	zolcs	semantic	false	2	gbsafe6	23846	59590	0.671	1.822	valid
32	true	dimacs	semantic	false	2	gbsafe6	90044	244155	2.273	20.980	valid
32	false	zolcs	semantic	false	2	gbsafe6	63116	120153	1.452	9.294	valid
32	false	dimacs	semantic	false	2	gbsafe6	175129	458270	4.417	24.165	valid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	true	zolcs	semantic	true	2	gbsafe6	23881	59656	0.661	2.624	valid
32	true	dimacs	semantic	true	2	gbsafe6	90076	244251	2.283	28.601	valid
32	false	zolcs	semantic	true	2	gbsafe6	63151	120219	1.442	12.558	valid
32	false	dimacs	semantic	true	2	gbsafe6	175161	458366	4.426	22.182	valid
32	true	zolcs	semantic	false	2	gbsafe6	23846	59590	0.671	1.833	valid
32	true	dimacs	semantic	false	2	gbsafe6	90044	244155	2.263	20.960	valid
32	false	zolcs	semantic	false	2	gbsafe6	63116	120153	1.452	9.304	valid
32	false	dimacs	semantic	false	2	gbsafe6	175129	458270	4.386	24.165	valid
32	true	zolcs	semantic	true	2	gbsafe6	23881	59656	0.661	2.614	valid
32	true	dimacs	semantic	true	2	gbsafe6	90076	244251	2.303	28.601	valid
32	false	zolcs	semantic	true	2	gbsafe6	63151	120219	1.452	12.548	valid
32	false	dimacs	semantic	true	2	gbsafe6	175161	458366	4.437	22.192	valid
32	true	zolcs	semantic	false	2	gbsafe6	23846	59590	0.661	1.822	valid
32	true	dimacs	semantic	false	2	gbsafe6	90044	244155	2.294	21.000	valid
32	false	zolcs	semantic	false	2	gbsafe6	63116	120153	1.453	9.304	valid
32	false	dimacs	semantic	false	2	gbsafe6	175129	458270	4.396	24.135	valid
50	true	zolcs	id	true	0	gbsafe6	75103	206042	2.453	67.406	valid
50	true	dimacs	id	true	0	gbsafe6	309801	860959			HySat error
50	false	zolcs	id	true	0	gbsafe6	215659	417213	6.199	191.154	valid
50	false	dimacs	id	true	0	gbsafe6	661191	1770398			HySat error
50	true	zolcs	id	false	0	gbsafe6	75099	206040	2.644	66.965	valid
50	true	dimacs	id	false	0	gbsafe6	309801	860959			HySat error
50	false	zolcs	id	false	0	gbsafe6	215655	417209	5.668	238.442	valid
50	false	dimacs	id	false	0	gbsafe6	661191	1770398			HySat error
50	true	zolcs	id	true	1	gbsafe6	75103	206042	2.534	67.115	valid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	true	dimacs	id	true	1	gbsafe6	309801	860959			HySat error
50	false	zolcs	id	true	1	gbsafe6	215659	417213	5.588	191.304	valid
50	false	dimacs	id	true	1	gbsafe6	661191	1770398			HySat error
50	true	zolcs	id	false	1	gbsafe6	75099	206040	2.503	67.076	valid
50	true	dimacs	id	false	1	gbsafe6	309801	860959			HySat error
50	false	zolcs	id	false	1	gbsafe6	215655	417209	5.698	238.563	valid
50	false	dimacs	id	false	1	gbsafe6	661191	1770398			HySat error
50	true	zolcs	id	true	2	gbsafe6	75103	206042	2.514	67.095	valid
50	true	dimacs	id	true	2	gbsafe6	309801	860959			HySat error
50	false	zolcs	id	true	2	gbsafe6	215659	417213	5.648	191.184	valid
50	false	dimacs	id	true	2	gbsafe6	661191	1770398			HySat error
50	true	zolcs	id	false	2	gbsafe6	75099	206040	2.513	67.005	valid
50	true	dimacs	id	false	2	gbsafe6	309801	860959			HySat error
50	false	zolcs	id	false	2	gbsafe6	215655	417209	5.698	238.433	valid
50	false	dimacs	id	false	2	gbsafe6	661191	1770398			HySat error
50	true	zolcs	syntactic	true	0	gbsafe6	72250	204667	2.474	36.252	valid
50	true	dimacs	syntactic	true	0	gbsafe6	300319	855759			HySat error
50	false	zolcs	syntactic	true	0	gbsafe6	212806	411609	5.358	193.478	valid
50	false	dimacs	syntactic	true	0	gbsafe6	604857	1608485			HySat error
50	true	zolcs	syntactic	false	0	gbsafe6	72197	204565	2.464	22.282	valid
50	true	dimacs	syntactic	false	0	gbsafe6	300269	855609			HySat error
50	false	zolcs	syntactic	false	0	gbsafe6	212753	411507	5.368	158.177	valid
50	false	dimacs	syntactic	false	0	gbsafe6	604807	1608335			HySat error
50	true	zolcs	syntactic	true	1	gbsafe6	72250	204667	2.414	36.202	valid
50	true	dimacs	syntactic	true	1	gbsafe6	300319	855759			HySat error

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	false	zolcs	syntactic	true	1	gbsafe6	212806	411609	5.348	193.388	valid
50	false	dimacs	syntactic	true	1	gbsafe6	604857	1608485			HySat error
50	true	zolcs	syntactic	false	1	gbsafe6	72197	204565	2.464	22.312	valid
50	true	dimacs	syntactic	false	1	gbsafe6	300269	855609			HySat error
50	false	zolcs	syntactic	false	1	gbsafe6	212753	411507	5.368	158.066	valid
50	false	dimacs	syntactic	false	1	gbsafe6	604807	1608335			HySat error
50	true	zolcs	syntactic	true	2	gbsafe6	72250	204667	2.423	36.242	valid
50	true	dimacs	syntactic	true	2	gbsafe6	300319	855759			HySat error
50	false	zolcs	syntactic	true	2	gbsafe6	212806	411609	5.328	193.428	valid
50	false	dimacs	syntactic	true	2	gbsafe6	604857	1608485			HySat error
50	true	zolcs	syntactic	false	2	gbsafe6	72197	204565	2.484	22.302	valid
50	true	dimacs	syntactic	false	2	gbsafe6	300269	855609			HySat error
50	false	zolcs	syntactic	false	2	gbsafe6	212753	411507	5.368	158.116	valid
50	false	dimacs	syntactic	false	2	gbsafe6	604807	1608335			HySat error
50	true	zolcs	semantic	true	2	gbsafe6	72250	204667	2.414	36.232	valid
50	true	dimacs	semantic	true	2	gbsafe6	300319	855759			HySat error
50	false	zolcs	semantic	true	2	gbsafe6	212806	411609	5.408	193.347	valid
50	false	dimacs	semantic	true	2	gbsafe6	604857	1608485			HySat error
50	true	zolcs	semantic	false	2	gbsafe6	72197	204565	2.444	22.292	valid
50	true	dimacs	semantic	false	2	gbsafe6	300269	855609			HySat error
50	false	zolcs	semantic	false	2	gbsafe6	212753	411507	5.358	158.116	valid
50	false	dimacs	semantic	false	2	gbsafe6	604807	1608335			HySat error
50	true	zolcs	semantic	true	2	gbsafe6	72250	204667	2.423	36.232	valid
50	true	dimacs	semantic	true	2	gbsafe6	300319	855759			HySat error
50	false	zolcs	semantic	true	2	gbsafe6	212806	411609	5.408	193.368	valid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	false	dimacs	semantic	true	2	gbsafe6	604857	1608485			HySat error
50	true	zolcs	semantic	false	2	gbsafe6	72197	204565	2.493	22.282	valid
50	true	dimacs	semantic	false	2	gbsafe6	300269	855609			HySat error
50	false	zolcs	semantic	false	2	gbsafe6	212753	411507	5.418	158.297	valid
50	false	dimacs	semantic	false	2	gbsafe6	604807	1608335			HySat error
50	true	zolcs	semantic	true	2	gbsafe6	72250	204667	2.433	36.192	valid
50	true	dimacs	semantic	true	2	gbsafe6	300319	855759			HySat error
50	false	zolcs	semantic	true	2	gbsafe6	212806	411609	5.398	193.407	valid
50	false	dimacs	semantic	true	2	gbsafe6	604857	1608485			HySat error
50	true	zolcs	semantic	false	2	gbsafe6	72197	204565	2.494	22.292	valid
50	true	dimacs	semantic	false	2	gbsafe6	300269	855609			HySat error
50	false	zolcs	semantic	false	2	gbsafe6	212753	411507	5.388	158.157	valid
50	false	dimacs	semantic	false	2	gbsafe6	604807	1608335			HySat error
32	true	zolcs	id	true	0	gbunsafe1	19425	30705	0.521	0.110	invalid
32	true	dimacs	id	true	0	gbunsafe1	31577	106027	1.051	0.371	invalid
32	false	zolcs	id	true	0	gbunsafe1	32515	61930	0.832	0.080	invalid
32	false	dimacs	id	true	0	gbunsafe1	103572	267843	2.734	15.402	invalid
32	true	zolcs	id	false	0	gbunsafe1	19422	30704	0.391	0.110	invalid
32	true	dimacs	id	false	0	gbunsafe1	31577	106027	0.931	0.371	invalid
32	false	zolcs	id	false	0	gbunsafe1	32512	61928	0.751	0.150	invalid
32	false	dimacs	id	false	0	gbunsafe1	103572	267843	2.504	15.482	invalid
32	true	zolcs	id	true	1	gbunsafe1	19425	30705	0.381	0.110	invalid
32	true	dimacs	id	true	1	gbunsafe1	31577	106027	0.901	0.391	invalid
32	false	zolcs	id	true	1	gbunsafe1	32515	61930	0.741	0.080	invalid
32	false	dimacs	id	true	1	gbunsafe1	103572	267843	2.513	15.623	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	true	zolcs	id	false	1	gbunsafe1	19422	30704	0.380	0.110	invalid
32	true	dimacs	id	false	1	gbunsafe1	31577	106027	0.902	0.381	invalid
32	false	zolcs	id	false	1	gbunsafe1	32512	61928	0.751	0.150	invalid
32	false	dimacs	id	false	1	gbunsafe1	103572	267843	2.493	15.973	invalid
32	true	zolcs	id	true	2	gbunsafe1	19425	30705	0.391	0.110	invalid
32	true	dimacs	id	true	2	gbunsafe1	31577	106027	0.911	0.391	invalid
32	false	zolcs	id	true	2	gbunsafe1	32515	61930	0.731	0.080	invalid
32	false	dimacs	id	true	2	gbunsafe1	103572	267843	2.504	16.183	invalid
32	true	zolcs	id	false	2	gbunsafe1	19422	30704	0.370	0.120	invalid
32	true	dimacs	id	false	2	gbunsafe1	31577	106027	0.922	0.371	invalid
32	false	zolcs	id	false	2	gbunsafe1	32512	61928	0.751	0.160	invalid
32	false	dimacs	id	false	2	gbunsafe1	103572	267843	2.493	15.562	invalid
32	true	zolcs	syntactic	true	0	gbunsafe1	18832	30705	0.380	0.110	invalid
32	true	dimacs	syntactic	true	0	gbunsafe1	30423	106027	0.912	1.192	invalid
32	false	zolcs	syntactic	true	0	gbunsafe1	31922	60809	0.721	0.080	invalid
32	false	dimacs	syntactic	true	0	gbunsafe1	95873	245934	2.254	13.249	invalid
32	true	zolcs	syntactic	false	0	gbunsafe1	18797	30672	0.371	0.120	invalid
32	true	dimacs	syntactic	false	0	gbunsafe1	30391	105995	0.951	0.761	invalid
32	false	zolcs	syntactic	false	0	gbunsafe1	31887	60743	0.711	0.110	invalid
32	false	dimacs	syntactic	false	0	gbunsafe1	95841	245838	2.303	9.113	invalid
32	true	zolcs	syntactic	true	1	gbunsafe1	18832	30705	0.360	0.120	invalid
32	true	dimacs	syntactic	true	1	gbunsafe1	30423	106027	0.962	1.182	invalid
32	false	zolcs	syntactic	true	1	gbunsafe1	31922	60809	0.721	0.080	invalid
32	false	dimacs	syntactic	true	1	gbunsafe1	95873	245934	2.263	13.128	invalid
32	true	zolcs	syntactic	false	1	gbunsafe1	18797	30672	0.361	0.120	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	true	dimacs	syntactic	false	1	gbunsafe1	30391	105995	0.951	0.781	invalid
32	false	zolcs	syntactic	false	1	gbunsafe1	31887	60743	0.701	0.100	invalid
32	false	dimacs	syntactic	false	1	gbunsafe1	95841	245838	2.273	9.293	invalid
32	true	zolcs	syntactic	true	2	gbunsafe1	18832	30705	0.370	0.120	invalid
32	true	dimacs	syntactic	true	2	gbunsafe1	30423	106027	0.942	1.162	invalid
32	false	zolcs	syntactic	true	2	gbunsafe1	31922	60809	0.721	0.080	invalid
32	false	dimacs	syntactic	true	2	gbunsafe1	95873	245934	2.263	13.409	invalid
32	true	zolcs	syntactic	false	2	gbunsafe1	18797	30672	0.380	0.120	invalid
32	true	dimacs	syntactic	false	2	gbunsafe1	30391	105995	0.942	0.781	invalid
32	false	zolcs	syntactic	false	2	gbunsafe1	31887	60743	0.701	0.100	invalid
32	false	dimacs	syntactic	false	2	gbunsafe1	95841	245838	2.273	9.223	invalid
32	true	zolcs	semantic	true	2	gbunsafe1	18832	30705	0.391	0.120	invalid
32	true	dimacs	semantic	true	2	gbunsafe1	30423	106027	0.991	1.152	invalid
32	false	zolcs	semantic	true	2	gbunsafe1	31922	60809	0.721	0.080	invalid
32	false	dimacs	semantic	true	2	gbunsafe1	95873	245934	2.273	13.129	invalid
32	true	zolcs	semantic	false	2	gbunsafe1	18797	30672	0.370	0.120	invalid
32	true	dimacs	semantic	false	2	gbunsafe1	30391	105995	0.952	0.781	invalid
32	false	zolcs	semantic	false	2	gbunsafe1	31887	60743	0.701	0.100	invalid
32	false	dimacs	semantic	false	2	gbunsafe1	95841	245838	2.283	9.143	invalid
32	true	zolcs	semantic	true	2	gbunsafe1	18832	30705	0.361	0.120	invalid
32	true	dimacs	semantic	true	2	gbunsafe1	30423	106027	0.951	1.162	invalid
32	false	zolcs	semantic	true	2	gbunsafe1	31922	60809	0.721	0.080	invalid
32	false	dimacs	semantic	true	2	gbunsafe1	95873	245934	2.293	13.069	invalid
32	true	zolcs	semantic	false	2	gbunsafe1	18797	30672	0.380	0.130	invalid
32	true	dimacs	semantic	false	2	gbunsafe1	30391	105995	0.952	0.761	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	false	zolcs	semantic	false	2	gbunsafe1	31887	60743	0.701	0.110	invalid
32	false	dimacs	semantic	false	2	gbunsafe1	95841	245838	2.293	9.243	invalid
32	true	zolcs	semantic	true	2	gbunsafe1	18832	30705	0.381	0.120	invalid
32	true	dimacs	semantic	true	2	gbunsafe1	30423	106027	0.961	1.162	invalid
32	false	zolcs	semantic	true	2	gbunsafe1	31922	60809	0.721	0.080	invalid
32	false	dimacs	semantic	true	2	gbunsafe1	95873	245934	2.263	13.048	invalid
32	true	zolcs	semantic	false	2	gbunsafe1	18797	30672	0.370	0.130	invalid
32	true	dimacs	semantic	false	2	gbunsafe1	30391	105995	0.951	0.771	invalid
32	false	zolcs	semantic	false	2	gbunsafe1	31887	60743	0.711	0.100	invalid
32	false	dimacs	semantic	false	2	gbunsafe1	95841	245838	2.263	9.143	invalid
50	true	zolcs	id	true	0	gbunsafe1	61692	104574	1.452	1.192	invalid
50	true	dimacs	id	true	0	gbunsafe1	98375	369952	3.485	3.495	invalid
50	false	zolcs	id	true	0	gbunsafe1	108544	210001	2.884	8.662	invalid
50	false	dimacs	id	true	0	gbunsafe1	356061	935367	10.525	239.545	invalid
50	true	zolcs	id	false	0	gbunsafe1	61689	104573	1.281	1.162	invalid
50	true	dimacs	id	false	0	gbunsafe1	98375	369952	3.315	3.485	invalid
50	false	zolcs	id	false	0	gbunsafe1	108541	209999	2.604	9.193	invalid
50	false	dimacs	id	false	0	gbunsafe1	356061	935367	9.083	240.075	invalid
50	true	zolcs	id	true	1	gbunsafe1	61692	104574	1.262	1.172	invalid
50	true	dimacs	id	true	1	gbunsafe1	98375	369952	3.045	3.465	invalid
50	false	zolcs	id	true	1	gbunsafe1	108544	210001	2.594	8.592	invalid
50	false	dimacs	id	true	1	gbunsafe1	356061	935367	8.622	240.957	invalid
50	true	zolcs	id	false	1	gbunsafe1	61689	104573	1.362	1.202	invalid
50	true	dimacs	id	false	1	gbunsafe1	98375	369952	3.035	3.455	invalid
50	false	zolcs	id	false	1	gbunsafe1	108541	209999	2.623	9.263	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	false	dimacs	id	false	1	gbunsafe1	356061	935367	9.173	243.981	invalid
50	true	zolcs	id	true	2	gbunsafe1	61692	104574	1.281	1.182	invalid
50	true	dimacs	id	true	2	gbunsafe1	98375	369952	3.164	3.445	invalid
50	false	zolcs	id	true	2	gbunsafe1	108544	210001	2.634	8.612	invalid
50	false	dimacs	id	true	2	gbunsafe1	356061	935367	9.183	242.068	invalid
50	true	zolcs	id	false	2	gbunsafe1	61689	104573	1.272	1.192	invalid
50	true	dimacs	id	false	2	gbunsafe1	98375	369952	3.195	3.465	invalid
50	false	zolcs	id	false	2	gbunsafe1	108541	209999	2.604	9.103	invalid
50	false	dimacs	id	false	2	gbunsafe1	356061	935367	9.073	240.336	invalid
50	true	zolcs	syntactic	true	0	gbunsafe1	60316	104574	1.262	8.382	invalid
50	true	dimacs	syntactic	true	0	gbunsafe1	95673	369952	3.185	18.226	invalid
50	false	zolcs	syntactic	true	0	gbunsafe1	107168	207350	2.473	0.862	invalid
50	false	dimacs	syntactic	true	0	gbunsafe1	329933	859737	8.322	204.574	invalid
50	true	zolcs	syntactic	false	0	gbunsafe1	60263	104523	1.262	1.172	invalid
50	true	dimacs	syntactic	false	0	gbunsafe1	95623	369902	3.215	11.506	invalid
50	false	zolcs	syntactic	false	0	gbunsafe1	107115	207248	2.494	25.476	invalid
50	false	dimacs	syntactic	false	0	gbunsafe1	329883	859587	8.332	150.006	invalid
50	true	zolcs	syntactic	true	1	gbunsafe1	60316	104574	1.262	8.272	invalid
50	true	dimacs	syntactic	true	1	gbunsafe1	95673	369952	3.214	18.356	invalid
50	false	zolcs	syntactic	true	1	gbunsafe1	107168	207350	2.474	0.852	invalid
50	false	dimacs	syntactic	true	1	gbunsafe1	329933	859737	8.252	203.092	invalid
50	true	zolcs	syntactic	false	1	gbunsafe1	60263	104523	1.252	1.172	invalid
50	true	dimacs	syntactic	false	1	gbunsafe1	95623	369902	3.214	11.516	invalid
50	false	zolcs	syntactic	false	1	gbunsafe1	107115	207248	2.474	25.446	invalid
50	false	dimacs	syntactic	false	1	gbunsafe1	329883	859587	8.322	151.828	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	true	zolcs	syntactic	true	2	gbunsafe1	60316	104574	1.252	8.242	invalid
50	true	dimacs	syntactic	true	2	gbunsafe1	95673	369952	3.215	18.436	invalid
50	false	zolcs	syntactic	true	2	gbunsafe1	107168	207350	2.414	0.852	invalid
50	false	dimacs	syntactic	true	2	gbunsafe1	329933	859737	8.292	202.260	invalid
50	true	zolcs	syntactic	false	2	gbunsafe1	60263	104523	1.242	1.162	invalid
50	true	dimacs	syntactic	false	2	gbunsafe1	95623	369902	3.205	11.506	invalid
50	false	zolcs	syntactic	false	2	gbunsafe1	107115	207248	2.494	25.376	invalid
50	false	dimacs	syntactic	false	2	gbunsafe1	329883	859587	8.312	149.565	invalid
50	true	zolcs	semantic	true	2	gbunsafe1	60316	104574	1.262	8.212	invalid
50	true	dimacs	semantic	true	2	gbunsafe1	95673	369952	3.204	18.306	invalid
50	false	zolcs	semantic	true	2	gbunsafe1	107168	207350	2.474	0.852	invalid
50	false	dimacs	semantic	true	2	gbunsafe1	329933	859737	8.202	201.860	invalid
50	true	zolcs	semantic	false	2	gbunsafe1	60263	104523	1.242	1.162	invalid
50	true	dimacs	semantic	false	2	gbunsafe1	95623	369902	3.194	11.506	invalid
50	false	zolcs	semantic	false	2	gbunsafe1	107115	207248	2.494	25.186	invalid
50	false	dimacs	semantic	false	2	gbunsafe1	329883	859587	8.362	149.766	invalid
50	true	zolcs	semantic	true	2	gbunsafe1	60316	104574	1.272	8.262	invalid
50	true	dimacs	semantic	true	2	gbunsafe1	95673	369952	3.195	18.216	invalid
50	false	zolcs	semantic	true	2	gbunsafe1	107168	207350	2.473	0.852	invalid
50	false	dimacs	semantic	true	2	gbunsafe1	329933	859737	8.171	203.243	invalid
50	true	zolcs	semantic	false	2	gbunsafe1	60263	104523	1.252	1.152	invalid
50	true	dimacs	semantic	false	2	gbunsafe1	95623	369902	3.235	11.466	invalid
50	false	zolcs	semantic	false	2	gbunsafe1	107115	207248	2.473	25.256	invalid
50	false	dimacs	semantic	false	2	gbunsafe1	329883	859587	8.312	151.548	invalid
50	true	zolcs	semantic	true	2	gbunsafe1	60316	104574	1.272	8.262	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	true	dimacs	semantic	true	2	gbunsafe1	95673	369952	3.185	18.286	invalid
50	false	zolcs	semantic	true	2	gbunsafe1	107168	207350	2.413	0.852	invalid
50	false	dimacs	semantic	true	2	gbunsafe1	329933	859737	8.222	201.931	invalid
50	true	zolcs	semantic	false	2	gbunsafe1	60263	104523	1.242	1.162	invalid
50	true	dimacs	semantic	false	2	gbunsafe1	95623	369902	3.225	11.466	invalid
50	false	zolcs	semantic	false	2	gbunsafe1	107115	207248	2.483	25.186	invalid
50	false	dimacs	semantic	false	2	gbunsafe1	329883	859587	8.312	150.777	invalid
32	true	zolcs	id	true	0	gbunsafe6	19425	30705	0.390	0.411	invalid
32	true	dimacs	id	true	0	gbunsafe6	52895	146980	1.171	4.135	invalid
32	false	zolcs	id	true	0	gbunsafe6	32515	61785	0.711	1.282	invalid
32	false	dimacs	id	true	0	gbunsafe6	124890	328431	2.824	18.937	invalid
32	true	zolcs	id	false	0	gbunsafe6	19422	30704	0.361	0.391	invalid
32	true	dimacs	id	false	0	gbunsafe6	52895	146980	1.182	4.166	invalid
32	false	zolcs	id	false	0	gbunsafe6	32512	61783	0.721	1.342	invalid
32	false	dimacs	id	false	0	gbunsafe6	124890	328431	2.694	18.967	invalid
32	true	zolcs	id	true	1	gbunsafe6	19425	30705	0.361	0.401	invalid
32	true	dimacs	id	true	1	gbunsafe6	52895	146980	1.172	4.156	invalid
32	false	zolcs	id	true	1	gbunsafe6	32515	61785	0.791	1.282	invalid
32	false	dimacs	id	true	1	gbunsafe6	124890	328431	2.744	18.967	invalid
32	true	zolcs	id	false	1	gbunsafe6	19422	30704	0.380	0.401	invalid
32	true	dimacs	id	false	1	gbunsafe6	52895	146980	1.181	4.155	invalid
32	false	zolcs	id	false	1	gbunsafe6	32512	61783	0.722	1.352	invalid
32	false	dimacs	id	false	1	gbunsafe6	124890	328431	2.964	18.987	invalid
32	true	zolcs	id	true	2	gbunsafe6	19425	30705	0.351	0.401	invalid
32	true	dimacs	id	true	2	gbunsafe6	52895	146980	1.172	4.135	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	false	zolcs	id	true	2	gbunsafe6	32515	61785	0.721	1.282	invalid
32	false	dimacs	id	true	2	gbunsafe6	124890	328431	2.774	18.907	invalid
32	true	zolcs	id	false	2	gbunsafe6	19422	30704	0.370	0.391	invalid
32	true	dimacs	id	false	2	gbunsafe6	52895	146980	1.321	4.166	invalid
32	false	zolcs	id	false	2	gbunsafe6	32512	61783	0.841	1.352	invalid
32	false	dimacs	id	false	2	gbunsafe6	124890	328431	3.545	18.937	invalid
32	true	zolcs	syntactic	true	0	gbunsafe6	19393	30705	0.370	0.381	invalid
32	true	dimacs	syntactic	true	0	gbunsafe6	51741	146980	1.322	3.444	invalid
32	false	zolcs	syntactic	true	0	gbunsafe6	32483	61721	0.742	1.021	invalid
32	false	dimacs	syntactic	true	0	gbunsafe6	117191	306522	2.894	16.393	invalid
32	true	zolcs	syntactic	false	0	gbunsafe6	19358	30672	0.370	0.160	invalid
32	true	dimacs	syntactic	false	0	gbunsafe6	51709	146948	1.342	3.525	invalid
32	false	zolcs	syntactic	false	0	gbunsafe6	32448	61655	0.741	0.951	invalid
32	false	dimacs	syntactic	false	0	gbunsafe6	117159	306426	2.874	15.903	invalid
32	true	zolcs	syntactic	true	1	gbunsafe6	19393	30705	0.361	0.381	invalid
32	true	dimacs	syntactic	true	1	gbunsafe6	51741	146980	1.322	3.444	invalid
32	false	zolcs	syntactic	true	1	gbunsafe6	32483	61721	0.741	1.021	invalid
32	false	dimacs	syntactic	true	1	gbunsafe6	117191	306522	2.864	16.483	invalid
32	true	zolcs	syntactic	false	1	gbunsafe6	19358	30672	0.371	0.160	invalid
32	true	dimacs	syntactic	false	1	gbunsafe6	51709	146948	1.312	3.545	invalid
32	false	zolcs	syntactic	false	1	gbunsafe6	32448	61655	0.731	0.961	invalid
32	false	dimacs	syntactic	false	1	gbunsafe6	117159	306426	2.864	15.872	invalid
32	true	zolcs	syntactic	true	2	gbunsafe6	19393	30705	0.371	0.391	invalid
32	true	dimacs	syntactic	true	2	gbunsafe6	51741	146980	1.312	3.495	invalid
32	false	zolcs	syntactic	true	2	gbunsafe6	32483	61721	0.741	1.011	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	false	dimacs	syntactic	true	2	gbunsafe6	117191	306522	2.905	16.373	invalid
32	true	zolcs	syntactic	false	2	gbunsafe6	19358	30672	0.360	0.160	invalid
32	true	dimacs	syntactic	false	2	gbunsafe6	51709	146948	1.322	3.535	invalid
32	false	zolcs	syntactic	false	2	gbunsafe6	32448	61655	0.742	0.961	invalid
32	false	dimacs	syntactic	false	2	gbunsafe6	117159	306426	2.875	15.883	invalid
32	true	zolcs	semantic	true	2	gbunsafe6	19393	30705	0.360	0.381	invalid
32	true	dimacs	semantic	true	2	gbunsafe6	51741	146980	1.342	3.475	invalid
32	false	zolcs	semantic	true	2	gbunsafe6	32483	61721	0.751	1.011	invalid
32	false	dimacs	semantic	true	2	gbunsafe6	117191	306522	2.894	16.553	invalid
32	true	zolcs	semantic	false	2	gbunsafe6	19358	30672	0.361	0.160	invalid
32	true	dimacs	semantic	false	2	gbunsafe6	51709	146948	1.332	3.545	invalid
32	false	zolcs	semantic	false	2	gbunsafe6	32448	61655	0.751	0.951	invalid
32	false	dimacs	semantic	false	2	gbunsafe6	117159	306426	2.894	16.023	invalid
32	true	zolcs	semantic	true	2	gbunsafe6	19393	30705	0.370	0.381	invalid
32	true	dimacs	semantic	true	2	gbunsafe6	51741	146980	1.322	3.464	invalid
32	false	zolcs	semantic	true	2	gbunsafe6	32483	61721	0.731	1.011	invalid
32	false	dimacs	semantic	true	2	gbunsafe6	117191	306522	2.894	16.433	invalid
32	true	zolcs	semantic	false	2	gbunsafe6	19358	30672	0.361	0.160	invalid
32	true	dimacs	semantic	false	2	gbunsafe6	51709	146948	1.332	3.545	invalid
32	false	zolcs	semantic	false	2	gbunsafe6	32448	61655	0.741	0.951	invalid
32	false	dimacs	semantic	false	2	gbunsafe6	117159	306426	2.904	15.852	invalid
32	true	zolcs	semantic	true	2	gbunsafe6	19393	30705	0.371	0.381	invalid
32	true	dimacs	semantic	true	2	gbunsafe6	51741	146980	1.332	3.485	invalid
32	false	zolcs	semantic	true	2	gbunsafe6	32483	61721	0.741	1.011	invalid
32	false	dimacs	semantic	true	2	gbunsafe6	117191	306522	2.895	16.393	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
32	true	zolcs	semantic	false	2	gbunsafe6	19358	30672	0.360	0.160	invalid
32	true	dimacs	semantic	false	2	gbunsafe6	51709	146948	1.332	3.525	invalid
32	false	zolcs	semantic	false	2	gbunsafe6	32448	61655	0.752	0.961	invalid
32	false	dimacs	semantic	false	2	gbunsafe6	117159	306426	2.885	16.323	invalid
50	true	zolcs	id	true	0	gbunsafe6	61692	104574	1.312	6.600	invalid
50	true	dimacs	id	true	0	gbunsafe6	172631	514486	4.667	56.511	invalid
50	false	zolcs	id	true	0	gbunsafe6	108544	209766	2.804	12.277	invalid
50	false	dimacs	id	true	0	gbunsafe6	430317	1150179	12.769	321.252	invalid
50	true	zolcs	id	false	0	gbunsafe6	61689	104573	1.382	6.640	invalid
50	true	dimacs	id	false	0	gbunsafe6	172631	514486	4.196	56.842	invalid
50	false	zolcs	id	false	0	gbunsafe6	108541	209764	2.534	13.629	invalid
50	false	dimacs	id	false	0	gbunsafe6	430317	1150179	11.736	323.795	invalid
50	true	zolcs	id	true	1	gbunsafe6	61692	104574	1.362	6.619	invalid
50	true	dimacs	id	true	1	gbunsafe6	172631	514486	4.337	56.671	invalid
50	false	zolcs	id	true	1	gbunsafe6	108544	209766	2.814	12.297	invalid
50	false	dimacs	id	true	1	gbunsafe6	430317	1150179	12.408	321.442	invalid
50	true	zolcs	id	false	1	gbunsafe6	61689	104573	1.242	6.610	invalid
50	true	dimacs	id	false	1	gbunsafe6	172631	514486	4.466	56.712	invalid
50	false	zolcs	id	false	1	gbunsafe6	108541	209764	2.533	13.629	invalid
50	false	dimacs	id	false	1	gbunsafe6	430317	1150179	11.877	321.522	invalid
50	true	zolcs	id	true	2	gbunsafe6	61692	104574	1.282	6.599	invalid
50	true	dimacs	id	true	2	gbunsafe6	172631	514486	4.436	56.571	invalid
50	false	zolcs	id	true	2	gbunsafe6	108544	209766	2.544	12.267	invalid
50	false	dimacs	id	true	2	gbunsafe6	430317	1150179	11.596	321.492	invalid
50	true	zolcs	id	false	2	gbunsafe6	61689	104573	1.332	6.620	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	true	dimacs	id	false	2	gbunsafe6	172631	514486	4.446	56.561	invalid
50	false	zolcs	id	false	2	gbunsafe6	108541	209764	2.563	13.639	invalid
50	false	dimacs	id	false	2	gbunsafe6	430317	1150179	11.567	322.253	invalid
50	true	zolcs	syntactic	true	0	gbunsafe6	61642	104574	1.281	5.928	invalid
50	true	dimacs	syntactic	true	0	gbunsafe6	169929	514486	4.426	51.043	invalid
50	false	zolcs	syntactic	true	0	gbunsafe6	108494	209666	2.524	20.168	invalid
50	false	dimacs	syntactic	true	0	gbunsafe6	404189	1074549	10.706	280.923	invalid
50	true	zolcs	syntactic	false	0	gbunsafe6	61589	104523	1.302	11.096	invalid
50	true	dimacs	syntactic	false	0	gbunsafe6	169879	514436	4.436	47.838	invalid
50	false	zolcs	syntactic	false	0	gbunsafe6	108441	209564	2.544	20.258	invalid
50	false	dimacs	syntactic	false	0	gbunsafe6	404139	1074399	10.565	261.255	invalid
50	true	zolcs	syntactic	true	1	gbunsafe6	61642	104574	1.271	5.978	invalid
50	true	dimacs	syntactic	true	1	gbunsafe6	169929	514486	4.456	50.893	invalid
50	false	zolcs	syntactic	true	1	gbunsafe6	108494	209666	2.514	20.188	invalid
50	false	dimacs	syntactic	true	1	gbunsafe6	404189	1074549	10.796	280.513	invalid
50	true	zolcs	syntactic	false	1	gbunsafe6	61589	104523	1.302	11.126	invalid
50	true	dimacs	syntactic	false	1	gbunsafe6	169879	514436	4.487	47.869	invalid
50	false	zolcs	syntactic	false	1	gbunsafe6	108441	209564	2.544	20.168	invalid
50	false	dimacs	syntactic	false	1	gbunsafe6	404139	1074399	10.636	261.225	invalid
50	true	zolcs	syntactic	true	2	gbunsafe6	61642	104574	1.272	5.929	invalid
50	true	dimacs	syntactic	true	2	gbunsafe6	169929	514486	4.467	50.973	invalid
50	false	zolcs	syntactic	true	2	gbunsafe6	108494	209666	2.523	20.218	invalid
50	false	dimacs	syntactic	true	2	gbunsafe6	404189	1074549	10.805	280.583	invalid
50	true	zolcs	syntactic	false	2	gbunsafe6	61589	104523	1.302	11.116	invalid
50	true	dimacs	syntactic	false	2	gbunsafe6	169879	514436	4.446	47.918	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	false	zolcs	syntactic	false	2	gbunsafe6	108441	209564	2.544	20.228	invalid
50	false	dimacs	syntactic	false	2	gbunsafe6	404139	1074399	10.616	262.387	invalid
50	true	zolcs	semantic	true	2	gbunsafe6	61642	104574	1.272	5.929	invalid
50	true	dimacs	semantic	true	2	gbunsafe6	169929	514486	4.447	51.054	invalid
50	false	zolcs	semantic	true	2	gbunsafe6	108494	209666	2.524	20.168	invalid
50	false	dimacs	semantic	true	2	gbunsafe6	404189	1074549	10.625	280.974	invalid
50	true	zolcs	semantic	false	2	gbunsafe6	61589	104523	1.292	11.106	invalid
50	true	dimacs	semantic	false	2	gbunsafe6	169879	514436	4.487	47.878	invalid
50	false	zolcs	semantic	false	2	gbunsafe6	108441	209564	2.513	20.188	invalid
50	false	dimacs	semantic	false	2	gbunsafe6	404139	1074399	10.425	261.635	invalid
50	true	zolcs	semantic	true	2	gbunsafe6	61642	104574	1.271	5.908	invalid
50	true	dimacs	semantic	true	2	gbunsafe6	169929	514486	4.466	51.204	invalid
50	false	zolcs	semantic	true	2	gbunsafe6	108494	209666	2.463	20.208	invalid
50	false	dimacs	semantic	true	2	gbunsafe6	404189	1074549	10.556	280.473	invalid
50	true	zolcs	semantic	false	2	gbunsafe6	61589	104523	1.302	11.096	invalid
50	true	dimacs	semantic	false	2	gbunsafe6	169879	514436	4.467	47.938	invalid
50	false	zolcs	semantic	false	2	gbunsafe6	108441	209564	2.524	20.198	invalid
50	false	dimacs	semantic	false	2	gbunsafe6	404139	1074399	10.345	261.655	invalid
50	true	zolcs	semantic	true	2	gbunsafe6	61642	104574	1.352	5.959	invalid
50	true	dimacs	semantic	true	2	gbunsafe6	169929	514486	4.476	51.003	invalid
50	false	zolcs	semantic	true	2	gbunsafe6	108494	209666	2.494	20.238	invalid
50	false	dimacs	semantic	true	2	gbunsafe6	404189	1074549	10.586	280.604	invalid
50	true	zolcs	semantic	false	2	gbunsafe6	61589	104523	1.292	11.106	invalid
50	true	dimacs	semantic	false	2	gbunsafe6	169879	514436	4.497	47.948	invalid
50	false	zolcs	semantic	false	2	gbunsafe6	108441	209564	2.503	20.198	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
50	false	dimacs	semantic	false	2	gbunsafe6	404139	1074399	10.405	261.586	invalid
24	true	zolcs	id	true	0	sched	63573	59392	1.462	7.841	valid
24	true	dimacs	id	true	0	sched	674893	1695065			HySat error
24	false	zolcs	id	true	0	sched	69423	118389	2.303	8.111	valid
24	false	dimacs	id	true	0	sched	1239418	3464187			HySat error
24	true	zolcs	id	false	0	sched	63573	59392	1.603	7.831	valid
24	true	dimacs	id	false	0	sched	674893	1695065			HySat error
24	false	zolcs	id	false	0	sched	69423	118389	2.334	8.111	valid
24	false	dimacs	id	false	0	sched	1239418	3464187			HySat error
24	true	zolcs	id	true	1	sched	60323	57067	1.452	7.400	valid
24	true	dimacs	id	true	1	sched	630043	1569644			HySat error
24	false	zolcs	id	true	1	sched	66173	113513	2.504	6.870	valid
24	false	dimacs	id	true	1	sched	1141918	3191387			HySat error
24	true	zolcs	id	false	1	sched	60323	57067	1.412	7.400	valid
24	true	dimacs	id	false	1	sched	630043	1569644			HySat error
24	false	zolcs	id	false	1	sched	66173	113513	2.203	6.900	valid
24	false	dimacs	id	false	1	sched	1141918	3191387			HySat error
24	true	zolcs	id	true	2	sched	60323	57067	1.742	7.390	valid
24	true	dimacs	id	true	2	sched	630043	1569644			HySat error
24	false	zolcs	id	true	2	sched	66173	113513	2.113	6.899	valid
24	false	dimacs	id	true	2	sched	1141918	3191387			HySat error
24	true	zolcs	id	false	2	sched	60323	57067	1.522	7.410	valid
24	true	dimacs	id	false	2	sched	630043	1569644			HySat error
24	false	zolcs	id	false	2	sched	66173	113513	2.073	6.899	valid
24	false	dimacs	id	false	2	sched	1141918	3191387			HySat error

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
24	true	zolcs	syntactic	true	0	sched	60973	59392	1.562	7.711	valid
24	true	dimacs	syntactic	true	0	sched	245893	675740	6.339	4.426	valid
24	false	zolcs	syntactic	true	0	sched	66823	114922	2.023	7.911	valid
24	false	dimacs	syntactic	true	0	sched	310243	862212	8.212	6.580	valid
24	true	zolcs	syntactic	false	0	sched	60648	59392	1.132	7.711	valid
24	true	dimacs	syntactic	false	0	sched	245568	675740	6.349	4.416	valid
24	false	zolcs	syntactic	false	0	sched	66498	114272	2.013	7.871	valid
24	false	dimacs	syntactic	false	0	sched	309918	861237	8.342	6.570	valid
24	true	zolcs	syntactic	true	1	sched	58048	57067	1.062	7.310	valid
24	true	dimacs	syntactic	true	1	sched	237118	633769	5.979	4.036	valid
24	false	zolcs	syntactic	true	1	sched	63898	110371	2.143	6.660	valid
24	false	dimacs	syntactic	true	1	sched	295618	820937	7.812	6.108	valid
24	true	zolcs	syntactic	false	1	sched	58048	57067	1.242	7.310	valid
24	true	dimacs	syntactic	false	1	sched	237118	633769	6.049	4.006	valid
24	false	zolcs	syntactic	false	1	sched	63898	110371	1.953	6.660	valid
24	false	dimacs	syntactic	false	1	sched	295618	820937	8.512	6.078	valid
24	true	zolcs	syntactic	true	2	sched	58048	57067	1.122	7.320	valid
24	true	dimacs	syntactic	true	2	sched	237118	633769	6.038	4.006	valid
24	false	zolcs	syntactic	true	2	sched	63898	110371	2.163	6.660	valid
24	false	dimacs	syntactic	true	2	sched	295618	820937	8.081	6.109	valid
24	true	zolcs	syntactic	false	2	sched	58048	57067	1.101	7.300	valid
24	true	dimacs	syntactic	false	2	sched	237118	633769	6.429	4.016	valid
24	false	zolcs	syntactic	false	2	sched	63898	110371	1.962	6.670	valid
24	false	dimacs	syntactic	false	2	sched	295618	820937	8.012	6.088	valid
24	true	zolcs	semantic	true	2	sched					Out of memory

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
24	true	dimacs	semantic	true	2	sched					Out of memory
24	false	zolcs	semantic	true	2	sched					Out of memory
24	false	dimacs	semantic	true	2	sched					Out of memory
24	true	zolcs	semantic	false	2	sched					Out of memory
24	true	dimacs	semantic	false	2	sched					Out of memory
24	false	zolcs	semantic	false	2	sched					Out of memory
24	false	dimacs	semantic	false	2	sched					Out of memory
24	true	zolcs	semantic	true	2	sched					Out of memory
24	true	dimacs	semantic	true	2	sched					Out of memory
24	false	zolcs	semantic	true	2	sched					Out of memory
24	false	dimacs	semantic	true	2	sched					Out of memory
24	true	zolcs	semantic	false	2	sched					Out of memory
24	true	dimacs	semantic	false	2	sched					Out of memory
24	false	zolcs	semantic	false	2	sched					Out of memory
24	false	dimacs	semantic	false	2	sched					Out of memory
24	true	zolcs	semantic	true	2	sched					Out of memory
24	true	dimacs	semantic	true	2	sched					Out of memory
24	false	zolcs	semantic	true	2	sched					Out of memory
24	false	dimacs	semantic	true	2	sched					Out of memory
24	true	zolcs	semantic	false	2	sched					Out of memory
24	true	dimacs	semantic	false	2	sched					Out of memory
24	false	zolcs	semantic	false	2	sched					Out of memory
24	false	dimacs	semantic	false	2	sched					Out of memory
24	true	zolcs	id	true	0	insched	47973	46614	1.152	1.002	invalid
24	true	dimacs	id	true	0	insched	487368	1209465	16.503	220.557	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
24	false	zolcs	id	true	0	insched	53823	92689	1.763	0.561	invalid
24	false	dimacs	id	true	0	insched	879318	2453987			HySat error
24	true	zolcs	id	false	0	insched	47973	46614	1.032	1.012	invalid
24	true	dimacs	id	false	0	insched	487368	1209465	12.398	220.166	invalid
24	false	zolcs	id	false	0	insched	53823	92689	1.613	0.581	invalid
24	false	dimacs	id	false	0	insched	879318	2453987			HySat error
24	true	zolcs	id	true	1	insched	46023	45264	0.961	0.901	invalid
24	true	dimacs	id	true	1	insched	462018	1134369	11.447	196.303	invalid
24	false	zolcs	id	true	1	insched	51873	89764	1.542	0.560	invalid
24	false	dimacs	id	true	1	insched	818868	2285087			HySat error
24	true	zolcs	id	false	1	insched	46023	45264	0.972	0.881	invalid
24	true	dimacs	id	false	1	insched	462018	1134369	11.526	196.613	invalid
24	false	zolcs	id	false	1	insched	51873	89764	1.552	0.570	invalid
24	false	dimacs	id	false	1	insched	818868	2285087			HySat error
24	true	zolcs	id	true	2	insched	46023	45264	0.971	0.901	invalid
24	true	dimacs	id	true	2	insched	462018	1134369	11.537	196.703	invalid
24	false	zolcs	id	true	2	insched	51873	89764	1.542	0.581	invalid
24	false	dimacs	id	true	2	insched	818868	2285087			HySat error
24	true	zolcs	id	false	2	insched	46023	45264	0.952	0.871	invalid
24	true	dimacs	id	false	2	insched	462018	1134369	11.576	196.473	invalid
24	false	zolcs	id	false	2	insched	51873	89764	1.532	0.550	invalid
24	false	dimacs	id	false	2	insched	818868	2285087			HySat error
24	true	zolcs	syntactic	true	0	insched	46023	46614	0.951	0.841	invalid
24	true	dimacs	syntactic	true	0	insched	197468	522340	4.597	18.276	invalid
24	false	zolcs	syntactic	true	0	insched	51873	90161	1.553	0.560	invalid

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
24	false	dimacs	syntactic	true	0	insched	276443	766012	7.090	25.596	invalid
24	true	zolcs	syntactic	false	0	insched	45698	46614	0.821	0.911	invalid
24	true	dimacs	syntactic	false	0	insched	197143	522340	4.606	18.907	invalid
24	false	zolcs	syntactic	false	0	insched	51548	89511	1.552	0.550	invalid
24	false	dimacs	syntactic	false	0	insched	276118	765037	6.870	25.456	invalid
24	true	zolcs	syntactic	true	1	insched	44398	45264	0.791	0.771	invalid
24	true	dimacs	syntactic	true	1	insched	192918	493369	4.326	18.376	invalid
24	false	zolcs	syntactic	true	1	insched	50248	87561	1.662	0.540	invalid
24	false	dimacs	syntactic	true	1	insched	268968	744887	6.690	24.725	invalid
24	true	zolcs	syntactic	false	1	insched	44398	45264	0.781	0.841	invalid
24	true	dimacs	syntactic	false	1	insched	192918	493369	4.347	18.656	invalid
24	false	zolcs	syntactic	false	1	insched	50248	87561	1.462	0.560	invalid
24	false	dimacs	syntactic	false	1	insched	268968	744887	6.700	24.705	invalid
24	true	zolcs	syntactic	true	2	insched	44398	45264	0.761	0.771	invalid
24	true	dimacs	syntactic	true	2	insched	192918	493369	4.647	18.356	invalid
24	false	zolcs	syntactic	true	2	insched	50248	87561	1.472	0.550	invalid
24	false	dimacs	syntactic	true	2	insched	268968	744887	6.870	24.755	invalid
24	true	zolcs	syntactic	false	2	insched	44398	45264	0.831	0.851	invalid
24	true	dimacs	syntactic	false	2	insched	192918	493369	4.376	18.646	invalid
24	false	zolcs	syntactic	false	2	insched	50248	87561	1.853	0.550	invalid
24	false	dimacs	syntactic	false	2	insched	268968	744887	8.782	24.746	invalid
24	true	zolcs	semantic	true	2	insched					Out of memory
24	true	dimacs	semantic	true	2	insched					Out of memory
24	false	zolcs	semantic	true	2	insched					Out of memory
24	false	dimacs	semantic	true	2	insched					Out of memory

k	Polarity optimisation	Output format	Formula recognition	NNF	DC simp. level	Problem	No. of variables	No. of constraints	Frontend time	Backend time	Result
24	true	zolcs	semantic	false	2	insched					Out of memory
24	true	dimacs	semantic	false	2	insched					Out of memory
24	false	zolcs	semantic	false	2	insched					Out of memory
24	false	dimacs	semantic	false	2	insched					Out of memory
24	true	zolcs	semantic	true	2	insched					Out of memory
24	true	dimacs	semantic	true	2	insched					Out of memory
24	false	zolcs	semantic	true	2	insched					Out of memory
24	false	dimacs	semantic	true	2	insched					Out of memory
24	true	zolcs	semantic	false	2	insched					Out of memory
24	true	dimacs	semantic	false	2	insched					Out of memory
24	false	zolcs	semantic	false	2	insched					Out of memory
24	false	dimacs	semantic	false	2	insched					Out of memory
24	true	zolcs	semantic	true	2	insched					Out of memory
24	true	dimacs	semantic	true	2	insched					Out of memory
24	false	zolcs	semantic	true	2	insched					Out of memory
24	false	dimacs	semantic	true	2	insched					Out of memory
24	true	zolcs	semantic	false	2	insched					Out of memory
24	true	dimacs	semantic	false	2	insched					Out of memory
24	false	zolcs	semantic	false	2	insched					Out of memory
24	false	dimacs	semantic	false	2	insched					Out of memory

Appendix I

Source Code

This appendix contains all of the source code for the BMC/DCValidator, but none of the source code for the tests (these are placed in Appendix G on page 126). The source code has been divided into sections by packages and follows in alphabetical order by package name and class name.

I.1 Package `bmc.constraint`

I.1.1 `Constraint.java`

```
1  package bmc.constraint;
2
3  import bmc.literal.*;
4  import bmc.util.*;
5
6  public abstract class Constraint {
7
8      protected IntBag literals;
9
10     public Constraint() {
11         this.literals = new IntBag();
12     }
13
14     public abstract void addTerm(Literal literal);
15     public abstract boolean isTriviallyTrue();
16     public abstract String toString();
17
18 }
```

I.1.2 `ConstraintWriter.java`

```
1  package bmc.constraint;
2
3  import bmc.constraint.*;
4  import bmc.literal.*;
5  import bmc.formula.*;
```

```
6  import bmc.util.*;
7
8  import java.io.*;
9  import java.util.*;
10
11 public class ConstraintWriter {
12     protected int numClauses;
13     protected RandomAccessFile output;
14     protected LiteralHandler literalHandler;
15
16     protected final static int NO_BLANKS = 10;
17
18     protected long noVarsPtr;
19
20     public ConstraintWriter(LiteralHandler literalHandler
           , Formula formula,
21     Settings settings, RandomAccessFile output) throws
           IOException {
22         this.literalHandler = literalHandler;
23         this.output = output;
24         this.numClauses = 0;
25         writeComment("k="+settings.getK());
26         writeComment("formula="+formula);
27         writeComment("polarityOpt="+settings.
               getPolarityOpt());
28         writeComment("dcSimpLevel="+settings.
               getDCSimpLevel());
29         writeComment("nnf="+settings.getNNF());
30         writeComment("outputFormat="+
31         (settings.getOutputFormat() == Settings.
               OUTPUT_FORMAT_ZOLCS ? "zolcs" : "dimacs"));
32         writeComment("fRecognition="+
33         (settings.getFRecognition() == Settings.
               F_RECOGNITION_ID ? "id" :
34             (settings.getFRecognition() == Settings.
                   F_RECOGNITION_SYNTACTIC ? "syntactic"
35             : "semantic")));
36             writeComment("outputFolder="+( settings.
                   getOutputFolder().equals("") ?
37             "(current folder)" : settings.getOutputFolder
                   ()));
38             output.writeBytes("p ");
39             if (settings.getOutputFormat() == Settings.
                   OUTPUT_FORMAT_ZOLCS) {
40                 output.writeBytes("zolcs");
41             }
42             else {
43                 output.writeBytes("cnf");
44             }
45             output.writeBytes(" ");
```

```
46              noVarsPtr = output.getFilePointer();
47              StringBuffer strBlank = new StringBuffer();
48              for (int i=0; i<NO_BLANKS*2+1; i++) {
49                  strBlank.append(" ");
50              }
51              output.writeBytes(strBlank.toString());
52              output.writeBytes("\n\n");
53
54              // adding constraint for true literal
55              if (settings.getOutputFormat() == Settings.
                    OUTPUT_FORMAT_ZOLCS) {
56                  output.writeBytes("1 "+
                        VariableNoGenerator.BOOL_LIT_NO+" 1 0\
                        n");
57              }
58              else {
59                  output.writeBytes(VariableNoGenerator.
                        BOOL_LIT_NO+" 0\n");
60              }
61              numClauses++;
62      }
63
64      public void finish() throws IOException {
65          String numVarStr = Integer.toString(
                literalHandler.getNumVariables());
66          String numCstrStr = Integer.toString(numClauses);
67          if (numVarStr.length() + numCstrStr.length() > 2*
                NO_BLANKS) {
68              throw new IllegalStateException("Not enough
                    space reserved in file");
69          }
70          output.seek(noVarsPtr);
71          output.writeBytes(numVarStr);
72          output.writeBytes(" ");
73          output.writeBytes(numCstrStr);
74      }
75
76      public void writeConstraint(Constraint constraint)
            throws IOException {
77          numClauses++;
78          output.writeBytes(constraint.toString());
79      }
80
81      private void writeComment(String comment)
82      throws IOException {
83          StringBuffer buffer = new StringBuffer();
84          if (comment.length() > 0) {
85              buffer.append("c ");
86              buffer.append(comment);
87          }
```

```
88          buffer.append("\n");
89          output.writeBytes(buffer.toString());
90      }
91
92  }
```

I.1.3 DIMACSConstraint.java

```
1   package bmc.constraint;
2
3   import java.util.*;
4   import bmc.literal.*;
5
6   public class DIMACSConstraint extends Constraint {
7
8       boolean isTriviallyTrue;
9
10      public DIMACSConstraint() {
11          this.isTriviallyTrue = false;
12      }
13
14      public boolean isTriviallyTrue() {
15          return isTriviallyTrue;
16      }
17
18      public void addTerm(Literal literal) {
19          int lit = literal.getSign() ? literal.getNumber()
                : -literal.getNumber();
20          if (lit == VariableNoGenerator.BOOL_LIT_NO) {
21              isTriviallyTrue = true;
22              literals.add(lit);
23              return;
24          }
25          if (lit == -VariableNoGenerator.BOOL_LIT_NO) {
26              return;
27          }
28          for (int i=0; i<literals.size(); i++) {
29              int lit2 = literals.get(i);
30              if (lit == lit2) {
31                  return;
32              }
33              if (lit == -lit2) {
34                  isTriviallyTrue = true;
35                  literals.add(lit);
36                  return;
37              }
38          }
39          literals.add(lit);
40      }
41
```

```
42      public String toString() {
43          StringBuffer buffer = new StringBuffer();
44          if (literals.size() > 0) {
45              for (int i=0; i<literals.size(); i++) {
46                  int lit = literals.get(i);
47                  buffer.append(lit).append(" ");
48              }
49          }
50          else {
51              buffer.append("-").append(VariableNoGenerator
                    .BOOL_LIT_NO).append(" ");
52          }
53          buffer.append("0\n");
54          return buffer.toString();
55      }
56
57  }
```

I.1.4 ZOLCSConstraint.java

```
1   package bmc.constraint;
2
3   import bmc.util.*;
4   import bmc.literal.*;
5
6   public class ZOLCSConstraint extends Constraint {
7       private IntBag weights;
8       private int threshold;
9
10      public ZOLCSConstraint() {
11          this(1);
12      }
13
14      public ZOLCSConstraint(int threshold) {
15          this.threshold = threshold;
16          this.weights = new IntBag();
17      }
18
19      public boolean isTriviallyTrue() {
20          return threshold <= 0;
21      }
22
23      public void addTerm(Literal literal) {
24          addTerm(literal, 1);
25      }
26
27      public void addTerm(Literal literal, int weight) {
28          int lit1 = literal.getSign() ? literal.getNumber
                () : -(literal.getNumber());
29          if (weight == 0) {
```

```
30              return;
31          }
32          if (lit1 == -VariableNoGenerator.BOOL_LIT_NO) {
33              return;
34          }
35          if (lit1 == VariableNoGenerator.BOOL_LIT_NO) {
36              threshold -= weight;
37              return;
38          }
39          for (int i=0; i<literals.size(); i++) {
40              int lit2 = literals.get(i);
41              if (Math.abs(lit1) == Math.abs(lit2)) { //
                    same literal no
42                  int weight2 = weights.get(i);
43                  if (lit1 == lit2) { // same literal no,
                        same sign
44                      weights.set(i, weight+weight2);
45                  }
46                  else { // different signs, same literal
                        no
47                      if (weight2 > weight) {
48                          weights.set(i, weight2-weight);
49                          threshold -= weight;
50                      }
51                      else if (weight2 == weight) {
52                          literals.remove(i);
53                          weights.remove(i);
54                          threshold -= weight;
55                      }
56                      else { // weight > weight2
57                          weights.set(i, weight-weight2);
58                          literals.set(i, lit1);
59                          threshold -= weight2;
60                      }
61                  }
62                  return;
63              }
64          }
65          literals.add(lit1);
66          weights.add(weight);
67      }
68
69      public String toString() {
70          StringBuffer buffer = new StringBuffer();
71          if (literals.size() > 0) {
72              for (int i=0; i<literals.size(); i++) {
73                  int lit = literals.get(i);
74                  int weight = weights.get(i);
75                  buffer.append(weight).append(" ");
76                  buffer.append(lit).append(" ");
```

```
77              }
78          }
79          else {
80              buffer.append("1 -").append(
                    VariableNoGenerator.BOOL_LIT_NO).append("
                    ");
81          }
82          buffer.append(threshold).append(" 0\n");
83          return buffer.toString();
84      }
85
86  }
```

I.2 Package `bmc.formula`

I.2.1 `Chop.java`

```
1   package bmc.formula;
2
3   import java.util.Map;
4   import java.lang.reflect.*;
5
6   public class Chop extends Connective {
7
8       private String symbol = ";";
9
10      public Chop(Formula left, Formula right) {
11          super(left, right);
12      }
13
14      public int getType() {
15          return Formula.TYPE_CHOP;
16      }
17
18      public String getSymbol() {
19          return symbol;
20      }
21  }
```

I.2.2 `Conjunction.java`

```
1   package bmc.formula;
2
3   public class Conjunction extends Connective {
4
5       private String symbol = "/\\";
6
7       public Conjunction(Formula left, Formula right) {
8           super(left, right);
9       }
```

```
10
11      public int getType() {
12          return Formula.TYPE_CONJUNCTION;
13      }
14
15      public String getSymbol() {
16          return symbol;
17      }
18
19  }
```

I.2.3 Connective.java

```
1   package bmc.formula;
2
3   import java.util.*;
4   import java.lang.reflect.*;
5
6   public abstract class Connective extends Formula {
7
8       protected Formula left;
9       protected Formula right;
10
11      protected static Method L_OP_SET_METHOD,
            R_OP_SET_METHOD;
12      static {
13          try {
14              L_OP_SET_METHOD = Connective.class.getMethod
                    ("setLeftOperand",Formula.CLASS_ARR);
15              R_OP_SET_METHOD = Connective.class.getMethod
                    ("setRightOperand",Formula.CLASS_ARR);
16          }
17          catch (Exception e) {System.out.println(e); }
18      }
19
20
21      public Connective(Formula left, Formula right) {
22          this.left = left;
23          this.right = right;
24          left.parent = this;
25          right.parent = this;
26          left.setSetMethod(L_OP_SET_METHOD);
27          right.setSetMethod(R_OP_SET_METHOD);
28      }
29
30      public Formula getLeftOperand() {
31          return left;
32      }
33
34      public Formula getRightOperand() {
```

```
35          return right;
36      }
37
38      public abstract String getSymbol();
39
40      public void setLeftOperand(Formula left) {
41          if (this.left != null) {
42              this.left.parent = null;
43          }
44          this.left = left;
45          left.parent = this;
46          left.setSetMethod(L_OP_SET_METHOD);
47          nullReps();
48      }
49
50      public void setRightOperand(Formula right) {
51          if (this.right != null) {
52              this.right.parent = null;
53          }
54          this.right = right;
55          right.parent = this;
56          right.setSetMethod(R_OP_SET_METHOD);
57          nullReps();
58      }
59
60      public Object clone() throws
            CloneNotSupportedException {
61          // Deep clone
62          Connective con = (Connective) super.clone();
63          con.left = (Formula) left.clone();
64          con.left.parent = con;
65          con.right = (Formula) right.clone();
66          con.right.parent = con;
67          return con;
68      }
69
70      public void getStates(Collection states) {
71          left.getStates(states);
72          right.getStates(states);
73      }
74
75      public String toString() {
76          if (stringRep == null) {
77              StringBuffer buffer = new StringBuffer();
78              buffer.append("(").append(left).append(" ").
                    append(this.getSymbol());
79              buffer.append(" ").append(right).append(")");
80              stringRep = buffer.toString();
81          }
82          return stringRep;
```

```
83      }
84
85  }
```

I.2.4 Disjunction.java

```
1   package bmc.formula;
2
3   public class Disjunction extends Connective {
4
5       private String symbol = "\\/";
6
7       public Disjunction(Formula left, Formula right) {
8           super(left, right);
9       }
10
11      public int getType() {
12          return Formula.TYPE_DISJUNCTION;
13      }
14
15      public String getSymbol() {
16          return symbol;
17      }
18
19  }
```

I.2.5 Duration.java

```
1   package bmc.formula;
2
3   import java.util.*;
4
5   public class Duration extends Formula {
6
7       private Formula stateAss;
8       private int bound;
9
10      public Duration(Formula stateAss, int bound) {
11          this.stateAss = stateAss;
12          if (bound <= 0) {
13              throw new IllegalArgumentException("Duration
                    bound must be greater than zero");
14          }
15          this.bound = bound;
16      }
17
18      public Formula getStateAssertion() {
19          return stateAss;
20      }
21
```

```
22      public int getBound() {
23          return bound;
24      }
25
26      public String toString() {
27          if (stringRep == null) {
28              StringBuffer buffer = new StringBuffer();
29              buffer.append("dur ").append(stateAss).append
                    (" >= ").append(bound);
30              stringRep = buffer.toString();
31          }
32          return stringRep;
33      }
34
35      public void setStateAssertion(Formula stateAss) {
36          this.stateAss = stateAss;
37          nullReps();
38      }
39
40      public int getType() {
41          return Formula.TYPE_DURATION;
42      }
43
44      public void getStates(Collection states) {
45          stateAss.getStates(states);
46      }
47
48      public Object clone() throws
            CloneNotSupportedException{
49          Duration clone = (Duration) super.clone();
50          clone.stateAss = (Formula) this.stateAss.clone();
51          clone.bound = this.bound;
52          return clone;
53      }
54
55  }
```

I.2.6 False.java

```
1   package bmc.formula;
2
3   import java.util.*;
4
5   public class False extends Formula {
6
7       public static final String FALSE_STR = "false";
8
9       public int getType() {
10          return Formula.TYPE_FALSE;
11      }
```

```
12
13      public String toString() {
14          return FALSE_STR;
15      }
16
17      public void getStates(Collection states) {
18      }
19
20  }
```

I.2.7 Formula.java

```
1   package bmc.formula;
2
3   import java.lang.reflect.*;
4   import java.util.*;
5   import bmc.robdd.*;
6
7   public abstract class Formula implements Cloneable {
8
9
10      private static int nextID = 1;
11      protected int id;
12
13      protected Method setMethod;
14      protected Formula parent;
15
16      public static final int TYPE_UNKNOWN = 0;
17      public static final int TYPE_DURATION = 1;
18      public static final int TYPE_CONJUNCTION = 2;
19      public static final int TYPE_NEGATION = 3;
20      public static final int TYPE_CHOP = 4;
21      public static final int TYPE_DISJUNCTION = 5;
22      public static final int TYPE_TRUE = 6;
23      public static final int TYPE_FALSE = 7;
24      public static final int TYPE_STATE = 8;
25
26      protected String stringRep;
27
28      protected ROBDD robdd;
29
30      protected static final Class[] CLASS_ARR = new Class
            []{Formula.class};
31
32      public Formula() {
33          reassignID();
34      }
35
36      public Formula getParent() {
37          return parent;
```

```
38      }
39
40      public Formula getGrandParent() {
41          return parent.parent;
42      }
43
44      public abstract int getType();
45
46      // Force subclass implementation of toString()
47      public abstract String toString();
48
49      protected void nullReps() {
50          stringRep = null;
51          robdd = null;
52          reassignID();
53          if (hasParent()) {
54              getParent().nullReps();
55          }
56      }
57
58      public int getID() {
59          return id;
60      }
61
62      public boolean hasParent() {
63          return parent != null;
64      }
65
66      public boolean hasGrandParent() {
67          return parent != null && parent.hasParent();
68      }
69
70      public abstract void getStates(Collection states);
71
72      protected void setSetMethod(Method setMethod) {
73          this.setMethod = setMethod;
74      }
75
76      public void updateParent(Formula formula)
77      throws Exception {
78          setMethod.invoke(parent, new Object[]{formula});
79      }
80
81      public void updateGrandParent(Formula formula)
82      throws Exception {
83          parent.setMethod.invoke(parent.parent, new Object
                []{formula});
84
85      }
86
```

```
87      protected void reassignID() {
88          this.id = nextID++;
89      }
90
91      public Object clone() throws
            CloneNotSupportedException {
92          Formula f = (Formula) super.clone();
93          f.parent = null;
94          return f;
95      }
96
97      public ROBDD getROBDD(ROBDDUtil robddUtil) throws
            CloneNotSupportedException {
98          if (robdd != null) {
99              return robdd;
100         }
101         robdd = robddUtil.makeROBDD(this);
102         return robdd;
103     }
104
105 }
```

I.2.8 Negation.java

```
1   package bmc.formula;
2
3   import java.util.*;
4   import java.lang.reflect.*;
5
6   public class Negation extends Formula {
7
8       private Formula operand;
9
10      protected static Method OP_SET_METHOD;
11      static {
12          try {
13              OP_SET_METHOD = Negation.class.getMethod("
                    setOperand",Formula.CLASS_ARR);
14          }
15          catch (Exception e) {System.out.println(e); }
16      }
17
18      public Negation(Formula operand) {
19          this.operand = operand;
20          operand.parent = this;
21          operand.setSetMethod(OP_SET_METHOD);
22      }
23
24      public Formula getOperand() {
25          return operand;
```

```
26      }
27
28      public void setOperand(Formula operand) {
29          this.operand = operand;
30          operand.parent = this;
31          operand.setSetMethod(OP_SET_METHOD);
32          nullReps();
33      }
34
35      public String toString() {
36          if (stringRep == null) {
37              stringRep = "~"+operand;
38          }
39          return stringRep;
40      }
41
42      public int getType() {
43          return Formula.TYPE_NEGATION;
44      }
45
46      public Object clone() throws
            CloneNotSupportedException {
47          // Deep clone
48          Negation neg = (Negation) super.clone();
49          neg.operand = (Formula) operand.clone();
50          neg.operand.parent = neg;
51          return neg;
52      }
53
54      public void getStates(Collection states) {
55          operand.getStates(states);
56      }
57
58  }
```

I.2.9 State.java

```
1   package bmc.formula;
2
3   import java.util.*;
4
5   public class State extends Formula implements Comparable
        {
6
7       private String name;
8
9       public State(String name) {
10          this.name = name;
11      }
12
```

```
13      public String getName() {
14          return name;
15      }
16
17      public int getType() {
18          return Formula.TYPE_STATE;
19      }
20
21      public String toString() {
22          return name;
23      }
24
25      public void getStates(Collection states) {
26          states.add(this);
27      }
28
29      public int compareTo(Object obj) {
30          State s = (State) obj;
31          return name.compareTo(s.name);
32      }
33
34  }
```

I.2.10 `True.java`

```
1   package bmc.formula;
2
3   import java.util.*;
4
5   public class True extends Formula {
6
7       public static final String TRUE_STR = "true";
8
9       public int getType() {
10          return Formula.TYPE_TRUE;
11      }
12
13      public String toString() {
14          return TRUE_STR;
15      }
16
17      public void getStates(Collection states) {
18      }
19
20  }
```

I.3 Package bmc.literal

I.3.1 `IDLookupLiteralHandler.java`

```
1  package bmc.literal;
2
3  import java.util.*;
4  import bmc.*;
5  import bmc.formula.*;
6  import bmc.util.*;
7
8  public class IDLookupLiteralHandler extends
       LiteralHandler {
9
10
11     public IDLookupLiteralHandler() {
12     }
13
14     protected int getNormalizedID(Formula formula) {
15         return formula.getID();
16     }
17
18 }
```

I.3.2 IndexedFormula.java

```
1  package bmc.literal;
2
3  import bmc.*;
4
5  public class IndexedFormula {
6
7      protected int formulaID;
8      protected int i;
9      protected int j;
10     protected int m;
11
12     public IndexedFormula(int formulaID, int i, int m,
           int j) {
13         this.formulaID = formulaID;
14         this.i = i;
15         this.j = j;
16         this.m = m;
17     }
18
19     public IndexedFormula(int formulaID, int i, int j) {
20         this(formulaID, i, -1, j);
21     }
22
23     public IndexedFormula(int formulaID, int i) {
24         this(formulaID, i, -1, -1);
25     }
26
27     public IndexedFormula(int formulaID) {
```

```
28          this(formulaID, -1, -1, -1);
29      }
30
31
32      public boolean equals(Object o) {
33          if (!(o instanceof IndexedFormula)) {
34              return false;
35          }
36          IndexedFormula iF = (IndexedFormula) o;
37          return iF.formulaID == formulaID &&
38          iF.i == i &&
39          iF.j == j &&
40          iF.m == m;
41      }
42
43      public int hashCode() {
44          return ((int) formulaID) ^ (i << 24) ^ (j << 16)
                ^ (m << 8);
45      }
46
47      public String toString() {
48          return "IF["+formulaID+","+i+","+m+","+j+"]";
49      }
50  }
```

I.3.3 Literal.java

```
1   package bmc.literal;
2
3   import java.util.List;
4
5   public class Literal implements Comparable {
6
7       private int number;
8       private boolean sign;
9
10      protected Literal(boolean sign) {
11          this.number = VariableNoGenerator.BOOL_LIT_NO;
12          this.sign = sign;
13      }
14
15      public Literal(int number, boolean sign) {
16          if (number == VariableNoGenerator.BOOL_LIT_NO) {
17              throw new IllegalArgumentException("number is
                    not allowed to have same value as
                   BOOL_LIT_NO");
18          }
19          if (number <= 0) {
20              throw new IllegalArgumentException("number is
                    not allowed to be less that on equal to
```

```
                0");
21          }
22          this.number = number;
23          this.sign = sign;
24      }
25
26      public static Literal getTrueLiteral() {
27          return new Literal(true);
28      }
29
30      public static Literal getFalseLiteral() {
31          return new Literal(false);
32      }
33
34      public int getNumber() {
35          return number;
36      }
37
38      public boolean getSign() {
39          return sign;
40      }
41
42      public Literal flipSign() {
43          if (number == VariableNoGenerator.BOOL_LIT_NO) {
44              return new Literal(!sign);
45          }
46          return new Literal(number, !sign);
47      }
48
49      public int compareTo(Object o) {
50          Literal l = (Literal) o;
51          int result = number - l.number;
52          if (result == 0) {
53              if (sign != l.sign) {
54                  if (sign) {
55                      return 1;
56                  }
57                  else {
58                      return -1;
59                  }
60              }
61          }
62          return result;
63      }
64
65      public String toString() {
66          return sign ? "x"+number : "~x"+number;
67      }
68
69      public boolean equals(Object o) {
```

```
70          if (!(o instanceof Literal)) {
71              return false;
72          }
73          Literal lit = (Literal) o;
74          if (lit.number != number) {
75              return false;
76          }
77          return lit.sign == sign;
78      }
79
80      public int hashCode() {
81          if (sign) {
82              return number;
83          }
84          else {
85              return -number;
86          }
87      }
88  }
```

I.3.4 LiteralHandler.java

```
1   package bmc.literal;
2
3   import bmc.*;
4   import bmc.formula.*;
5   import bmc.util.*;
6
7   import java.util.*;
8
9   public abstract class LiteralHandler {
10      protected Map formulaIDToNormID;
11      protected Map normIDToLiteral;
12      protected Set normIDs;
13      protected Literal falseLit, trueLit;
14      protected VariableNoGenerator varNoGenerator;
15
16      public LiteralHandler() {
17          this.varNoGenerator = new VariableNoGenerator();
18          formulaIDToNormID = new HashMap();
19          normIDToLiteral = new HashMap();
20          normIDs = new HashSet();
21          trueLit = Literal.getTrueLiteral();
22          falseLit = Literal.getFalseLiteral();
23      }
24
25
26      public int getNumVariables() {
27          return varNoGenerator.getNumVariables();
28      }
```

```
29
30
31      public Literal getLiteral(Formula formula, int i)
32      throws CloneNotSupportedException {
33          return getLiteral(formula, i, -1, -1);
34      }
35
36      public Literal getLiteral(Formula formula, int i, int
            j)
37      throws CloneNotSupportedException {
38          return getLiteral(formula, i, -1, j);
39      }
40
41      public Literal getLiteral(Formula formula, int i, int
            m, int j)
42      throws CloneNotSupportedException {
43          boolean sign = true;
44          while (formula.getType() == Formula.TYPE_NEGATION
                ) {
45              sign = !sign;
46              formula = ((Negation) formula).getOperand();
47          }
48          if (formula.getType() == Formula.TYPE_TRUE &&
                sign ||
49          formula.getType() == Formula.TYPE_FALSE && !sign)
                 {
50              return trueLit;
51          }
52          else if (formula.getType() == Formula.TYPE_FALSE
                && sign ||
53          formula.getType() == Formula.TYPE_TRUE && !sign)
                {
54              return falseLit;
55          }
56          else {
57              Integer formulaIDObj = new Integer(formula.
                    getID());
58              Integer normIDObj = (Integer)
                    formulaIDToNormID.get(formulaIDObj);
59              int normID;
60              if (normIDObj == null) {
61                  normID = getNormalizedID(formula);
62                  normIDObj = new Integer(normID);
63                  formulaIDToNormID.put(formulaIDObj,
                        normIDObj);
64              }
65              else {
66                  normID = normIDObj.intValue();
67              }
```

```
68              IndexedFormula iF = new IndexedFormula(normID
                    , i, m, j);
69              Literal lit = (Literal) normIDToLiteral.get(
                    iF);
70              if (lit == null) {
71                  lit = new Literal(varNoGenerator.
                        getNextLiteralNo(), true);
72                  normIDToLiteral.put(iF, lit);
73                  if (Main.debugLiterals) {
74                      System.out.println("lit=["+lit+"] ->
                            Formula=["+formula+"] iF=["+iF
                            +"]");
75                  }
76              }
77              if (sign) {
78                  return lit;
79              }
80              else {
81                  return lit.flipSign();
82              }
83          }
84      }
85
86
87      protected abstract int getNormalizedID(Formula
            formula)
88      throws CloneNotSupportedException;
89
90      public boolean hasDef(Formula formula, boolean
            polarity)
91      throws CloneNotSupportedException {
92          boolean sign = polarity;
93          while (formula.getType() == Formula.TYPE_NEGATION
                ) {
94              sign = !sign;
95              formula = ((Negation) formula).getOperand();
96          }
97          Integer formulaIDObj = new Integer(formula.getID
                ());
98          Integer normIDObj = (Integer) formulaIDToNormID.
                get(formulaIDObj);
99          int normID;
100         if (normIDObj == null) {
101             normID = getNormalizedID(formula);
102             normIDObj = new Integer(normID);
103             formulaIDToNormID.put(formulaIDObj, normIDObj
                    );
104         }
105         else {
106             normID = normIDObj.intValue();
```

```
107             }
108             if (sign) {
109                 return normIDs.contains(normIDObj);
110             }
111             else {
112                 return normIDs.contains(new Integer(-normID))
                        ;
113             }
114         }
115 
116         public boolean hasDef(State formula)
117         throws CloneNotSupportedException {
118             Integer formulaIDObj = new Integer(formula.getID
                    ());
119             Integer normIDObj = (Integer) formulaIDToNormID.
                    get(formulaIDObj);
120             int normID;
121             if (normIDObj == null) {
122                 normID = getNormalizedID(formula);
123                 normIDObj = new Integer(normID);
124                 formulaIDToNormID.put(formulaIDObj, normIDObj
                        );
125             }
126             else {
127                 normID = normIDObj.intValue();
128             }
129             if (normIDs.contains(normIDObj)) {
130                 return true;
131             }
132             else {
133                 normIDObj = new Integer(-normID);
134                 return normIDs.contains(normIDObj);
135             }
136         }
137 
138         public void addDef(Formula formula, boolean polarity)
139         throws CloneNotSupportedException {
140             boolean sign = polarity;
141             while (formula.getType() == Formula.TYPE_NEGATION
                    ) {
142                 sign = !sign;
143                 formula = ((Negation) formula).getOperand();
144             }
145             Integer formulaIDObj = new Integer(formula.getID
                    ());
146             Integer normIDObj = (Integer) formulaIDToNormID.
                    get(formulaIDObj);
147             int normID;
148             if (normIDObj == null) {
149                 normID = getNormalizedID(formula);
```

```
150                 normIDObj = new Integer(normID);
151                 formulaIDToNormID.put(formulaIDObj, normIDObj
                        );
152             }
153             else {
154                 normID = normIDObj.intValue();
155             }
156             if (!sign) {
157                 normIDObj = new Integer(-normID);
158             }
159             normIDs.add(normIDObj);
160         }
161 
162 }
```

I.3.5 SemanticLookupLiteralHandler.java

```
1  package bmc.literal;
2  
3  import java.util.*;
4  import bmc.*;
5  import bmc.formula.*;
6  import bmc.util.*;
7  
8  public class SemanticLookupLiteralHandler extends
       LiteralHandler {
9      protected Map strRepToNormID;
10     protected CNFTranslator cnfTrans;
11 
12     public SemanticLookupLiteralHandler() {
13         strRepToNormID = new HashMap();
14         cnfTrans = new CNFTranslator();
15     }
16 
17     protected String getNormalizedRep(Formula formula)
18     throws CloneNotSupportedException {
19         Formula cnfFormula = cnfTrans.translate((Formula)
               formula.clone());
20         SortedSet reps = new TreeSet();
21         getNormalizedRep(cnfFormula, reps);
22         Iterator it = reps.iterator();
23         StringBuffer buffer = new StringBuffer();
24         while (it.hasNext()) {
25             buffer.append(it.next());
26             if (it.hasNext()) {
27                 buffer.append(" & ");
28             }
29         }
30         return buffer.toString();
31 
```

```
32      }
33
34      protected void getNormalizedRep(Formula formula,
            SortedSet reps)
35      throws CloneNotSupportedException {
36          if (formula.getType() == Formula.TYPE_CONJUNCTION
                ) {
37              Conjunction con = (Conjunction) formula;
38              getNormalizedRep(con.getLeftOperand(), reps);
39              getNormalizedRep(con.getRightOperand(), reps)
                    ;
40          }
41          else {
42              SortedSet clauseReps = new TreeSet();
43              getNormalizedClauseRep(formula, clauseReps);
44              Iterator it = clauseReps.iterator();
45              StringBuffer buffer = new StringBuffer();
46              while (it.hasNext()) {
47                  buffer.append(it.next());
48                  if (it.hasNext()) {
49                      buffer.append(" | ");
50                  }
51              }
52              reps.add(buffer.toString());
53          }
54      }
55
56      protected void getNormalizedClauseRep(Formula formula
            , SortedSet reps)
57      throws CloneNotSupportedException {
58          switch (formula.getType()) {
59              case Formula.TYPE_CHOP: {
60                  Chop f1 = (Chop) formula;
61                  reps.add("("+getNormalizedRep(f1.
                        getLeftOperand())+" ; "+
62                  getNormalizedRep(f1.getRightOperand())+")
                        ");
63                  break;
64              }
65              case Formula.TYPE_DISJUNCTION: {
66                  Disjunction f1 = (Disjunction) formula;
67                  getNormalizedClauseRep(f1.getLeftOperand
                        (), reps);
68                  getNormalizedClauseRep(f1.getRightOperand
                        (), reps);
69                  break;
70              }
71              case Formula.TYPE_DURATION:{
72                  Duration f1 = (Duration) formula;
```

```
73                  reps.add("dur("+getNormalizedRep(f1.
                        getStateAssertion())+") >="+f1.
                        getBound());
74                  break;
75              }
76              case Formula.TYPE_FALSE:{
77                  reps.add(False.FALSE_STR);
78                  break;
79              }
80              case Formula.TYPE_NEGATION: {
81                  reps.add("~"+getNormalizedRep(((Negation)
                        formula).getOperand()));
82                  break;
83              }
84              case Formula.TYPE_STATE:{
85                  reps.add(((State) formula).getName());
86                  break;
87              }
88              case Formula.TYPE_TRUE:{
89                  reps.add(True.TRUE_STR);
90                  break;
91              }
92              default:
93                  throw new IllegalStateException("
                        Unexpected formula type");
94          }
95
96      }
97
98      public int getNormalizedID(Formula formula)
99      throws CloneNotSupportedException {
100         String strRep = getNormalizedRep((Formula)
                formula.clone());
101         Integer normIDObj = (Integer) strRepToNormID.get(
                strRep);
102         int normID;
103         if (normIDObj == null) {
104             normID = formula.getID();
105             strRepToNormID.put(strRep, new Integer(normID
                    ));
106         }
107         else {
108             normID = normIDObj.intValue();
109         }
110         return normID;
111     }
112
113 }
```

I.3.6 SyntacticLiteralHandler.java

```
1  package bmc.literal;
2
3  import java.util.*;
4  import bmc.*;
5  import bmc.formula.*;
6  import bmc.util.*;
7
8  public class SyntacticLookupLiteralHandler extends
       LiteralHandler {
9      protected Map strRepToNormID;
10
11     public SyntacticLookupLiteralHandler() {
12         strRepToNormID = new HashMap();
13     }
14
15     protected int getNormalizedID(Formula formula) {
16         String strRep = formula.toString();
17         Integer normIDObj = (Integer) strRepToNormID.get(
               strRep);
18         int normID;
19         if (normIDObj == null) {
20             normID = formula.getID();
21             strRepToNormID.put(strRep, new Integer(normID
                   ));
22         }
23         else {
24             normID = normIDObj.intValue();
25         }
26         return normID;
27     }
28
29 }
```

I.3.7 VariableNoGenerator.java

```
1  package bmc.literal;
2
3  public class VariableNoGenerator {
4
5      public static final int BOOL_LIT_NO = 1;
6
7      protected int nextLiteralNo;
8
9      public VariableNoGenerator() {
10         nextLiteralNo = BOOL_LIT_NO+1;
11     }
12
13     public int getNextLiteralNo() {
```

```
14          return nextLiteralNo++;
15      }
16
17      public int getNumVariables() {
18          return nextLiteralNo - 1;
19      }
20  }
```

I.4 Package bmc.parser

I.4.1 BMCPreprocessor.java

```
1   package bmc.parser;
2
3   import java_cup.runtime.*;
4   import java.util.*;
5   import java.text.ParseException;
6   import java.io.IOException;
7
8   public class BMCPreprocessor {
9       private BMCLex scanner;
10      private StringBuffer buffer;
11      private Map defs;
12      private LinkedList stack;
13
14      public static final int STATE_DEFAULT = 1;
15      public static final int STATE_LINE_START = 2;
16      public static final int STATE_LINE_ID = 3;
17      public static final int STATE_IN_PARAMS = 4;
18      public static final int STATE_IN_DEF = 5;
19
20      public BMCPreprocessor() {
21          this.buffer = new StringBuffer();
22          this.stack = new LinkedList();
23          this.defs = new HashMap();
24      }
25
26      public void setScanner(BMCLex scanner) {
27          this.scanner = scanner;
28      }
29
30      private Token nextToken() throws IOException,
            ParseException {
31          if (stack.size() > 0) {
32              return (Token) stack.removeFirst();
33          }
34          else {
35              return new Token(scanner.next_token(),
                    scanner.getTokenStr());
36          }
```

```
37      }
38
39      private void pushToken(Token token) {
40          stack.addFirst(token);
41      }
42
43      private void pushTokens(List tokens) {
44          for (int i=tokens.size()-1; i>=0; i--) {
45              pushToken((Token) tokens.get(i));
46          }
47      }
48
49      private boolean checkReplace(boolean pushDef) throws
            IOException, ParseException {
50          Token defName = nextToken();
51          if (defName.getSymbol() != BMCSymbols.ID) {
52              // No macro name, so nothing to replace
53              pushToken(defName);
54              return false;
55          }
56          List def = getDef(defName.getStr());
57          if (def == null) {
58              // No macro by that name
59              pushToken(defName);
60              return false;
61          }
62
63          // A macro by that name exists, parse parameters
64          List params = parseParams();
65          if (params == null) {
66              // No (valid) params
67              params = new LinkedList();
68          }
69
70          List defParams = getParams(defName.getStr());
71          if (defParams.size() != params.size()) {
72              throw new ParseException("Macro \""+defName.
                    getStr()+
73              "\" takes "+defParams.size()+" parameters",
                    scanner.getLineNo());
74          }
75          applyMDef(defName.getStr(), params, pushDef);
76          return true;
77      }
78
79      private List parseParams() throws IOException,
            ParseException {
80          Token token = nextToken();
81          if (token.getSymbol() != BMCSymbols.LPAREN) {
82              pushToken(token);
```

```
83              return null;
84          }
85          LinkedList params = new LinkedList();
86          LinkedList tmp = new LinkedList();
87          List param = new LinkedList();
88          loop: while ((token = nextToken()).getSymbol() !=
                BMCSymbols.EOF) {
89              switch (token.getSymbol()) {
90                  case BMCSymbols.ID:
91                      pushToken(token);
92                      if (checkReplace(false)) {
93                          // Token(s) replaced by macro,
                               try again
94                          continue;
95                      }
96                      else {
97                          // Token not replaced
98                          // pop token again
99                          nextToken();
100                         param.add(token);
101                     }
102                     break;
103                 case BMCSymbols.COMMA:
104                     // Done with this parameter
105                     if (param.size() == 0) {
106                         throw new ParseException("
                               Argument list malformed",
107                         scanner.getLineNo());
108                     }
109                     params.add(param);
110                     param = new LinkedList();
111                     break;
112                 case BMCSymbols.RPAREN:
113                     // Done with all parameters
114                     if (param.size() > 0) {
115                         params.add(param);
116                     }
117                     return params;
118                 case BMCSymbols.DOT:
119                 case BMCSymbols.LINE:
120                     // Recall token for abort
121                     tmp.add(token);
122                     // Abort, no RPAREN found
123                     break loop;
124                 default:
125                     // Add to parameter
126                     param.add(token);
127             }
128             // Recall token in case we need to abort
129             tmp.add(token);
```

```
130             }
131             // No RPAREN found, abort
132             pushTokens(tmp);
133             return null;
134         }
135 
136         private void appendToken(Token token) {
137             buffer.append(token.getStr());
138             if (token.getSymbol() == BMCSymbols.DOT) {
139                 buffer.append("\n");
140             }
141             else {
142                 buffer.append(" ");
143             }
144         }
145 
146         private void appendTokens(List tokens) {
147             Iterator it = tokens.iterator();
148             while (it.hasNext()) {
149                 Token token = (Token) it.next();
150                 appendToken(token);
151             }
152         }
153 
154         public String preprocess() throws IOException,
                ParseException {
155             buffer.setLength(0);
156             Token token;
157             List tmp = new LinkedList();
158             String macroName = null;
159             List params = null;
160             List defBody = null;
161             int state = STATE_DEFAULT;
162             while ((token = nextToken()) != null && token.
                    getSymbol() != BMCSymbols.EOF) {
163                 switch (state) {
164                     case STATE_DEFAULT:
165                         switch (token.getSymbol()) {
166                             case BMCSymbols.LINE:
167                                 state = STATE_LINE_START;
168                                 tmp.clear();
169                                 tmp.add(token);
170                                 break;
171                             case BMCSymbols.ID:
172                                 pushToken(token);
173                                 if (checkReplace(false)) {
174                                     continue;
175                                 }
176                                 else {
177                                     nextToken();
```

```
178                             appendToken(token);
179                         }
180                         break;
181                     default:
182                         appendToken(token);
183                         break;
184                 }
185                 break;
186             case STATE_LINE_START:
187                 switch (token.getSymbol()) {
188                     case BMCSymbols.ID:
189                         pushToken(token);
190                         if (checkReplace(false)) {
191                             continue;
192                         }
193                         else {
194                             nextToken();
195                             state = STATE_LINE_ID;
196                             macroName = token.getStr
                                    ();
197                         }
198                         break;
199                     default:
200                         tmp.add(token);
201                         appendTokens(tmp);
202                         state = STATE_DEFAULT;
203                 }
204                 break;
205             case STATE_LINE_ID:
206                 switch (token.getSymbol()) {
207                     case BMCSymbols.LPAREN:
208                         params = new LinkedList();
209                         state = STATE_IN_PARAMS;
210                         break;
211                     case BMCSymbols.DEF:
212                         state = STATE_IN_DEF;
213                         params = new LinkedList();
214                         defBody = new LinkedList();
215                         break;
216                     default:
217                         throw new ParseException("
                                Malformed definition of
                                macro \""+
218                         macroName + "\"",scanner.
                                getLineNo());
219                 }
220                 break;
221             case STATE_IN_PARAMS:
222                 switch (token.getSymbol()) {
223                     case BMCSymbols.RPAREN: {
```

```
224                         Token token2 = nextToken();
225                         if (token2.getSymbol() !=
                                BMCSymbols.DEF) {
226                             throw new ParseException
                                    ("Malformed definition
                                     of macro \""+
227                             macroName + "\"",scanner.
                                    getLineNo());
228                         }
229                         state = STATE_IN_DEF;
230                         defBody = new LinkedList();
231                         break;
232                     }
233                     case BMCSymbols.ID: {
234                         if (getDef(token.getStr()) !=
                                 null) {
235                             throw new ParseException
                                    ("Macros not allowed
                                    in definition of
                                    formal parameters",
236                             scanner.getLineNo());
237                         }
238                         params.add(token.getStr());
239                         break;
240                     }
241                     case BMCSymbols.COMMA: {
242                         if (params.size() == 0) {
243                             throw new ParseException
                                    ("Malformed definition
                                     of macro \""+
244                             macroName + "\"",scanner.
                                    getLineNo());
245                         }
246                         // Peek at next symbol
247                         Token token2 = nextToken();
248                         if (token2.getSymbol() !=
                                BMCSymbols.ID) {
249                             throw new ParseException
                                    ("Malformed definition
                                     of macro \""+
250                             macroName + "\"",scanner.
                                    getLineNo());
251                         }
252                         pushToken(token2);
253                         break;
254                     }
255                     default:
256                         throw new ParseException("
                                Malformed definition of
                                macro \""+
```

```
257                                 macroName + "\"",scanner.
                                        getLineNo());
258                         }
259                         break;
260                     case STATE_IN_DEF:
261                         switch (token.getSymbol()) {
262                             case BMCSymbols.DOT:
263                                 addDef(macroName, defBody,
                                        params);
264                                 state = STATE_DEFAULT;
265                                 break;
266                             case BMCSymbols.ID:
267                                 if (macroName.equals(token.
                                        getStr())) {
268                                     throw new ParseException
                                            ("Recursive macro
                                            definition not allowed
                                            ", scanner.getLineNo()
                                            );
269                                 }
270                                 pushToken(token);
271                                 if (!checkReplace(true)) {
272                                     nextToken();
273                                     defBody.add(token);
274                                 }
275                                 break;
276                             default:
277                                 defBody.add(token);
278                         }
279                         break;
280                     default:
281                         throw new IllegalStateException("
                                Unknown parse state");
282                 }
283             }
284             return buffer.toString();
285         }
286 
287 
288         private void applyMDef(String id, List vl, boolean
                pushDef)
289         throws ParseException  {
290             List result = new LinkedList();
291             List macro = getDef(id);
292             if (macro == null) {
293                 throw new ParseException(
294                 "Macro \""+id+"\" undefined",
295                 scanner.getLineNo());
296             }
297             List params = getParams(id);
```

```
298             if (params == null) {
299                 throw new IllegalStateException("No params");
300             }
301             if (params.size() != vl.size()) {
302                 throw new ParseException(
303                 "Macro \""+id+"\" takes "+params.size()+"
                        parameters",
304                 scanner.getLineNo());
305             }
306             Map m = new HashMap();
307             for(int i=0; i<vl.size(); i++) {
308                 m.put(params.get(i),vl.get(i));
309             }
310             Iterator it = macro.iterator();
311             while (it.hasNext()) {
312                 Token elem = (Token) it.next();
313                 if (elem.getSymbol() == BMCSymbols.ID) {
314                     List replace = (List) m.get(elem.getStr()
                            );
315                     if (replace != null) {
316                         result.addAll(replace);
317                     }
318                     else {
319                         result.add(elem);
320                     }
321                 }
322                 else {
323                     result.add(elem);
324                 }
325             }
326             if (pushDef) {
327                 pushTokens(result);
328             }
329             else {
330                 appendTokens(result);
331             }
332         }
333 
334         private void addDef(String fname, List text, List
                paramList)
335         throws ParseException {
336             if (defs.get(fname) != null) {
337                 throw new ParseException("Redefinition of
                        macro \""+fname+
338                 "\" not allowed", scanner.getLineNo());
339             }
340             defs.put(fname, text);
341             if (paramList == null) {
342                 paramList = new LinkedList();
343             }
```

```
344             defs.put(fname+".args", paramList);
345         }
346 
347         private List getDef(String fname) {
348             return (List) defs.get(fname);
349         }
350 
351         private List getParams(String fname) {
352             return (List) defs.get(fname+".args");
353         }
354 
355         public class Token {
356             private Symbol symbol;
357             String str;
358             public Token(Symbol symbol, String str) {
359                 this.symbol = symbol;
360                 this.str = str;
361             }
362 
363             public int getSymbol() {
364                 return symbol.sym;
365             }
366 
367             public String getStr() {
368                 return str;
369             }
370 
371             public String toString() {
372                 return symbol+ ": "+str;
373             }
374         }
375 }
```

I.5 Package `bmc.robdd`

I.5.1 `NodeDesc.java`

```
1  package bmc.robdd;
2 
3  public class NodeDesc {
4 
5      protected String varID;
6      protected int low, high;
7 
8      public NodeDesc(String varID, int low, int high) {
9          this.varID = varID;
10         this.low = low;
11         this.high = high;
12     }
13 
```

```
14      public String getVarID() {
15          return varID;
16      }
17
18      public int getLow() {
19          return low;
20      }
21
22      public int getHigh() {
23          return high;
24      }
25
26      public boolean equals(Object o) {
27          if (!(o instanceof NodeDesc)) {
28              return false;
29          }
30          NodeDesc d = (NodeDesc) o;
31          return varID.equals(d.varID) && (low == d.low) &&
                (high == d.high);
32      }
33
34      public String toString() {
35          return "v="+varID+" l="+low+" h="+high;
36      }
37
38      public int hashCode() {
39          return varID.hashCode() ^ (low << 16) ^ high;
40      }
41  }
```

I.5.2 NodePair.java

```
1   package bmc.robdd;
2
3   public class NodePair {
4       protected int u1, u2;
5
6       public NodePair(int u1, int u2) {
7           this.u1 = u1;
8           this.u2 = u2;
9       }
10
11      public int getU1() {
12          return u1;
13      }
14
15      public int getU2() {
16          return u2;
17      }
18
```

```
19      public boolean equals(Object o) {
20          if (!(o instanceof NodePair)) {
21              return false;
22          }
23          NodePair np = (NodePair) o;
24          return u1 == np.u1 && u2 == np.u2;
25      }
26
27      public int hashCode() {
28          return u1 ^ (u2 << 16);
29      }
30  }
```

I.5.3 ROBDD.java

```
1   package bmc.robdd;
2
3   import java.util.*;
4
5   public class ROBDD implements Cloneable {
6       ArrayList tableT;
7       int root;
8
9       public ROBDD(ArrayList tableT, int root) {
10          this.tableT = tableT;
11          this.root = root;
12      }
13
14      public ArrayList getTableT() {
15          return tableT;
16      }
17
18      public int getRoot() {
19          return root;
20      }
21
22      public boolean isTrue() {
23          return root == 1;
24      }
25
26      public boolean isFalse() {
27          return root == 0;
28      }
29
30      public ROBDD negate() {
31          ArrayList nTableT = (ArrayList) tableT.clone();
32          ListIterator it = nTableT.listIterator();
33          while (it.hasNext()) {
34              NodeDesc desc = (NodeDesc) it.next();
35              String varID = desc.getVarID();
```

```
36              int low = desc.getLow();
37              int high = desc.getHigh();
38              if (low == 0) {
39                  low = 1;
40              }
41              else if (low == 1) {
42                  low = 0;
43              }
44              if (high == 0) {
45                  high = 1;
46              }
47              else if (high == 1) {
48                  high = 0;
49              }
50              it.remove();
51              it.add(new NodeDesc(varID, low, high));
52          }
53          int nRoot = root;
54          if (root == 1) {
55              nRoot = 0;
56          }
57          else if (root == 0) {
58              nRoot = 1;
59          }
60          return new ROBDD(nTableT, nRoot);
61      }
62
63      public boolean equals(Object o) {
64          if (!(o instanceof ROBDD)) {
65              return false;
66          }
67          ROBDD r = (ROBDD) o;
68          if (root != r.root) {
69              return false;
70          }
71          return tableT.equals(r.tableT);
72      }
73
74      public String toString() {
75          return "root: "+root+" tableT: "+tableT.toString
                ();
76      }
77
78  }
```

I.5.4 ROBDDUtil.java

```
1   package bmc.robdd;
2
3   import java.util.*;
```

```
4
5  import bmc.formula.*;
6  import bmc.*;
7
8  public class ROBDDUtil {
9      protected ArrayList tableT;
10     protected Map tableH, tableG;
11     protected int mode;
12
13     public static final int OP_IMPLICATION = 1;
14     public static final int OP_CONJUNCTION = 2;
15     public static final int OP_DISJUNCTION = 3;
16
17     public ROBDDUtil() {
18         tableH = new HashMap();
19         tableG = new HashMap();
20     }
21
22     public ROBDD makeROBDD(Formula formula)
23     throws CloneNotSupportedException {
24         init();
25         List states = new LinkedList();
26         formula.getStates(states);
27         Collections.sort(states);
28         int root = build(formula, states, 0);
29         return new ROBDD(tableT, root);
30     }
31
32     protected void init() {
33         tableT = new ArrayList();
34         tableT.add(new NodeDesc(False.FALSE_STR,-1,-1));
35         tableT.add(new NodeDesc(True.TRUE_STR,-1,-1));
36         tableH.clear();
37         tableG.clear();
38     }
39
40     public ROBDD apply(ROBDD left, ROBDD right, int
           operator) {
41         init();
42         int root = apply(left.getTableT(), left.getRoot()
               ,
43         right.getTableT(), right.getRoot(), operator);
44         return new ROBDD(tableT, root);
45     }
46
47     private int apply(List tableT1, int u1, List tableT2,
           int u2, int operator) {
48         NodePair np = new NodePair(u1,u2);
49         Integer result = (Integer) tableG.get(np);
50         if (result != null) {
```

```
51              return result.intValue();
52          }
53          int intResult = -1;
54          NodeDesc desc1 = (NodeDesc) tableT1.get(u1);
55          String var1 = desc1.getVarID();
56          NodeDesc desc2 = (NodeDesc) tableT2.get(u2);
57          String var2 = desc2.getVarID();
58          if (u1 <= 1) {
59              if (u2 <= 1) {
60                  // Both left and right are constants
61                  switch (operator) {
62                      case OP_IMPLICATION:
63                          intResult = u1==1? u2 : 1;
64                          break;
65                      case OP_CONJUNCTION:
66                          intResult = u1 & u2;
67                          break;
68                      case OP_DISJUNCTION:
69                          intResult = u1 | u2;
70                          break;
71                      default:
72                          throw new
                                IllegalArgumentException("
                                Unsupported operator");
73                  }
74              }
75              else {
76                  // Left is constant, while right is not
77                  intResult = mk(var2, apply(tableT1,u1,
                        tableT2,
78                  desc2.getLow(),operator), apply(tableT1,
                        u1,tableT2,
79                  desc2.getHigh(),operator));
80              }
81          }
82          else if (u2 <= 1) {
83              // Right is constant, while left is not
84              intResult = mk(var1, apply(tableT1,desc1.
                    getLow(),
85              tableT2,u2,operator), apply(tableT1,desc1.
                    getHigh(),tableT2,
86              u2,operator));
87          }
88          else {
89              // Neither left nor right are constants
90              int comp = var1.compareTo(var2);
91              if (comp == 0) {
92                  intResult = mk(var1, apply(tableT1,desc1.
                        getLow(),
```

```
93                  tableT2,desc2.getLow(),operator), apply(
                        tableT1,desc1.getHigh(),tableT2,
94                  desc2.getHigh(),operator));
95              }
96              else if (comp < 0) {
97                  intResult = mk(var1, apply(tableT1,desc1.
                        getLow(),
98                  tableT2,u2,operator), apply(tableT1,desc1
                        .getHigh(),tableT2,
99                  u2,operator));
100             }
101             else {
102                 intResult = mk(var2, apply(tableT1,u1,
                        tableT2,
103                 desc2.getLow(),operator), apply(tableT1,
                        u1,tableT2,
104                 desc2.getHigh(),operator));
105             }
106         }
107         result = new Integer(intResult);
108         tableG.put(np, result);
109         return intResult;
110     }
111
112     protected int mk(String state, int l, int h) {
113         if (l == h) {
114             return l;
115         }
116         else {
117             NodeDesc n = new NodeDesc(state, l, h);
118             Integer u = (Integer) tableH.get(n);
119             if (u == null) {
120                 u = new Integer(tableT.size());
121                 tableT.add(n);
122                 tableH.put(n, u);
123             }
124             return u.intValue();
125         }
126     }
127
128     protected int build(Formula formula, List states, int
            index) throws CloneNotSupportedException {
129         if (index >= states.size()) {
130             if (states.size() == 0) {
131                 // Replace has never been called. Invoke
                        it once to ensure that
132                 // the formula collapses to a single True
                         or False.
133                 formula = replace((Formula) formula.clone
                        (), null, true);
```

```
134             }
135             switch (formula.getType()) {
136                 case Formula.TYPE_FALSE:
137                     return 0;
138                 case Formula.TYPE_TRUE:
139                     return 1;
140                 default:
141                     throw new IllegalStateException("
                           Unexpected formula type");
142             }
143         }
144         else {
145             String state = ((State) states.get(index)).
                getName();
146             int v0 = build(replace((Formula) formula.
                clone(), state, false),
147             states, index+1);
148             int v1 = build(replace((Formula) formula.
                clone(), state, true),
149             states, index+1);
150             return mk(state, v0, v1);
151         }
152     }
153
154
155     protected Formula replace(Formula formula, String
            varID, boolean value) {
156         switch (formula.getType()) {
157             case Formula.TYPE_STATE: {
158                 State state = (State) formula;
159                 if (state.getName().equals(varID)) {
160                     if (value) {
161                         return new True();
162                     }
163                     else {
164                         return new False();
165                     }
166                 }
167                 else {
168                     return formula;
169                 }
170             }
171             case Formula.TYPE_FALSE:
172             case Formula.TYPE_TRUE:
173                 return formula;
174             case Formula.TYPE_NEGATION: {
175                 Negation neg = (Negation) formula;
176                 Formula op = replace(neg.getOperand(),
                        varID, value);
177                 switch (op.getType()) {
```

```
178                     case Formula.TYPE_FALSE:
179                         return new True();
180                     case Formula.TYPE_TRUE:
181                         return new False();
182                     default:
183                         neg.setOperand(op);
184                         return neg;
185                 }
186             }
187             case Formula.TYPE_CONJUNCTION: {
188                 Connective con = (Connective) formula;
189                 Formula left = replace(con.getLeftOperand
                        (), varID, value);
190                 Formula right = replace(con.
                        getRightOperand(), varID, value);
191                 if (left.getType() == Formula.TYPE_FALSE
                        ||
192                 right.getType() == Formula.TYPE_FALSE) {
193                     return new False();
194                 }
195                 else if (left.getType() == Formula.
                        TYPE_TRUE) {
196                     if (right.getType() == Formula.
                            TYPE_TRUE) {
197                         return new True();
198                     }
199                     else {
200                         return right;
201                     }
202                 }
203                 else if (right.getType() == Formula.
                        TYPE_TRUE) {
204                     return left;
205                 }
206                 else {
207                     con.setLeftOperand(left);
208                     con.setRightOperand(right);
209                     return con;
210                 }
211             }
212             case Formula.TYPE_DISJUNCTION: {
213                 Connective con = (Connective) formula;
214                 Formula left = replace(con.getLeftOperand
                        (), varID, value);
215                 Formula right = replace(con.
                        getRightOperand(), varID, value);
216                 if (left.getType() == Formula.TYPE_TRUE
                        ||
217                 right.getType() == Formula.TYPE_TRUE) {
218                     return new True();
```

```
219                 }
220                 else if (left.getType() == Formula.
                        TYPE_FALSE) {
221                     if (right.getType() == Formula.
                            TYPE_FALSE) {
222                         return new False();
223                     }
224                     else {
225                         return right;
226                     }
227                 }
228                 else if (right.getType() == Formula.
                        TYPE_FALSE) {
229                     return left;
230                 }
231                 else {
232                     con.setLeftOperand(left);
233                     con.setRightOperand(right);
234                     return con;
235                 }
236             }
237             default:
238                 throw new IllegalStateException("
                        Unexpected formula type");
239         }
240     }
241 }
```

I.6 Package `bmc.simp`

I.6.1 `DCSimplifier.java`

```
1  package bmc.simp;
2
3  import java.util.*;
4
5  import bmc.util.*;
6  import bmc.robdd.*;
7  import bmc.formula.*;
8
9  public class DCSimplifier {
10
11     protected int level;
12     protected int k;
13     protected double simpTime;
14
15     public static final int FORMULA_TYPE_DC = 1;
16     public static final int FORMULA_TYPE_SA = 2;
17
18     public DCSimplifier(Settings settings) {
```

```
19          this.level = settings.getDCSimpLevel();
20          this.k = settings.getK();
21      }
22
23      public Formula simplifyDC(Formula formula)
24      throws Exception {
25          if (level == Settings.DC_SIMP_LEVEL_NONE) {
26              this.simpTime = 0;
27              return formula;
28          }
29          formula = simplifyTree(formula, Formula.
                TYPE_UNKNOWN, FORMULA_TYPE_DC);
30          return formula;
31      }
32
33      protected Formula simplifySA(Formula formula)
34      throws Exception {
35          if (level == Settings.DC_SIMP_LEVEL_NONE) {
36              return formula;
37          }
38          formula = simplifyTree(formula, Formula.
                TYPE_UNKNOWN, FORMULA_TYPE_SA);
39          return formula;
40      }
41
42      protected Formula simplifyTree(Formula formula, int
            parentType, int formulaType )
43      throws Exception {
44          switch (formula.getType()) {
45              case Formula.TYPE_DURATION: {
46                  if (formulaType == FORMULA_TYPE_SA) {
47                      throw new IllegalArgumentException("
                            Formula is not an SA formula");
48                  }
49                  Duration dur = (Duration) formula;
50                  Formula sa = dur.getStateAssertion();
51                  sa = simplifySA(sa);
52                  if (sa != dur.getStateAssertion()) {
53                      dur.setStateAssertion(sa);
54                  }
55                  formula = applyLevel1Simp(formula,
                        formula.getType());
56                  return formula;
57              }
58              case Formula.TYPE_STATE:
59                  if (formulaType == FORMULA_TYPE_DC) {
60                      throw new IllegalArgumentException("
                            Formula is not a DC formula");
61                  }
62                  return formula;
```

```
63              case Formula.TYPE_CHOP: {
64                  if (formulaType == FORMULA_TYPE_SA) {
65                      throw new IllegalArgumentException("
                            Formula is not an SA formula");
66                  }
67                  Connective conn = (Connective) formula;
68                  Formula left = simplifyTree(conn.
                        getLeftOperand(), conn.getType(),
                        formulaType);
69                  if (left != conn.getLeftOperand()) {
70                      conn.setLeftOperand(left);
71                  }
72                  Formula right = simplifyTree(conn.
                        getRightOperand(), conn.getType(),
                        formulaType);
73                  if (right != conn.getRightOperand()) {
74                      conn.setRightOperand(right);
75                  }
76                  formula = applyLevel1Simp(formula, conn.
                        getType());
77                  return formula;
78              }
79              case Formula.TYPE_CONJUNCTION:
80              case Formula.TYPE_DISJUNCTION: {
81                  Connective conn = (Connective) formula;
82                  Formula left = simplifyTree(conn.
                        getLeftOperand(), conn.getType(),
                        formulaType);
83                  if (left != conn.getLeftOperand()) {
84                      conn.setLeftOperand(left);
85                  }
86                  Formula right = simplifyTree(conn.
                        getRightOperand(), conn.getType(),
                        formulaType);
87                  if (right != conn.getRightOperand()) {
88                      conn.setRightOperand(right);
89                  }
90                  if (parentType != conn.getType()) {
91                      if (level >= Settings.
                            DC_SIMP_LEVEL_WITH_BDD) {
92                          if (formulaType ==
                                FORMULA_TYPE_DC) {
93                              formula = applyDCLevel2Simp(
                                    formula);
94                          }
95                          else { // SA
96                              formula = applySALevel2Simp(
                                    formula);
97                          }
98                      }
```

```
99                      formula = applyLevel1Simp(formula,
                            conn.getType());
100                 }
101                 return formula;
102             }
103             case Formula.TYPE_NEGATION: {
104                 Negation neg = (Negation) formula;
105                 Formula op = simplifyTree(neg.getOperand
                        (), neg.getType(), formulaType);
106                 if (neg.getOperand() != op) {
107                     neg.setOperand(op);
108                 }
109                 formula = applyLevel1Simp(formula, neg.
                        getType());
110                 return formula;
111             }
112             case Formula.TYPE_TRUE:
113                 return formula;
114             case Formula.TYPE_FALSE:
115                 return formula;
116             default:
117                 throw new IllegalArgumentException("
                        Unknown formula type: "+formula.
                        getType());
118         }
119     }
120
121     protected Formula applyLevel1Simp(Formula formula,
            int type) {
122         if (formula.getType() != type) {
123             return formula;
124         }
125         switch (formula.getType()) {
126             case Formula.TYPE_DURATION: {
127                 Duration dur = (Duration) formula;
128                 if (dur.getStateAssertion().getType() ==
                        Formula.TYPE_FALSE) {
129                     return new False();
130                 }
131                 else if (dur.getBound() > k) {
132                     return new False();
133                 }
134                 return formula;
135             }
136             case Formula.TYPE_CONJUNCTION: {
137                 Conjunction con = (Conjunction) formula;
138                 Formula left = con.getLeftOperand();
139                 Formula right = con.getRightOperand();
140                 left = applyLevel1Simp(left, type);
141                 right = applyLevel1Simp(right, type);
```

```
142                 if (left.getType() == Formula.TYPE_FALSE
                        ||
143                 right.getType() == Formula.TYPE_FALSE) {
144                     return new False();
145                 }
146                 else if (left.getType() == Formula.
                        TYPE_TRUE) {
147                     return right;
148                 }
149                 else if (right.getType() == Formula.
                        TYPE_TRUE) {
150                     return left;
151                 }
152                 else {
153                     if (left != con.getLeftOperand()) {
154                         con.setLeftOperand(left);
155                     }
156                     if (right != con.getRightOperand()) {
157                         con.setRightOperand(right);
158                     }
159                 }
160                 return formula;
161             }
162             case Formula.TYPE_DISJUNCTION: {
163                 Disjunction dis = (Disjunction) formula;
164                 Formula left = dis.getLeftOperand();
165                 Formula right = dis.getRightOperand();
166                 left = applyLevel1Simp(left, type);
167                 right = applyLevel1Simp(right, type);
168                 if (left.getType() == Formula.TYPE_TRUE
                        ||
169                 right.getType() == Formula.TYPE_TRUE) {
170                     return new True();
171                 }
172                 else if (left.getType() == Formula.
                        TYPE_FALSE) {
173                     return right;
174                 }
175                 else if (right.getType() == Formula.
                        TYPE_FALSE) {
176                     return left;
177                 }
178                 else {
179                     if (left != dis.getLeftOperand()) {
180                         dis.setLeftOperand(left);
181                     }
182                     if (right != dis.getRightOperand()) {
183                         dis.setRightOperand(right);
184                     }
185                 }
```

```
186                 return formula;
187             }
188             case Formula.TYPE_CHOP: {
189                 Chop chop = (Chop) formula;
190                 Formula left = chop.getLeftOperand();
191                 Formula right = chop.getRightOperand();
192                 left = applyLevel1Simp(left, type);
193                 right = applyLevel1Simp(right, type);
194                 if (left.getType() == Formula.TYPE_FALSE
                        ||
195                 right.getType() == Formula.TYPE_FALSE) {
196                     return new False();
197                 }
198                 else {
199                     if (left != chop.getLeftOperand()) {
200                         chop.setLeftOperand(left);
201                     }
202                     if (right != chop.getRightOperand())
                            {
203                         chop.setRightOperand(right);
204                     }
205                 }
206                 return formula;
207             }
208             case Formula.TYPE_NEGATION: {
209                 Negation neg = (Negation) formula;
210                 Formula op = neg.getOperand();
211                 if (op.getType() == Formula.TYPE_TRUE) {
212                     return new False();
213                 }
214                 else if (op.getType() == Formula.
                        TYPE_FALSE) {
215                     return new True();
216                 }
217                 else if (op.getType() == Formula.
                        TYPE_NEGATION) {
218                     return ((Negation) op).getOperand();
219                 }
220                 else {
221                     if (op != neg.getOperand()) {
222                         neg.setOperand(op);
223                     }
224                 }
225                 return formula;
226             }
227             case Formula.TYPE_TRUE:
228                 return formula;
229             case Formula.TYPE_FALSE:
230                 return formula;
231             default:
```

```
232                 throw new IllegalArgumentException("
                        Unknown formula type: "+formula.
                        getType());
233             }
234         }
235
236         protected Formula applyDCLevel2Simp(Formula formula)
                throws Exception {
237             LinkedList compareList = new LinkedList();
238             ROBDDUtil robddUtil = new ROBDDUtil();
239             populateDCCompareList(formula, compareList,
                    formula.getType());
240             switch (formula.getType()) {
241                 case Formula.TYPE_CONJUNCTION: {
242                     for (int i=0 ; i<compareList.size() ; i
                            ++) {
243                         for (int j=i+1 ; j<compareList.size()
                                ; j++) {
244                             Formula f1 = (Formula)
                                    compareList.get(i);
245                             Formula f2 = (Formula)
                                    compareList.get(j);
246                             if (f1.getType() == Formula.
                                    TYPE_NEGATION &&
247                             f2.getType() == Formula.
                                    TYPE_NEGATION) {
248                                 Duration dur1 = (Duration) ((
                                        Negation) f1).getOperand()
                                        ;
249                                 Duration dur2 = (Duration) ((
                                        Negation) f2).getOperand()
                                        ;
250                                 Formula sa1 = dur1.
                                        getStateAssertion();
251                                 Formula sa2 = dur2.
                                        getStateAssertion();
252                                 if (dur1.getBound() >= dur2.
                                        getBound()) {
253                                     ROBDD robdd = robddUtil.
                                            apply(sa1.getROBDD(
                                            robddUtil), sa2.
                                            getROBDD(robddUtil),
                                            ROBDDUtil.
                                            OP_IMPLICATION);
254                                     if (robdd.isTrue()) {
255                                         compareList.remove(i)
                                                ;
256                                         f1.updateParent(new
                                                True());
257                                         break;
```

```
258                         }
259                     }
260                     if (dur2.getBound() >= dur1.
                            getBound()) {
261                         ROBDD robdd = robddUtil.
                                apply(sa2.getROBDD(
                                robddUtil), sa1.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
262                         if (robdd.isTrue()) {
263                             compareList.remove(j)
                                    ;
264                             f2.updateParent(new
                                    True());
265                             continue;
266                         }
267                     }
268                 }
269                 else if (f1.getType() == Formula.
                        TYPE_NEGATION) {
270                     Duration dur1 = (Duration) ((
                            Negation) f1).getOperand()
                            ;
271                     Duration dur2 = (Duration) f2
                            ;
272                     Formula sa1 = dur1.
                            getStateAssertion();
273                     Formula sa2 = dur2.
                            getStateAssertion();
274                     if (dur2.getBound() >= dur1.
                            getBound()) {
275                         ROBDD robdd = robddUtil.
                                apply(sa2.getROBDD(
                                robddUtil), sa1.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
276                         if (robdd.isTrue()) {
277                             return new False();
278                         }
279                     }
280                 }
281                 else if (f2.getType() == Formula.
                        TYPE_NEGATION) {
282                     Duration dur1 = (Duration) f1
                            ;
283                     Duration dur2 = (Duration) ((
                            Negation) f2).getOperand()
                            ;
```

```
284                     Formula sa1 = dur1.
                            getStateAssertion();
285                     Formula sa2 = dur2.
                            getStateAssertion();
286                     if (dur1.getBound() >= dur2.
                            getBound()) {
287                         ROBDD robdd = robddUtil.
                                apply(sa1.getROBDD(
                                robddUtil), sa2.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
288                         if (robdd.isTrue()) {
289                             return new False();
290                         }
291                     }
292                 }
293                 else { // both durations
294                     Duration dur1 = (Duration) f1
                            ;
295                     Duration dur2 = (Duration) f2
                            ;
296                     Formula sa1 = dur1.
                            getStateAssertion();
297                     Formula sa2 = dur2.
                            getStateAssertion();
298                     if (dur1.getBound() >= dur2.
                            getBound()) {
299                         ROBDD robdd = robddUtil.
                                apply(sa1.getROBDD(
                                robddUtil), sa2.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
300                         if (robdd.isTrue()) {
301                             compareList.remove(j)
                                    ;
302                             f2.updateParent(new
                                    True());
303                             continue;
304                         }
305                     }
306                     if (dur2.getBound() >= dur1.
                            getBound()) {
307                         ROBDD robdd = robddUtil.
                                apply(sa2.getROBDD(
                                robddUtil), sa1.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
```

```
308                                 if (robdd.isTrue()) {
309                                     compareList.remove(i)
                                            ;
310                                     f1.updateParent(new
                                            True());
311                                     break;
312                                 }
313                             }
314                         }
315                     }
316                 }
317                 return formula;
318             }
319             case Formula.TYPE_DISJUNCTION: {
320                 for (int i=0 ; i<compareList.size() ; i
                        ++) {
321                     for (int j=i+1 ; j<compareList.size()
                            ; j++) {
322                         Formula f1 = (Formula)
                                compareList.get(i);
323                         Formula f2 = (Formula)
                                compareList.get(j);
324                         if (f1.getType() == Formula.
                                TYPE_NEGATION &&
325                         f2.getType() == Formula.
                                TYPE_NEGATION) {
326                             Duration dur1 = (Duration) ((
                                    Negation) f1).getOperand()
                                    ;
327                             Duration dur2 = (Duration) ((
                                    Negation) f2).getOperand()
                                    ;
328                             Formula sa1 = dur1.
                                    getStateAssertion();
329                             Formula sa2 = dur2.
                                    getStateAssertion();
330                             if (dur1.getBound() >= dur2.
                                    getBound()) {
331                                 ROBDD robdd = robddUtil.
                                        apply(sa1.getROBDD(
                                        robddUtil), sa2.
                                        getROBDD(robddUtil),
                                        ROBDDUtil.
                                        OP_IMPLICATION);
332                                 if (robdd.isTrue()) {
333                                     compareList.remove(j)
                                            ;
334                                     f2.updateParent(new
                                            False());
335                                     continue;
```

```
336                         }
337                     }
338                     if (dur2.getBound() >= dur1.
                            getBound()) {
339                         ROBDD robdd = robddUtil.
                                apply(sa2.getROBDD(
                                robddUtil), sa1.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
340                         if (robdd.isTrue()) {
341                             compareList.remove(i)
                                    ;
342                             f1.updateParent(new
                                    False());
343                             break;
344                         }
345                     }
346                 }
347                 else if (f1.getType() == Formula.
                        TYPE_NEGATION) {
348                     Duration dur1 = (Duration) ((
                            Negation) f1).getOperand()
                            ;
349                     Duration dur2 = (Duration) f2
                            ;
350                     Formula sa1 = dur1.
                            getStateAssertion();
351                     Formula sa2 = dur2.
                            getStateAssertion();
352                     if (dur1.getBound() >= dur2.
                            getBound()) {
353                         ROBDD robdd = robddUtil.
                                apply(sa1.getROBDD(
                                robddUtil), sa2.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
354                         if (robdd.isTrue()) {
355                             return new True();
356                         }
357                     }
358                 }
359                 else if (f2.getType() == Formula.
                        TYPE_NEGATION) {
360                     Duration dur1 = (Duration) f1
                            ;
361                     Duration dur2 = (Duration) ((
                            Negation) f2).getOperand()
                            ;
```

```
362                     Formula sa1 = dur1.
                            getStateAssertion();
363                     Formula sa2 = dur2.
                            getStateAssertion();
364                     if (dur2.getBound() >= dur1.
                            getBound()) {
365                         ROBDD robdd = robddUtil.
                                apply(sa2.getROBDD(
                                robddUtil), sa1.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
366                         if (robdd.isTrue()) {
367                             return new True();
368                         }
369                     }
370                 }
371                 else { // both durations
372                     Duration dur1 = (Duration) f1
                            ;
373                     Duration dur2 = (Duration) f2
                            ;
374                     Formula sa1 = dur1.
                            getStateAssertion();
375                     Formula sa2 = dur2.
                            getStateAssertion();
376                     if (dur1.getBound() >= dur2.
                            getBound()) {
377                         ROBDD robdd = robddUtil.
                                apply(sa1.getROBDD(
                                robddUtil), sa2.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
378                         if (robdd.isTrue()) {
379                             compareList.remove(i)
                                    ;
380                             f1.updateParent(new
                                    False());
381                             break;
382                         }
383                     }
384                     if (dur2.getBound() >= dur1.
                            getBound()) {
385                         ROBDD robdd = robddUtil.
                                apply(sa2.getROBDD(
                                robddUtil), sa1.
                                getROBDD(robddUtil),
                                ROBDDUtil.
                                OP_IMPLICATION);
```

```
386                                 if (robdd.isTrue()) {
387                                     compareList.remove(j)
                                            ;
388                                     f2.updateParent(new
                                            False());
389                                     continue;
390                                 }
391                             }
392                         }
393                     }
394                 }
395                 return formula;
396             }
397             case Formula.TYPE_DURATION:
398                 //fall through
399             case Formula.TYPE_NEGATION:
400                 //fall through
401             case Formula.TYPE_TRUE:
402                 //fall through
403             case Formula.TYPE_FALSE:
404                 return formula;
405             default:
406                 throw new IllegalArgumentException("
                        Formula is not a DC formula");
407         }
408
409     }
410
411     protected Formula applySALevel2Simp(Formula formula)
            throws Exception {
412         LinkedList compareList = new LinkedList();
413         ROBDDUtil robddUtil = new ROBDDUtil();
414         populateSACompareList(formula, compareList,
                formula.getType());
415         switch (formula.getType()) {
416             case Formula.TYPE_CONJUNCTION: {
417                 if (compareList.size() > 1) {
418                     SubsequenceROBDDConstructor src = new
                            SubsequenceROBDDConstructor(
                            compareList);
419                     for (int i=0 ; i<compareList.size() ;
                            i++) {
420                         ROBDD others = src.getROBDD(i,
                                ROBDDUtil.OP_CONJUNCTION);
421                         ROBDD me = robddUtil.makeROBDD((
                                Formula) compareList.get(i));
422                         ROBDD notme = me.negate();
423                         ROBDD implme = robddUtil.apply(
                                others, me, ROBDDUtil.
                                OP_IMPLICATION);
```

```
424                         if (implme.isTrue()) {
425                             Formula f = (Formula)
                                    compareList.get(i);
426                             Formula f2 = new True();
427                             f.updateParent(f2);
428                             compareList.set(i, f2);
429                         }
430                         ROBDD implnotme = robddUtil.apply
                                (others, notme, ROBDDUtil.
                                OP_IMPLICATION);
431                         if (implnotme.isTrue()) {
432                             Formula f = (Formula)
                                    compareList.get(i);
433                             return new False();
434                         }
435                     }
436                 }
437                 break;
438             }
439             case Formula.TYPE_DISJUNCTION: {
440                 if (compareList.size() > 1) {
441                     SubsequenceROBDDConstructor src = new
                            SubsequenceROBDDConstructor(
                            compareList);
442                     for (int i=0 ; i<compareList.size() ;
                            i++) {
443                         ROBDD others = src.getROBDD(i,
                                ROBDDUtil.OP_DISJUNCTION);
444                         ROBDD me = robddUtil.makeROBDD((
                                Formula) compareList.get(i));
445                         ROBDD notme = me.negate();
446                         ROBDD implme = robddUtil.apply(me
                                , others, ROBDDUtil.
                                OP_IMPLICATION);
447                         if (implme.isTrue()) {
448                             Formula f = (Formula)
                                    compareList.get(i);
449                             Formula f2 = new False();
450                             f.updateParent(f2);
451                             compareList.set(i, f2);
452                         }
453                         ROBDD implnotme = robddUtil.apply
                                (notme, others, ROBDDUtil.
                                OP_IMPLICATION);
454                         if (implnotme.isTrue()) {
455                             Formula f = (Formula)
                                    compareList.get(i);
456                             return new True();
457                         }
458                     }
```

```
459                 }
460                 break;
461             }
462             default:
463                 throw new IllegalStateException("
                    Conjunction or disjunction expected");
464         }
465         return formula;
466     }
467
468     protected void populateDCCompareList(Formula formula,
        LinkedList compareToList, int type) {
469         switch (formula.getType()) {
470             case Formula.TYPE_CONJUNCTION:
471             case Formula.TYPE_DISJUNCTION:
472                 if (formula.getType() != type) {
473                     return;
474                 }
475                 Connective conn = (Connective) formula;
476                 populateDCCompareList(conn.getLeftOperand
                    (), compareToList, type);
477                 populateDCCompareList(conn.
                    getRightOperand(), compareToList, type
                    );
478                 break;
479             case Formula.TYPE_DURATION:
480                 compareToList.add(formula);
481                 break;
482             case Formula.TYPE_NEGATION:
483                 Negation neg = (Negation) formula;
484                 if (neg.getOperand().getType() == Formula
                    .TYPE_DURATION) {
485                     compareToList.add(formula);
486                 }
487                 break;
488             case Formula.TYPE_TRUE:
489                 break;
490             case Formula.TYPE_FALSE:
491                 break;
492             case Formula.TYPE_CHOP:
493                 break;
494             default:
495                 throw new IllegalArgumentException("
                    Formula is not a DC formula");
496         }
497     }
498
499     protected void populateSACompareList(Formula formula,
        LinkedList compareToList, int type) {
500         switch (formula.getType()) {
```

```
501                 case Formula.TYPE_CONJUNCTION:
502                 case Formula.TYPE_DISJUNCTION:
503                     if (formula.getType() != type) {
504                         compareToList.add(formula);
505                         break;
506                     }
507                     Connective conn = (Connective) formula;
508                     populateSACompareList(conn.getLeftOperand
                            (), compareToList, type);
509                     populateSACompareList(conn.
                            getRightOperand(), compareToList, type
                            );
510                     break;
511                 case Formula.TYPE_STATE:
512                     compareToList.add(formula);
513                     break;
514                 case Formula.TYPE_NEGATION:
515                     compareToList.add(formula);
516                     break;
517                 case Formula.TYPE_TRUE:
518                     break;
519                 case Formula.TYPE_FALSE:
520                     break;
521                 default:
522                     throw new IllegalArgumentException("
                            Formula is not an SA formula");
523             }
524         }
525 }
```

I.6.2 SubsequenceROBDDConstructor.java

```
1  package bmc.simp;
2
3  import java.util.*;
4
5  import bmc.robdd.*;
6  import bmc.formula.*;
7
8  public class SubsequenceROBDDConstructor {
9
10     protected LinkedList compareList;
11     protected Map prefixRobdds;
12     protected Map postfixRobdds;
13     protected ROBDDUtil robddUtil;
14     protected int lastIndex;
15
16     public SubsequenceROBDDConstructor(LinkedList
           compareList) {
17         if (compareList.size() <= 1) {
```

```
18              throw new IllegalArgumentException("
                    compareList must contain more than one
                    element");
19          }
20          this.compareList = compareList;
21          this.prefixRobdds = new HashMap(compareList.size
                ());
22          this.postfixRobdds = new HashMap(compareList.size
                ());
23          this.robddUtil = new ROBDDUtil();
24          this.lastIndex = -1;
25      }
26
27      public ROBDD getROBDD(int excludeId, int operator)
            throws CloneNotSupportedException {
28          if (excludeId <= lastIndex) {
29              throw new IllegalArgumentException("
                    Subsequence iteration order "+
30              "must be strictly increasing");
31          }
32          lastIndex = excludeId;
33          ROBDD prefixRobdd = getPrefixROBDD(excludeId,
                operator);
34          ROBDD postfixRobdd = getPostfixROBDD(excludeId,
                operator);
35
36          if (prefixRobdd == null) {
37              return postfixRobdd;
38          }
39          else if (postfixRobdd == null) {
40              return prefixRobdd;
41          }
42          else {
43              return robddUtil.apply(prefixRobdd,
                    postfixRobdd, operator);
44          }
45      }
46
47      protected ROBDD getPrefixROBDD(int excludeId, int
            operator) throws CloneNotSupportedException {
48          Integer prefixId = new Integer(excludeId);
49          ROBDD robdd = (ROBDD) prefixRobdds.get(prefixId);
50          if (robdd != null) {
51              // Prefix ROBDD was already constructed
52              return robdd;
53          }
54          if (excludeId > 0) {
55              Formula formula = (Formula) compareList.get(
                    excludeId-1);
```

```
56              ROBDD formulaRobdd = formula.getROBDD(
                    robddUtil);
57              if (excludeId == 1) {
58                  robdd = formulaRobdd;
59              }
60              else {
61                  ROBDD previousRobdd = getPrefixROBDD(
                        excludeId-1, operator);
62                  robdd = robddUtil.apply(previousRobdd,
                        formulaRobdd, operator);
63              }
64              prefixRobdds.put(prefixId, robdd);
65          }
66          return robdd;
67      }
68
69      protected ROBDD getPostfixROBDD(int excludeId, int
            operator) throws CloneNotSupportedException {
70          Integer postfixId = new Integer(excludeId);
71          ROBDD robdd = (ROBDD) postfixRobdds.get(postfixId
                );
72          if (robdd != null) {
73              // Postfix ROBDD was already constructed
74              return robdd;
75          }
76          if (excludeId < compareList.size()-1) {
77              Formula formula = (Formula) compareList.get(
                    excludeId+1);
78              ROBDD formulaRobdd = formula.getROBDD(
                    robddUtil);
79              if (excludeId == compareList.size()-2) {
80                  robdd = formulaRobdd;
81              }
82              else {
83                  ROBDD nextRobdd = getPostfixROBDD(
                        excludeId+1, operator);
84                  robdd = robddUtil.apply(formulaRobdd,
                        nextRobdd, operator);
85              }
86              postfixRobdds.put(postfixId, robdd);
87          }
88          return robdd;
89      }
90
91  }
```

I.7 Package `bmc.tracer`

I.7.1 `HySatSolutionParser.java`

```
1  package bmc.tracer;
2
3  import java.io.*;
4  import java.text.*;
5  import java.util.*;
6  import bmc.*;
7  import bmc.formula.*;
8  import bmc.literal.*;
9  import bmc.util.*;
10
11 public class HySatSolutionParser {
12
13     public static final int VALUE_TRUE = 1;
14     public static final int VALUE_WILDCARD = 0;
15     public static final int VALUE_FALSE = -1;
16
17     public static final int STATUS_ERROR = -1;
18     public static final int STATUS_SAT = 1;
19     public static final int STATUS_UNSAT = 2;
20
21     private static final String SOLUTION_STR = "Solution
           :";
22     private static final String STATUS_STR = "STATUS:";
23     private static final String SAT_STR = "Satisfiable";
24     private static final String NO_SOL_STR = "No solution
           ";
25     private static final String CPU_TIME_STR = "cpu time
           for solving";
26
27     protected Goal goal;
28     protected Reader input;
29     protected LiteralHandler literalHandler;
30     protected int status;
31     protected Set stateTraceSet;
32     protected boolean displayTrace;
33     protected double cpuTime;
34
35     public HySatSolutionParser(Reader input,
           LiteralHandler literalHandler, Goal goal, boolean
           displayTrace) {
36         this.goal = goal;
37         this.input = input;
38         this.literalHandler = literalHandler;
39         this.status = STATUS_ERROR;
40         this.displayTrace = displayTrace;
41         this.stateTraceSet = new TreeSet();
42     }
43
44     public void parse() throws ParseException,
           IOException, CloneNotSupportedException {
```

```
45          LineNumberReader reader = new LineNumberReader(
                input);
46          String line;
47          while ((line = reader.readLine()) != null) {
48              if (line.startsWith(STATUS_STR)) {
49                  if (line.indexOf(SAT_STR) >= 0) {
50                      status = STATUS_SAT;
51                      break;
52                  }
53                  else if (line.indexOf(NO_SOL_STR) >= 0) {
54                      status = STATUS_UNSAT;
55                      break;
56                  }
57                  else {
58                      throw new ParseException("Unknown
                            status", reader.getLineNumber());
59                  }
60              }
61          }
62          if (line == null) {
63              throw new ParseException("No status line",
                    reader.getLineNumber());
64          }
65          while ((line = reader.readLine()) != null) {
66              if (line.startsWith(CPU_TIME_STR)) {
67                  StringTokenizer stok = new
                        StringTokenizer(line);
68                  while (stok.hasMoreTokens() && !":".
                        equals(stok.nextToken())) {}
69                  if (!stok.hasMoreTokens()) {
70                      throw new ParseException("No CPU time
                            ", reader.getLineNumber());
71                  }
72                  String timeStr = stok.nextToken();
73                  cpuTime = Double.parseDouble(timeStr);
74                  break;
75              }
76          }
77          if (line == null) {
78              throw new ParseException("No CPU time",reader
                    .getLineNumber());
79          }
80          if (displayTrace && status == STATUS_SAT) {
81              while ((line = reader.readLine()) != null) {
82                  if (line.equals(SOLUTION_STR)) {
83                      break;
84                  }
85              }
86              if (line == null) {
```

```
87                  throw new ParseException("Solution token
                        not found", reader.getLineNumber());
88              }
89              if ((line = reader.readLine()) == null) {
90                  throw new ParseException("Solution token
                        found but no data lines found",
91                  reader.getLineNumber());
92              }
93              StringTokenizer stok = new StringTokenizer(
                    line);
94              int[] varValues = new int[literalHandler.
                    getNumVariables()+1];
95              while (stok.hasMoreTokens()) {
96                  String token = stok.nextToken();
97                  char firstChar = token.charAt(0);
98                  if (firstChar == '*') {
99                      varValues[Integer.parseInt(token.
                            substring(1))]=
100                     HySatSolutionParser.VALUE_WILDCARD;
101                 }
102                 else if (firstChar == '-') {
103                     varValues[Integer.parseInt(token.
                            substring(1))]=
104                     HySatSolutionParser.VALUE_FALSE;
105                 }
106                 else {
107                     varValues[Integer.parseInt(token)]=
108                     HySatSolutionParser.VALUE_TRUE;
109                 }
110             }
111
112             Iterator it = goal.getStates().iterator();
113             int k = goal.getSettings().getK();
114             while (it.hasNext()) {
115                 State state = (State) it.next();
116                 int[] stateValues = new int[k];
117                 for (int i=0; i<k; i++) {
118                     if (literalHandler.hasDef(state)) {
119                         Literal literal = literalHandler.
                                getLiteral(state, i);
120                         stateValues[i] = varValues[
                                literal.getNumber()];
121                     }
122                     else {
123                         stateValues[i] =
                                HySatSolutionParser.
                                VALUE_WILDCARD;
124                     }
125                 }
```

```
126                 stateTraceSet.add(new StateTrace(state,
                        stateValues));
127             }
128         }
129     }
130
131     public Set getStateTraceSet() {
132         return stateTraceSet;
133     }
134
135     public int getStatus() {
136         return status;
137     }
138
139     public double getCPUTime() {
140         return cpuTime;
141     }
142 }
```

I.7.2 StateTrace.java

```
1  package bmc.tracer;
2
3  import bmc.formula.*;
4
5  public class StateTrace implements Comparable {
6
7      private State state;
8      private int[] values;
9
10     public StateTrace(State state, int[] values) {
11         this.state = state;
12         this.values = values;
13     }
14
15     public State getState() {
16         return state;
17     }
18
19     public int[] getValues() {
20         return values;
21     }
22
23     public int compareTo(Object o) {
24         if (o instanceof StateTrace) {
25             State oState = ((StateTrace) o).getState();
26             return state.getName().compareTo(oState.
                   getName());
27         }
28         else {
```

```
29              throw new IllegalArgumentException("CompareTo
                    -object must be of type StateTrace");
30          }
31      }
32
33      public String toString() {
34          StringBuffer sb = new StringBuffer();
35          sb.append(state.getName()).append(" [");
36          for (int i=0; i<values.length ; i++) {
37              sb.append(values[i]);
38              if (i < values.length-1) {
39                  sb.append(",");
40              }
41          }
42          sb.append("]");
43          return sb.toString();
44      }
45
46
47  }
```

I.7.3 StateTraceFrame.java

```
1   package bmc.tracer;
2
3   import javax.swing.*;
4   import java.awt.event.*;
5   import java.util.*;
6   import bmc.*;
7   import bmc.tracer.*;
8   import bmc.formula.*;
9   import bmc.util.*;
10
11  public class StateTraceFrame extends JFrame {
12
13      public static final int DISPLAY_MODE_NONE = 0;
14      public static final int DISPLAY_MODE_GUI = 1;
15
16      public StateTraceFrame(Settings settings, Set
            stateTraceSet) {
17          super("BMC/DC Translator: Counter Example Found")
                ;
18          getContentPane().add(new StateTracePanel(settings
                , stateTraceSet));
19          pack();
20          addWindowListener(new StateTraceWindowAdapter(
                this));
21      }
22
```

```
23      private class StateTraceWindowAdapter extends
            WindowAdapter {
24          StateTraceFrame frame;
25          public StateTraceWindowAdapter(StateTraceFrame
                frame) {
26              this.frame = frame;
27          }
28
29          public void windowClosing(WindowEvent e) {
30              frame.dispose();
31          }
32      }
33
34      public static void display(Settings settings, Set
            stateTraceSet) {
35          StateTraceFrame frame = new StateTraceFrame(
                settings, stateTraceSet);
36          frame.show();
37      }
38  }
```

I.7.4 StateTracePanel.java

```
1   package bmc.tracer;
2
3   import javax.swing.*;
4   import java.awt.*;
5   import java.awt.font.*;
6   import java.util.*;
7   import bmc.*;
8   import bmc.tracer.*;
9   import bmc.util.*;
10
11  public class StateTracePanel extends JComponent {
12
13      protected Set stateTraces;
14      protected int k;
15
16      protected static final int STATE_NAME_WIDTH = 70;
17
18      protected final int scaleWidth;
19
20      // Height of state trace slot
21      protected static final int STATE_TRACE_HEIGHT = 50;
22
23      // Distance from start of state trace slot to tt
            value
24      // and distance from ff value to end of state trace
            slot
```

```
25      protected static final int VALUE_SKIP = (int) (
            STATE_TRACE_HEIGHT*.35);
26
27      protected static final int HORIZONTAL_INSETS = 10;
28      protected static final int VERTICAL_INSETS = 15;
29      protected static final int Y_LEGEND_WIDTH = 5;
30
31      protected final int panelWidth;
32
33      protected int stepCount;
34
35      protected static final int STEP_WIDTH = 20;
36
37      protected static final int MARK_SIZE = 8;
38
39      protected static final int AXIS_STEP = 5;
40
41      protected static final String TT_STR = "tt";
42      protected static final String FF_STR = "ff";
43
44      public StateTracePanel(Settings settings, Set
            stateTraces) {
45          this.k = settings.getK();
46          this.stateTraces = stateTraces;
47          stepCount = (int) Math.ceil(k/5.0)*5;
48          scaleWidth = (stepCount)*STEP_WIDTH;
49          panelWidth = HORIZONTAL_INSETS + STATE_NAME_WIDTH
                + HORIZONTAL_INSETS + Y_LEGEND_WIDTH +
50          HORIZONTAL_INSETS + scaleWidth +HORIZONTAL_INSETS
                ;
51
52      }
53
54      public void paint(Graphics graphics) {
55          Graphics2D g = (Graphics2D) graphics;
56          g.setBackground(Color.white);
57          FontMetrics metrics = g.getFontMetrics();
58          int x = HORIZONTAL_INSETS;
59          int y = VERTICAL_INSETS;
60          int x2 = HORIZONTAL_INSETS + STATE_NAME_WIDTH +
                HORIZONTAL_INSETS + Y_LEGEND_WIDTH +
61          HORIZONTAL_INSETS;
62          Iterator it = stateTraces.iterator();
63          while (it.hasNext()) {
64              g.setColor(Color.black);
65              StateTrace trace = (StateTrace) it.next();
66              String stateName = trace.getState().getName()
                    ;
67              String ldots = "...";
```

```
68              if (metrics.stringWidth(stateName) >
                    STATE_NAME_WIDTH) {
69                  while (metrics.stringWidth(stateName+
                        ldots) > STATE_NAME_WIDTH) {
70                      stateName = stateName.substring(0,
                            stateName.length()-1);
71                  }
72                  stateName += ldots;
73              }
74              g.drawString(stateName, x, y+((int) (0.6*
                    STATE_TRACE_HEIGHT)));
75              int yTT = y+VALUE_SKIP;
76              int yFF = y+STATE_TRACE_HEIGHT-VALUE_SKIP;
77              LineMetrics lm = metrics.getLineMetrics(
                    TT_STR, g);
78              g.drawString(TT_STR, x2-Y_LEGEND_WIDTH-
                    HORIZONTAL_INSETS,yTT+4);
79              g.drawString(FF_STR, x2-Y_LEGEND_WIDTH-
                    HORIZONTAL_INSETS,yFF+4);
80              int[] values = trace.getValues();
81              int lastYVal = yFF;
82              int currYVal = yFF;
83              int lastVal = HySatSolutionParser.VALUE_FALSE
                    ;
84              for(int i = 0; i<values.length ; i++) {
85                  int value = values[i];
86                  if (value == HySatSolutionParser.
                        VALUE_WILDCARD) {
87                      g.setColor(Color.gray);
88                      g.fillRect(x2+i*STEP_WIDTH, Math.min(
                            yFF,yTT), STEP_WIDTH, Math.abs(yTT
                            -yFF));
89                      g.setColor(Color.black);
90                  }
91                  else {
92                      if (value == HySatSolutionParser.
                            VALUE_TRUE) {
93                          currYVal = yTT;
94                      }
95                      else {
96                          currYVal = yFF;
97                      }
98                      if (lastVal != value) {
99                          g.drawLine(x2+i*STEP_WIDTH,
                                lastYVal, x2+i*STEP_WIDTH,
                                currYVal);
100                     }
101                     g.drawLine(x2+i*STEP_WIDTH,currYVal,
                            x2+(i+1)*STEP_WIDTH,currYVal);
102                 }
```

```
103                     lastYVal = currYVal;
104                     lastVal = value;
105                 }
106                 y += STATE_TRACE_HEIGHT;
107             }
108             g.drawLine(x2, VERTICAL_INSETS, x2, y);
109             g.drawLine(x2, y, x2+scaleWidth, y);
110             for (int i=0; i<=stepCount; i++) {
111                 int x3 = x2+i*STEP_WIDTH;
112                 g.drawLine(x3, y-MARK_SIZE/2, x3, y+MARK_SIZE
                        /2);
113                 if (i % AXIS_STEP == 0) {
114                     String figure = String.valueOf(i);
115                     g.drawString(figure, x3-((int) (0.5*
                            metrics.stringWidth(figure))), y+
                            MARK_SIZE+metrics.getHeight());
116                 }
117             }
118         }
119
120         public Dimension getPreferredSize() {
121             return new Dimension(panelWidth,
                    STATE_TRACE_HEIGHT * (stateTraces.size()+1));
122         }
123     }
```

I.8 Package `bmc.trans`

I.8.1 `DCToDIMACSTranslator.java`

```
1   package bmc.trans;
2
3   import bmc.literal.*;
4   import bmc.util.*;
5   import bmc.formula.*;
6   import bmc.constraint.*;
7   import bmc.*;
8
9   public class DCToDIMACSTranslator extends DCTranslator {
10
11      public DCToDIMACSTranslator(LiteralHandler
            literalHandler, Settings settings,
            ConstraintWriter cstrWriter) {
12          super(literalHandler, settings, cstrWriter);
13      }
14
15      public void translate(Formula formula) throws
            Exception {
16          formula = flattenDeepDurations(formula);
17
```

```
18          translateDC(formula, true);
19          if (!polarityOpt) {
20              translateDC(formula, false);
21          }
22
23          Constraint cstr = new DIMACSConstraint();
24          for (int i=0; i<=k; i++) {
25              Literal lit = literalHandler.getLiteral(
                    formula,0,i);
26              cstr.addTerm(lit);
27          }
28          addConstraint(cstr);
29      }
30
31      protected void translateDC(Formula formula, boolean
            polarity)
32      throws Exception {
33          if (literalHandler.hasDef(formula, polarity)) {
34              if (Main.debugOccurrence) {
35                  System.out.println("Reoccurence: "+
                        formula+","+polarity);
36              }
37              return;
38          }
39          if (Main.debugOccurrence) {
40              System.out.println("New occurence: "+formula
                    +","+polarity);
41          }
42          switch (formula.getType()) {
43              case Formula.TYPE_TRUE:
44                  break;
45              case Formula.TYPE_FALSE:
46                  break;
47              case Formula.TYPE_DURATION:
48                  translateDC((Duration) formula, polarity)
                        ;
49                  break;
50              case Formula.TYPE_CONJUNCTION:
51                  translateDC((Conjunction) formula,
                        polarity);
52                  break;
53              case Formula.TYPE_DISJUNCTION:
54                  translateDC((Disjunction) formula,
                        polarity);
55                  break;
56              case Formula.TYPE_CHOP:
57                  translateDC((Chop) formula, polarity);
58                  break;
59              case Formula.TYPE_NEGATION:
```

```
60                  translateDC((Negation) formula, polarity)
                        ;
61                  break;
62              default:
63                  throw new IllegalStateException("Unknown
                        formula type");
64          }
65          literalHandler.addDef(formula, polarity);
66      }
67
68
69      protected void translateDC(Conjunction con, boolean
            polarity)
70      throws Exception {
71          Formula p1 = con.getLeftOperand();
72          Formula p2 = con.getRightOperand();
73          if (polarity) {
74              translateDC(p1, true);
75              translateDC(p2, true);
76              for (int i=0; i<=k; i++) {
77                  for (int j=i; j<=k; j++) {
78                      Constraint cstr1 = new
                            DIMACSConstraint();
79                      Literal nconij = literalHandler.
                            getLiteral(con, i, j).flipSign();
80                      cstr1.addTerm(nconij);
81                      Literal p1ij = literalHandler.
                            getLiteral(p1, i, j);
82                      cstr1.addTerm(p1ij);
83                      addConstraint(cstr1);
84
85                      Constraint cstr2 = new
                            DIMACSConstraint();
86                      cstr2.addTerm(nconij);
87                      Literal p2ij = literalHandler.
                            getLiteral(p2, i, j);
88                      cstr2.addTerm(p2ij);
89                      addConstraint(cstr2);
90                  }
91              }
92          }
93          else {
94              translateDC(p1, false);
95              translateDC(p2, false);
96              for (int i=0; i<=k; i++) {
97                  for (int j=i; j<=k; j++) {
98                      Constraint cstr1 = new
                            DIMACSConstraint();
99                      Literal conij = literalHandler.
                            getLiteral(con, i, j);
```

```
100                     cstr1.addTerm(conij);
101                     Literal np1ij = literalHandler.
                            getLiteral(p1, i, j).flipSign();
102                     cstr1.addTerm(np1ij);
103                     Literal np2ij = literalHandler.
                            getLiteral(p2, i, j).flipSign();
104                     cstr1.addTerm(np2ij);
105                     addConstraint(cstr1);
106                 }
107             }
108         }
109     }
110
111     protected void translateDC(Disjunction dis, boolean
            polarity)
112     throws Exception {
113         Formula p1 = dis.getLeftOperand();
114         Formula p2 = dis.getRightOperand();
115         if (polarity) {
116             translateDC(p1, true);
117             translateDC(p2, true);
118             for (int i=0; i<=k; i++) {
119                 for (int j=i; j<=k; j++) {
120                     Constraint cstr1 = new
                            DIMACSConstraint();
121                     Literal ndisij = literalHandler.
                            getLiteral(dis, i, j).flipSign();
122                     cstr1.addTerm(ndisij);
123                     Literal p1ij = literalHandler.
                            getLiteral(p1, i, j);
124                     cstr1.addTerm(p1ij);
125                     Literal p2ij = literalHandler.
                            getLiteral(p2, i, j);
126                     cstr1.addTerm(p2ij);
127                     addConstraint(cstr1);
128                 }
129             }
130         }
131         else {
132             translateDC(p1, false);
133             translateDC(p2, false);
134             for (int i=0; i<=k; i++) {
135                 for (int j=i; j<=k; j++) {
136                     Constraint cstr1 = new
                            DIMACSConstraint();
137                     Literal disij = literalHandler.
                            getLiteral(dis, i, j);
138                     cstr1.addTerm(disij);
139                     Literal np1ij = literalHandler.
                            getLiteral(p1, i, j).flipSign();
```

```
140                     cstr1.addTerm(np1ij);
141                     addConstraint(cstr1);
142
143                     Constraint cstr2 = new
                            DIMACSConstraint();
144                     cstr2.addTerm(disij);
145                     Literal np2ij = literalHandler.
                            getLiteral(p2, i, j).flipSign();
146                     cstr2.addTerm(np2ij);
147                     addConstraint(cstr2);
148                 }
149             }
150         }
151     }
152
153     protected void translateDC(Negation neg, boolean
            polarity)
154     throws Exception {
155         if (polarity) {
156             translateDC(neg.getOperand(), false);
157         }
158         else {
159             translateDC(neg.getOperand(), true);
160         }
161     }
162
163     protected void translateDC(Duration dur, boolean
            polarity)
164     throws Exception {
165         if (dur.getBound() > 1) {
166             throw new IllegalArgumentException("Duration
                    bound is not allowed to be greater than
                    one in DIMACS");
167         }
168         Formula sa = dur.getStateAssertion();
169         if (polarity) {
170             translateSA(sa, true);
171             for (int i=0; i<=k-1; i++) {
172                 for (int j=i+1; j<=k; j++) {
173                     Constraint cstr1 = new
                            DIMACSConstraint();
174                     Literal ndurij = literalHandler.
                            getLiteral(dur, i, j).flipSign();
175                     cstr1.addTerm(ndurij);
176                     Literal sai = literalHandler.
                            getLiteral(sa, i);
177                     cstr1.addTerm(sai);
178                     Literal duri1j = literalHandler.
                            getLiteral(dur, i+1, j);
179                     cstr1.addTerm(duri1j);
```

```
180                     addConstraint(cstr1);
181                 }
182             }
183             for (int i=0; i<=k; i++) {
184                 Constraint cstr1 = new DIMACSConstraint()
                        ;
185                 Literal ndurii = literalHandler.
                        getLiteral(dur, i, i).flipSign();
186                 cstr1.addTerm(ndurii);
187                 addConstraint(cstr1);
188             }
189         }
190         else {
191             translateSA(sa, false);
192             for (int i=0; i<=k-1; i++) {
193                 for (int j=i+1; j<=k; j++) {
194                     Constraint cstr1 = new
                            DIMACSConstraint();
195                     Literal durij = literalHandler.
                            getLiteral(dur, i, j);
196                     cstr1.addTerm(durij);
197                     Literal nsai = literalHandler.
                            getLiteral(sa, i).flipSign();
198                     cstr1.addTerm(nsai);
199                     addConstraint(cstr1);
200
201                     Constraint cstr2 = new
                            DIMACSConstraint();
202                     cstr2.addTerm(durij);
203                     Literal nduri1j = literalHandler.
                            getLiteral(dur, i+1, j).flipSign()
                            ;
204                     cstr2.addTerm(nduri1j);
205                     addConstraint(cstr2);
206                 }
207             }
208         }
209     }
210
211     protected void translateDC(Chop chop, boolean
            polarity)
212     throws Exception {
213         // Make t^k[[phi]] and t^k[[psi]]
214         Formula p1 = chop.getLeftOperand();
215         Formula p2 = chop.getRightOperand();
216         if (polarity) {
217             translateDC(p1, true);
218             translateDC(p2, true);
219             for (int i=0; i<=k; i++) {
220                 for (int j=i; j<=k; j++) {
```

```
221                     Literal p1p2 = literalHandler.
                            getLiteral(chop,i,j);
222                     Literal np1p2 = p1p2.flipSign();
223                     Constraint cstr1 = new
                            DIMACSConstraint();
224                     cstr1.addTerm(np1p2);
225                     for (int m=i; m<=j; m++) {
226                         Literal combi = literalHandler.
                                getLiteral(chop,i,m,j);
227                         cstr1.addTerm(combi);
228                     }
229                     addConstraint(cstr1);
230
231                     for (int m=i; m<=j; m++) {
232                         Constraint cstr2 = new
                                DIMACSConstraint();
233                         Literal ncombi = literalHandler.
                                getLiteral(chop,i,m,j).
                                flipSign();
234                         cstr2.addTerm(ncombi);
235                         Literal p1im = literalHandler.
                                getLiteral(p1, i, m);
236                         cstr2.addTerm(p1im);
237                         addConstraint(cstr2);
238
239                         Constraint cstr3 = new
                                DIMACSConstraint();
240                         cstr3.addTerm(ncombi);
241                         Literal p2mj = literalHandler.
                                getLiteral(p2, m, j);
242                         cstr3.addTerm(p2mj);
243                         addConstraint(cstr3);
244                     }
245                 }
246             }
247         }
248         else { // !polarity
249             translateDC(p1, false);
250             translateDC(p2, false);
251             for (int i=0; i<=k; i++) {
252                 for (int j=i; j<=k; j++) {
253                     for (int m=i; m<=j; m++) {
254                         Constraint cstr1 = new
                                DIMACSConstraint();
255                         Literal chopij = literalHandler.
                                getLiteral(chop,i,j);
256                         cstr1.addTerm(chopij);
257                         Literal p1im = literalHandler.
                                getLiteral(p1, i, m);
258                         Literal np1im = p1im.flipSign();
```

```
259                         cstr1.addTerm(np1im);
260                         Literal p2mj = literalHandler.
                                getLiteral(p2, m, j);
261                         Literal np2mj = p2mj.flipSign();
262                         cstr1.addTerm(np2mj);
263                         addConstraint(cstr1);
264                     }
265                 }
266             }
267         }
268     }
269
270
271
272     protected void translateSA(Negation neg, boolean
            polarity)
273     throws Exception {
274         if (polarity) {
275             translateSA(neg.getOperand(), false);
276         }
277         else {
278             translateSA(neg.getOperand(), true);
279         }
280     }
281     protected void translateSA(Conjunction con, boolean
            polarity)
282     throws Exception {
283         Formula s1 = con.getLeftOperand();
284         Formula s2 = con.getRightOperand();
285         if (polarity) {
286             translateSA(s1, true);
287             translateSA(s2, true);
288             for (int i=0; i<=k-1; i++) {
289                 Constraint cstr1 = new DIMACSConstraint()
                        ;
290                 Literal nconi = literalHandler.getLiteral
                        (con, i).flipSign();
291                 cstr1.addTerm(nconi);
292                 Literal s1i = literalHandler.getLiteral(
                        s1, i);
293                 cstr1.addTerm(s1i);
294                 addConstraint(cstr1);
295
296                 Constraint cstr2 = new DIMACSConstraint()
                        ;
297                 cstr2.addTerm(nconi);
298                 Literal s2i = literalHandler.getLiteral(
                        s2, i);
299                 cstr2.addTerm(s2i);
300                 addConstraint(cstr2);
```

```
301             }
302         }
303         else {
304             translateSA(s1, false);
305             translateSA(s2, false);
306             for (int i=0; i<=k-1; i++) {
307                 Constraint cstr1 = new DIMACSConstraint()
                        ;
308                 Literal coni = literalHandler.getLiteral(
                        con, i);
309                 cstr1.addTerm(coni);
310                 Literal ns1i = literalHandler.getLiteral(
                        s1, i).flipSign();
311                 cstr1.addTerm(ns1i);
312                 Literal ns2i = literalHandler.getLiteral(
                        s2, i).flipSign();
313                 cstr1.addTerm(ns2i);
314                 addConstraint(cstr1);
315             }
316         }
317     }
318     protected void translateSA(Disjunction dis, boolean
            polarity)
319     throws Exception {
320         Formula s1 = dis.getLeftOperand();
321         Formula s2 = dis.getRightOperand();
322         if (polarity) {
323             translateSA(s1, true);
324             translateSA(s2, true);
325             for (int i=0; i<=k-1; i++) {
326                 Constraint cstr1 = new DIMACSConstraint()
                        ;
327                 Literal ndisi = literalHandler.getLiteral
                        (dis, i).flipSign();
328                 cstr1.addTerm(ndisi);
329                 Literal s1i = literalHandler.getLiteral(
                        s1, i);
330                 cstr1.addTerm(s1i);
331                 Literal s2i = literalHandler.getLiteral(
                        s2, i);
332                 cstr1.addTerm(s2i);
333                 addConstraint(cstr1);
334             }
335         }
336         else {
337             translateSA(s1, false);
338             translateSA(s2, false);
339             for (int i=0; i<=k-1; i++) {
340                 Constraint cstr1 = new DIMACSConstraint()
                        ;
```

```
341                 Literal disi = literalHandler.getLiteral(
                        dis, i);
342                 cstr1.addTerm(disi);
343                 Literal ns1i = literalHandler.getLiteral(
                        s1, i).flipSign();
344                 cstr1.addTerm(ns1i);
345                 addConstraint(cstr1);
346
347                 Constraint cstr2 = new DIMACSConstraint()
                        ;
348                 cstr2.addTerm(disi);
349                 Literal ns2i = literalHandler.getLiteral(
                        s2, i).flipSign();
350                 cstr2.addTerm(ns2i);
351                 addConstraint(cstr2);
352             }
353         }
354     }
355
356
357     protected void translateSA(Formula formula, boolean
            polarity)
358     throws Exception {
359         if (literalHandler.hasDef(formula, polarity)) {
360             // Do not output anything
361             return;
362         }
363         switch (formula.getType()) {
364             case Formula.TYPE_STATE:
365                 // Do nothing
366                 break;
367             case Formula.TYPE_TRUE:
368                 // Do nothing
369                 break;
370             case Formula.TYPE_FALSE:
371                 // Do nothing
372                 break;
373             case Formula.TYPE_NEGATION: {
374                 translateSA((Negation) formula, polarity)
                        ;
375                 break;
376             }
377             case Formula.TYPE_CONJUNCTION: {
378                 translateSA((Conjunction) formula,
                        polarity);
379                 break;
380             }
381             case Formula.TYPE_DISJUNCTION: {
382                 translateSA((Disjunction) formula,
                        polarity);
```

```
383                 break;
384             }
385             default:
386                 throw new IllegalStateException("Unknown
                        SA formula type");
387         }
388         literalHandler.addDef(formula, polarity);
389     }
390
391     private Formula flattenDeepDurations(Formula formula)
392     throws CloneNotSupportedException {
393         switch (formula.getType()) {
394             case Formula.TYPE_DURATION: {
395                 Duration dur = (Duration) formula;
396                 if (dur.getBound() > 1) {
397                     return chopUp(dur.getBound(),
398                     new Duration(dur.getStateAssertion(),
                            1));
399                 }
400                 else {
401                     return formula;
402                 }
403             }
404             case Formula.TYPE_CONJUNCTION:
405             case Formula.TYPE_DISJUNCTION:
406             case Formula.TYPE_CHOP: {
407                 Connective con = (Connective) formula;
408                 Formula flatLeft = flattenDeepDurations(
                        con.getLeftOperand());
409                 if (flatLeft != con.getLeftOperand()) {
410                     con.setLeftOperand(flatLeft);
411                 }
412                 Formula flatRight = flattenDeepDurations(
                        con.getRightOperand());
413                 if (flatRight != con.getRightOperand()) {
414                     con.setRightOperand(flatRight);
415                 }
416                 return formula;
417             }
418             case Formula.TYPE_NEGATION: {
419                 Negation neg = (Negation) formula;
420                 Formula flatOp = flattenDeepDurations(neg
                        .getOperand());
421                 if (flatOp != neg.getOperand()) {
422                     neg.setOperand(flatOp);
423                 }
424                 return formula;
425             }
426             default:
427                 return formula;
```

```
428         }
429     }
430
431
432
433     private Formula chopUp(int n, Duration dur1)
434     throws CloneNotSupportedException {
435         if (n == 1) {
436             return (Formula) dur1.clone();
437         }
438         else if (n % 2 == 0) {
439             Formula left = chopUp(n/2, dur1);
440             Formula right = (Formula) left.clone();
441             return new Chop(left, right);
442         }
443         else {
444             Formula left = chopUp(n-1, dur1);
445             Formula right = chopUp(1, dur1);
446             return new Chop(left, right);
447         }
448     }
449
450 }
```

I.8.2 DCToZOLCSTranslator.java

```
1  package bmc.trans;
2
3  import bmc.formula.*;
4  import bmc.constraint.*;
5  import bmc.literal.*;
6  import bmc.util.*;
7  import bmc.*;
8
9  import java.util.*;
10
11
12 public class DCToZOLCSTranslator extends DCTranslator {
13
14     public DCToZOLCSTranslator(LiteralHandler
           literalHandler, Settings settings,
           ConstraintWriter cstrWriter) {
15         super(literalHandler, settings, cstrWriter);
16     }
17
18
19     public void translate(Formula formula) throws
           Exception {
20         translateDC(formula, true);
21         if (!polarityOpt) {
```

```
22              translateDC(formula, false);
23          }
24
25          Constraint cstr = new ZOLCSConstraint();
26          for (int i=0; i<=k; i++) {
27              Literal lit = literalHandler.getLiteral(
                    formula,0,i);
28              cstr.addTerm(lit);
29          }
30          addConstraint(cstr);
31      }
32
33      protected void translateDC(Formula formula, boolean
            polarity)
34      throws Exception {
35          if (literalHandler.hasDef(formula, polarity)) {
36              if (Main.debugOccurrence) {
37                  System.out.println("Reoccurence: "+
                        formula+","+polarity);
38              }
39              return;
40          }
41          if (Main.debugOccurrence) {
42              System.out.println("New occurence: "+formula
                    +","+polarity);
43          }
44          switch (formula.getType()) {
45              case Formula.TYPE_TRUE:
46                  // Do nothing
47                  break;
48              case Formula.TYPE_FALSE:
49                  // Do nothing
50                  break;
51              case Formula.TYPE_DURATION:
52                  translateDC((Duration) formula, polarity)
                        ;
53                  break;
54              case Formula.TYPE_CONJUNCTION:
55                  translateDC((Conjunction) formula,
                        polarity);
56                  break;
57              case Formula.TYPE_DISJUNCTION:
58                  translateDC((Disjunction) formula,
                        polarity);
59                  break;
60              case Formula.TYPE_CHOP:
61                  translateDC((Chop) formula, polarity);
62                  break;
63              case Formula.TYPE_NEGATION:
```

```
64                  translateDC((Negation) formula, polarity)
                        ;
65                  break;
66              default:
67                  throw new IllegalStateException("Unknown
                        formula type");
68          }
69          literalHandler.addDef(formula, polarity);
70      }
71
72      protected void translateDC(Chop chop, boolean
            polarity)
73      throws Exception {
74          Formula p1 = chop.getLeftOperand();
75          Formula p2 = chop.getRightOperand();
76          if (polarity) {
77              translateDC(p1, true);
78              translateDC(p2, true);
79              for (int i=0; i<=k; i++) {
80                  for (int j=i; j<=k; j++) {
81                      Literal p1p2 = literalHandler.
                            getLiteral(chop,i,j);
82                      Literal np1p2 = p1p2.flipSign();
83                      ZOLCSConstraint cstr1 = new
                            ZOLCSConstraint();
84                      cstr1.addTerm(np1p2);
85                      for (int m=i; m<=j; m++) {
86                          Literal combi = literalHandler.
                                getLiteral(chop,i,m,j);
87                          cstr1.addTerm(combi);
88                      }
89                      addConstraint(cstr1);
90
91                      for (int m=i; m<=j; m++) {
92                          ZOLCSConstraint cstr2 = new
                                ZOLCSConstraint(2);
93                          Literal ncombi = literalHandler.
                                getLiteral(chop,i,m,j).
                                flipSign();
94                          cstr2.addTerm(ncombi,2);
95                          Literal p1im = literalHandler.
                                getLiteral(p1, i, m);
96                          cstr2.addTerm(p1im);
97                          Literal p2mj = literalHandler.
                                getLiteral(p2, m, j);
98                          cstr2.addTerm(p2mj);
99                          addConstraint(cstr2);
100                     }
101                 }
102             }
```

```
103         }
104         else { // !polarity
105             translateDC(p1, false);
106             translateDC(p2, false);
107             for (int i=0; i<=k; i++) {
108                 for (int j=i; j<=k; j++) {
109                     for (int m=i; m<=j; m++) {
110                         ZOLCSConstraint cstr1 = new
                                ZOLCSConstraint();
111                         Literal chopij = literalHandler.
                                getLiteral(chop,i,j);
112                         cstr1.addTerm(chopij);
113                         Literal p1im = literalHandler.
                                getLiteral(p1, i, m);
114                         Literal np1im = p1im.flipSign();
115                         cstr1.addTerm(np1im);
116                         Literal p2mj = literalHandler.
                                getLiteral(p2, m, j);
117                         Literal np2mj = p2mj.flipSign();
118                         cstr1.addTerm(np2mj);
119                         addConstraint(cstr1);
120                     }
121                 }
122             }
123         }
124     }
125     protected void translateDC(Duration dur, boolean
            polarity)
126     throws Exception {
127         Formula sa = dur.getStateAssertion();
128         int n = dur.getBound();
129         if (polarity) {
130             translateSA(sa, true);
131             for (int i=0; i<=k; i++) {
132                 for (int j=i; j<=k; j++) {
133                     ZOLCSConstraint cstr1 = new
                            ZOLCSConstraint(n);
134                     Literal ndurij = literalHandler.
                            getLiteral(dur, i, j).flipSign();
135                     cstr1.addTerm(ndurij, n);
136                     for (int m=i; m<=j-1; m++) {
137                         Literal sam = literalHandler.
                                getLiteral(sa, m);
138                         cstr1.addTerm(sam);
139                     }
140                     addConstraint(cstr1);
141                 }
142             }
143         }
144         else {
```

```
145             translateSA(sa, false);
146             for (int i=0; i<=k-n; i++) {
147                 for (int j=i+n; j<=k; j++) {
148                     ZOLCSConstraint cstr1 = new
                            ZOLCSConstraint(j-i+1-n);
149                     Literal durij = literalHandler.
                            getLiteral(dur, i, j);
150                     cstr1.addTerm(durij, j-i+1-n);
151                     for (int m=i; m<=j-1; m++) {
152                         Literal nsam = literalHandler.
                                getLiteral(sa, m).flipSign();
153                         cstr1.addTerm(nsam);
154                     }
155                     addConstraint(cstr1);
156                 }
157             }
158         }
159
160     }
161     protected void translateDC(Conjunction con, boolean
            polarity)
162     throws Exception {
163         Formula p1 = con.getLeftOperand();
164         Formula p2 = con.getRightOperand();
165         if (polarity) {
166             translateDC(p1, true);
167             translateDC(p2, true);
168             for (int i=0; i<=k; i++) {
169                 for (int j=i; j<=k; j++) {
170                     ZOLCSConstraint cstr1 = new
                            ZOLCSConstraint(2);
171                     Literal np1p2 = literalHandler.
                            getLiteral(con,i,j).flipSign();
172                     cstr1.addTerm(np1p2, 2);
173                     Literal p1ij = literalHandler.
                            getLiteral(p1, i, j);
174                     cstr1.addTerm(p1ij);
175                     Literal p2ij = literalHandler.
                            getLiteral(p2, i, j);
176                     cstr1.addTerm(p2ij);
177                     addConstraint(cstr1);
178                 }
179             }
180         }
181         else { // !polarity
182             translateDC(p1, false);
183             translateDC(p2, false);
184             for (int i=0; i<=k; i++) {
185                 for (int j=i; j<=k; j++) {
```

```
186                     ZOLCSConstraint cstr1 = new
                            ZOLCSConstraint();
187                     Literal p1p2 = literalHandler.
                            getLiteral(con,i,j);
188                     cstr1.addTerm(p1p2);
189                     Literal np1ij = literalHandler.
                            getLiteral(p1, i, j).flipSign();
190                     cstr1.addTerm(np1ij);
191                     Literal np2ij = literalHandler.
                            getLiteral(p2, i, j).flipSign();
192                     cstr1.addTerm(np2ij);
193                     addConstraint(cstr1);
194                 }
195             }
196         }
197     }
198
199
200     protected void translateDC(Disjunction dis, boolean
            polarity)
201     throws Exception {
202         Formula p1 = dis.getLeftOperand();
203         Formula p2 = dis.getRightOperand();
204         if (polarity) {
205             translateDC(p1, true);
206             translateDC(p2, true);
207             for (int i=0; i<=k; i++) {
208                 for (int j=i; j<=k; j++) {
209                     ZOLCSConstraint cstr1 = new
                            ZOLCSConstraint();
210                     Literal np1p2 = literalHandler.
                            getLiteral(dis,i,j).flipSign();
211                     cstr1.addTerm(np1p2);
212                     Literal p1ij = literalHandler.
                            getLiteral(p1, i, j);
213                     cstr1.addTerm(p1ij);
214                     Literal p2ij = literalHandler.
                            getLiteral(p2, i, j);
215                     cstr1.addTerm(p2ij);
216                     addConstraint(cstr1);
217                 }
218             }
219         }
220         else { // !polarity
221             translateDC(p1, false);
222             translateDC(p2, false);
223             for (int i=0; i<=k; i++) {
224                 for (int j=i; j<=k; j++) {
225                     ZOLCSConstraint cstr1 = new
                            ZOLCSConstraint(2);
```

```
226                     Literal p1p2 = literalHandler.
                            getLiteral(dis,i,j);
227                     cstr1.addTerm(p1p2, 2);
228                     Literal np1ij = literalHandler.
                            getLiteral(p1, i, j).flipSign();
229                     cstr1.addTerm(np1ij);
230                     Literal np2ij = literalHandler.
                            getLiteral(p2, i, j).flipSign();
231                     cstr1.addTerm(np2ij);
232                     addConstraint(cstr1);
233                 }
234             }
235         }
236     }
237
238     protected void translateDC(Negation neg, boolean
            polarity)
239     throws Exception {
240         if (polarity) {
241             translateDC(neg.getOperand(), false);
242         }
243         else {
244             translateDC(neg.getOperand(), true);
245         }
246     }
247
248     protected void translateSA(Negation neg, boolean
            polarity)
249     throws Exception {
250         if (polarity) {
251             translateSA(neg.getOperand(), false);
252         }
253         else {
254             translateSA(neg.getOperand(), true);
255         }
256     }
257     protected void translateSA(Conjunction con, boolean
            polarity)
258     throws Exception {
259         Formula p1 = con.getLeftOperand();
260         Formula p2 = con.getRightOperand();
261         if (polarity) {
262             translateSA(p1, true);
263             translateSA(p2, true);
264             for (int i=0; i<=k-1; i++) {
265                 ZOLCSConstraint cstr1 = new
                        ZOLCSConstraint(2);
266                 Literal np1p2 = literalHandler.getLiteral
                        (con,i).flipSign();
267                 cstr1.addTerm(np1p2, 2);
```

```
268                 Literal p1i = literalHandler.getLiteral(
                        p1, i);
269                 cstr1.addTerm(p1i);
270                 Literal p2i = literalHandler.getLiteral(
                        p2, i);
271                 cstr1.addTerm(p2i);
272                 addConstraint(cstr1);
273             }
274         }
275         else {
276             translateSA(p1, false);
277             translateSA(p2, false);
278             for (int i=0; i<=k-1; i++) {
279                 ZOLCSConstraint cstr1 = new
                        ZOLCSConstraint();
280                 Literal p1p2 = literalHandler.getLiteral(
                        con,i);
281                 cstr1.addTerm(p1p2);
282                 Literal np1i = literalHandler.getLiteral(
                        p1, i).flipSign();
283                 cstr1.addTerm(np1i);
284                 Literal np2i = literalHandler.getLiteral(
                        p2, i).flipSign();
285                 cstr1.addTerm(np2i);
286                 addConstraint(cstr1);
287             }
288         }
289     }
290
291     protected void translateSA(Disjunction dis, boolean
            polarity)
292     throws Exception {
293         Formula p1 = dis.getLeftOperand();
294         Formula p2 = dis.getRightOperand();
295         if (polarity) {
296             translateSA(p1, true);
297             translateSA(p2, true);
298             for (int i=0; i<=k; i++) {
299                 ZOLCSConstraint cstr1 = new
                        ZOLCSConstraint();
300                 Literal np1p2 = literalHandler.getLiteral
                        (dis,i).flipSign();
301                 cstr1.addTerm(np1p2);
302                 Literal p1i = literalHandler.getLiteral(
                        p1, i);
303                 cstr1.addTerm(p1i);
304                 Literal p2i = literalHandler.getLiteral(
                        p2, i);
305                 cstr1.addTerm(p2i);
306                 addConstraint(cstr1);
```

```
307             }
308         }
309         else { // !polarity
310             translateSA(p1, false);
311             translateSA(p2, false);
312             for (int i=0; i<=k; i++) {
313                 ZOLCSConstraint cstr1 = new
                        ZOLCSConstraint(2);
314                 Literal p1p2 = literalHandler.getLiteral(
                        dis,i);
315                 cstr1.addTerm(p1p2, 2);
316                 Literal np1i = literalHandler.getLiteral(
                        p1, i).flipSign();
317                 cstr1.addTerm(np1i);
318                 Literal np2i = literalHandler.getLiteral(
                        p2, i).flipSign();
319                 cstr1.addTerm(np2i);
320                 addConstraint(cstr1);
321             }
322         }
323     }
324
325
326     protected void translateSA(Formula formula, boolean
            polarity)
327     throws Exception {
328         if (literalHandler.hasDef(formula, polarity)) {
329             // Do not output anything
330             return;
331         }
332         switch (formula.getType()) {
333             case Formula.TYPE_STATE: {
334                 // Do nothing
335                 break;
336             }
337             case Formula.TYPE_TRUE:
338                 // Do nothing
339                 break;
340             case Formula.TYPE_FALSE:
341                 // Do nothing
342                 break;
343             case Formula.TYPE_NEGATION: {
344                 translateSA((Negation) formula, polarity)
                        ;
345                 break;
346             }
347             case Formula.TYPE_CONJUNCTION: {
348                 translateSA((Conjunction) formula,
                        polarity);
349                 break;
```

```
350             }
351             case Formula.TYPE_DISJUNCTION: {
352                 translateSA((Disjunction) formula,
                        polarity);
353                 break;
354             }
355             default:
356                 throw new IllegalStateException("Unknown
                        SA formula type");
357         }
358         literalHandler.addDef(formula, polarity);
359     }
360 }
```

I.8.3 DCTranslator.java

```
1  package bmc.trans;
2
3  import bmc.*;
4  import bmc.literal.*;
5  import bmc.util.*;
6  import bmc.formula.*;
7  import bmc.constraint.*;
8
9  public abstract class DCTranslator {
10
11     private double translationTime;
12     protected LiteralHandler literalHandler;
13     protected int k;
14     protected boolean polarityOpt;
15     protected int outputFormat;
16     protected ConstraintWriter constraintWriter;
17
18     public DCTranslator(LiteralHandler literalHandler,
           Settings settings, ConstraintWriter
           constraintWriter) {
19         this.literalHandler = literalHandler;
20         this.k = settings.getK();
21         this.polarityOpt = settings.getPolarityOpt();
22         this.outputFormat = settings.getOutputFormat();
23         this.constraintWriter = constraintWriter;
24     }
25
26     public abstract void translate(Formula formula)
           throws Exception;
27
28     protected void addConstraint(Constraint constraint)
           throws Exception {
29         if (!constraint.isTriviallyTrue()) {
30             constraintWriter.writeConstraint(constraint);
```

```
31          }
32          if (Main.debugConstraints) {
33              System.out.print("Constraint added: "+
                    constraint);
34              if (constraint.isTriviallyTrue()) {
35                  System.out.println("--> but was trivially
                        true and thus ignored");
36              }
37          }
38      }
39
40      public LiteralHandler getLiteralHandler() {
41          return literalHandler;
42      }
43
44      public double getTranslationTime() {
45          return translationTime;
46      }
47  }
```

I.9 Package `bmc.util`

I.9.1 `CNFTranslator.java`

```
1   package bmc.util;
2
3   import bmc.formula.*;
4
5   public class CNFTranslator {
6
7       public CNFTranslator() {
8       }
9
10      public Formula translate(Formula formula)
11      throws CloneNotSupportedException {
12          // Translate using first to negation normal form
                (NNF)
13          // and then to clause normal form (CNF).
14          formula = (Formula) formula.clone();
15          Formula nnfFormula = translateToNNF(formula, true
                );
16          return translateToCNF(nnfFormula);
17      }
18
19      /**
20       *  Translates a formula in NNF to CNF
21       */
22      protected Formula translateToCNF(Formula formula)
23      throws CloneNotSupportedException {
24          switch (formula.getType()) {
```

```
25              case Formula.TYPE_STATE:
26              case Formula.TYPE_TRUE:
27              case Formula.TYPE_FALSE:
28              case Formula.TYPE_NEGATION:
29                  return formula;
30              case Formula.TYPE_DURATION: {
31                  Duration dur = (Duration) formula;
32                  Formula sa = dur.getStateAssertion();
33                  Formula sa2 = translateToCNF(sa);
34                  if (sa != sa2) {
35                      return new Duration(sa2, dur.getBound
                            ());
36                  }
37                  else {
38                      return formula;
39                  }
40              }
41              case Formula.TYPE_CHOP:
42                  Chop chop = (Chop) formula;
43                  chop.setLeftOperand(translateToCNF(chop.
                        getLeftOperand()));
44                  chop.setRightOperand(translateToCNF(chop.
                        getRightOperand()));
45                  return formula;
46              case Formula.TYPE_CONJUNCTION:
47                  Conjunction con = (Conjunction) formula;
48                  con.setLeftOperand(translateToCNF(con.
                        getLeftOperand()));
49                  con.setRightOperand(translateToCNF(con.
                        getRightOperand()));
50                  return formula;
51              case Formula.TYPE_DISJUNCTION:
52                  Disjunction dis = (Disjunction) formula;
53                  Formula left = translateToCNF(dis.
                        getLeftOperand());
54                  Formula right = translateToCNF(dis.
                        getRightOperand());
55                  if (left.getType() == Formula.
                        TYPE_CONJUNCTION) {
56                      if (right.getType() == Formula.
                            TYPE_CONJUNCTION) {
57                          // (a and b) or (c and d) ->
58                          // (a or c) and (a or d) and (b
                                or c) and (c or d)
59                          Conjunction leftC = (Conjunction)
                                left;
60                          Conjunction rightC = (Conjunction
                                ) right;
61                          Formula a = leftC.getLeftOperand
                                ();
```

```
62                  Formula b = leftC.getRightOperand
                        ();
63                  Formula c = rightC.getLeftOperand
                        ();
64                  Formula d = rightC.
                        getRightOperand();
65                  // Make "a or c"
66                  Formula dis1;
67                  dis1 = new Disjunction((Formula)
                        a.clone(), (Formula) c.clone()
                        );
68                  // Recurse "a or c"
69                  Formula aorc = translateToCNF(
                        dis1);
70                  // Make "a or d"
71                  Disjunction dis2;
72                  dis2 = new Disjunction((Formula)
                        a.clone(), (Formula) d.clone()
                        );
73                  // Recuse "a or d"
74                  Formula aord = translateToCNF(
                        dis2);
75                  // Make "b or c"
76                  Disjunction dis3;
77                  dis3 = new Disjunction((Formula)
                        b.clone(), (Formula) c.clone()
                        );
78                  // Recuse "b or c"
79                  Formula borc = translateToCNF(
                        dis3);
80                  // Make "b or d"
81                  Disjunction dis4;
82                  dis4 = new Disjunction((Formula)
                        b.clone(),
83                  (Formula) d.clone());
84                  // Recuse "b or d"
85                  Formula bord = translateToCNF(
                        dis4);
86                  // Make "(a or c) and (a or d)"
87                  leftC.setLeftOperand(aorc);
88                  leftC.setRightOperand(aord);
89                  // Make "(b or c) and (b or d)"
90                  rightC.setLeftOperand(borc);
91                  rightC.setRightOperand(bord);
92                  // Make the whole formula
93                  return new Conjunction(leftC,
                        rightC);
94              }
95              else {
```

```
96                      // (a and b) or c -> (a or c) and
                            (b or c)
97                      // Make "a or c"
98                      Conjunction leftC = (Conjunction)
                            left;
99                      Formula a = leftC.getLeftOperand
                            ();
100                     Formula b = leftC.getRightOperand
                            ();
101                     Formula c = right;
102                     dis.setLeftOperand(a);
103                     dis.setRightOperand(c);
104                     // Recuse "a or c"
105                     Formula aorc = translateToCNF(dis
                            );
106                     // Make "b or c"
107                     Disjunction dis2;
108                     dis2 = new Disjunction(b,
109                     (Formula) c.clone());
110                     // Recuse "b or c"
111                     Formula borc = translateToCNF(
                            dis2);
112                     // Make the whole formula
113                     leftC.setLeftOperand(aorc);
114                     leftC.setRightOperand(borc);
115                     return leftC;
116                 }
117             }
118             else if (right.getType() == Formula.
                    TYPE_CONJUNCTION) {
119                 // a or (b and c) -> (a or b) and (a
                        or c)
120                 Conjunction rightC = (Conjunction)
                        right;
121                 Formula a = left;
122                 Formula b = rightC.getLeftOperand();
123                 Formula c = rightC.getRightOperand();
124                 // Make "a or b"
125                 dis.setLeftOperand(a);
126                 dis.setRightOperand(b);
127                 // Recuse "a or b"
128                 Formula aorb = translateToCNF(dis);
129                 // Make "a or c"
130                 Disjunction dis2;
131                 dis2 = new Disjunction((Formula) a.
                        clone(), c);
132                 // Recuse "a or c"
133                 Formula aorc = translateToCNF(dis2);
134                 // Make the whole formula
135                 rightC.setLeftOperand(aorb);
```

```
136                     rightC.setRightOperand(aorc);
137                     return rightC;
138                 }
139                 else {
140                     // Simply update with the translated
                           subformulas
141                     dis.setLeftOperand(left);
142                     dis.setRightOperand(right);
143                     return formula;
144                 }
145             default:
146                 throw new IllegalStateException("Unknown
                       formula type");
147         }
148     }
149
150     public Formula translateToNNF(Formula formula,
           boolean sign)
151     throws CloneNotSupportedException {
152         switch (formula.getType()) {
153             case Formula.TYPE_TRUE:
154                 if (sign) {
155                     return formula;
156                 }
157                 else {
158                     return new False();
159                 }
160             case Formula.TYPE_FALSE:
161                 if (sign) {
162                     return formula;
163                 }
164                 else {
165                     return new True();
166                 }
167             case Formula.TYPE_CONJUNCTION:
168                 Conjunction con = (Conjunction) formula;
169                 if (sign) {
170                     // Just recurse
171                     con.setLeftOperand(
172                     translateToNNF(con.getLeftOperand(),
                           true));
173                     con.setRightOperand(
174                     translateToNNF(con.getRightOperand(),
                           true));
175                     return con;
176                 }
177                 else {
178                     // Negate the conjunction:
179                     // ~(a and b) -> ~a or ~b
```

```
180                     Formula left = translateToNNF(con.
                            getLeftOperand(), false);
181                     Formula right = translateToNNF(con.
                            getRightOperand(), false);
182                     return new Disjunction(left, right);
183                 }
184             case Formula.TYPE_DISJUNCTION:
185                 Disjunction dis = (Disjunction) formula;
186                 if (sign) {
187                     // Just recurse
188                     dis.setLeftOperand(
189                     translateToNNF(dis.getLeftOperand(),
                            true));
190                     dis.setRightOperand(
191                     translateToNNF(dis.getRightOperand(),
                             true));
192                     return dis;
193                 }
194                 else {
195                     // Negate the disjunction:
196                     // ~(a or b) ->  ~a and ~b
197                     Formula dleft = translateToNNF(dis.
                            getLeftOperand(), false);
198                     Formula dright = translateToNNF(dis.
                            getRightOperand(), false);
199                     return new Conjunction(dleft, dright)
                            ;
200                 }
201             case Formula.TYPE_CHOP: {
202                 Chop chop = (Chop) formula;
203                 chop.setLeftOperand(
204                 translateToNNF(chop.getLeftOperand(),
                        true));
205                 chop.setRightOperand(
206                 translateToNNF(chop.getRightOperand(),
                        true));
207
208                 if (!sign) {
209                     return new Negation(chop);
210                 }
211                 return chop;
212             }
213             case Formula.TYPE_NEGATION:
214                 // Change sign
215                 Negation neg = (Negation) formula;
216                 return translateToNNF(neg.getOperand(), !
                        sign);
217             case Formula.TYPE_DURATION:
218                 Duration dur = (Duration) formula;
219                 Formula sa = dur.getStateAssertion();
```

```
220                 Formula sa2 = translateToNNF(sa, true);
221                 if (sa != sa2) {
222                     formula = new Duration(sa2, dur.
                            getBound());
223                 }
224                 if (sign) {
225                     return formula;
226                 }
227                 else {
228                     return new Negation(formula);
229                 }
230             case Formula.TYPE_STATE:
231                 if (sign) {
232                     return formula;
233                 }
234                 else {
235                     return new Negation(formula);
236                 }
237             default:
238                 throw new IllegalStateException("Unknown
                        formula type");
239         }
240     }
241 }
```

I.9.2 CannotFindKException.java

```
1  package bmc.util;
2
3  import bmc.formula.*;
4  public class CannotFindKException extends java.lang.
       Exception {
5
6      public CannotFindKException(Formula formula) {
7          super("Cannot find k from formula: "+formula);
8      }
9  }
```

I.9.3 Command.java

```
1  package bmc.util;
2
3  public class Command {
4
5      private String commandText;
6
7      public Command(String commandText) {
8          this.commandText = commandText;
9      }
10
```

```
11      public String getCommandText() {
12          return commandText;
13      }
14
15      public String toString() {
16          return commandText;
17      }
18  }
```

I.9.4 IntBag.java

```
1   package bmc.util;
2
3   // This example is from the book "Java in a Nutshell,
        Second Edition".
4   // Written by David Flanagan.  Copyright (c) 1997 O'
        Reilly & Associates.
5   // You may distribute this source code for non-commercial
         purposes only.
6   // You may study, modify, and use this example for any
        purpose, as long as
7   // this notice is retained.  Note that this example is
        provided "as is",
8   // WITHOUT WARRANTY of any kind either expressed or
        implied.
9
10  // Modified by Anne-Sofie Nielsen and Jacob Enslev, 2004
11
12  import java.io.*;
13
14  public class IntBag {
15      protected int[] nums = new int[8]; // An array to
            store the numbers.
16      protected int size = 0;  // Index of next unused
            element of nums[].
17
18      /** Return an element of the array */
19      public int get(int index) throws
            ArrayIndexOutOfBoundsException {
20          if (index >= size || index < 0) throw new
                ArrayIndexOutOfBoundsException(index);
21          else return nums[index];
22      }
23
24      /** Replaces one element of the array with another */
25      public void set(int index, int value) throws
            ArrayIndexOutOfBoundsException {
26          if (index >= size || index < 0) throw new
                ArrayIndexOutOfBoundsException(index);
27          else nums[index] = value;
```

```
28      }
29
30      /** Add an int to the array, growing the array if
            necessary */
31      public void add(int x) {
32          if (nums.length == size) resize(nums.length*2);
                // Grow array, if needed.
33          nums[size++] = x;
                // Store the int in it.
34      }
35
36      /** Removes the element in the array, and swaps the
            last element of
37       *  the array into the empty slot */
38      public void remove(int index) throws
            ArrayIndexOutOfBoundsException {
39          if (index >= size || index < 0) throw new
                ArrayIndexOutOfBoundsException(index);
40          else {
41              nums[index] = nums[--size];
42          }
43      }
44
45      public int size() {
46          return size;
47      }
48
49      /** An internal method to change the allocated size
            of the array */
50      private void resize(int newsize) {
51          int[] oldnums = nums;
52          nums = new int[newsize];                        //
                Create a new array.
53          System.arraycopy(oldnums, 0, nums, 0, size); //
                Copy array elements.
54      }
55
56      public String toString() {
57          String result = "[";
58          for (int i=0 ; i<size ;i++) {
59              result += nums[i];
60              if (i < size -1 ) {
61                  result += ",";
62              }
63          }
64          return result+"]";
65      }
66  }
```

I.9.5 KValFinder.java

```
1  package bmc.util;
2
3  import bmc.formula.*;
4  import bmc.util.*;
5
6  public class KValFinder {
7      public KValFinder() {}
8
9      public int findK(Formula formula) throws
           CannotFindKException {
10         KValInfo info = findK2(formula);
11         if (info.reqGuard) {
12             throw new CannotFindKException(formula);
13         }
14         return info.k;
15     }
16
17     private KValInfo findK2(Formula formula) {
18         switch (formula.getType()) {
19             case Formula.TYPE_TRUE:
20             case Formula.TYPE_FALSE:
21                 return new KValInfo(0,false,false);
22             case Formula.TYPE_DURATION: {
23                 Duration dur = (Duration) formula;
24                 return new KValInfo(dur.getBound(),false,
                       false);
25             }
26             case Formula.TYPE_NEGATION: {
27                 Negation neg = (Negation) formula;
28                 Formula op = neg.getOperand();
29                 if (op.getType() == Formula.TYPE_DURATION
                       ){
30                     Duration dur = (Duration) op;
31                     boolean sIsTrue = dur.
                           getStateAssertion().getType() ==
                           Formula.TYPE_TRUE;
32                     return new KValInfo(dur.getBound()-1,
                           false,sIsTrue);
33                 }
34                 else if (op.getType() == Formula.
                       TYPE_CHOP) {
35                     return new KValInfo(0, true, false);
36                 }
37                 else {
38                     throw new IllegalStateException("
                           Formula is expected to be in NNF
                           ["+formula+"]");
39                 }
```

```
40              }
41              case Formula.TYPE_CONJUNCTION: {
42                  Conjunction con = (Conjunction) formula;
43                  KValInfo info1 = findK2(con.
                        getLeftOperand());
44                  KValInfo info2 = findK2(con.
                        getRightOperand());
45                  if (info1.isGuard && info2.isGuard) {
46                      return new KValInfo(Math.min(info1.k,
                            info2.k), false, true);
47                  }
48                  else if (info1.isGuard) {
49                      return new KValInfo(info1.k, false,
                            true);
50                  }
51                  else if (info2.isGuard) {
52                      return new KValInfo(info2.k, false,
                            true);
53                  }
54                  else {
55                      return new KValInfo(info1.k + info2.k
                            ,
56                      info1.reqGuard || info2.reqGuard,
                            false);
57                  }
58              }
59              case Formula.TYPE_DISJUNCTION: {
60                  Disjunction dis = (Disjunction) formula;
61                  KValInfo info1 = findK2(dis.
                        getLeftOperand());
62                  KValInfo info2 = findK2(dis.
                        getRightOperand());
63                  return new KValInfo(Math.max(info1.k,
                        info2.k),
64                  info1.reqGuard || info2.reqGuard,
65                  info1.isGuard && info2.isGuard);
66              }
67              case Formula.TYPE_CHOP: {
68                  Chop chop = (Chop) formula;
69                  KValInfo info1 = findK2(chop.
                        getLeftOperand());
70                  KValInfo info2 = findK2(chop.
                        getRightOperand());
71                  return new KValInfo(info1.k + info2.k,
72                  info1.reqGuard || info2.reqGuard,
73                  false);
74              }
75              default:
76                  throw new IllegalStateException("Unknown
                        formula type");
```

```
77          }
78
79      }
80
81      private class KValInfo {
82          public int k;
83          public boolean isGuard, reqGuard;
84
85          public KValInfo(int k, boolean reqGuard, boolean
                isGuard) {
86              this.k = k;
87              this.reqGuard = reqGuard;
88              this.isGuard = isGuard;
89          }
90      }
91  }
```

I.9.6 Settings.java

```
1   package bmc.util;
2
3   import java.io.*;
4   import bmc.*;
5
6
7   public class Settings implements Cloneable {
8
9       private boolean polarityOpt;
10      private boolean nnf;
11      private boolean findK;
12      private int dcSimpLevel;
13      private int outputFormat;
14      private int fRecognition;
15      private String outputFolder;
16
17      public static final int OUTPUT_FORMAT_ZOLCS = 1;
18      public static final int OUTPUT_FORMAT_DIMACS = 2;
19
20      public static final int F_RECOGNITION_ID = 1;
21      public static final int F_RECOGNITION_SYNTACTIC = 2;
22      public static final int F_RECOGNITION_SEMANTIC = 3;
23
24      public static final int DC_SIMP_LEVEL_NONE = 0;
25      public static final int DC_SIMP_LEVEL_BASIC = 1;
26      public static final int DC_SIMP_LEVEL_WITH_BDD = 2;
27
28      private int k = 0;
29
30      public Settings() {
31          setStdValues();
```

```
32      }
33
34      public boolean getPolarityOpt() {
35          return polarityOpt;
36      }
37
38      public boolean getNNF() {
39          if (findK) {
40              return true;
41          }
42          return nnf;
43      }
44
45      public int getDCSimpLevel() {
46          if (fRecognition == F_RECOGNITION_SEMANTIC) {
47              return DC_SIMP_LEVEL_WITH_BDD;
48          }
49          return dcSimpLevel;
50      }
51
52      public int getK() {
53          return k;
54      }
55
56      public boolean getFindK() {
57          return findK;
58      }
59
60      public int getOutputFormat() {
61          return outputFormat;
62      }
63
64      public String getOutputFolder() {
65          return outputFolder;
66      }
67
68      public int getFRecognition() {
69          return fRecognition;
70      }
71
72      public void setPolarityOpt(boolean polarityOpt) {
73          this.polarityOpt = polarityOpt;
74      }
75
76      public void setNNF(boolean nnf) {
77          this.nnf = nnf;
78      }
79
80
81      public void setDCSimpLevel(int dcSimpLevel) {
```

```
82          if (dcSimpLevel < DC_SIMP_LEVEL_NONE ||
                dcSimpLevel > DC_SIMP_LEVEL_WITH_BDD) {
83              throw new IllegalArgumentException("
                    DCsimpLevel not recognized");
84          }
85          this.dcSimpLevel = dcSimpLevel;
86      }
87
88      public void setFindK(boolean findK) {
89          this.findK = findK;
90      }
91
92      public void setOutputFolder(String outputFolder) {
93          this.outputFolder = outputFolder;
94      }
95
96      public void setOutputFormat(int outputFormat) {
97          if (outputFormat < OUTPUT_FORMAT_ZOLCS ||
                outputFormat > OUTPUT_FORMAT_DIMACS) {
98              throw new IllegalArgumentException("
                    outputFormat not recognized");
99          }
100         this.outputFormat = outputFormat;
101     }
102
103     public void setFRecognition(int fRecognition) {
104         if (fRecognition < F_RECOGNITION_ID ||
                fRecognition > F_RECOGNITION_SEMANTIC) {
105             throw new IllegalArgumentException("
                    fRecognition not recognized");
106         }
107         this.fRecognition = fRecognition;
108     }
109
110     public void setK(int k) {
111         if (k < 0) {
112             throw new IllegalArgumentException("k must be
                        greater than or equal to zero");
113         }
114         this.k = k;
115     }
116
117     public String toString() {
118         return "k="+k+"\n"+
119         "polarityOpt="+polarityOpt+"\n"+
120         "dcSimpLevel="+dcSimpLevel+"\n"+
121         "outputFormat="+outputFormat+"\n"+
122         "fRecognition="+fRecognition+"\n"+
123         "findK="+findK+"\n"+
```

```
124         "outputFolder="+( outputFolder.equals("") ? "(
                current folder)" : outputFolder)+"\n";
125     }
126
127     public void setStdValues() {
128         this.setK(1);
129         this.setPolarityOpt(true);
130         this.setFindK(false);
131         this.setNNF(false);
132         this.setOutputFolder("");
133         this.setDCSimpLevel(DC_SIMP_LEVEL_BASIC);
134         this.setOutputFormat(OUTPUT_FORMAT_ZOLCS);
135         this.setFRecognition(F_RECOGNITION_SYNTACTIC);
136     }
137
138     public Object clone() throws
            CloneNotSupportedException {
139         return super.clone();
140     }
141
142 }
```

I.9.7 ShellExecutor.java

```
1  package bmc.util;
2
3  import java.io.*;
4  import java.util.*;
5  import bmc.*;
6
7  public class ShellExecutor {
8
9      protected String[] cmd;
10
11     public ShellExecutor() {
12         this(null);
13     }
14
15     public ShellExecutor(String[] cmd) {
16         if (cmd != null) {
17             this.cmd = new String[cmd.length+1];
18             System.arraycopy(cmd, 0, this.cmd, 0, cmd.
                   length);
19         }
20         else {
21             this.cmd = new String[3];
22             String osName = System.getProperty("os.name")
                   ;
23             if (osName.equals("Linux") || osName.equals("
                   SunOS")) {
```

```
24                  this.cmd[0] = "/bin/sh" ;
25                  this.cmd[1] = "-c" ;
26              }
27              else if(osName.equals("Windows 95")) {
28                  this.cmd[0] = "command.com" ;
29                  this.cmd[1] = "/C" ;
30              }
31              else if(osName.indexOf("Win") >= 0 ) {
32                  this.cmd[0] = "cmd.exe" ;
33                  this.cmd[1] = "/C" ;
34              }
35              else
36                  throw new IllegalStateException("
                        Unsupported OS: "+osName);
37          }
38      }
39
40
41      public Process execCmd(String commandText) throws
            Exception {
42          cmd[cmd.length-1] = commandText;
43          Runtime rt = Runtime.getRuntime();
44          if (Main.debugExec) {
45              System.out.print("Execing ");
46              for (int i=0 ; i<cmd.length ; i++) {
47                  System.out.print(cmd[i] + " ");
48              }
49              System.out.println();
50          }
51          return rt.exec(cmd);
52      }
53
54
55      public int execCmdAndFinish(String commandText)
            throws Exception {
56          Process proc = execCmd(commandText);
57
58          // any error message?
59          StreamGobbler errorGobbler = new
60          StreamGobbler(proc.getErrorStream());
61
62          // any output?
63          StreamGobbler outputGobbler = new
64          StreamGobbler(proc.getInputStream());
65
66          // kick them off
67          new Thread(errorGobbler).start();
68          new Thread(outputGobbler).start();
69
70          int exitVal = proc.waitFor();
```

```
71          proc.destroy();
72          return exitVal;
73      }
74  }
```

I.9.8 StreamGobbler.java

```
1   package bmc.util;
2
3   import java.io.*;
4
5   public class StreamGobbler implements Runnable {
6
7       private InputStream is;
8
9       public StreamGobbler(InputStream is) {
10          this.is = is;
11      }
12
13      public void run() {
14          try {
15              while (!Thread.interrupted()) {
16                  if (is.available() > 0) {
17                      is.read();
18                  }
19                  else {
20                      Thread.sleep(100);
21                  }
22              }
23          }
24          catch (Exception ignored) {
25          }
26      }
27  }
```

I.10 Package bmc

I.10.1 Main.java

```
1   package bmc;
2
3   import java.io.*;
4   import java.text.*;
5   import java.util.*;
6   import java.util.regex.*;
7
8   import bmc.parser.*;
9   import bmc.tracer.*;
10  import bmc.simp.*;
11  import bmc.util.*;
```

```
12 import bmc.literal.*;
13 import bmc.tracer.*;
14 import bmc.formula.*;
15 import bmc.trans.*;
16 import bmc.constraint.*;
17
18 public class Main {
19
20     private ShellExecutor shellExec;
21     private String solverCmd;
22     private KValFinder kValFinder;
23
24     private int displayMode;
25
26
27     public static boolean debugLiterals = false;
28     public static boolean debugGeneral = true;
29     public static boolean debugSimplify = true;
30     public static boolean debugExec = false;
31     public static boolean debugOccurrence = false;
32     public static boolean debugConstraints = false;
33     public static boolean resultPrintOut = true;
34
35
36     public Main(String[] shellCmd, String solverCmd, int
           displayMode) {
37         shellExec = new ShellExecutor(shellCmd);
38         kValFinder = new KValFinder();
39         this.solverCmd = solverCmd;
40         this.displayMode = displayMode;
41     }
42
43     public int runGoal(Goal goal) throws IOException {
44         long startTime = System.currentTimeMillis();
45
46         DCTranslator dcTranslator;
47         LiteralHandler literalHandler;
48         ConstraintWriter constraintWriter;
49         RandomAccessFile output = null;
50         Thread errStreamGobbler = null;
51
52         try {
53
54             Settings settings = goal.getSettings();
55             String filename = goal.getName();
56
57             Formula formula = goal.getFormula();
58             if (settings.getNNF()) {
59                 CNFTranslator cnfTranslator = new
                       CNFTranslator();
```

```
60                  formula = cnfTranslator.translateToNNF(
                        formula, false);
61              }
62              else if (formula.getType() == Formula.
                    TYPE_NEGATION) {
63                  formula = ((Negation) formula).getOperand
                        ();
64              }
65              else {
66                  formula = new Negation(formula);
67              }
68
69
70              if (debugGeneral) {
71                  System.out.println("**** DCformula: "+
                        goal.getFormula());
72              }
73
74              if (settings.getFindK()) {
75                  settings.setK(Integer.MAX_VALUE);
76              }
77
78              DCSimplifier dcSimplifier = new DCSimplifier(
                    settings);
79              formula = dcSimplifier.simplifyDC(formula);
80
81              if (debugSimplify) {
82                  System.out.println("**** Simplified,
                        Negated DCformula: "+formula);
83              }
84
85              if (settings.getFindK()) {
86                  int foundKVal = kValFinder.findK(formula)
                        ;
87                  settings.setK(foundKVal);
88                  if (debugGeneral) {
89                      System.out.println("**** Found kval
                            ="+foundKVal);
90                  }
91              }
92
93              File outputFolder = new File(settings.
                    getOutputFolder());
94              switch (settings.getOutputFormat()) {
95                  case Settings.OUTPUT_FORMAT_DIMACS:
96                      filename += ".dimacs";
97                      break;
98                  case Settings.OUTPUT_FORMAT_ZOLCS:
99                      filename += ".zolcs";
100                     break;
```

```
101                 default:
102                     throw new IllegalArgumentException("
                            Unknown output format");
103             }
104             File outputFile = new File(outputFolder.
                    getCanonicalPath(), filename);
105             output = new RandomAccessFile(outputFile, "rw
                    ");
106
107             switch (settings.getFRecognition()) {
108                 case Settings.F_RECOGNITION_ID:
109                     literalHandler = new
                            IDLookupLiteralHandler();
110                     break;
111                 case Settings.F_RECOGNITION_SYNTACTIC:
112                     literalHandler = new
                            SyntacticLookupLiteralHandler();
113                     break;
114                 case Settings.F_RECOGNITION_SEMANTIC:
115                     literalHandler = new
                            SemanticLookupLiteralHandler();
116                     break;
117                 default:
118                     throw new IllegalArgumentException("
                            Unknown literal handler");
119             }
120             constraintWriter = new ConstraintWriter(
                    literalHandler, formula, settings, output)
                    ;
121
122             switch (settings.getOutputFormat()) {
123                 case Settings.OUTPUT_FORMAT_DIMACS:
124                     dcTranslator = new
                            DCToDIMACSTranslator(
                            literalHandler,
125                     settings, constraintWriter);
126                     break;
127                 case Settings.OUTPUT_FORMAT_ZOLCS:
128                     dcTranslator = new
                            DCToZOLCSTranslator(literalHandler
                            ,
129                     settings, constraintWriter);
130                     break;
131                 default:
132                     throw new IllegalArgumentException("
                            Unknown output format");
133             }
134
135             dcTranslator.translate(formula);
136             constraintWriter.finish();
```

```
137                 double frontEndTime = ((double)(System.
                        currentTimeMillis() - startTime))/1000;
138                 output.close();
139
140                 // Run solver
141                 StringBuffer myCmd = new StringBuffer(
                        solverCmd);
142                 if (settings.getOutputFormat() == Settings.
                        OUTPUT_FORMAT_DIMACS) {
143                     myCmd.append(" --dimacs ");
144                 }
145                 else // zolcs
146                     myCmd.append(" --zolcs ");
147
148                 myCmd.append("\"");
149                 if (settings.getOutputFolder().length() != 0)
                        {
150                     myCmd.append(settings.getOutputFolder()).
                            append("/");
151                 }
152                 myCmd.append(filename).append("\"");
153                 Process proc = shellExec.execCmd(myCmd.
                        toString());
154                 Reader hysatOutput = new InputStreamReader(
                        proc.getInputStream());
155                 // Read error stream
156                 errStreamGobbler = new Thread(new
                        StreamGobbler(proc.getErrorStream()));
157                 errStreamGobbler.start();
158                 // Read solver's response
159                 boolean displayTrace = displayMode !=
                        StateTraceFrame.DISPLAY_MODE_NONE;
160                 HySatSolutionParser hySatParser = new
                        HySatSolutionParser(
161                 hysatOutput,
162                 literalHandler, goal, displayTrace);
163                 hySatParser.parse();
164                 errStreamGobbler.interrupt();
165                 proc.destroy();
166                 int status = hySatParser.getStatus();
167                 if (resultPrintOut) {
168                     System.out.println();
169                     System.out.println
                            ("################################################")
                            ;
170                     if (status == HySatSolutionParser.
                            STATUS_SAT) {
171                         System.out.println("Formula is
                                invalid");
172                     }
```

```
173                 else if (status == HySatSolutionParser.
                        STATUS_UNSAT) {
174                     System.out.println("Formula is valid
                            ");
175                 }
176                 else {
177                     System.out.println("Error occurred
                            during translation");
178                 }
179                 System.out.println();
180                 printTimes(frontEndTime,
181                 hySatParser.getCPUTime());
182                 System.out.println();
183                 if (displayTrace && status ==
                        HySatSolutionParser.STATUS_SAT) {
184                     StateTraceFrame.display(settings,
                            hySatParser.getStateTraceSet());
185                 }
186                 System.out.println
                        ("################################################")
                        ;
187             }
188             return status;
189         }
190         catch (CannotFindKException e) {
191             if (output!=null) {
192                 output.close();
193             }
194             System.out.println("Error: "+e.getMessage());
195             return HySatSolutionParser.STATUS_ERROR;
196         }
197         catch (java.text.ParseException e) {
198             if (output!=null) {
199                 output.close();
200             }
201             System.out.println("Syntax error: "+e.
                    getMessage() + " at line "+(e.
                    getErrorOffset()+1));
202             return HySatSolutionParser.STATUS_ERROR;
203         }
204         catch (Exception ex) {
205             if (output!=null) {
206                 output.close();
207             }
208             System.out.println(ex);
209             ex.printStackTrace(System.out);
210             return HySatSolutionParser.STATUS_ERROR;
211         }
212         catch (Error e) {
213             if (output!=null) {
```

```
214                 output.close();
215             }
216             System.out.println(e);
217             e.printStackTrace(System.out);
218             return HySatSolutionParser.STATUS_ERROR;
219         }
220         finally {
221             if (errStreamGobbler != null) {
222                 errStreamGobbler.interrupt();
223             }
224         }
225     }
226
227     public List parse(Reader input) throws Exception {
228         BMCParser parser = new BMCParser();
229         BMCPreprocessor preproc = new BMCPreprocessor();
230         preproc.setScanner(new BMCLex(input));
231         String result = preproc.preprocess();
232         input.close();
233         StringReader input2 = new StringReader(result);
234         parser.setScanner(new BMCLex(input2));
235         return  parser.parseTasks();
236     }
237
238     public void run(Reader input) throws Exception {
239         List tasks = parse(input);
240         Iterator it = tasks.iterator();
241         while (it.hasNext()) {
242             Object task = it.next();
243             if (task instanceof Goal) {
244                 runGoal((Goal) task);
245             }
246             else if (task instanceof Command) {
247                 Command command = (Command) task;
248                 shellExec.execCmdAndFinish(command.
                        getCommandText());
249             }
250         }
251     }
252
253     public static void printUsage() {
254         System.out.println("Usage: bmc.Main [-shellCmd
                CMD] [-displayTrace] -solverCmd CMD INPUT_FILE
                ");
255     }
256
257     public static void main(String[] args) throws
            Exception {
258         String[] shellCmd = null;
259         String solverCmd = null;
```

```
260         String inputFileName = null;
261         int displayMode = StateTraceFrame.
                DISPLAY_MODE_NONE;
262         Pattern pattern = Pattern.compile(" ");
263         for (int i=0; i<args.length; i++) {
264             if (args[i].equals("-shellCmd")) {
265                 if ((++i) == args.length) {
266                     printUsage();
267                     return;
268                 }
269                 shellCmd = pattern.split(args[i]);
270             }
271             else if (args[i].equals("-solverCmd")) {
272                 if ((++i) == args.length) {
273                     printUsage();
274                     return;
275                 }
276                 solverCmd = args[i];
277             }
278             else if (args[i].equals("-displayTrace")) {
279                 displayMode = StateTraceFrame.
                        DISPLAY_MODE_GUI;
280             }
281             else {
282                 if (inputFileName != null) {
283                     printUsage();
284                     return;
285                 }
286                 inputFileName = args[i];
287             }
288         }
289         if (inputFileName == null || solverCmd == null) {
290             printUsage();
291             return;
292         }
293 
294         BufferedReader br = new BufferedReader(new
                FileReader(inputFileName));
295         Main main = new Main(shellCmd, solverCmd,
                displayMode);
296         main.run(br);
297         br.close();
298     }
299 
300     public String getSolverCmd() {
301         return solverCmd;
302     }
303 
304     public static void setDebugging(boolean value) {
305         debugExec =
```

```
306             debugGeneral =
307             debugLiterals =
308             debugSimplify =
309             debugOccurrence =
310             debugConstraints = value;
311         }
312
313         public static void setPrintOut(boolean value) {
314             resultPrintOut = value;
315         }
316
317         public static void printTimes(double frontendTime,
                double backendTime) {
318             int DEC_COUNT = 3;
319             frontendTime = ((double) Math.round(frontendTime*
                    Math.pow(10,DEC_COUNT)))/Math.pow(10,DEC_COUNT
                    );
320             backendTime = ((double) Math.round(backendTime*
                    Math.pow(10,DEC_COUNT)))/Math.pow(10,DEC_COUNT
                    );
321             double validationTime = ((double) Math.round((
                    frontendTime + backendTime)*Math.pow(10,
                    DEC_COUNT)))/Math.pow(10,DEC_COUNT);
322             String frontendStr = Double.toString(frontendTime
                    );
323             String backendStr = Double.toString(backendTime);
324             String valStr = Double.toString(validationTime);
325             int dotFrontIdx = frontendStr.indexOf(".");
326             int dotBackIdx = backendStr.indexOf(".");
327             int dotValIdx = valStr.indexOf(".");
328             for (int i=0; i<dotValIdx-dotBackIdx; i++) {
329                 backendStr = " "+backendStr;
330             }
331             for (int i=0; i<dotValIdx-dotFrontIdx; i++) {
332                 frontendStr = " "+frontendStr;
333             }
334             int backDecimals = backendStr.length()-dotValIdx
                    -1;
335             int frontDecimals = frontendStr.length()-
                    dotValIdx-1;
336             int valDecimals = valStr.length()-dotValIdx-1;
337             for (int i=0; i<DEC_COUNT-frontDecimals; i++) {
338                 frontendStr = frontendStr + "0";
339             }
340             for (int i=0; i<DEC_COUNT-backDecimals; i++) {
341                 backendStr = backendStr + "0";
342             }
343             for (int i=0; i<DEC_COUNT-valDecimals; i++) {
344                 valStr = valStr + "0";
345             }
```

```
346             String frontendLine = "Frontend time: "+
                    frontendStr+" s";
347             System.out.println(frontendLine);
348             System.out.println("Backend time : "+backendStr+
                    " s");
349             StringBuffer tmp = new StringBuffer();
350             for (int i=0 ; i<frontendLine.length() ; i++) {
351                 tmp.append("-");
352             }
353             System.out.println(tmp.toString());
354             System.out.println("Total time    : "+valStr+ " s
                    ");
355             System.out.println(tmp.toString());
356         }
357 }
```