# Static Analysis of Stochastic Process Algebras

Fan Yang

Kongens Lyngby 2007
IMM-MSC-2007-9

# Summary

The Performance Evaluation Process Algebra, PEPA, is introduced by Jane Hillston as a stochastic process algebra for modelling distributed systems and especially suitable for performance evaluation. A range of tools has already been developed that apply this algebra to various application areas for different purposes.

In this thesis, we present a static analysis more precisely approximating the *control structure* of processes expressed in PEPA. The analysis technique we adopted is Data Flow Analysis which is often associated with the efficient implementation of classical imperative programming languages. We begin the analysis by defining an appropriate transfer function, then with the classical worklist algorithm we construct a *finite automaton* that captures *all* possible interactions among processes. With the help of the novel methodology of annotating *label* and *layer* to the PEPA program, the approximating result is very precise.

Later we try to accelerate the analysis by two approaches, and develop algorithms for validating the deadlock property of the PEPA program. In addition, the thesis comes out with a tool that fully implements the analyses and it could be used to verify the deadlock property of the PEPA programs in a certain scale.

**Keywords:** PEPA, Date Flow Analysis, Control Structure, Finite Automaton, Deadlock, Static Analysis, Stochastic Process Algebra.

# Preface

This thesis is the result of my work at the Informatics and Mathematical Modelling department of Technical University of Denmark, for obtaining a M.Sc degree of Computer System Engineering. It corresponds to 30 ECTs points and is carried out in the period through 1$^{st}$ August 2006 to 31$^{st}$ January 2007, under the supervision of Professor Hanne Riis Nielson.

Lyngby, January 2007

Fan Yang

# Acknowledgements

First of all, I would like to thank Hanne Riis Nielson, my supervisor, for her excellent guidance through out the whole project. I could always get a lot of inspiration from talking with her, which makes the project going pretty smooth and efficient.

Then I would like to thank our project partner Stephen Gilmore. He provides me a series of excellent jobshop examples which show preciousness at the last stage of the project. I would also like to thank Raghav Karol, who gave me very useful suggestions on Latex documentation and always try to share his knowledge with me. Then I would like to thank the people at the Language Based Technology group who make me a pleasant stay when I work on the thesis.

Lastly, I would like to give special thanks to my beloved girlfriend Ziyan Feng, for her emotional support and proofreading my thesis several times. Of course, I always feel very grateful for the support from my parents, and thank them for trying to make my life easier when I study abroad.

# Contents

Chapter 1

# Introduction

In computer science, the *process calculi* (or process algebras) are a diverse family of related approaches to formally modelling concurrent systems. They provide a tool for the high-level description of interactions, communications, and synchronizations between a collection of independent processes [35]. The most famous ones include: Communicating Sequential Processes(CSP)[24] ,Calculus of Communicating Systems (CCS) [26] and Algebra of Communicating Processes(ACP)[9].

Among them there is a branch of process algebras extended with probabilistic or stochastic information, which are usually named *stochastic process algebras* (SPAs). For example, in CSP tradition, there is Timed CSP [20], in CCS tradition, there is PEPA [22], and $pr\mathrm{ACP}_I^-$ [8] is in ACP tradition. The SPAs have gained acceptance as one of the techniques available for performance analysis. For example, in large computer and communication systems, they could be used to model the system and predict the behavior of a system with respect to dynamic properties such as throughput and response time[23].

However, the system modelling with SPAs always inherits the process algebras's concurrent essentials and behaves in a complex way. When dynamically executing the system, we sometimes need to be ensured that there *must not* arise any abnormal event. For example, the system shouldn't terminate unexpectedly(no deadlock). Furthermore, sometimes even though we are notified that the system might go into the deadlock states, we are not satisfied. We are also curious in which steps might we reach those states, because this information would be very useful for understanding the cause of the abnormity and then help people to revise the system to avoid deadlock.

In this thesis, we are going to develop a static analysis for one kind of SPAs: Performance Evaluation Process Algebra (PEPA) [22], aiming at answering the following two questions for the system modelled with PEPA:

- Does the system potentially have chance to go into the deadlock states?
- If there exist deadlock states, how does the system behave before reaching those states?

In the following subsections, first we will introduce the theoretical background of our work. Then we give a short description of the real work we accomplished. In the last of this chapter we will outline the structure of the thesis.

## 1.1 Theoretical Background

### 1.1.1 Related Works

In recently years, there are several methods of applying static analysis techniques to highly concurrent languages and a variety of process algebras. For process algebras, the works are mainly based on three approaches:

- One line of work is to adapt Type Systems from the functional and object-oriented languages to express meaningful properties of the process algebras(e.g. [27, 34]).
- Another line of work is based on Control Flow Analysis. The process algebras that have been extensively researched are: pi-calculus [10], variants of mobile ambients(e.g. [12, 29]) and process algebras for cryptographic protocols such as Lysa [11].
- The last line of work just emerges recently, and it uses the classical approach – Data Flow Analysis to focus on analyzing *transitions* instead of *configurations* of the models (Configurations are the main concern of the previous two approaches). The process algebras that has already been done includes CCS [15, 16].

Our work adopts the last approach, because its special feature of transition tracking would help us easily answer not only the first question (verify deadlock property) but also the second question mentioned above (find paths leading to deadlock states). And the work in [15, 16] inspires our own work quite a lot.

### 1.1.2 Analysis Techniques

Our work is based on the category of *program analysis*, in particular the *Data Flow Analysis*, which will be introduced briefly as follows.

*Program analysis* offers static compile-time techniques for predicting *safe* and computable *approximations* to the set of values or behaviors arising dynamically at run-time when executing a program on a computer[28].

The *safe* here means that analysis is based on formal semantics (Our job is based on the semantics of PEPA). The *approximations* are usually divided into three classes:(a)Over-approximation captures the entire behavior of the program. (b)Under-approximation captures a subset of all possible behavior of the program in reality.(c)Undecidable-approximation can't decide whether the the approximation behaviors belong to the program or not. In our work, we will use both (a) and (b) approaches for approximation.

The *Data Flow Analysis* is among four classical program analysis approaches, which are Data Flow Analysis, Constraint Based Analysis, Abstract Interpretation and Type and Effect Systems. In Data Flow Analysis, it is customary to think of a program (written in traditional programming language) as a graph: the nodes are basic blocks and the edges describe how control might pass from one basic block to another. The transfer functions associated with basic blocks are often specified as Bitvector Frameworks or more general as Monotone Frameworks. The transfer functions in Bitvector Frameworks will always remove information no longer appropriate, and at the same time generate appropriate information to the basic blocks which will form new basic block.

## 1.2 Our work

Our work is based on the Data Flow Analysis. We build a graph (actually we generate an automaton from the program while the graph is the graphical representation of the automaton) for the program written in PEPA, which could capture the control structure of the program. And this automaton could be used to verify deadlock property and illustrate paths leading to deadlock states.

Concretely, we first specify the *kill* and *gen* function based on the semantics of the PEPA and make our own transfer function. For the safe approximation, we perform under-approximation for the kill function and the over-approximation for the gen function: it is always safe to kill less information and gen more information than the exact information. We shall see that the *labels* and *layers* of individual actions(labels and layers are annotation to the PEPA program while action is a term in PEPA, please refer to chapter 2 in details) will correspond to the basic blocks of Data Flow analysis and there is a need to introduce two domains : *extended multisets* and *extra extended multisets* to replace the Bitvectors for these functions.

Later we will use worklist algorithm to construct a finite control flow graph (automaton) for PEPA process: the nodes describe the *exposed actions*(will be introduced in chapter 3) for the various configuration (state) that may arise dynamically during the execution; the edges describe the interactions of actions(transitions) that tell how one configuration evolves to another configuration. For the termination of the algorithm, we adopt a suitable granularity function and the widening operator.

Lastly, to improve the analysis efficiency, we develop two methods which could significantly speed up the analysis when some information is ignored.

After developing these analyses, we utilize them first to verify several small programs and then study the deadlock property of a series of larger programs: variants of Milner's process for a jobshop [31].

## 1.3 Thesis Organization

Chapter 2 introduces the syntax of PEPA as well as its semantics. In addition, layers and labels are equipped to PEPA.

Chapter 3 first introduces the concept of exposed actions for PEPA program and two domains: *extend multiset* and *extra extend multiset*. Later two functions $\mathcal{E}^s_\star$ and $\mathcal{E}^p_\star$ are developed to compute the information for initialization worklist algorithm.

Chapter 4 describes the development of function kill($\mathcal{K}^s_\star, \mathcal{K}^p_\star$), gen($\mathcal{G}^s_\star, \mathcal{G}^p_\star$) and finally the `transfer` function which will be invoked in the worklist algorithm. The domain *extend multimap* and *extra extend multimap* are introduced.

Chapter 5 describes the constructing of the automaton by worklist algorithm. Some auxiliary functions are developed to cooperate with the worklist algorithm, such as `update`, `enabled`, $\mathcal{Y}_{\star}^{s}$, $\mathcal{Y}_{\star}^{p}$ etc.

Chapter 6 discusses two methods for increasing the analysis speed (constructing the automaton) on the condition that the structure of PEPA program meet some requirements.

Chapter 7 shows how to use the analysis developed in the previous chapters to verify the deadlock property of PEPA programs and how to find the paths leading to each deadlock state.

Chapter 8 concludes the thesis and points out some future work.

Chapter 2

# Performance Evaluation Process Algebra

In this chapter we shall first introduce the syntax of PEPA programs and then review the semantics as presented in [22]. Lastly we will equip PEPA with labels and layers that would be helpful for the analysis developed later.

## 2.1 Syntax

PEPA models are described as interaction of *components*. Each component itself contains a series of *activities* that give the behavior of concrete actions. Each activity, $a \in \mathcal{A}ct$, is defined as a pair $(\alpha, r)$ where $\alpha \in \mathcal{A}$ is the action type and $r \in \mathbb{R}^+$ is the activity rate that indicates the duration of this activity. And we know $\mathcal{A}ct \subseteq \mathcal{A} \times \mathbb{R}^+$.

PEPA also provides a set of *combinators* to build up complex behavior from simpler behavior, which means, the combinators could combine different simpler components together and thus influence the activities (represent the behavior) within them. There are five type of combinators in PEPA, namely *prefix*(.), *Choice*(+),*Cooperation*( $\underset{L}{\bowtie}$ ), *Hiding*(/) and *Constant*($\overset{def}{=}$).

**Prefix:** $(\alpha, r).S$**:** Prefix is the basic mechanism by which the behaviors of components are constructed. It means after the component has carried out activity $(\alpha, r)$, it will behave as component $S$.

**Choice:** $S_1 + S_2$**:** This combinator represents a system which may behave either as component $S_1$ or as $S_2$. $S_1 + S_2$ enables all the current activities of $S_1$ and all the current activities of $S_2$. The first activity to complete distinguishes one out of the two components, $S_1$ or $S_2$ and the other component of the choice is then discarded.

**Cooperation:** $P_1 \underset{L}{\bowtie} P_2$**:** This combinator describes the synchronization of the $P_1$ and $P_2$ over the activities in the *cooperation set* $L$. The components may proceed independently with activities whose types do not belong to this set. In contrast, the components should cooperate with the other components if the types of their activities (to be executed) both fall into this set.

**Hiding:** $P/L$**:** It behaves as $P$ except that any activities of types within the set $L$ are *hidden*, meaning that their type is not witnessed upon completion. A hidden activity is witnessed only by its delay and the unknown type $\tau$, and it can't be carried out in cooperation with any other component.

**Constant:** $C \overset{def}{=} S$ **or** $S \overset{def}{=} P$**:** Constant are components whose meaning is given by a defining equation such as $C \overset{def}{=} S$ or $S \overset{def}{=} P$ which gives the constant $C$ the behavior of the component $S$ or $S$ given the behavior of $P$ respectively. This is how we assign names to components (behaviors).

The syntax is formally defined by means of the following grammar.

$$\begin{aligned} S &::= (\alpha, r).S \mid S_1 + S_2 \mid C \\ P &::= P_1 \underset{L}{\bowtie} P_2 \mid P/L \mid S \end{aligned}$$

where $S$ denotes a *sequential component* and $P$ denotes a *model component*. A sequential component could be formed by *sequential combinators* that is either prefix, choice or constant $C$ where $C$ stands for sequential component. A model component could be formed by *model combinators* that is either cooperation, hiding or constant $S$ where $S$ stands for sequential component or model component.

We shall be interested in programs of the form

$$\text{let} \quad \underbrace{C_1 \triangleq S_1; \cdots; C_k \triangleq S_k}_{Sequential\ Component\ definition} \quad \text{in} \quad \underbrace{P_0}_{Model\ Component\ definition}$$

where the Sequential processes (sequential components) named $C_1, \cdots, C_k (\in$ **PN**) are mutually recursively defined and may be used in the main model process $P_0$ ($P_0$ is formed by model components connected with model combinators) as well as in the sequential process bodies $S_1, \cdots, S_k$. We shall require that $C_1, \cdots, C_k$ are pairwise distinct and that they are the only sequential process names used. In turn, $P_0$ is the main model component which is essentially described by $\underset{L}{\bowtie}$ and / combinators to join the sequential processes $C_1, \cdots, C_k$. And these sequential processes are also called model components in model component definition.

Another thing should be mentioned here is that cooperation between several different components using differing cooperation sets may be regarded as being built up in layers. Each cooperation combining just two components which themselves might be composed from cooperation between components at a lower level. For example:

$$(P_1 \underset{L}{\bowtie} P_2) \underset{K}{\bowtie} P_3$$

In this case, the top layer could be $Q_1 \underset{K}{\bowtie} Q_2$ where, at the lower level, if $\equiv$ denotes syntactic equivalence, $Q_1 \equiv P_1 \underset{L}{\bowtie} P_2$ and $Q_2 \equiv P_3$.

**Example 2.1** *Let's transform a program written in PEPA syntax into the program with our form:*

$$\begin{array}{rcl} S & \overset{\text{def}}{=} & (g, r_1).(p, r_2).S \\ Q & \overset{\text{def}}{=} & (g, r_3).(h, r_4).Q + (p, r_5).Q \\ & & S \underset{\{g,p\}}{\bowtie} Q \end{array}$$

*would be transformed into*

$$\begin{array}{l} \text{let } S \triangleq (g, r_1).(p, r_2).S \\ \quad\;\; Q \triangleq (g, r_3).(h, r_4).Q + (p, r_5).Q \\ \text{in} \;\; S \underset{\{g,p\}}{\bowtie} Q \end{array}$$

*Here we also give other two programs: the sequential components remain the same as above, while the main model component are defined as* $(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ *and* $(S \underset{\{g,p\}}{\bowtie} S) \underset{g,p}{\bowtie} Q$ *respectively. These three programs would be furthered discussed in the following examples.*

## 2.2 Semantics

Following [22] the calculus is equipped with a reduction semantics. The *reduction relation* $E \rightarrow_{(\alpha,r)} E'$ is specified in Table 2.1 and expresses that the process $E$ in one step may evolve into the process $E'$.

**Prefix**

$$\frac{}{(\alpha, r).E \rightarrow_{(\alpha,r)} E}$$

**Cooperation**

$$\frac{E \rightarrow_{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \rightarrow_{(\alpha,r)} E' \underset{L}{\bowtie} F} \quad (\alpha \notin L)$$

$$\frac{F \rightarrow_{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \rightarrow_{(\alpha,r)} E \underset{L}{\bowtie} F'} \quad (\alpha \notin L)$$

$$\frac{E \rightarrow_{(\alpha,r_1)} E' \quad F \rightarrow_{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \rightarrow_{(\alpha,R)} E' \underset{L}{\bowtie} F'} \quad (\alpha \in L) \quad \text{where } R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \ \min(r_\alpha(E), r_\alpha(F))$$

**Hiding**

$$\frac{E \rightarrow_{(\alpha,r)} E'}{E/L \rightarrow_{(\alpha,r)} E'/L} \quad (\alpha \notin L)$$

$$\frac{E \rightarrow_{(\alpha,r)} E'}{E/L \rightarrow_{(\tau,r)} E'/L} \quad (\alpha \in L)$$

**Choice**

$$\frac{E \rightarrow_{(\alpha,r)} E'}{E + F \rightarrow_{(\alpha,r)} E'}$$

$$\frac{F \rightarrow_{(\alpha,r)} F'}{E + F \rightarrow_{(\alpha,r)} F'}$$

**Constant**

$$\frac{E \rightarrow_{(\alpha,r)} E'}{A \rightarrow_{(\alpha,r)} E'} \quad (A \overset{\mathit{def}}{=} E)$$

Table 2.1: Operational semantics of PEPA

**Example 2.2** *Using the formal semantics we can express the first steps of the reductions of Example 2.1 as follows:*

$$\begin{array}{lll} S \underset{\{g,p\}}{\bowtie} Q & \equiv & ((g,r_1).(p,r_2).S) \underset{\{g,p\}}{\bowtie} ((g,r_3).(h,r_4).Q + (p,r_5).Q) \\ & \rightarrow_{(g,R_1)} & ((p,r_2).S) \underset{\{g,p\}}{\bowtie} ((h,r_4).Q) \\ & \rightarrow_{(h,r_4)} & ((p,r_2).S) \underset{\{g,p\}}{\bowtie} Q \\ & \equiv & ((p,r_2).S) \underset{\{g,p\}}{\bowtie} ((g,r_3).(h,r_4).Q + (p,r_5).Q) \\ & \rightarrow_{(p,R_2)} & S \underset{\{g,p\}}{\bowtie} Q \end{array}$$

*Here $R_1$ is a function of $r_1$ and $r_3$ while $R_2$ is a function of $r_2$ and $r_5$. Since our analysis doesn't touch upon the* **rate** *of the action. so we will not explain the accurate meaning of this parameter. For the brevity of the thesis, we skip this parameter if necessary.*

## 2.3 The introduction of Label and Layer to PEPA

In order to capture the control structure of the process, we add additional information to each prefix component $(\alpha, r).S$. The actions $(\alpha, r)$ are annotated with two markers: labels $\ell \in$ **Lab** and layers $\imath \in$ **Layer**. These two markers serves as pointers into the process. They have no semantic significance and will only be used in the analysis that will be presented shortly.

**Lab:** Labels $\ell$ are *directly* assigned to each actions. They are *only* determined by the sequential component definition of the program.

The rule for allocating label to action: each action will be assigned a unique number $n \in \mathbb{N}$ that start from 1. Even two actions have the same action type will have different labels.

**Example 2.3** *Let's take the sequential component definition of program in Example 2.1, we will label them as follows:*

$$S \triangleq (g^1, r_1).(p^2, r_2).S$$
$$Q \triangleq (g^3, r_3).(h^4, r_4).Q + (p^5, r_5).Q$$

**Layer:** Layers $\imath$ *eventually* will also be assigned to each action. First each model component will be issued the layer $\imath$ which is determined by the model component definition of the program. Then the model component will assign its layer to the sequential components(actions) it represents.

The rule for allocating layer to model component: each model component will be assigned a unique vector $v$ (represents the layer) that contains element $e \in \{0,1\}$. $v$ is composed depending on the tree structure of the cooperation combinator. $e$ will be appended to $v$ from the top layer to the lower layer of the combinator tree in sequence. $v$ has two part: the *location* that takes up the rightmost position of the vector, and the *current layer* that consists of all elements left to the rightmost element. And the lefthanded side of combinator will be assigned 0 while righthanded side will be assigned 1. The top layer is assumed to be assigned 0. For example:

**Example 2.4** *Let's take one of the model component definition of program in Example 2.1, we will add layer to each model component as follows:*

$$(\underbrace{S}_{000} \underset{\{\}}{\bowtie} \underbrace{S}_{001}) \underset{\{g,p\}}{\bowtie} \underbrace{Q}_{01} \qquad \text{or} \qquad (S^{000} \underset{\{\}}{\bowtie} S^{001}) \underset{\{g,p\}}{\bowtie} Q^{01}$$

*where S on the left has layer 000: its* current layer *is 00, and since it is on the lefthanded side of* $\underset{\{\}}{\bowtie}$*, its* location *is 0. Q has layer 01: its* current layer *is 0 (the top layer), and due to the fact it is on the righthanded side of* $\underset{\{g,p\}}{\bowtie}$*, its* location *is 1.*
*Then we will grant layer to actions from model component. Let's take S from sequential component definition of program in Example 2.1 with layer 000 as the example:*

$$S \triangleq (g, r_1)^{000}.(p, r_2)^{000}.S$$

*Here all actions(g and p) will be assigned their model component's layer 000.*

If we annotate the syntax of the prefix component $(\alpha, r).S$ with two new notations, eventually it will change to $(\alpha^{\ell}, r)^{\imath}.S$.

**Example 2.5** *If we take S from Example 2.1, label it as in Example 2.3 and layer it with 000 as in Example 2.4, each action within S will be marked as follows:*

$$S \triangleq (g^1, r_1)^{000}.(p^2, r_2)^{000}.S$$

The above annotation doesn't impact the significance of the semantics at all. However, to facilitate the analysis, we present two versions of semantics as follows for different purposes.

Table 2.2 shows the semantics of the sequential component of PEPA only equipped with label but omit layer. This is because the layer of a certain sequential component always remain the same even when it evolves to other form of sequential

**Sequential Component**

| | |
|---|---|
| Prefix | $\dfrac{}{(\alpha^{\ell}, r).S \rightarrow_{\alpha^{\ell}} S}$ |
| Choice1 | $\dfrac{S_1 \rightarrow_{\alpha^{\ell_1}} S_1'}{S_1 + S_2 \rightarrow_{\alpha^{\ell_1}} S_1'}$ |
| Choice2 | $\dfrac{S_2 \rightarrow_{\alpha^{\ell_2}} S_2'}{S_1 + S_2 \rightarrow_{\alpha^{\ell_2}} S_2'}$ |
| ConstC | $\dfrac{S \rightarrow_{\alpha^{\ell}} S'}{C \rightarrow_{\alpha^{\ell}} S'} \quad (C \overset{def}{=} S)$ |

Table 2.2: Semantics of the sequential components of PEPA equipped with label

component. The label itself is enough to clearly illustrate the effect of each transition among sequential components.

Table 2.3 demonstrates the semantics of PEPA equipped with both label and layer. It contains two parts explaining exactly the syntax of PEPA introduced in Subsection 2.1.

**Sequential Component**

Prefix
$$\frac{}{[(\alpha^{\ell}, r).S]^{\imath_n} \rightarrow_{\alpha^{(\ell,\imath_n)}} [S]^{\imath_n}}$$

Choice1
$$\frac{[S_1]^{\imath_n} \rightarrow_{\alpha^{(\ell_1,\imath_n)}} [S_1']^{\imath_n}}{[S_1 + S_2]^{\imath_n} \rightarrow_{\alpha^{(\ell_1,\imath_n)}} [S_1']^{\imath_n}}$$

Choice2
$$\frac{[S_2]^{\imath_n} \rightarrow_{\alpha^{(\ell_2,\imath_n)}} [S_2']^{\imath_n}}{[S_1 + S_2]^{\imath_n} \rightarrow_{\alpha^{(\ell_2,\imath_n)}} [S_2']^{\imath_n}}$$

ConstC
$$\frac{[S]^{\imath_n} \rightarrow_{\alpha^{(\ell,\imath_n)}} [S']^{\imath_n}}{[C]^{\imath_n} \rightarrow_{\alpha^{(\ell,\imath_n)}} [S']^{\imath_n}} \quad (C \overset{def}{=} S)$$

**Model Component**

Coop1
$$\frac{[P_1]^{\imath_n 0} \rightarrow_{\alpha^{(\ell_1,\imath_1)}} [P_1']^{\imath_n 0}}{[P_1 \underset{L}{\bowtie} P_2]^{\imath_n} \rightarrow_{\alpha^{(\ell_1,\imath_1)}} [P_1' \underset{L}{\bowtie} P_2]^{\imath_n}} \quad (\alpha \notin L)$$

Coop2
$$\frac{[P_2]^{\imath_n 1} \rightarrow_{\alpha^{(\ell_2,\imath_2)}} [P_2']^{\imath_n 1}}{[P_1 \underset{L}{\bowtie} P_2]^{\imath_n} \rightarrow_{\alpha^{(\ell_2,\imath_2)}} [P_1 \underset{L}{\bowtie} P_2']^{\imath_n}} \quad (\alpha \notin L)$$

Coop3
$$\frac{[P_1]^{\imath_n 0} \rightarrow_{\alpha^{(\ell_1,\imath_1)}} [P_1']^{\imath_n 0} \quad [P_2]^{\imath_n 1} \rightarrow_{\alpha^{(\ell_2,\imath_2)}} [P_2']^{\imath_n 1}}{[P_1 \underset{L}{\bowtie} P_2]^{\imath_n} \rightarrow_{\alpha^{(\ell_1,\imath_1)(\ell_2,\imath_2)}} [P_1' \underset{L}{\bowtie} P_2']^{\imath_n}} \quad (\alpha \in L)$$

Hiding1
$$\frac{[P]^{\imath_n} \rightarrow_{\alpha^{(\ell,\imath)}} [P']^{\imath_n}}{[P/L]^{\imath_n} \rightarrow_{\alpha^{(\ell,\imath)}} [P'/L]^{\imath_n}} \quad (\alpha \notin L)$$

Hiding2
$$\frac{[P]^{\imath_n} \rightarrow_{\alpha^{(\ell,\imath)}} [P']^{\imath_n}}{[P/L]^{\imath_n} \rightarrow_{\tau^{(\ell,\imath)}} [P'/L]^{\imath_n}} \quad (\alpha \in L)$$

ConstS
$$\frac{P \rightarrow_{\alpha^{\ell}} P'}{[S]^{\imath_n} \rightarrow_{\alpha^{(\ell,\imath_n)}} [P']^{\imath_n}} \quad (S \overset{def}{=} P)$$

Table 2.3: Semantics of PEPA equipped with label and layer

CHAPTER 3

# Exposed Actions

An *exposed action* is an action that *may* participate in the next interaction. For instance, the sequential component $S$ in Example 2.1 will have action $g$ as exposed action while $Q$ will have both $g$ and $p$ as exposed actions. In either case, they will have one occurrence of each type. However, in general, a process may contain many, even infinitely many, occurrences of the same action (identified by the same label and layer) and it may be the case that several of them are ready to participate in the next interaction. For example:

$$\begin{array}{rcl} S & \triangleq & (\alpha, r).S \\ P & \triangleq & S + S \end{array}$$

Here $S$ only has one occurrence of $\alpha$ as exposed action, at the same time $P$ will have two occurrence of the same action as exposed actions and both of them are ready for next interaction.

## 3.1 Extended Multiset $M$ and Extra Extended Multiset $M_{ex}$

To capture this in [16] the authors define an *extended multiset* $M$ (the domain that our $\mathcal{E}^s_\star$ works on, $\mathcal{E}^s_\star$ will be presented in Subsection 3.2 ) and we introduce it in Subsection 3.1.1, and we are going to define the *extra extended multiset* $M_{ex}$ (the domain that our $\mathcal{E}^p_\star$ works on, $\mathcal{E}^p_\star$ will be presented in Subsection 3.2) in Subsection 3.1.2 for catering to our new scenario. In the following section 3.2, we will introduce the abstraction function $\mathcal{E}^s_\star$ and $\mathcal{E}^p_\star$ that specify an extended multiset and extra extended multiset of the program.

### 3.1.1 Extended Multiset $M$

$M$ is defined as an element of

$$\mathfrak{M} = \textbf{Lab} \to \mathbb{N} \cup \{\top\}$$

$M(\ell)$ records the number of occurrences of the label $\ell$; there may be a finite number in which case $M(\ell) \in \mathbb{N}$ or an max finite number in which case $M(\ell) = \top$. Here $\top \in \mathbb{N}$ that varies from different programs.

We equip the set $\mathfrak{M} = \textbf{Lab} \to \mathbb{N} \cup \{\top\}$ with a partial ordering $\leq_{\mathfrak{M}}$ defined by:

$$M \leq_{\mathfrak{M}} M' \quad \text{iff} \quad \forall \ell : M(\ell) \leq M'(\ell) \vee M'(\ell) = \top$$

**Fact 3.1** *The domain*$(\mathfrak{M}, \leq_{\mathfrak{M}})$ is a complete lattice with *least element* $\bot_{\mathfrak{M}}$ given by $\forall \ell : \bot_{\mathfrak{M}}(\ell) = 0$ and *largest element* $\top_{\mathfrak{M}}$ given by $\forall \ell : \top_{\mathfrak{M}}(\ell) = \top$.

The least upper bound and greatest lower bound operators of $\mathfrak{M}$ are denoted $\sqcup_{\mathfrak{M}}$ and $\sqcap_{\mathfrak{M}}$, respectively, and they are defined by:

$$(M \sqcup_{\mathfrak{M}} M')(\ell) = \begin{cases} max\{M(\ell), M'(\ell)\} & \text{if } M(\ell) \in \mathbb{N} \wedge M'(\ell) \in \mathbb{N} \\ \top & \text{otherwise} \end{cases}$$

$$(M \sqcap_{\mathfrak{M}} M')(\ell) = \begin{cases} min\{M(\ell), M'(\ell)\} & \text{if } M(\ell) \in \mathbb{N} \wedge M'(\ell) \in \mathbb{N} \\ M(\ell) & \text{if } M'(\ell) = \top \\ M'(\ell) & \text{if } M(\ell) = \top \end{cases}$$

We also define addition and substraction on extended multisets, they are defined by:

$$(M +_{\mathfrak{M}} M')(\ell) = \begin{cases} M(\ell) + M'(\ell) & \text{if } M(\ell) \in \mathbb{N} \wedge M'(\ell) \in \mathbb{N} \\ \top & \text{otherwise} \end{cases}$$

$$(M -_{\mathfrak{M}} M')(\ell) = \begin{cases} M(\ell) - M'(\ell) & \text{if } M(\ell) \in \mathbb{N} \wedge M(\ell) \geq M'(\ell) \\ 0 & \text{if } M(\ell) \in \mathbb{N} \wedge M(\ell) < M'(\ell) \\ & \text{or } M(\ell) \in \mathbb{N} \wedge M'(\ell) = \top \\ \top & \text{if } M(\ell) = \top \end{cases}$$

**Fact 3.2** *The operations enjoy the following properties:*

1. *$\sqcup_{\mathfrak{M}}$ and $+_{\mathfrak{M}}$ are monotonic in both arguments and they both observe the laws of Abelian monoid with $\perp_{\mathfrak{M}}$ as neutral element.*
2. *$\sqcap_{\mathfrak{M}}$ is monotonic in both arguments and it observes the laws of an Abelian monoid with $\top_{\mathfrak{M}}$ as neutral element.*
3. *$-_{\mathfrak{M}}$ is monotonic in its left argument and anti-monotonic in its right argument.*

**Fact 3.3** *The operations $+_{\mathfrak{M}}$ and $-_{\mathfrak{M}}$ satisfy the following laws:*

1. *$M -_{\mathfrak{M}} (M_1 +_{\mathfrak{M}} M_2) = (M -_{\mathfrak{M}} M_1) -_{\mathfrak{M}} M_2$*
2. *If $M \leq_{\mathfrak{M}} M_1$ then $(M_1 -_{\mathfrak{M}} M) +_{\mathfrak{M}} M_2 = (M_1 +_{\mathfrak{M}} M_2) -_{\mathfrak{M}} M$.*
3. *If $M'_1 \leq_{\mathfrak{M}} M_1$ and $M'_2 \leq_{\mathfrak{M}}$ then $(M_1 -_{\mathfrak{M}} M'_1) +_{\mathfrak{M}} (M_2 -_{\mathfrak{M}} M'_2) = (M_1 +_{\mathfrak{M}} M_2) -_{\mathfrak{M}} (M'_1 +_{\mathfrak{M}} M'_2)$.*

In the following we write $M[\ell \mapsto n]$ for the extended multiset $M$ that $\ell$ is mapped to $n \in \mathbb{N} \cup \{\top\}$. We write $\texttt{dom}(M)$ for the set $\{\ell \mid M(\ell) \neq 0\}$.

### 3.1.2 Extra Extended Multiset $M_{ex}$

If **Labex** = (**Lab**, **Layer**) then

$$\mathfrak{M}_{ex} = \textbf{Labex} \rightarrow \mathbb{N} \cup \{\top\}$$

We let $\ell_{ex} = (\ell, \imath) \in$ **Labex**, then $M_{ex}(\ell_{ex})$ records the number of occurrences of the label $\ell$ at layer $\imath$; there may be a finite number in which case $M_{ex}(\ell_{ex}) \in \mathbb{N}$ or an max finite number in which case $M_{ex}(\ell_{ex}) = \top$. And the value of $\top$ varies from different programs.

**Fact 3.4** *The domain*$(\mathfrak{M}_{ex}, \leq_{\mathfrak{M}_{ex}})$ is a complete lattice with *least element* $\perp_{\mathfrak{M}_{ex}}$ given by $\forall \ell_{ex} : \perp_{\mathfrak{M}_{ex}}(\ell_{ex}) = 0$ and *largest element* $\top_{\mathfrak{M}_{ex}}$ given by $\forall \ell_{ex} : \top_{\mathfrak{M}_{ex}}(\ell_{ex}) = \top$.

The least upper bound, greatest lower bound, addition and substraction of $M_{ex}$ are defined very close to the counterpart of $M$, which are listed as follows:

$$(M_{ex} \sqcup_{\mathfrak{M}_{ex}} M'_{ex})(\ell_{ex}) = \begin{cases} max\{M_{ex}(\ell_{ex}), M'_{ex}(\ell_{ex})\} & \text{if } M_{ex}(\ell_{ex}) \in \mathbb{N} \wedge M'_{ex}(\ell_{ex}) \in \mathbb{N} \\ \top & \text{otherwise} \end{cases}$$

$$(M_{ex} \sqcap_{\mathfrak{M}_{ex}} M'_{ex})(\ell) = \begin{cases} min\{M_{ex}(\ell_{ex}), M'_{ex}(\ell_{ex})\} & \text{if } M_{ex}(\ell_{ex}) \in \mathbb{N} \wedge M'_{ex}(\ell_{ex}) \in \mathbb{N} \\ M_{ex}(\ell_{ex}) & \text{if } M'_{ex}(\ell_{ex}) = \top \\ M'_{ex}(\ell_{ex}) & \text{if } M_{ex}(\ell_{ex}) = \top \end{cases}$$

$$(M_{ex} +_{\mathfrak{M}_{ex}} M'_{ex})(\ell) = \begin{cases} M_{ex}(\ell_{ex}) + M'_{ex}(\ell_{ex}) & \text{if } M_{ex}(\ell_{ex}) \in \mathbb{N} \wedge M'_{ex}(\ell_{ex}) \in \mathbb{N} \\ \top & \text{otherwise} \end{cases}$$

$$(M_{ex} -_{\mathfrak{M}_{ex}} M'_{ex})(\ell) = \begin{cases} M_{ex}(\ell_{ex}) - M'_{ex}(\ell_{ex}) & \text{if } M_{ex}(\ell_{ex}) \in \mathbb{N} \wedge M_{ex}(\ell_{ex}) \geq M'_{ex}(\ell_{ex}) \\ 0 & \text{if } M_{ex}(\ell_{ex}) \in \mathbb{N} \wedge M(\ell) < M'_{ex}(\ell_{ex}) \\ & \text{or } M_{ex}(\ell_{ex}) \in \mathbb{N} \wedge M'_{ex}(\ell_{ex}) = \top \\ \top & \text{if } M_{ex}(\ell_{ex}) = \top \end{cases}$$

**Fact 3.5** *The operations enjoy the following properties:*

1. $\sqcup_{\mathfrak{M}_{ex}}$ *and* $+_{\mathfrak{M}_{ex}}$ *are monotonic in both arguments and they both observe the laws of Abelian monoid with* $\perp_{\mathfrak{M}_{ex}}$ *as neutral element.*
2. $\sqcap_{\mathfrak{M}_{ex}}$ *is monotonic in both arguments and it observes the laws of an Abelian monoid with* $\top_{\mathfrak{M}_{ex}}$ *as neutral element.*
3. $-_{\mathfrak{M}_{ex}}$ *is monotonic in its left argument and anti-monotonic in its right argument.*

**Fact 3.6** *The operations* $+_{\mathfrak{M}_{ex}}$ *and* $-_{\mathfrak{M}_{ex}}$ *satisfy the following laws:*

1. $M -_{\mathfrak{M}_{ex}} (M_1 +_{\mathfrak{M}_{ex}} M_2) = (M -_{\mathfrak{M}_{ex}} M_1) -_{\mathfrak{M}_{ex}} M_2$

2. *If* $M \leq_{\mathfrak{M}_{ex}} M_1$ *then* $(M_1 -_{\mathfrak{M}_{ex}} M) +_{\mathfrak{M}_{ex}} M_2 = (M_1 +_{\mathfrak{M}_{ex}} M_2) -_{\mathfrak{M}_{ex}} M$.
3. *If* $M'_1 \leq_{\mathfrak{M}_{ex}} M_1$ *and* $M'_2 \leq_{\mathfrak{M}_{ex}}$ *then* $(M_1 -_{\mathfrak{M}_{ex}} M'_1) +_{\mathfrak{M}_{ex}} (M_2 -_{\mathfrak{M}_{ex}} M'_2) = (M_1 +_{\mathfrak{M}_{ex}} M_2) -_{\mathfrak{M}_{ex}} (M'_1 +_{\mathfrak{M}_{ex}} M'_2)$.

We write `domExlayer`$(M_{ex}, \imath)$ for the set $\{\ell \mid (\ell, \imath) \in \mathbf{Labex}$ of $M_{ex}\}$. We write `domEx`$(M_{ex})$ for the set $\{(\ell, \imath) \mid M_{ex}(\ell, \imath) \neq 0\}$.

## 3.2 Calculating Exposed Actions

The information of key interest is the collection of *extra extended multisets* of exposed actions of the model component processes. However, to obtain that we need first to get the *extended multisets* of exposed actions of the sequential component processes. The first step is computed by abstraction function $\mathcal{E}^s_\star$, and the second step by function namely $\mathcal{E}^p_\star$.

To motivate the definition let us first consider the combination of choice and prefix of two sequential processes $(\alpha_1^{\ell_1}, r_1).S_1 + (\alpha_2^{\ell_2}, r_2).S_2$. Here both of the actions $\alpha_1$ and $\alpha_2$ are ready to interact but actions from $S_1$ and $S_2$ are not, so we shall take:

$$\mathcal{E}^s[\![(\alpha_1^{\ell_1}, r_1).S_1 + (\alpha_2^{\ell_2}, r_2).S_2]\!]env = \perp_{\mathfrak{M}}[\ell_1 \mapsto 1] +_{\mathfrak{M}} \perp_{\mathfrak{M}}[\ell_2 \mapsto 1]$$

If the two labels happen to be equal ($\ell_1 = \ell_2$) the overall count will become 2 since we have used the pointwise addition operator $+_{\mathfrak{M}}$.

Second, if we turn to cooperation combinator, we shall have the following function formula, and this time we use $\mathcal{E}^p$ instead:

$$\mathcal{E}^p[\![(\alpha_1^{\ell_1}, r_1)^{00}.S \bowtie (\alpha_2^{\ell_2}, r_2)^{01}.S]\!] = \perp_{\mathfrak{M}_{ex}}[(\ell_1, 00) \mapsto 1] +_{\mathfrak{M}_{ex}} \perp_{\mathfrak{M}_{ex}}[(\ell_2, 01) \mapsto 1]$$

In this case, even though $\ell_1$ and $\ell_2$ have the same label, According to the $+_{\mathfrak{M}_{ex}}$ operation, the overall count couldn't become 2, because these two labels are not in the same layer.

### Function $\mathcal{E}$

To handle the general case we shall introduce two functions

$$\mathcal{E}^s : \mathbf{S}_{proc} \rightarrow (\mathbf{PN} \rightarrow \mathfrak{M}) \rightarrow \mathfrak{M}$$

$$\mathcal{E}^p : \mathbf{P}_{proc} \rightarrow \mathfrak{M}_{ex}$$

For function $\mathcal{E}^s$, it takes an environment as the additional parameter which holds the required information for the process names. The function is defined in Table 3.1 for arbitrary processes; in the case of choice and prefix, it generalizes the clauses shown above. Turning to the clause for sequence constants we simply consult the environment *env* provided as the first argument to $\mathcal{E}^s$.

As shown in Table 3.1, there defines a function $\mathcal{F}_{\mathcal{E}}$: $(\mathbf{PN} \to \mathfrak{M}) \to (\mathbf{PN} \to \mathfrak{M})$. Since the operations involved in its definition are all monotonic (cf. Fact 3.1). we have a monotonic function defined on a complete lattice (cf. Fact 3.2) and Tarski's fixed point theorem ensures that it has a fixed point which is denoted $env_{\mathcal{E}}$ in Table 3.1. Since all sequential processes are finite it follows that $\mathcal{F}_{\mathcal{E}}$ is continuous and hence that the Kleene formulation of the fixed point is permissible. We can now define the function

$$\mathcal{E}^s_\star : \mathbf{S}_{proc} \to \mathfrak{M}$$

Simply as $\mathcal{E}^s_\star[\![S]\!] = \mathcal{E}^s[\![S]\!]env_{\mathcal{E}}$.

For function $\mathcal{E}^p$, it will always take layer $\imath$ as parameters and finally append the correct layer to each $S$ sequential component. Specifically, for cooperation operator, it will combine the result of two components by $+_{\mathfrak{M}_{ex}}$ operation. The clause for the $P/L$ will simply ignores the hidden set $L$. The clause for constant model combinator will borrow the result get from $\mathcal{E}^s_\star$ step and use the denotation of this sequential component to compute the overall $\mathfrak{M}_{ex}$. It is worth pointing out that only the label and layer pair belonging to the $\mathfrak{M}_{ex}$ set should be set up by each constant combinator clause.

We can now define the function

$$\mathcal{E}^p_\star : \mathbf{P}_{proc} \to \mathfrak{M}_{ex}$$

Simply as $\mathcal{E}^p_\star[\![P]\!] = \mathcal{E}^p[\![P]\!]^0$.The parameter 0 takes charge of layer initialization and is the value issued to the top layer. (cf.section 2.3).

**Example 3.1** *For the running example of Example 2.1 we have*

$$\begin{aligned}
\mathcal{E}^p_\star[\![S \underset{\{g,p\}}{\bowtie} Q]\!] &= [(1,00) \mapsto 1, (2,00) \mapsto 0, (3,01) \mapsto 1, (4,01) \mapsto 0, \\
&\quad (5,01) \mapsto 1] \\
\mathcal{E}^p_\star[\![(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q]\!] &= [(1,000) \mapsto 1, (2,000) \mapsto 0, (1,001) \mapsto 1, (2,001) \mapsto 0, \\
&\quad (3,01) \mapsto 1, (4,01) \mapsto 0, (5,01) \mapsto 1] \\
\mathcal{E}^p_\star[\![(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q]\!] &= [(1,000) \mapsto 1, (2,000) \mapsto 0, (1,001) \mapsto 1, (2,001) \mapsto 0, \\
&\quad (3,01) \mapsto 1, (4,01) \mapsto 0, (5,01) \mapsto 1]
\end{aligned}$$

Exposed actions for let $C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k$ in $P_0$

$$\begin{array}{rcl}
\mathcal{E}^s[\![(\alpha^\ell, r).S]\!]env &=& \bot_{\mathfrak{M}}[\ell \mapsto 1] \\
\mathcal{E}^s[\![S_1 + S_2]\!]env &=& \mathcal{E}^s[\![S_1]\!]env +_{\mathfrak{M}} \mathcal{E}^s[\![S_2]\!]env \\
\mathcal{E}^s[\![C]\!]env &=& env(C) \\
\mathcal{E}^s_\star[\![S]\!] &=& \mathcal{E}^s[\![S]\!]env_{\mathcal{E}} \\
\text{where}\mathcal{F}_{\mathcal{E}}(env) &=& [C_1 \mapsto \mathcal{E}^s[\![S_1]\!]env, \cdots, C_k \mapsto \mathcal{E}^s[\![S_k]\!]env] \\
\text{and } env_{\bot_{\mathfrak{M}}} &=& [C_1 \mapsto \bot_{\mathfrak{M}}, \cdots, C_k \mapsto \bot_{\mathfrak{M}}] \\
\text{and } env_{\mathcal{E}} &=& \sqcup_{j\geq 0}\mathcal{F}^j_{\mathcal{E}}(env_{\bot_{\mathfrak{M}}}) \\
\mathcal{E}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^\imath &=& \mathcal{E}^p[\![P_1]\!]^{\imath 0} +_{\mathfrak{M}ex} \mathcal{E}^p[\![P_2]\!]^{\imath 1} \\
\mathcal{E}^p[\![P/L]\!]^\imath &=& \mathcal{E}^p[\![P]\!]^\imath \\
\mathcal{E}^p[\![S]\!]^\imath &=& \text{let } (\ell_1 \mapsto n_1, \cdots, \ell_n \mapsto n_n) = \mathcal{E}^s_\star[\![S]\!] \\
&& \text{in } \bot_{\mathfrak{M}ex}[(\ell_j, \imath) \mapsto n_j] \text{ where } \ell_j \in \text{domExlayer}(\bot_{\mathfrak{M}ex}, \imath) \\
&& \ \text{and } j \in \{1, \cdots, n\} \\
\mathcal{E}^p_\star[\![P]\!] &=& \mathcal{E}^p[\![P]\!]^0
\end{array}$$

Table 3.1: $\mathcal{E}^s$ and $\mathcal{E}^p$ function

*We could see that the first case has 5 elements (5 label-layer pairs) in its extra extended multiset while the last two cases have 7 elements in each of them.* All *label-layer pairs for each program is listed in Table above. However, we could simplify them with the help of $\bot_{\mathfrak{M}ex}$ and remove the element that doesn't ready for transition(the label-layer pair maps to 0). Here we have another version of the result:*

$$\begin{array}{rcl}
\mathcal{E}^p_\star[\![S \underset{\{g,p\}}{\bowtie} Q]\!] &=& \bot_{\mathfrak{M}ex}[(1,00) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1] \\
\mathcal{E}^p_\star[\![(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q]\!] &=& \bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1] \\
\mathcal{E}^p_\star[\![(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q]\!] &=& \bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]
\end{array}$$

## 3.3 Termination

When deal with the calculation of $\mathcal{E}^s_\star$, it is not trivial to implement the computation of the least fixed point of Table 3.1. In [16], the authors propose an Lemma which fit in our case pretty well, here we just give the lemma without proof.

**Lemma 3.7** *Using the notation of Table 3.1 we have*

$$env_{\mathcal{E}} \quad = \quad \mathcal{F}_{\mathcal{E}}^{k}(env_{\bot_{\mathfrak{M}}}) \bowtie \mathcal{F}_{\mathcal{E}}^{2k}(env_{\bot_{\mathfrak{M}}})$$

*where $k$ is number of sequential components in the program and $\bowtie$ is the point-wise extension of the operation $\bowtie_{\mathfrak{M}}$ defined by*

$$(M \bowtie_{\mathfrak{M}} M')(\ell) = \begin{cases} M(\ell) & \textit{if } M(\ell) = M'(\ell) \\ \top & \textit{otherwise} \end{cases}$$

From this lemma, we could calculate the $env_{\mathcal{E}}$ and $\mathcal{E}_{\star}^{s}[\![S]\!]$ without any problem. Consequently, $\mathcal{E}_{\star}^{p}[\![P]\!]$ could be computed based on the result of $\mathcal{E}_{\star}^{s}[\![S]\!]$.

Chapter 4

# Transfer Functions

The abstraction functions $\mathcal{E}^s_\star$ and $\mathcal{E}^p_\star$ only give us the information of interest for the initial process and we shall now present auxiliary functions allowing us to approximate how the information evolves during the execution of the process.

Once an action has participated in an interaction, some new actions maybe expose and some actions just cease to be exposed. We say the new exposure actions would be *generated* by the default interaction while the action no longer expose anymore would be *killed* by the same interaction. For instance, recall $S$ is marked as $S = (g^1, r_1)^{000}.(p^2, r_2)^{000}.S$ in the Example 2.5 . Initially $(g^1, r_1)^{000}$ is exposed but once it has been executed it will no longer be exposed in the next interaction(we say it is killed), but at the same time the action $(p^2, r_2)^{000}$ would get exposed(we say it is generated).

Thus, in order to capture how the program evolve, we shall now introduce two functions $\mathcal{G}^s_\star$ and $\mathcal{G}^p_\star$ to describe the information generated by execution of the processes and two functions $\mathcal{K}^s_\star$ and $\mathcal{K}^p_\star$ to describe the information killed by execution of the processes.

## 4.1 Extended Multimap $T$ and Extra Extended Multimap $T_{ex}$

Just as we introduce the domains on which $\mathcal{E}^s_\star$ and $\mathcal{E}^p_\star$ work, we will first describe relevant domains that functions $\mathcal{G}^s_\star$, $\mathcal{G}^p_\star$, $\mathcal{K}^s_\star$ and $\mathcal{K}^p_\star$ ground on.

### 4.1.1 Extended Multimap $T$

The information generated by $\mathcal{G}^s_\star$ or $\mathcal{K}^s_\star$ will be an element of:

$$\mathfrak{T} = \mathbf{Lab} \rightarrow \mathfrak{M} \qquad (= \mathbf{Lab} \rightarrow (\mathbf{Lab} \rightarrow \mathbb{N} \cup \{\top\}))$$

As for exposed actions it is not sufficient to use sets: there may be more than one occurrence of an action that is either generated or killed by another action. The ordering $\leq_\mathfrak{T}$ is defined as the pointwise extension of $\leq_\mathfrak{M}$:

$$T_1 \leq_\mathfrak{T} T_2 \qquad \text{iff} \qquad \forall \ell : T_1(\ell) \leq_\mathfrak{M} T_2(\ell)$$

In analogy with Fact 3.1 this turns$(\mathfrak{T}, \leq_\mathfrak{T})$ into a complete lattice with least element $\perp_\mathfrak{T}$ and greatest element $\top_\mathfrak{T}$ defined as expected. The operators $\sqcup_\mathfrak{T}, \sqcap_\mathfrak{T}, +_\mathfrak{T}$ and $-_\mathfrak{T}$ on $\mathfrak{T}$ are defined as the pointwise extensions of the corresponding operators on $\mathfrak{M}$ and they enjoy properties corresponding to those of Fact 3.2 and Fact 3.3. We shall occasionally write $T(\ell_1\ell_2)$ as an abbreviation for $T(\ell_1) +_\mathfrak{M} T(\ell_2)$.

### 4.1.2 Extended Multimap $T_{ex}$

If $\mathbf{Labex} = (\mathbf{Lab}, \mathbf{Layer})$ then the information generated by $\mathcal{G}^p_\star$ or $\mathcal{K}^p_\star$ will be an element of:

$$\mathfrak{T}_{ex} = \mathbf{Labex} \rightarrow \mathfrak{M}_{ex} \qquad (= \mathbf{Labex} \rightarrow (\mathbf{Labex} \rightarrow \mathbb{N} \cup \{\top\}))$$

The operators $\sqcup_{\mathfrak{T}ex}, \sqcap_{\mathfrak{T}ex}, +_{\mathfrak{T}ex}$ and $-_{\mathfrak{T}ex}$ on $\mathfrak{T}_{ex}$ are defined as the pointwise extensions of the corresponding operators on $\mathfrak{M}_{ex}$. We shall occasionally write $T_{ex}(\ell^1_{ex}\ell^2_{ex})$ as an abbreviation for $T_{ex}(\ell^1_{ex}) +_{\mathfrak{M}ex} T_{ex}(\ell^2_{ex})$.

## 4.2 Generated Actions

To motivate the definitions of $\mathcal{G}^s_\star$ and $\mathcal{G}^p_\star$, let us consider prefix combinator as expressed in the process $(\alpha^\ell, r).S$. Clearly, once $(\alpha^\ell, r)$ has been executed it will no longer be exposed whereas the actions of $\mathcal{E}^s_\star[\![S]\!]$ will become exposed. Thus a first suggestion may be to take $\mathcal{G}^s_\star[\![(\alpha^\ell, r).S]\!](\ell) = \mathcal{E}^s_\star[\![S]\!]$. However, to cater for the case where the same label may occur several times in a sequential process (as the case when $\ell$ is used inside $S$) we have to modify these formula slightly to ensure that they correctly combines the information available about $\ell$. The function $\mathcal{G}^s_\star$ will compute an *over*-approximation as it takes the *least upper bound* of the information available.

$$\mathcal{G}^s_\star[\![(\alpha^\ell, r).S]\!]\ell' = \begin{cases} \mathcal{E}^s_\star[\![S]\!] \sqcup_{\mathfrak{M}} \mathcal{G}^s_\star[\![S]\!]\ell' & \text{if } \ell' = \ell \\ \mathcal{G}^s_\star[\![S]\!]\ell' & \text{if } \ell' \neq \ell \end{cases}$$

it could be rewritten as:

$$\mathcal{G}^s_\star[\![(\alpha^\ell, r).S]\!] = \bot_{\mathfrak{T}}[\ell \mapsto \mathcal{E}^s_\star[\![S]\!]] \sqcup_{\mathfrak{T}} \mathcal{G}^s_\star[\![S]\!]$$

**Function** $\mathcal{G}$
To cater for the general case, we shall define two functions:

$$\mathcal{G}^s : \mathbf{S}_{proc} \rightarrow (PN \rightarrow \mathfrak{T}) \rightarrow \mathfrak{T}$$

$$\mathcal{G}^p : \mathbf{P}_{proc} \rightarrow \mathfrak{T}_{ex}$$

For function $\mathcal{G}^s$, it takes an environment as the parameter which provides relevant information for the process names and is defined in Table 4.1. The clauses are much as one should expect from the explanation above, in particular we may note that the operation $\sqcup_{\mathfrak{T}}$ is used to combine information throughout the clauses and represents the *over*-approximation characteristic of this function. The recursive definitions in prefix clause give rise to a monotonic function $\mathcal{F}_{\mathcal{G}}$: $(\mathbf{PN} \rightarrow \mathfrak{T}) \rightarrow (\mathbf{PN} \rightarrow \mathfrak{T})$ on a complete lattice (cf. Fact 3.1,Fact 3.2). and hence Tarski's fixed point theorem ensures that the least fixed point $env_{\mathcal{G}}$ exists. Once more the function turns out to be continuous and hence the Kleene formulation of the fixed point is permissible. And we could define function

$$\mathcal{G}^s_\star : \mathbf{S}_{proc} \rightarrow \mathfrak{T}$$

and this function will give us all information generated by sequential component in the program.

For function $\mathcal{G}^p$, it will always take layer $\imath$ as parameters and finally append the correct layer to each $S$ sequential component. Specifically, for cooperation operator, it will combine the result of two components by $\sqcup_{\mathfrak{T}_{ex}}$ operation. The clause for the $P/L$ will simply ignores the hidden set $L$. The clause for constant model combinator will borrow the result get from $\mathcal{G}^s_\star$ step and compute the denotation of this sequential component to the overall $\mathfrak{T}_{ex}$. It is worth pointing

Exposed actions for let $C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k$ in $P_0$

$$
\begin{array}{rcl}
\mathcal{G}^s[\![(\alpha^\ell, r).S]\!]env & = & \bot_{\mathfrak{T}}[\ell \mapsto \mathcal{E}^s[\![S]\!]env_\varepsilon] \sqcup_{\mathfrak{T}} \mathcal{G}^s[\![S]\!]env \\
\mathcal{G}^s[\![S_1 + S_2]\!]env & = & \mathcal{G}^s[\![S_1]\!]env \sqcup_{\mathfrak{T}} \mathcal{G}^s[\![S_2]\!]env \\
\mathcal{G}^s[\![C]\!]env & = & env(C) \\
\mathcal{G}^s_\star[\![S]\!] & = & \mathcal{G}^s[\![S]\!]env_{\mathcal{G}} \\
\text{where}\mathcal{F}_{\mathcal{G}} & = & [C_1 \mapsto \mathcal{G}^s[\![S_1]\!]env, \cdots, C_k \mapsto \mathcal{G}^s[\![S_k]\!]env] \\
\text{and } env_{\bot_{\mathfrak{T}}} & = & [C_1 \mapsto \bot_{\mathfrak{T}}, \cdots, C_k \mapsto \bot_{\mathfrak{T}}] \\
\text{and } env_{\mathcal{G}} & = & \sqcup_{j\geq 0}\mathcal{F}^j_{\mathcal{G}}(env_{\bot_{\mathfrak{T}}}) \\
\mathcal{G}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^\imath & = & \mathcal{G}^p[\![P_1]\!]^{\imath 0} \sqcup_{\mathfrak{T}ex} \mathcal{G}^p[\![P_2]\!]^{\imath 1} \\
\mathcal{G}^p[\![P/L]\!]^\imath & = & \mathcal{G}^p[\![P]\!]^\imath \\
\mathcal{G}^p[\![S]\!]^\imath & = & \text{let } [\ell_1 \mapsto \{\ell_1 \mapsto n_{11}, \cdots, \ell_n \mapsto n_{1n}\}, \cdots, \\
 & & \qquad \ell_n \mapsto \{\ell_1 \mapsto n_{n1}, \cdots, \ell_n \mapsto n_{nn}\}] = \mathcal{G}^s_\star[\![S]\!] \\
 & & \text{in } \bot_{\mathfrak{T}ex}[(\ell_j, \imath) \mapsto \bot_{\mathfrak{M}ex}[(\ell_k, \imath) \mapsto n_{jk}]] \\
 & & \ \text{where } \ell_j, \ell_k \in \text{domExlayer}(\bot_{\mathfrak{M}ex}, \imath) \text{ and } j, k \in \{1, \cdots, n\} \\
\mathcal{G}^p_\star[\![P]\!] & = & \mathcal{G}^p[\![P]\!]^0
\end{array}
$$

Table 4.1: $\mathcal{G}^s$ and $\mathcal{G}^p$ function

out that only the label and layer pair belonging to the $\mathfrak{T}_{ex}$ set should be set up by each constant combinator clause and we use $j$ and $k$ to control it.

We can now define the function

$$\mathcal{G}^p_\star : \mathbf{P}_{proc} \to \mathfrak{T}_{ex}$$

Simply as $\mathcal{G}^p_\star[\![P]\!] = \mathcal{G}^p[\![P]\!]^0$.The parameter 0 takes charge of layer initialization and is the value issued to the top layer.

**Example 4.1** *We will compute the $\mathcal{G}^p_\star$ for programs introduced in Example 2.1.*

| $\ell_{ex}$ | $\mathcal{G}^p_\star[\![S \underset{\{g,p\}}{\bowtie} Q]\!](\ell_{ex})$ |
|---|---|
| $(1, 00)$ | $\bot_{\mathfrak{M}ex}[(2, 00) \mapsto 1]$ |
| $(2, 00)$ | $\bot_{\mathfrak{M}ex}[(1, 00) \mapsto 1]$ |
| $(3, 01)$ | $\bot_{\mathfrak{M}ex}[(4, 01) \mapsto 1]$ |
| $(4, 01)$ | $\bot_{\mathfrak{M}ex}[(3, 01) \mapsto 1, (5, 01) \mapsto 1]$ |
| $(5, 01)$ | $\bot_{\mathfrak{M}ex}[(3, 01) \mapsto 1, (5, 01) \mapsto 1]$ |

| $\ell_{ex}$ | $\mathcal{G}^p_\star[\![(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q]\!](\ell_{ex})$ | $\mathcal{G}^p_\star[\![(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q]\!](\ell_{ex})$ |
|---|---|---|
| $(1,000)$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1]$ |
| $(2,000)$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1]$ |
| $(1,001)$ | $\perp_{\mathfrak{M}ex}[(2,001) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(2,001) \mapsto 1]$ |
| $(2,001)$ | $\perp_{\mathfrak{M}ex}[(1,001) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(1,001) \mapsto 1]$ |
| $(3,01)$ | $\perp_{\mathfrak{M}ex}[(4,01)]$ | $\perp_{\mathfrak{M}ex}[(4,01)]$ |
| $(4,01)$ | $\perp_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $(5,01)$ | $\perp_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ |

*It could be seen clearly that* $(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ *and* $(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ *have exactly the same results from* $\mathcal{G}^p_\star$ *functions.*

**Lemma 4.1** *If* $P \to_{\tilde{\ell}} Q$ *then* $\mathcal{G}^s_\star[\![Q]\!] \leq_{\mathfrak{T}} \mathcal{G}^s_\star[\![P]\!]$.

**Proof.** We proceed by induction on the inference of $P \to_{\tilde{\ell}} Q$ as defined in Table 2.2.

**The case [Prefix ]:**

$$\begin{aligned} \mathcal{G}^s_\star[\![(\alpha^\ell, r).S]\!] &= \mathcal{G}^s[\![(\alpha^\ell, r).S]\!]env_G \\ &= \perp_{\mathfrak{T}}[\ell \mapsto \mathcal{E}^s[\![S]\!]env_\varepsilon] \sqcup_{\mathfrak{T}} \mathcal{G}^s[\![S]\!]env_G \\ &\geq_{\mathfrak{T}} \mathcal{G}^s[\![S]\!]env_G = \mathcal{G}^s_\star[\![S]\!] \end{aligned}$$

as required.

**The case [Choice1 ]:** From Induction hypothesis, we know $\mathcal{G}^s_\star[\![S_1']\!] \leq_{\mathfrak{T}} \mathcal{G}^s_\star[\![S_1]\!]$ which means $\mathcal{G}^s[\![S_1']\!]env_G \leq_{\mathfrak{T}} \mathcal{G}^s[\![S_1]\!]env_G$.
So we calculate

$$\begin{aligned} \mathcal{G}^s_\star[\![S_1 + S_2]\!] &= \mathcal{G}^s[\![S_1 + S_2]\!]env_G = \mathcal{G}^s[\![S_1]\!]env_G \sqcup_{\mathfrak{T}} \mathcal{G}^s[\![S_2]\!]env_G \\ &\geq_{\mathfrak{T}} \mathcal{G}^s[\![S_1]\!]env_G \geq_{\mathfrak{T}} \mathcal{G}^s[\![S_1']\!]env_G = \mathcal{G}^s_\star[\![S_1']\!] \end{aligned}$$

this proves the result.

**The case [Choice2 ]:** Analogous.

**The case [ConstC ]:** From the induction hypothesis, we know $\mathcal{G}^s_\star[\![S']\!] \leq_{\mathfrak{T}} \mathcal{G}^s_\star[\![S]\!]$. We also know $\mathcal{G}^s_\star[\![C]\!] = \mathcal{G}^s_\star[\![S]\!]$ because $C \overset{def}{=} S$, so $\mathcal{G}^s_\star[\![S']\!] \leq_{\mathfrak{T}} \mathcal{G}^s_\star[\![C]\!]$ as required.

□

**Lemma 4.2** *If $P \rightarrow_{\alpha^{(\ell,\imath)}} Q$ then $\mathcal{G}^p[\![Q]\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P]\!]^{\imath_n}$.*
*If $P \rightarrow_{\alpha^{(\ell,\imath)}} Q$ then $\mathcal{G}^p_\star[\![Q]\!] \leq_{\mathfrak{T}ex} \mathcal{G}^p_\star[\![P]\!]$.*

**Proof.** We will prove the first part, which will immediately illustrate the correctness of the second part.

We proceed by induction on the inference of $P \rightarrow_{\widetilde{\ell\imath}} Q$ as defined in Table 2.3. There are two parts in Table 2.3: Sequential component part and Model component part. We also know that all sequential components essentially are the elements of model component part, so the sequential components could be considered as the "axioms" among all inference rules. In addition, ConstS in Model component will represent any sequential components, so if we successfully prove ConstS, it is straightforward to see the result applies to all sequential components: Prefix, Choice1, Choice2 and ConstC.

**The case [Prefix,Choice1,Choice2 and ConstC ]**: Refer proof of ConstS.

**The case [ConstS ]**: This component could be regarded as the axiom of all model component rule. The induction hypothesis is $P \rightarrow_{\alpha^\ell} P'$, from Lemma 4.1, we have $\mathcal{G}^s_\star[\![P']\!] \leq_{\mathfrak{T}} \mathcal{G}^s_\star[\![P]\!]$, we also have $\mathcal{G}^s_\star[\![P]\!] = \mathcal{G}^s_\star[\![S]\!]$. Thus we get $\mathcal{G}^s_\star[\![P']\!] \leq_{\mathfrak{T}} \mathcal{G}^s_\star[\![S]\!]$.

From the definition of $\mathcal{G}^p[\![P']\!]^{\imath_n}$, $\mathcal{G}^p[\![S]\!]^{\imath_n}$ and the fact $\mathcal{G}^s_\star[\![P']\!] \leq_{\mathfrak{T}} \mathcal{G}^s_\star[\![S]\!]$, it is easily seen that $\mathcal{G}^p[\![P']\!]^{\imath_n} \leq_{\mathfrak{T}} \mathcal{G}^p[\![S]\!]^{\imath_n}$, and this proves the result.

**The case [Coop1 ]**: From the induction hypothesis we have
$\mathcal{G}^p[\![P_1']\!]^{\imath_n 0} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P_1]\!]^{\imath_n 0}$.
If we add $\sqcup_{\mathfrak{T}ex}\mathcal{G}^p[\![P_2]\!]^{\imath_n 1}$ to its both sides, we get

$$\begin{array}{rl} & \mathcal{G}^p[\![P_1']\!]^{\imath_n 0} \sqcup_{\mathfrak{T}ex} \mathcal{G}^p[\![P_2]\!]^{\imath_n 1} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P_1]\!]^{\imath_n 0} \sqcup_{\mathfrak{T}ex} \mathcal{G}^p[\![P_2]\!]^{\imath_n 1} \\ \Leftrightarrow & \mathcal{G}^p[\![P_1' \underset{L}{\bowtie} P_2]\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} \end{array}$$

this proves the result.

**The case [Coop2 ]**: Analogous.

**The case [Coop3 ]**: From the induction hypothesis we have
$\mathcal{G}^p[\![P_1']\!]^{\imath_n 0} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P_1]\!]^{\imath_n 0}$ and $\mathcal{G}^p[\![P_2']\!]^{\imath_n 1} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P_2]\!]^{\imath_n 1}$.
so we have

$$\begin{array}{rl} & \mathcal{G}^p[\![P_1']\!]^{\imath_n 0} \sqcup_{\mathfrak{T}ex} \mathcal{G}^p[\![P_2']\!]^{\imath_n 1} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P_1]\!]^{\imath_n 0} \sqcup_{\mathfrak{T}ex} \mathcal{G}^p[\![P_2]\!]^{\imath_n 1} \\ \Leftrightarrow & \mathcal{G}^p[\![P_1' \underset{L}{\bowtie} P_2']\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} \end{array}$$

this proves the result.

**The case [Hiding1 ]:** From the induction hypothesis we have $\mathcal{G}^p[\![P']\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P]\!]^{\imath_n}$. We also know $\mathcal{G}^p[\![P']\!]^{\imath_n} = \mathcal{G}^p[\![P'/L]\!]^{\imath_n}$ and $\mathcal{G}^p[\![P]\!]^{\imath_n} = \mathcal{G}^p[\![P/L]\!]^{\imath_n}$. So it is straightforward to see that

$$\mathcal{G}^p[\![P'/L]\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{G}^p[\![P/L]\!]^{\imath_n}.$$

this proves the result.

**The case [Hiding2 ]:** Analogous.

This complete the proof of the first part. For the second part, since $\mathcal{G}^p_\star[\![Q]\!] = \mathcal{G}^p[\![Q]\!]^{\imath_0}$ and $\mathcal{G}^p_\star[\![P]\!] = \mathcal{G}^p[\![P]\!]^{\imath_0}$, when combined with the first part, it is straightforward to see the second part holds. □

Since $(\mathfrak{T}, \leq_{\mathfrak{T}})$ admits infinite ascending chains we need to show that the naive implementation of calculating $\mathcal{G}^s_\star$ will in fact terminate.In [16], the authors propose an Lemma which fit in our case pretty well, here we just give the lemma without proof.

**Lemma 4.3** *Using the notation of Table 4.1 we have*

$$env_{\mathcal{G}} \quad = \quad \mathcal{F}^k_{\mathcal{G}}(env_{\perp_{\mathfrak{T}}})$$

*where k is number of sequential components of the program.*

From this lemma, we could calculate the $env_{\mathcal{G}}$ and $\mathcal{G}^s_\star[\![S]\!]$ without any problem. Consequently, $\mathcal{G}^p_\star[\![P]\!]$ could be computed based on the result of $\mathcal{G}^s_\star[\![S]\!]$.

## 4.3 Killed Actions

Now we turn to find the definitions of $\mathcal{K}^s_\star$ and $\mathcal{K}^p_\star$. Similarly to the generated action, let us consider prefix combinator as expressed in the process $(\alpha^\ell, r).S$. Clearly, once $(\alpha^\ell, r)$ has been executed it will no longer be exposed. Thus a first suggestion may be to take $\mathcal{K}^s_\star[\![(\alpha^\ell, r).S]\!](\ell) = \perp_{\mathfrak{M}}[\ell \mapsto 1]$. However, like $\mathcal{G}^s_\star$, we need to consider same label may occur several times in a process, thus we will compute an *under*-approximation as it takes the *greatest lower bound* of the information available

$$\mathcal{K}^s_\star[\![(\alpha^\ell, r).S]\!]\ell' = \begin{cases} \perp_{\mathfrak{M}}[\ell \mapsto 1] \sqcap_{\mathfrak{M}} \mathcal{K}^s_\star[\![S]\!]\ell' & \text{if } \ell' = \ell \\ \mathcal{K}^s_\star[\![S]\!]\ell' & \text{if } \ell' \neq \ell \end{cases}$$

Exposed actions for let $C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k$ in $P_0$

$$\begin{array}{rcl}
\mathcal{K}^s[\![(\alpha^\ell, r).S]\!]env & = & \top_{\mathfrak{T}}[\ell \mapsto \bot_{\mathfrak{M}}[\ell \mapsto 1]] \sqcap_{\mathfrak{T}} \mathcal{K}^s[\![S]\!]env \\
\mathcal{K}^s[\![S_1 + S_2]\!]env & = & \text{let } [(\alpha_1^{\ell_1}, r_1).Q_1, \cdots, (\alpha_n^{\ell_n}, r_n).Q_n] = \mathcal{H}[\![S_1 + S_2]\!] \\
 & & \text{in } \sqcap_{\mathfrak{T}\ i\in(1,\cdots,n)}(\top_{\mathfrak{T}}[\ell_i \mapsto M] \sqcap_{\mathfrak{T}} \mathcal{K}^s[\![Q_i]\!]env) \\
 & & \qquad \textit{where } M = \mathcal{E}^s[\![\Sigma_{i\in(1,\cdots,n)}(\alpha_i^{\ell_i}, r_i).Q_i]\!]env_{\mathcal{E}} \\
\mathcal{K}^s[\![C]\!]env & = & env(C) \\
\mathcal{K}^s_\star[\![S]\!] & = & \mathcal{K}^s[\![S]\!]env_{\mathcal{K}} \\
\text{where}\mathcal{F}_{\mathcal{K}} & = & [C_1 \mapsto \mathcal{K}^s[\![S_1]\!]env, \cdots, C_k \mapsto \mathcal{K}^s[\![S_k]\!]env] \\
\text{and } env_{\top_{\mathfrak{T}}} & = & [C_1 \mapsto \top_{\mathfrak{T}}, \cdots, C_k \mapsto \top_{\mathfrak{T}}] \\
\text{and } env_{\mathcal{K}} & = & \sqcap_{j\geq 0}\mathcal{F}^j_{\mathcal{K}}(env_{\top_{\mathfrak{T}}}) \\
\mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^\imath & = & \mathcal{K}^p[\![P_1]\!]^{\imath 0} \sqcap_{\mathfrak{T}ex} \mathcal{K}^p[\![P_2]\!]^{\imath 1} \\
\mathcal{K}^p[\![P/L]\!]^\imath & = & \mathcal{K}^p[\![P]\!]^\imath \\
\mathcal{K}^p[\![S]\!]^\imath & = & \text{let } [\ell_1 \mapsto \{\ell_1 \mapsto n_{11}, \cdots, \ell_n \mapsto n_{1n}\}, \cdots, \\
 & & \qquad \ell_n \mapsto \{\ell_1 \mapsto n_{n1}, \cdots, \ell_n \mapsto n_{nn}\}] = \mathcal{K}^s_\star[\![S]\!] \\
 & & \text{in } \top_{\mathfrak{T}ex}[(\ell_j, \imath) \mapsto \bot_{\mathfrak{M}ex}[(\ell_k, \imath) \mapsto n_{jk}]] \\
 & & \ \text{where } \ell_j, \ell_k \in \text{domExlayer}(\bot_{\mathfrak{M}ex}, \imath) \text{ and } j,k \in \{1, \cdots, n\} \\
\mathcal{K}^p_\star[\![P]\!] & = & \mathcal{K}^p[\![P]\!]^0
\end{array}$$

Table 4.2: $\mathcal{K}^s$ and $\mathcal{K}^p$ function

it could be rewritten as:

$$\mathcal{K}^s_\star[\![(\alpha^\ell, r).S]\!] = \top_{\mathfrak{T}}[\ell \mapsto M] \sqcap_{\mathfrak{T}} \mathcal{K}^s_\star[\![S]\!] \qquad \text{where } M = \bot_{\mathfrak{M}}[\ell \mapsto 1]$$

$M$ also equals to $\mathcal{E}^s_\star[\![(\alpha^\ell, r).S]\!]$.

Now we are going to define the killed function formally, and we will use *under*-approximation as it always safe to kill fewer actions.

1. **Function** $\mathcal{K}$

$$\mathcal{K}^s : \mathbf{S}_{proc} \rightarrow (PN \rightarrow \mathfrak{T}) \rightarrow \mathfrak{T}$$

$$\mathcal{K}^p : \mathbf{P}_{proc} \rightarrow \mathfrak{T}_{ex}$$

For function $\mathcal{K}^s$, it takes an environment as the parameter which provides relevant information for the process names and is defined in Table 4.2. The prefix clauses are much as one should expect from the explanation above, in particular we may note that the operation $\sqcap_{\mathfrak{T}}$ is used to combine information throughout other clauses and represents the *under*-approximation characteristic of this function. Also we should notice that the $M$ in the clause for summations actually equals $\mathcal{E}^s[\![\Sigma_{i\in(1,\cdots,n)}(\alpha_i^{\ell_i}, r_i).Q_i]\!]env_{\mathcal{E}}$, reflecting that *all* the exposed actions of *all* the prefix clauses that the $S_1$ and $S_2$ could reach are indeed killed when one of them has been selected for the reduction step. And all the reachable prefix clauses from $S_1$ and $S_2$ could be calculated by function $\mathcal{H}$ that will be discussed later.

The recursive definitions in prefix clause give rise to a monotonic function $\mathcal{F}_{\mathcal{K}}$: $(\mathbf{PN} \rightarrow \mathfrak{T}) \rightarrow (\mathbf{PN} \rightarrow \mathfrak{T})$ on a complete lattice (cf. Fact 3.1,Fact 3.2). and hence Tarski's fixed point theorem ensures that the least fixed point $env_{\mathcal{K}}$ exists. Once more the function turns out to be co-continuous because $\mathfrak{T}$ contains no infinite decreasing chains and hence the Kleene formulation of the fixed point is permissible. And we could define function

$$\mathcal{K}^s_\star : \mathbf{S}_{proc} \rightarrow \mathfrak{T}$$

and this function will give us all information killed by sequential component in the program.

For function $\mathcal{K}^p$, it will always take layer $\imath$ as parameters and finally append the correct layer to each $S$ sequential component. Specifically, for cooperation operator, it will combine the result of two components by $\sqcap_{\mathfrak{T}_{ex}}$ operation. The clause for the $P/L$ will simply ignores the hidden set $L$. The clause for constant model combinator will borrow the result get from $\mathcal{K}^s_\star$ step and compute the denotation of this sequential component to the overall $\mathfrak{T}_{ex}$. It is worth pointing out that only the label and layer pair belong to the $\mathfrak{T}_{ex}$ set should be set up by each constant combinator clause and we use $j$ and $k$ to control it.

We can now define the function

$$\mathcal{K}^p_\star : \mathbf{P}_{proc} \rightarrow \mathfrak{T}_{ex}$$

Simply as $\mathcal{K}^p_\star[\![P]\!] = \mathcal{K}^p[\![P]\!]^0$.The parameter 0 takes charge of layer initialization and represents the value issued to the top layer.

2. **Function** $\mathcal{H}$
Function $\mathcal{H}$ will collect all prefix clauses from any sequential component. We define it as follows:

$$\mathcal{H} : \mathbf{Proc} \rightarrow \wp(\mathbf{Prefix})$$

where **Prefix** is the domain of prefix components in the program. $\wp$(**Prefix**) is the set of those components. The formal description is in Table 4.3. And the $\cup$ operation could be found in each clause, meaning that the function will collect *all* relevant prefix clauses.

Exposed actions for let $C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k$ in $P_0$

$$\begin{array}{rcl} \mathcal{H}[\![(\alpha^\ell, r).S]\!] & = & \text{let } R = \emptyset \\ & & \text{in } R \cup (\alpha^\ell, r).S \\ \mathcal{H}[\![S_1 + S_2]\!] & = & \text{let } R_1 = \mathcal{H}[\![S_1]\!] and R_2 = \mathcal{H}[\![S_2]\!] \\ & & \text{in } R_1 \cup R_2 \\ \mathcal{H}[\![C_k]\!]_{k \in I} & = & \mathcal{H}[\![S_k]\!] \end{array}$$

Table 4.3: $\mathcal{H}$ function

**Example 4.2** *We will compute the $\mathcal{K}^p_\star$ for programs introduced in Example 2.1.*

| $\ell_{ex}$ | $\mathcal{K}^p_\star[\![S \underset{\{g,p\}}{\bowtie} Q]\!](\ell_{ex})$ |
|---|---|
| $(1,00)$ | $\bot_{\mathfrak{M}ex}[(1,00) \mapsto 1]$ |
| $(2,00)$ | $\bot_{\mathfrak{M}ex}[(2,00) \mapsto 1]$ |
| $(3,01)$ | $\bot_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $(4,01)$ | $\bot_{\mathfrak{M}ex}[(4,01) \mapsto 1]$ |
| $(5,01)$ | $\bot_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ |

| $\ell_{ex}$ | $\mathcal{K}^p_\star[\![(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q]\!](\ell_{ex})$ | $\mathcal{K}^p_\star[\![(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q]\!](\ell_{ex})$ |
|---|---|---|
| $(1,000)$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1]$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1]$ |
| $(2,000)$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1]$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1]$ |
| $(1,001)$ | $\bot_{\mathfrak{M}ex}[(1,001) \mapsto 1]$ | $\bot_{\mathfrak{M}ex}[(1,001) \mapsto 1]$ |
| $(2,001)$ | $\bot_{\mathfrak{M}ex}[(2,001) \mapsto 1]$ | $\bot_{\mathfrak{M}ex}[(2,001) \mapsto 1]$ |
| $(3,01)$ | $\bot_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ | $\bot_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $(4,01)$ | $\bot_{\mathfrak{M}ex}[(4,01) \mapsto 1]$ | $\bot_{\mathfrak{M}ex}[(4,01) \mapsto 1]$ |
| $(5,01)$ | $\bot_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ | $\bot_{\mathfrak{M}ex}[(3,01) \mapsto 1, (5,01) \mapsto 1]$ |

*It could be seen clearly that $(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ and $(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ have exactly the same results from $\mathcal{K}^p_\star$ functions.*

**Lemma 4.4** *If $P \rightarrow_{\tilde{\ell}} Q$ then $\mathcal{K}^s_\star[\![P]\!] \leq_{\mathfrak{T}} \mathcal{K}^s_\star[\![Q]\!]$ and $\mathcal{K}^s_\star[\![P]\!](\tilde{\ell}) \leq_{\mathfrak{M}} \mathcal{E}^s_\star[\![P]\!]$.*

**Proof.** We proceed by induction on the inference of $P \rightarrow_{\tilde{\ell}} Q$ as defined in Table 2.2. For each case, we use index 1 and 2 to differentiate the first and second part of the proof.

**The case** [**Prefix** ]:

1.

$$\begin{array}{rcl}\mathcal{K}^s_\star[\![(\alpha^\ell, r).S]\!] & = & \mathcal{K}^s[\![(\alpha^\ell, r).S]\!]env_K \\ & = & \top_{\mathfrak{T}}[\ell \mapsto \bot_{\mathfrak{M}}[\ell \mapsto 1]] \sqcap_{\mathfrak{T}} \mathcal{K}^s[\![S]\!]env_K \\ & \leq_{\mathfrak{T}} & \mathcal{K}^s[\![S]\!]env_K = \mathcal{K}^s_\star[\![S]\!]\end{array}$$

as required.

2.

$$\begin{array}{rcl}\mathcal{K}^s_\star[\![(\alpha^\ell, r).S]\!](\tilde{\ell}) & = & \mathcal{K}^s[\![(\alpha^\ell, r).S]\!]env_K(\tilde{\ell}) \\ & = & \top_{\mathfrak{T}}[\ell \mapsto \bot_{\mathfrak{M}}[\ell \mapsto 1]](\tilde{\ell}) \sqcap_{\mathfrak{T}} \mathcal{K}^s[\![S]\!]env_K(\tilde{\ell}) \\ & = & \mathcal{E}^s_\star[\![(\alpha^\ell, r).S]\!] \sqcap_{\mathfrak{T}} \mathcal{K}^s[\![S]\!]env_K(\tilde{\ell}) \\ & \leq_{\mathfrak{M}} & \mathcal{E}^s_\star[\![(\alpha^\ell, r).S]\!]\end{array}$$

as required.

**The case** [**Choice1** ]:

1. From Induction hypothesis, we know
   $\mathcal{K}^s_\star[\![S_1']\!] \leq_{\mathfrak{T}} \mathcal{K}^s_\star[\![S_1]\!]$ which means $\mathcal{K}^s[\![S_1]\!]env_K \leq_{\mathfrak{T}} \mathcal{K}^s[\![S_1']\!]env_K$.
   Since $S_1$ is part of $S_1 + S_2$, when calculating $\mathcal{K}^s[\![S_1]\!]env_K$, we should take the effect caused by $S_1 + S_2$ into account. Here we shall have

   $$\begin{array}{rcl}\mathcal{K}^s[\![S_1]\!]env^{S_1+S_2} & = & \text{let } [(\alpha_1^{\ell_1}, r_1).Q_1, \cdots, (\alpha_n^{\ell_n}, r_n).Q_n] = \mathcal{H}[\![S_1 + S_2]\!] \\ & & \ \text{and } [(\alpha_1^{\ell_1}, r_1).Q_1, \cdots, (\alpha_k^{\ell_k}, r_k).Q_k] = \mathcal{H}[\![S_1]\!] \\ & & \ \text{where } k <= n \\ & & \text{in } \sqcap_{\mathfrak{T}\ i\in(1,\cdots,k)}(\top_{\mathfrak{T}}[\ell_i \mapsto M] \sqcap_{\mathfrak{T}} \mathcal{K}^s[\![Q_i]\!]env) \\ & & \qquad \textit{where } M = \mathcal{E}^s[\![\Sigma_{i\in(1,\cdots,n)}(\alpha_i^{\ell_i}, r_i).Q_i]\!]env_{\mathcal{E}}\end{array}$$

   recall that

   $$\begin{array}{rcl}\mathcal{K}^s[\![S_1 + S_2]\!]env & = & \text{let } [(\alpha_1^{\ell_1}, r_1).Q_1, \cdots, (\alpha_n^{\ell_n}, r_n).Q_n] = \mathcal{H}[\![S_1 + S_2]\!] \\ & & \text{in } \sqcap_{\mathfrak{T}\ i\in(1,\cdots,n)}(\top_{\mathfrak{T}}[\ell_i \mapsto M] \sqcap_{\mathfrak{T}} \mathcal{K}^s[\![Q_i]\!]env) \\ & & \qquad \textit{where } M = \mathcal{E}^s[\![\Sigma_{i\in(1,\cdots,n)}(\alpha_i^{\ell_i}, r_i).Q_i]\!]env_{\mathcal{E}}\end{array}$$

   thus, it is straightforward to see that $\mathcal{K}^s[\![S_1+S_2]\!]env_K \leq_{\mathfrak{T}} \mathcal{K}^s[\![S_1]\!]env_K^{S_1+S_2}$, so we have $\mathcal{K}^s[\![S_1 + S_2]\!]env_K \leq_{\mathfrak{T}} \mathcal{K}^s[\![S_1]\!]env_K^{S_1+S_2} \leq_{\mathfrak{T}} \mathcal{K}^s[\![S_1']\!]env_K$ and this proves the result.

2. from $\mathcal{K}^s[\![S_1 + S_2]\!]env$ above, it is easy to see that
   $\mathcal{K}^s_\star[\![S_1+S_2]\!](\tilde{\ell}) \leq_{\mathfrak{M}} \mathcal{E}^s_\star[\![S_1+S_2]\!]$ since clause for $M$ equals the exposed actions of $S_1 + S_2$.

**The case [Choice2 ]**: Analogous.

**The case [ConstC ]**:

1. From the induction hypothesis, we know $\mathcal{K}^s_\star[\![S]\!] \leq_{\mathfrak{T}} \mathcal{K}^s_\star[\![S']\!]$. We also know $\mathcal{K}^s_\star[\![C]\!] = \mathcal{K}^s_\star[\![S]\!]$ because $C \overset{def}{=} S$, so $\mathcal{K}^s_\star[\![C]\!] \leq_{\mathfrak{T}} \mathcal{K}^s_\star[\![S']\!]$ as required.
2. From the induction hypothesis, we know $\mathcal{K}^s_\star[\![S]\!](\tilde{\ell}) \leq_{\mathfrak{M}} \mathcal{E}^s_\star[\![S]\!]$ We also know $\mathcal{K}^s_\star[\![C]\!](\tilde{\ell}) = \mathcal{K}^s_\star[\![S]\!](\tilde{\ell})$ because $C \overset{def}{=} S$, so $\mathcal{K}^s_\star[\![C]\!](\tilde{\ell}) \leq_{\mathfrak{M}} \mathcal{E}^s_\star[\![S]\!]$

□

**Lemma 4.5** *If $P \rightarrow_{\alpha^{(\ell,\imath)}} Q$ then $\mathcal{K}^p[\![P]\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![Q]\!]^{\imath_n}$ and $\mathcal{K}^p[\![P]\!]^{\imath_n}(\widetilde{\ell\imath}) \leq_{\mathfrak{M}ex} \mathcal{E}^p[\![P]\!]^{\imath_n}$.*
*If $P \rightarrow_{\alpha^{(\ell,\imath)}} Q$ then $\mathcal{K}^p_\star[\![P]\!] \leq_{\mathfrak{T}ex} \mathcal{K}^p_\star[\![Q]\!]$ and $\mathcal{K}^s_\star[\![P]\!](\widetilde{\ell\imath}) \leq_{\mathfrak{M}ex} \mathcal{E}^s_\star[\![P]\!]$.*

**Proof.** We will prove the first part, which will immediately illustrate the correctness of the second part.

We proceed by induction on the inference of $P \rightarrow_{\widetilde{\ell\imath}} Q$ as defined in Table 2.3. The method we adopt here is exactly the same as the one used when we prove the counterpart in Lemma 4.2.

**The case [Prefix,Choice1,Choice2 and ConstC ]**: Refer proof of ConstS.

**The case [ConstS ]**:

1. This component could be regarded as the axiom of all model component rule. The induction hypothesis is $P \rightarrow_{\alpha^\ell} P'$, from Lemma 4.4, we have $\mathcal{K}^s_\star[\![P]\!] \leq_{\mathfrak{T}} \mathcal{K}^s_\star[\![P']\!]$, we also have $\mathcal{K}^s_\star[\![P]\!] = \mathcal{K}^s_\star[\![S]\!]$. Thus we get $\mathcal{K}^s_\star[\![S]\!] \leq_{\mathfrak{T}} \mathcal{K}^s_\star[\![P']\!]$.
   From the definition of $\mathcal{K}^p[\![P']\!]^{\imath_n}$, $\mathcal{K}^p[\![S]\!]^{\imath_n}$ and the fact $\mathcal{K}^s_\star[\![S]\!] \leq_{\mathfrak{T}} \mathcal{K}^s_\star[\![P']\!]$, it is easily seen that $\mathcal{K}^p[\![S]\!]^{\imath_n} \leq_{\mathfrak{T}} \mathcal{K}^p[\![P']\!]^{\imath_n}$, and this proves the result.
2. The induction hypothesis is $P \rightarrow_{\alpha^\ell} P'$, from Lemma 4.4, we have $\mathcal{K}^s_\star[\![P]\!](\ell) \leq_{\mathfrak{M}} \mathcal{E}^s_\star[\![P]\!]$, we also have $\mathcal{K}^s_\star[\![P]\!] = \mathcal{K}^s_\star[\![S]\!]$ and $\mathcal{E}^s_\star[\![P]\!] = \mathcal{E}^s_\star[\![S]\!]$, so we have $\mathcal{K}^s_\star[\![S]\!](\ell) \leq_{\mathfrak{M}} \mathcal{E}^s_\star[\![S]\!]$.
   From the definition of $\mathcal{K}^p[\![S]\!]^{\imath_n}$, $\mathcal{E}^p[\![S]\!]^{\imath_n}$ and the fact $\mathcal{K}^s_\star[\![S]\!](\ell) \leq_{\mathfrak{M}} \mathcal{E}^s_\star[\![S]\!]$, it is easily seen that $\mathcal{K}^p[\![S]\!]^{\imath_n}(\ell, \imath_n) \leq_{\mathfrak{M}ex} \mathcal{E}^p[\![S]\!]^{\imath_n}$, and this proves the result.

**The case [Coop1 ]:**

1. From the induction hypothesis we have
$\mathcal{K}^p[\![P_1]\!]^{\imath_n 0} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P_1']\!]^{\imath_n 0}$.
If we add $\sqcap_{\mathfrak{T}ex}\mathcal{K}^p[\![P_2]\!]^{\imath_n 1}$ to its both sides, we get

$$\begin{array}{rl} & \mathcal{K}^p[\![P_1]\!]^{\imath_n 0} \sqcap_{\mathfrak{T}ex} \mathcal{K}^p[\![P_2]\!]^{\imath_n 1} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P_1']\!]^{\imath_n 0} \sqcap_{\mathfrak{T}ex} \mathcal{K}^p[\![P_2]\!]^{\imath_n 1} \\ \Leftrightarrow & \mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P_1' \underset{L}{\bowtie} P_2]\!]^{\imath_n} \end{array}$$

this proves the result.

2. From the induction hypothesis we have
$\mathcal{K}^p[\![P_1]\!]^{\imath_n 0}(\ell_1, \imath_1) \leq_{\mathfrak{M}ex} \mathcal{E}^p[\![P_1]\!]^{\imath_n 0}$, we can calculate

$$\begin{array}{rcl} \mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} & = & \mathcal{K}^p[\![P_1]\!]^{\imath 0} \sqcap_{\mathfrak{T}ex} \mathcal{K}^p[\![P_2]\!]^{\imath 1} \\ & \leq_{\mathfrak{T}ex} & \mathcal{K}^p[\![P_1]\!]^{\imath 0} \end{array}$$

So we have

$$\begin{array}{rcl} \mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n}(\ell_1, \imath_1) & \leq_{\mathfrak{M}ex} & \mathcal{K}^p[\![P_1]\!]^{\imath 0}(\ell_1, \imath_1) \\ & \leq_{\mathfrak{M}ex} & \mathcal{E}^p[\![P_1]\!]^{\imath_n 0} \\ & \leq_{\mathfrak{M}ex} & \mathcal{E}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} \end{array}$$

As required.

**The case [Coop2 ]:** Analogous.

**The case [Coop3 ]:**

1. From the induction hypothesis we have
$\mathcal{K}^p[\![P_1]\!]^{\imath_n 0} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P_1']\!]^{\imath_n 0}$ and $\mathcal{K}^p[\![P_2]\!]^{\imath_n 1} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P_2']\!]^{\imath_n 1}$.
so we have

$$\begin{array}{rl} & \mathcal{K}^p[\![P_1]\!]^{\imath_n 0} \sqcap_{\mathfrak{T}ex} \mathcal{K}^p[\![P_2]\!]^{\imath_n 1} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P_1']\!]^{\imath_n 0} \sqcap_{\mathfrak{T}ex} \mathcal{K}^p[\![P_2']\!]^{\imath_n 1} \\ \Leftrightarrow & \mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P_1' \underset{L}{\bowtie} P_2']\!]^{\imath_n} \end{array}$$

this proves the result.

2. From the induction hypothesis we have
$\mathcal{K}^p[\![P_1]\!]^{\imath_n 0}(\ell_1, \imath_1) \leq_{\mathfrak{M}ex} \mathcal{E}^p[\![P_1]\!]^{\imath_n 0}$ and $\mathcal{K}^p[\![P_2]\!]^{\imath_n 1}(\ell_1, \imath_1) \leq_{\mathfrak{M}ex} \mathcal{E}^p[\![P_2]\!]^{\imath_n 1}$.
We calculate

$$\mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} = \mathcal{K}^p[\![P_1]\!]^{\imath_n 0} \sqcap_{\mathfrak{T}ex} \mathcal{K}^p[\![P_2]\!]^{\imath_n 1}$$

so we have

$$\begin{array}{rcl} \mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n}(\ell_l, \imath_1)(\ell_2, \imath_2) & = & \mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n}(\ell_l, \imath_1) +_{\mathfrak{M}ex} \\ & & \mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n}(\ell_2, \imath_2) \\ & \leq_{\mathfrak{M}ex} & \mathcal{K}^p[\![P_1]\!]^{\imath_n 0}(\ell_1, \imath_1) +_{\mathfrak{M}ex} \mathcal{K}^p[\![P_2]\!]^{\imath_n 1}(\ell_2, \imath_2) \\ & \leq_{\mathfrak{M}ex} & \mathcal{E}^p[\![P_1]\!]^{\imath_n 0} +_{\mathfrak{M}ex} \mathcal{E}^p[\![P_2]\!]^{\imath_n 1} \\ & = & \mathcal{E}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} \end{array}$$

as required.

**The case [Hiding1 ]:**

1. From the induction hypothesis we have $\mathcal{K}^p[\![P]\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P']\!]^{\imath_n}$. We also know $\mathcal{K}^p[\![P']\!]^{\imath_n} = \mathcal{K}^p[\![P'/L]\!]^{\imath_n}$ and $\mathcal{K}^p[\![P]\!]^{\imath_n} = \mathcal{K}^p[\![P/L]\!]^{\imath_n}$. So it is straightforward to see that
$$\mathcal{K}^p[\![P/L]\!]^{\imath_n} \leq_{\mathfrak{T}ex} \mathcal{K}^p[\![P'/L]\!]^{\imath_n}.$$
this proves the result.
2. From the induction hypothesis we have $\mathcal{K}^p[\![P]\!]^{\imath_n}(\ell,\imath) \leq_{\mathfrak{M}ex} \mathcal{E}^p[\![P]\!]^{\imath_n}$ We also know $\mathcal{K}^p[\![P]\!]^{\imath_n} = \mathcal{K}^p[\![P/L]\!]^{\imath_n}$ and $\mathcal{E}^p[\![P]\!]^{\imath_n} = \mathcal{E}^p[\![P/L]\!]^{\imath_n}$. So it is straightforward to see that
$$\mathcal{K}^p[\![P/L]\!]^{\imath_n}(\ell,\imath) \leq_{\mathfrak{M}ex} \mathcal{E}^p[\![P/L]\!]^{\imath_n}.$$

**The case [Hiding2 ]:** Analogous.

This complete the proof of the first part. For the second part, since $\mathcal{K}^p_\star[\![Q]\!] = \mathcal{K}^p[\![Q]\!]^{\imath_0}$, $\mathcal{K}^p_\star[\![P]\!] = \mathcal{K}^p[\![P]\!]^{\imath_0}$ and $\mathcal{E}^p_\star[\![P]\!] = \mathcal{E}^p[\![P]\!]^{\imath_0}$, when combined with the first part, it is straightforward to see the second part holds. □

Turning to the implementation of the least fixed point we use a simple iterative procedure that is terminated when $\mathcal{F}^j_{\mathcal{K}}(env_{\top_{\mathfrak{T}}}) = \mathcal{F}^{j+1}_{\mathcal{K}}(env_{\top_{\mathfrak{T}}})$. This procedure works because the lattice of interest admits no infinite descending chains.

## 4.4 The Transfer Function

In the classic Bit Vector Frameworks the transfer function looks like the following formula:
$$f_{block}(E) = (E \setminus kill_{block}) \cup gen_{block}$$
For a forward analysis, $E$ is the information holding at the entry of the block, $kill_{block}$ is the information killed by executing this block, while $gen_{block}$ is the information created by it. In our scenario, $E$ is the extra extended multiset $\mathfrak{M}_{ex}$ that tells all current available exposed actions. The block itself will be identified by '$\widetilde{\ell\imath}$' that may be executed, where $\widetilde{\ell\imath} \in \textbf{Labex} \cup (\textbf{Labex} \times \textbf{Labex})$. We come up with our transfer function which takes the following form:
$$\texttt{transfer}_{\widetilde{\ell\imath}}(E) = (E -_{\mathfrak{M}} \mathcal{K}^p_\star[\![P]\!](\widetilde{\ell\imath})) +_{\mathfrak{M}} \mathcal{G}^p_\star[\![P]\!](\widetilde{\ell\imath})$$

We used $\mathcal{K}^p_\star$ and$\mathcal{G}^p_\star$ which are discussed in the previous section. And the transfer function depicts how the information evolves during the execution of the model processes.

From the formula we also know that the $\texttt{transfer}_{\widetilde{\ell\imath}}$ function is monotonic.

The following result tells that the transfer functions $\texttt{transfer}_{\widetilde{\ell\imath}}$ defined above provide safe approximations to the exposed actions of the resulting process:

**Proposition 4.6** *If* $P \rightarrow_{\widetilde{\ell\imath}} Q$ *then* $\mathcal{E}^p[\![Q]\!]^{\imath_n} \leq_{\mathfrak{M}ex} \texttt{transfer}_{\widetilde{\ell\imath}}(\mathcal{E}^p[\![P]\!]^{\imath_n})$.

*If* $P \rightarrow_{\widetilde{\ell\imath}} Q$ *then* $\mathcal{E}^p_\star[\![Q]\!] \leq_{\mathfrak{M}ex} \texttt{transfer}_{\widetilde{\ell\imath}}(\mathcal{E}^p_\star[\![P]\!])$.

**Proof.** We will proof the first part, which will immediately illustrate the correctness of the second part.

We proceed by induction on the inference of $P \rightarrow_{\widetilde{\ell\imath}} Q$ as defined in Table 2.3.

**The case [Prefix ]:** First observe that $\mathcal{G}^p[\![(\alpha^\ell, r).S]\!]^{\imath_n}(\ell, \imath_n) \geq_{\mathfrak{M}ex} \mathcal{E}^p[\![S]\!]^{\imath_n}$. Then

$$\begin{aligned}&(\mathcal{E}^p[\![(\alpha^\ell, r).S]\!]^{\imath_n} -_{\mathfrak{M}ex} \mathcal{K}^p[\![(\alpha^\ell, r).S]\!]^{\imath_n}(\ell, \imath_n)) +_{\mathfrak{M}ex} \mathcal{G}^p[\![(\alpha^\ell, r).S]\!]^{\imath_n}(\ell, \imath_n)\\ &\geq_{\mathfrak{M}ex} \mathcal{G}^p[\![(\alpha^\ell, r).S]\!]^{\imath_n}(\ell, \imath_n)\\ &\geq_{\mathfrak{M}ex} \mathcal{E}^p[\![S]\!]^{\imath_n}\end{aligned}$$

as required.

**The case [Choice1 ]:** From the induction hypothesis we have

$$(\mathcal{E}^p[\![S_1]\!]^{\imath_n} -_{\mathfrak{M}ex} \mathcal{K}^p[\![S_1]\!]^{\imath_n}(\ell_1, \imath_n)) +_{\mathfrak{M}ex} \mathcal{G}^p[\![S_1]\!]^{\imath_n}(\ell_1, \imath_n) \geq_{\mathfrak{M}ex} \mathcal{E}^p[\![S_1']\!]^{\imath_n}$$

We have $\mathcal{K}^p[\![S_1]\!]^{\imath_n}(\ell_1, \imath_n) \geq_{\mathfrak{M}ex} \mathcal{K}^p[\![S_1{+}S_2]\!]^{\imath_n}(\ell_1, \imath_n)$ and $\mathcal{G}^p[\![S_1]\!]^{\imath_n}(\ell_1, \imath_n) \leq_{\mathfrak{M}ex} \mathcal{G}^p[\![S_1 + S_2]\!]^{\imath_n}(\ell_1, \imath_n)$, we also have $\mathcal{E}^p[\![S_1 + S_2]\!]^{\imath_n} \geq_{\mathfrak{M}ex} \mathcal{E}^p[\![S1]\!]^{\imath_n}$.

So we get

$$\begin{aligned}&(\mathcal{E}^p[\![S_1 + S_2]\!]^{\imath_n} -_{\mathfrak{M}ex} \mathcal{K}^p[\![S_1 + S_2]\!]^{\imath_n}(\ell_1, \imath_n)) +_{\mathfrak{M}ex} \mathcal{G}^p[\![S_1 + S_2]\!]^{\imath_n}(\ell_1, \imath_n)\\ &\geq_{\mathfrak{M}ex} (\mathcal{E}^p[\![S_1]\!]^{\imath_n} -_{\mathfrak{M}ex} \mathcal{K}^p[\![S_1]\!]^{\imath_n}(\ell_1, \imath_n)) +_{\mathfrak{M}ex} \mathcal{G}^p[\![S_1]\!]^{\imath_n}(\ell_1, \imath_n)\\ &\geq_{\mathfrak{M}ex} \mathcal{E}^p[\![S_1']\!]^{\imath_n}\end{aligned}$$

where we have use the monotonicity of $+_{\mathfrak{M}ex}$ and that $-_{\mathfrak{M}ex}$ is monotonic in its left argument and anti-monotonic in its right argument as stated in Fact 3.5. This proves the result.

**The case [Choice2 ]**: Analogous.

**The case [ConstC ]**: From the induction hypothesis we have

$$(\mathcal{E}^p\llbracket S\rrbracket^{\imath_n} -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket S\rrbracket^{\imath_n}(\ell,\imath_n)) +_{\mathfrak{M}_{ex}} \mathcal{G}^p\llbracket S\rrbracket^{\imath_n}(\ell,\imath_n) \geq_{\mathfrak{M}_{ex}} \mathcal{E}^p\llbracket S'\rrbracket^{\imath_n}$$

Because $C \overset{def}{=} S$, we have $\mathcal{E}^p\llbracket S\rrbracket^{\imath_n} = \mathcal{E}^p\llbracket C\rrbracket^{\imath_n}, \mathcal{K}^p\llbracket S\rrbracket^{\imath_n} = \mathcal{K}^p\llbracket C\rrbracket^{\imath_n} and \mathcal{G}^p\llbracket S\rrbracket^{\imath_n} = \mathcal{G}^p\llbracket C\rrbracket^{\imath_n}$.

Thus we have

$$(\mathcal{E}^p\llbracket C\rrbracket^{\imath_n} -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket C\rrbracket^{\imath_n}(\ell,\imath_n)) +_{\mathfrak{M}_{ex}} \mathcal{G}^p\llbracket C\rrbracket^{\imath_n}(\ell,\imath_n) \geq_{\mathfrak{M}_{ex}} \mathcal{E}^p\llbracket S'\rrbracket^{\imath_n}$$

And this proves the result.

**The case [Coop1 ]**: From the induction hypothesis we have

$$\begin{array}{lll}
\mathcal{E}^p\llbracket P_1'\rrbracket^{\imath_n} & \leq_{\mathfrak{M}_{ex}} & (\mathcal{E}^p\llbracket P_1\rrbracket^{\imath_n 0} -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1\rrbracket^{\imath_n 0}(\ell_1,\imath_1)) +_{\mathfrak{M}_{ex}} \mathcal{G}^p\llbracket P_1\rrbracket^{\imath_n 0}(\ell_1,\imath_1) \\
 & \leq_{\mathfrak{M}_{ex}} & (\mathcal{E}^p\llbracket P_1\rrbracket^{\imath_n 0} -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)) \\
 & & +_{\mathfrak{M}_{ex}}\mathcal{G}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)
\end{array}$$

using $\mathcal{K}^p\llbracket P_1\rrbracket^{\imath_n 0}(\ell_1,\imath_1) \geq_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1+P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)$, $\mathcal{G}^p\llbracket P_1\rrbracket^{\imath_n 0}(\ell_1,\imath_1) \leq_{\mathfrak{M}_{ex}} \mathcal{G}^p\llbracket P_1+P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)$ and the monotonicity of $+_{\mathfrak{M}_{ex}}$ and that $-_{\mathfrak{M}_{ex}}$ is monotonic in its left argument and anti-monotonic in its right argument as stated in Fact 3.5.

We can calculate

$$\begin{array}{lll}
\mathcal{E}^p\llbracket P_1' \underset{L}{\bowtie} P_2\rrbracket^{\imath_n} & = & \mathcal{E}^p\llbracket P_1'\rrbracket^{\imath_{n0}} +_{\mathfrak{M}_{ex}} \mathcal{E}^p\llbracket P_2\rrbracket^{\imath_{n1}} \\
 & \leq_{\mathfrak{M}_{ex}} & (\mathcal{E}^p\llbracket P_1\rrbracket^{\imath_n 0} -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)) \\
 & & +_{\mathfrak{M}_{ex}}\mathcal{G}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)) +_{\mathfrak{M}_{ex}} \mathcal{E}^p\llbracket P_2\rrbracket^{\imath_1} \\
 & = & (\mathcal{E}^p\llbracket P_1\rrbracket^{\imath_n 0} +_{\mathfrak{M}_{ex}} \mathcal{E}^p\llbracket P_2\rrbracket^{\imath_1}) -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n}(\ell_1,\imath_1) \\
 & & +_{\mathfrak{M}_{ex}}\mathcal{G}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)) \\
 & = & (\mathcal{E}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n} -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)) \\
 & & +_{\mathfrak{M}_{ex}}\mathcal{G}^p\llbracket P_1 \underset{L}{\bowtie} P_2\rrbracket^{\imath_n}(\ell_1,\imath_1))
\end{array}$$

where we use $\mathcal{E}^p\llbracket P_1\rrbracket^{\imath_n 0} \geq_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1\rrbracket^{\imath_n 0}(\ell_1,\imath_1) \geq_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1+P_2\rrbracket^{\imath_n}(\ell_1,\imath_1)$ (Lemma 4.4) and Fact 3.6. And this proves the result.

**The case [Coop2 ]**: Analogous.

**The case [Coop3 ]**: As in the previous case, from induction hypothesis we have

$$\begin{array}{lll}
\mathcal{E}^p\llbracket P_1'\rrbracket^{\imath_n} & \leq_{\mathfrak{M}_{ex}} & \mathcal{E}^p\llbracket P_1\rrbracket^{\imath_n 0} -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_1\rrbracket^{\imath_n 0}(\ell_1,\imath_1) +_{\mathfrak{M}_{ex}} \mathcal{G}^p\llbracket P_1\rrbracket^{\imath_n 0}(\ell_1,\imath_1) \\
\mathcal{E}^p\llbracket P_2'\rrbracket^{\imath_n} & \leq_{\mathfrak{M}_{ex}} & \mathcal{E}^p\llbracket P_2\rrbracket^{\imath_n 1} -_{\mathfrak{M}_{ex}} \mathcal{K}^p\llbracket P_2\rrbracket^{\imath_n 1}(\ell_2,\imath_2) +_{\mathfrak{M}_{ex}} \mathcal{G}^p\llbracket P_2\rrbracket^{\imath_n 1}(\ell_2,\imath_2)
\end{array}$$

We can calculate

$$\begin{array}{rcl}
\mathcal{E}^p[\![P_1' \underset{L}{\bowtie} P_2']\!]^{\imath_n} & = & \mathcal{E}^p[\![P_1']\!]^{\imath_{n0}} +_{\mathfrak{M}_{ex}} \mathcal{E}^p[\![P_2']\!]^{\imath_{n1}} \\
 & \leq_{\mathfrak{M}_{ex}} & \mathcal{E}^p[\![P_1]\!]^{\imath_n 0} -_{\mathfrak{M}_{ex}} \mathcal{K}^p[\![P_1]\!]^{\imath_n 0}(\ell_1, \imath_1) +_{\mathfrak{M}_{ex}} \mathcal{G}^p[\![P_1]\!]^{\imath_n 0}(\ell_1, \imath_1) \\
 & & +_{\mathfrak{M}_{ex}} \mathcal{E}^p[\![P_2]\!]^{\imath_n 1} -_{\mathfrak{M}_{ex}} \mathcal{K}^p[\![P_2]\!]^{\imath_n 1}(\ell_2, \imath_2) +_{\mathfrak{M}_{ex}} \mathcal{G}^p[\![P_2]\!]^{\imath_n 1}(\ell_2, \imath_2) \\
 & = & (\mathcal{E}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n} -_{\mathfrak{M}_{ex}} \mathcal{K}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n}(\ell_1, \imath_1)) \\
 & & +_{\mathfrak{M}_{ex}} \mathcal{G}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath_n}(\ell_1, \imath_1))
\end{array}$$

using Lemma 4.4 and Fact 3.6. This proves the result.

**The case [Hiding1 ]**: From the induction hypothesis we have

$$(\mathcal{E}^p[\![P]\!]^{\imath_n} -_{\mathfrak{M}_{ex}} \mathcal{K}^p[\![P]\!]^{\imath_n}(\ell, \imath)) +_{\mathfrak{M}_{ex}} \mathcal{G}^p[\![P]\!]^{\imath_n}(\ell, \imath) \geq_{\mathfrak{M}_{ex}} \mathcal{E}^p[\![P']\!]^{\imath_n}$$

Because we have $\mathcal{E}^p[\![P'/L]\!]^{\imath_n} = \mathcal{E}^p[\![P']\!]^{\imath_n}, \mathcal{E}^p[\![P/L]\!]^{\imath_n} = \mathcal{E}^p[\![P]\!]^{\imath_n}$, $\mathcal{K}^p[\![P/L]\!]^{\imath_n} = \mathcal{K}^p[\![P]\!]^{\imath_n}$ and $\mathcal{G}^p[\![P/L]\!]^{\imath_n} = \mathcal{G}^p[\![P]\!]^{\imath_n}$.

Thus we have

$$(\mathcal{E}^p[\![P/L]\!]^{\imath_n} -_{\mathfrak{M}_{ex}} \mathcal{K}^p[\![P/L]\!]^{\imath_n}(\ell, \imath)) +_{\mathfrak{M}_{ex}} \mathcal{G}^p[\![P/L]\!]^{\imath_n}(\ell, \imath) \geq_{\mathfrak{M}_{ex}} \mathcal{E}^p[\![P'/L]\!]^{\imath_n}$$

And this proves the result.

**The case [Hiding2 ]**: Analogous.

**The case [ConstS ]**: Analogous to the case ConstS.

This complete the proof of the first part. For the second part, since $\mathcal{E}^p_\star[\![Q]\!] = \mathcal{E}^p[\![Q]\!]^0$ and $\mathcal{E}^p_\star[\![P]\!] = \mathcal{E}^p[\![P]\!]^0$, when combined with the first part, it is straightforward to see the second part holds.

□

Chapter 5

# Constructing the Automaton

Given a program

$$\text{let } C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k \text{ in } P_0$$

we shall now construct a finite automaton that would reflect the potentially infinite transition of the system by a finite transition of the the automaton. Our automaton will have the following important components:

**Q:** It describes a set of states. Each state $q \in \mathrm{Q}$ is associated with an extra extended multiset $\mathtt{E}[q]$ that represent the current exposed actions of that state. In particular, $q$ represent a certain process $P$ with $\mathcal{E}^p_\star[\![P]\!] \leq_{\mathfrak{M}} \mathtt{E}[\mathrm{q}]$.

**q$_0$:** $q_0$ is the initial state of the automaton, it directly determines the exposed actions $\mathcal{E}^p_\star[\![P_0]\!]$ of the initial process.

**E:** E is a map that associate each state $q \in \mathrm{Q}$ to the exposed actions of that state.

**$\delta$:** A transition relation $\delta$ containing transitions of the following two forms:

- $q_s \Rightarrow^{\alpha}_{(\ell_1,\imath_1)(\ell_2,\imath_2)} q_t$ reflecting that in state $q_s$ two processes with the same cooperation action type under the same effective cooperation scope(cf. Subsection 5.3) labeled $\ell_1$ and $\ell_2$ on layers $\imath_1$ and $\imath_2$, respectively, may interact and give rise to the state $q_t$.

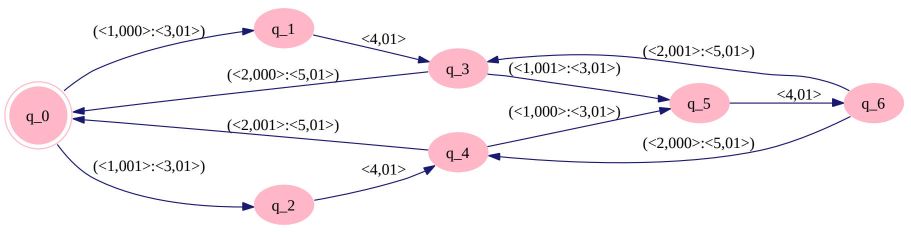

Figure 5.1: Automaton of
$(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$

- $q_s \Rightarrow^{\alpha}_{\ell,\imath} q_t$ reflecting that in state $q_s$ a process that does not need to cooperate with other processes for action $\alpha$, could proceed independently on action $\alpha$ that identified by $(\ell, \imath)$ and give rise to the state $q_t$.

We denote the automaton by (Q, $q_0$, $\delta$, E).

This automaton represents an arbitrary non-deterministic automaton since it doesn't have any final state: once it start from $q_0$, it will never stop and will always have transition toward other state(except a deadlock occur). Furthermore, we could call it *partially deterministic* in the sense that if $q_s \Rightarrow^{\alpha}_{\tilde{\ell\imath}} q_1$ and $q_s \Rightarrow^{\alpha}_{\tilde{\ell\imath}} q_2$ it could be ensured that $q_1 = q_2$. However, we could also simply convert this automaton to deterministic by adding a $fail$ state $q_f$ and a transition $q_s \Rightarrow^{\alpha}_{\tilde{\ell\imath}} q_f$ whenever there does not exist a state $q_t$ with $q_s \Rightarrow^{\alpha}_{\tilde{\ell\imath}} q_t$.

**Example 5.1** *For $(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ from Example 2.1, we obtain the finite automaton shown in Figure 5.1. The initial state is $q_0$ and there are seven states in the automaton: $q_0, \cdots, q_6$. The transitions among these states are clearly illustrated in the figure 5.1. The exposed actions(the extra extended multisets) correspond to each state are listed below.*

| $q$ | E[$q$] |
|---|---|
| $q_0$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_2$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_4$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_5$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_6$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |

The key algorithm for constructing the automaton is a *worklist algorithm* and it is represented in Subsection 5.1. It starts out from the initial state $q_0$ and constructs the automaton by adding more and more states and transitions. The algorithm makes use of several auxiliary functions that will be further introduced in the subsequent subsections:

- Given a state $q_s$ representing some exposed actions we need to select those labels $\widetilde{\ell\imath}$ (which represent actions) that may interact in the next step; this is done using the procedure `enabled` described in Subsection 5.3. However, to help solving the *scope* problem (we will talk about it later), we also developed the auxiliary function $\mathcal{Y}_\star$.
- Once the label-layer pair $\widetilde{\ell\imath}$ has been selected we can use the function $\texttt{transfer}_{\widetilde{\ell\imath}}$ already introduced in the previous chapter and this could determine the denotation of exposed actions from *this* transition for the target state.
- Finally, a target state $q_t$ should be generated by combining information of the previous states and the contribution to the target state given by the *current* transfer function.

  In the last part of this chapter, we will give result of the overall correctness of the construction.

## 5.1 The worklist algorithm

The main data structures of the algorithm are:

- a set Q of the states introduced so far; for each state $q$ the table E will specify the associated extra extended multiset E[q]$\in \mathfrak{M}_{ex}$ .
- a worklist W being a subset of Q, which contains those states that have to be processed.
- a set $\delta$ of triples $(q_s, \widetilde{\ell\imath}, q_t)$ defining the current transitions; here $q_s \in Q$ is the source state, $q_t \in Q$ is the target state and $\widetilde{\ell\imath} \in \textbf{Labex} \cup (\textbf{Labex} \times \textbf{Labex})$ is the label-layer pair of edge.

The overall algorithm has been displayed in Table 5.1 and is explained below.

Line(1) describes the initialization of the program: first the set of states Q is initialized to contain $q_0$ and the associated entry in table E is set to $\mathcal{E}^p_\star[\![P_0]\!]$. $q_0$

```
(1) $\mathrm{Q} := \{q_0\}; \mathrm{E}[q_0] := \mathcal{E}^p_\star[\![P_0]\!]; \mathrm{W} := \{q_0\}; \delta := \emptyset;$
(2) while $\mathrm{W} \neq \emptyset$ do
(3)     select $q_s$ from $\mathrm{W}; \mathrm{W} := \mathrm{W} \setminus \{q_s\};$
(4)     for each $\widetilde{\ell\imath} \in$ enabled$(\mathrm{E}[q_s])$ do
(5)         let $E =$ transfer$_{\widetilde{\ell\imath}}(\mathrm{E}[q_s])$
(6)         in update$(q_s, \widetilde{\ell\imath}, E)$
```

Table 5.1: The worklist algorithm for constructing the automaton.

has been added to Worklist W as well that indicates it will be proceeded in the next step. The transition relation $\delta$ will be empty.

Line(2) introduces the classical loop inspecting the contents of the worklist W. For each loop, a state $q_s$ will be picked up from the W and removed at the same time, which is shown in line(3). Line(4) shows that for this $q_s$, we will go through each $\widetilde{\ell\imath}$ from the set of all potential interactions calculated by `enabled`$(\mathrm{E}[q_s])$ function, which will be introduced in the following subsection. Here we just keep in mind that `enabled`$(\mathrm{E}[q_s]) \subseteq (\mathrm{E}[q_s] \cup (\mathrm{E}[q_s] \times \mathrm{E}[q_s])$. It reflects that either one of the actions or a pair of actions from $\mathrm{E}[q_s]$ will take part in the next interaction.

Line(5) and (6) describe once a potential transition $\widetilde{\ell\imath}$ is picked up, how it influence the previous constructed states and transitions of the automaton. In line(6), the function `transfer`$_{\widetilde{\ell\imath}}(\mathrm{E}[q_s])$ will return a extra extended multiset $E$ representing denotation to the target state in term of exposed actions calculated from this transition. According to this denotation $E$, Line(6) will call function `update` to refresh the properties of constructing automaton: states Q, the table E and transition relationship $\delta$. The basic idea for this function is: first to find whether an existing state could be reused, if not a new state will be introduced; second the table E and all relative transitions $\delta$ will be updated. This function would be presented in the next subsection.

## 5.2 The procedure `update`

The procedure `update`$(q_s, \widetilde{\ell\imath}, E)$ is specified in Table 5.2. The three parameters are connected as follows: $E$ is the extra extended multiset describing the contribution of the target state(we will call it $q_t$ later in the Table 5.2) to which there should be a transition marked $\widetilde{\ell\imath}$ that emerges from $q_s$. The procedure proceeds

```
(1) if there exists $q \in \mathrm{Q}$ with $H(\mathrm{E}[q]) = H(E)$
(2) then $q_t := q$
(3) else select $q_t$ from outside $\mathrm{Q}$;
(4)         $\mathrm{Q} := \mathrm{Q} \cup \{q_t\}; \mathrm{E}[q_t] := \perp_{\mathfrak{M}ex}$;
(5) if $\neg(\mathrm{E}[q_t] \geq_{\mathfrak{M}ex} E)$
(6) then $\mathrm{E}[q_t] := \mathrm{E}[q_t] \nabla_{\mathfrak{M}ex} E; \mathrm{W} := \mathrm{W} \cup \{q_t\}$;
(7) $\delta := \delta \setminus \{(q_s, \widetilde{\ell\imath}, q) \mid q \in \mathrm{Q}\} \cup \{(q_s, \widetilde{\ell\imath}, q_t)\}$;
(8) clean-up$(\mathrm{Q}, \mathrm{W}, \delta)$
```

Table 5.2: Processing enabled actions: update$(q_s, \widetilde{\ell\imath}, E)$

in three steps:

1. First it determines the *target state* $q_t$: In line(1)-(4), we first check whether an existing state in Q could be used as $q_t$ directly, if not a new state would be constructed for $q_t$ and consequently the entry of the new state in E should be appended and initialized to $\perp_{\mathfrak{M}ex}$, the new state should be appended to set Q as well. The thing worth pointing here is: in line(1), we use $H(\mathrm{E}[q]) = H(E)$ to determine whether an existing state $q$ has the desired exposed actions which is directly equal to (or its $H$ function's value equal to) what $E$ describes(or its $H$ function describes).$H$ is called the *granularity function*, and we will discuss it in later.

2. Second it determines the entry of $q_t$ in table E that should be updated or not. This is checked in line(5) which makes a judgement whether the description $\mathrm{E}[q_t]$ includes the required information $E$. If it does not, in line(6) we first use *widening operator* $\nabla_{\mathfrak{M}}$ to combine the old and the new extra extended multisets in such a way that termination of the overall algorithm is ensured. This *widening operator* $\nabla_{\mathfrak{M}}$ will be defined later. In this case, since $q_t$ and $\mathrm{E}[q_t]$ is considered modified, we should re-process it by putting it to worklist W.

3. Lastly, line(7)-(8) will update the transition relation that meets the changes made above. line(7) add the triple $(q_s, \widetilde{\ell\imath}, q_t)$ to $\delta$ as expected, however, it also removes any previous transitions from $q_s$ with label $\widetilde{\ell\imath}$ as its target state $q_t$ might be modified and isn't correct any more. As a consequence the automaton may contain unreachable parts and in line(8) the procedure `clean-up` in Table 5.3 will be invoked to remove the parts of Q, W and $\delta$ that can't be reached from the initial state $q_0$.

(1) $\mathrm{Q_{reach}} := \{q_0\} \cup \{q \mid \exists n, \exists q_1, \cdots, q_n : (q_0, \cdots, q_1) \in \delta \wedge \cdots \wedge (q_n, \cdots, q) \in \delta\}$;
(2) $\mathrm{Q} := \mathrm{Q} \cap \mathrm{Q_{reach}}$;
(3) $\delta := \delta \cap (\mathrm{Q_{reach}} \times \mathbf{Labex} \cup (\mathbf{Labex} \times \mathbf{Labex}) \times \mathrm{Q_{reach}})$;
(4) $\mathrm{W} := \mathrm{W} \cap \mathrm{Q_{reach}}$;

Table 5.3: The clean-up procedure: `clean-up`(Q,W,$\delta$)

### 5.2.1 The `widening` operator

The *widening operator* $\nabla_{\mathfrak{M}_{ex}} : \mathfrak{M}_{ex} \times \mathfrak{M}_{ex} \to \mathfrak{M}_{ex}$ used in line (6) of Table 5.2 combines extra extended multisets, and is defined by:

$$(M_1 \nabla_{\mathfrak{M}_{ex}} M_2)(\ell_{ex}) = \begin{cases} M_1(\ell_{ex}) & \text{if } M_2(\ell_{ex}) \leq M_1(\ell_{ex}) \\ M_2(\ell_{ex}) & \text{if } M_1(\ell_{ex}) = 0 \wedge M_2(\ell_{ex}) > 0 \\ \top & \text{otherwise} \end{cases}$$

It will ensure that the chain of values taken by E[$q_t$] in line(6) always stabilizes after a finite number of steps. Please refer to [13][28] for the definition of that. Here we merely establish the correctness of our choice.

**Fact 5.1** $\nabla_{\mathfrak{M}_{ex}}$ *is a widening operator, in particular* $M_1 \sqcup_{\mathfrak{M}_{ex}} M_2 \leq_{\mathfrak{M}_{ex}} M_1 \nabla_{\mathfrak{M}_{ex}} M_2$.

### 5.2.2 The `clean-up` procedure

The `clean-up` procedure in Table 5.3 will do some cleaning job after the target state $q_t$ is updated in Table 5.2. In line(1), $\mathrm{Q}_{reach}$ will collect all states that could be reached from the initial state $q_0$, line(2)-(4) will simply make intersection of the Q, W with $\mathrm{Q}_{reach}$ and remove transition relations which include source states or target states that are not reachable.

### 5.2.3 The `granularity` function

We now return to the choice of *granularity function* $H$ that is used in line (1) of table 5.2.

$$H : \mathfrak{M}_{ex} \to \mathfrak{H}$$

The function $H_{L,k}$ (for $k \in \mathbb{N} \cup \{\top\}$ and $L \subseteq \mathbf{Labex}$) is one example of a granularity function:

$$\begin{array}{rcl} H_{L,k}(E) & = & \{(\ell_{ex}, n) \mid \ell_{ex} \in L \wedge E(\ell_{ex}) = n \leq k\} \\ & \cup & \{(\ell_{ex}, \top) \mid \ell_{ex} \in L \wedge E(\ell_{ex}) = n > k \vee E(\ell_{ex}) = \top\} \end{array}$$

To ensure the correct operation of the algorithm we shall be interested in granularity functions with certain properties:

- $H$ is *finitary* if for all choices of finite sets $\mathbf{Labex}_{fin} \subseteq \mathbf{Labex}$, $H$ should be
$$H : (\mathbf{Labex}_{fin} \rightarrow \mathbb{N} \cup \{\top\}) \rightarrow \mathfrak{H}_{fin}$$
for some finite subset $\mathfrak{H}_{fin} \subseteq \mathfrak{H}$.
- $H$ is *stable* if:
$$H(E_1) = H(E_2) \text{ implies } H(E_1 \nabla_{\mathfrak{M}ex} E_2) = H(E_i) \text{ for } i = 1, 2$$

**Fact 5.2** *The granularity function $H_{L,k}$ is finitary as well as stable (for all choices of $L \subseteq$ **Labex** and $k \geq 0$, but $k \neq \top$).*

Now we state a general termination result for the construction of the finite automaton:

**Theorem 5.3** *Whenever the algorithm of Table 5.1 terminates it produces a partially deterministic automaton.*
*If the granularity function H is stable then the automaton satisfy the following injective property:*

$$\forall q_1, q_2 \in \mathrm{Q} : H(\mathrm{E}[q_1]) = H(\mathrm{E}[q_2]) \;\Rightarrow\; q_1 = q_2$$

If the granularity function H is finitary then the algorithm always terminates.

**Proof.** The proof is similar to the counterpart exhibits in [16], we omit it here. □

Even though Fact 5.2 doesn't guarantee that when $k = \top$, $H_{L,k}$ is still finitary as well as stable (which could ensure the algorithm of Table 5.1 terminate (Refer to Theorem 5.3)), we observed that if PEPA programs fulfills certain condition, Fact 5.2 will still hold even $k = \top$. We will discuss this condition in Subsetion 5.5.

In the following part, if there is no specific declaration, we will equip $H_{L,\top}$ to our analysis (all the programs in this thesis fulfills that condition, so it is safe to use $k = \top$), for obtaining the best precision from this granularity function to our analysis.

**Example 5.2** *To illustrate the worklist algorithm consider the program introduced before:* $(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$. *We adopt the granularity function* $H_{L,\top}$. *which guarantees that we get the automaton with best precision from* $H_{L,k}$(for $k \in \mathbb{N} \cup \{\top\}$). *Initially the automaton will only have one state* $q_0$ *with*

$$\mathrm{E}[q_0] = \bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1].$$

*Then the automaton will evolve as we execute the while loop from Table 5.1. After eight rounds of running that loop we get the constructed automaton, which ends up with eight states in total.*

*Figure 5.2 to 5.9 demonstrate the process of building up the automaton step by step. Each figure shows the nodes and edges already established after a specific round. There are nodes with three colors in these figures: the pink nodes represent the states that have already been proceeded; the blue and gold nodes all together represent states in the W set waiting for processing and in particular, the gold node corresponds to the state that will be processed in the next round (it is randomly selected from blue nodes). There are edges with two colors in these figures: the blue edges represent transitions already established before the current round while the red edges represent transitions be created just at the current round. We also append eight tables to supplement demonstrating these figures and each of them is included with the extra extended multisets relevant to each state.*

*After the first round of executing the while loop from Table 5.1, we get a automaton with four states, among which* $q_1, q_2$ *and* $q_3$ *are generated just in this round. At this round, the* $\mathrm{enabled}[q_s]$ *function at Line (4) from Table 5.1 (this function will be described in detail in Section 5.3, here* $q_s = q_0$*) returns three pairs of possible interactions:* $\{(1,000) : (3,01), (1,001) : (3,01), (1,000) : (1,001)\}$. *If we take* $(1,001) : (3,001)$, *then the transition function requires that the target state should have the denotation* $E = \bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$. *In the mean time, the granularity function* $H_{L,\top}$ *at Line (1) from Table 5.2 forces us to select a new state(here* $q_2$*) and then put this denotation to* $\mathrm{E}[q_2]$ *with* $\nabla_{\mathfrak{M}ex}$ *operator at Line(6) from Table 5.2. In this way we generate* $q_1, q_2$

*and $q_3$ and put them to the $W$ set for processing in the following rounds, among which $q_1$ will be processed in next rounds.*

*In the second round, $q_1$ is considered and thus there are three possible interactions: $\{(2,000) : (2,001), (2,001) : (5,01), (2,000) : (5,01)\}$. Picking up $(2,001) : (5,01)$ and $(2,000) : (5,01)$ will lead the creation of two new states $q_4$ and $q_5$ as performed in round one. However, the transition from $(2,000) : (2,001)$ make the transfer function require target state having the denotation of $E = \bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$. This time the granularity function $H_{L,\top}$ at Line (1) from Table 5.2 enables us to reuse the state $q_0$ which has the same denotation.*

*The three to eight rounds perform the similar operations and we will not explains them here in detail.*

| $q$ | E[$q$] W = $\{q_1, q_2, q_3\}$ |
|---|---|
| $q_0$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_2$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| | Round1 |

| $q$ | E[$q$] W = $\{q_2, q_3, q_4, q_5\}$ |
|---|---|
| $q_0$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_2$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_4$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_5$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| | Round2 |

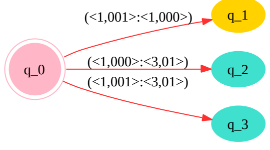

Figure 5.2: Round 1

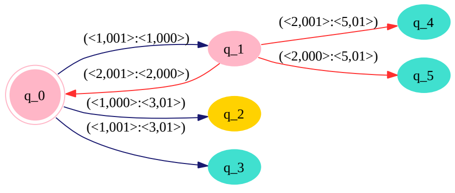

Figure 5.3: Round 2

| $q$ | E[$q$] W $= \{q_3, q_4, q_5\}$ |
|---|---|
| $q_0$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_2$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_4$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_5$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| | Round3 |

| $q$ | E[$q$] W $= \{q_4, q_5\}$ |
|---|---|
| $q_0$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_2$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_4$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_5$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| | Round4 |

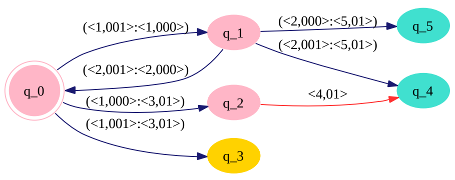

Figure 5.4: Round 3

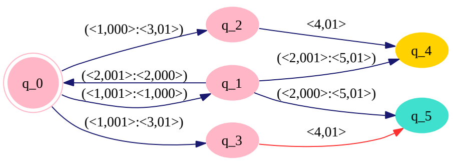

Figure 5.5: Round 4

| $q$ | E[$q$] W $= \{q_5, q_6\}$ |
|---|---|
| $q_0$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_2$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_4$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_5$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_6$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| | Round5 |

| $q$ | E[$q$] W $= \{q_6\}$ |
|---|---|
| $q_0$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_2$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_4$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_5$ | $\bot_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_6$ | $\bot_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| | Round6 |

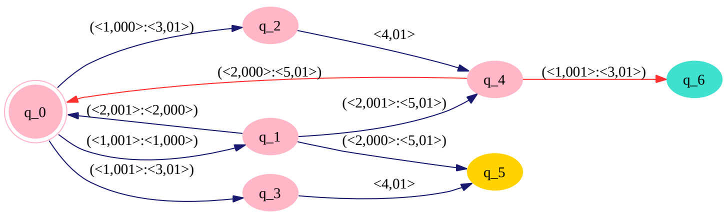

Figure 5.6: Round 5

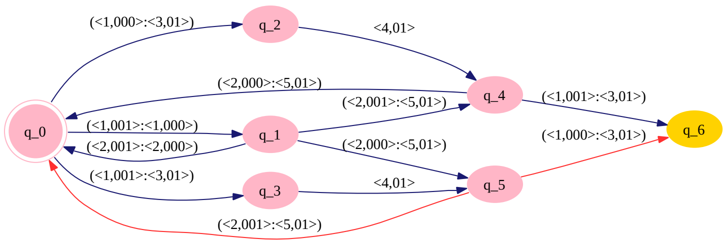

Figure 5.7: Round 6

| $q$ | E[$q$] W = $\{q_7\}$ |
|---|---|
| $q_0$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_2$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_4$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_5$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_6$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_7$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| | Round7 |

| $q$ | E[$q$] W = $\{\}$ |
|---|---|
| $q_0$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_2$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_3$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_4$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (1,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_5$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (2,001) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_6$ | $\perp_{\mathfrak{M}ex}[(2,000) \mapsto 1, (2,001) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_7$ | $\perp_{\mathfrak{M}ex}[(1,000) \mapsto 1, (1,001) \mapsto 1, (4,01) \mapsto 1]$ |
| | Round8 |

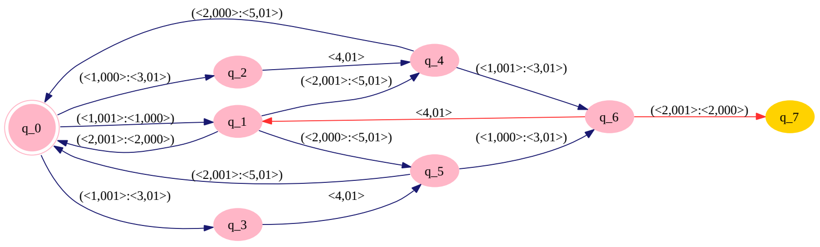

Figure 5.8: Round 7

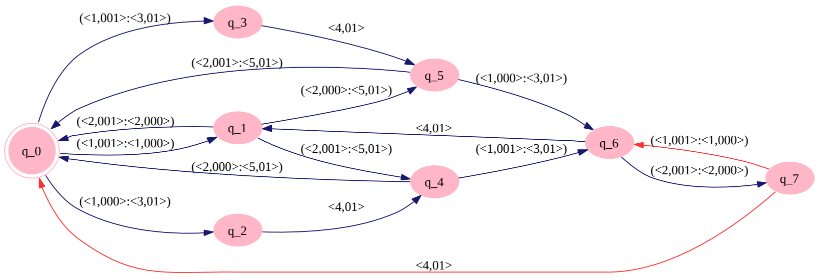

Figure 5.9: Round 8

## 5.3 The computation of enabled exposed actions

We now return to the construction of procedure `enabled`$(E)$ used in line (4) of Table 5.1 for worklist algorithm. Here $E$ taking the form E$[q_s]$ is an extra extended multiset that describes *all potential* enabled exposed actions at that state $q_s$.

From Table 2.1, we know there are two kinds of transition that could take place among processes in PEPA.

1. The transition only involves one process. In this case, The action of this transition should not be surrounded by any cooperation set containing the same type. Put it more accurately, this action can't be under *any effective cooperation scope* that covers this type.

2. The transition involves two processes cooperating on the same action type within the *same effective cooperation scope.*

Sequential component definition

$$P \overset{def}{=} (\alpha, 1).P$$
$$Q \overset{def}{=} (\alpha, 2).(\beta, 3).Q$$

Model component definition

| | |
|---|---|
| `case(1):` | $P$ |
| `case(2):` | $P \underset{\alpha}{\bowtie} Q$ |
| `case(3):` | $P \underset{\beta}{\bowtie} Q$ |
| `case(4):` | $P \underset{\alpha}{\bowtie} (Q/\{\alpha\})$ |

Table 5.4: A example for demonstrating cooperation environment

Before introducing the concept of *effective cooperation scope*, we would present *cooperation environment, cooperation scope* first, then we come to the term *effective cooperation scope.*

**Cooperation environment** For the sake of determining whether an action should be proceeded independently or cooperate with other processes that have the same action type, it is not enough just to analyze those processes with the suitable action type. One also need to consider what is their *cooperation environment*: the cooperation dependency among different processes due to the use of cooperation and hiding combinators.

**Example 5.3** *We define a program shown in Table 5.4: first we define two sequential components P and Q, then select different model components to illustrate the impact of various* cooperation environment *to action* $(\alpha, 1)$ *in process P.*

*In case(1), action* $(\alpha, 1)$ *is proceeded itself. In case(2), action*$(\alpha, 1)$ *in P must cooperate with action* $(\alpha, 2)$ *in Q, due to the fact they set up a cooperation set*$\{\alpha\}$ *between themselves. However, in case(3), since the cooperation set is* $\{\beta\}$ *that doesn't cover action* $\alpha$*,* $(\alpha, 1)$ *could still be proceeded on its own. For case(4), even though P and Q share a cooperation set*$\{\alpha\}$*,*$(\alpha, 1)$ *needn't cooperate with* $(\alpha, 2)$ *because the hidden operator make* $(\alpha, 2)$ *invisible to the process outside Q.*

**Cooperation scope** Up to now, it could be seen clearly that different *cooperation environment* would make sequential components behave dissimilarly.

If take this step further, we will discuss how the cooperation and hidden combinators impose their semantics force on the various sequential processes. To make it simple, here we first present an important concept *cooperation scope* to depict such semantics force: within a given program, a set of sequential processes might be influenced by a certain cooperation or hidden combinator, we say this set of sequential processes is under the *cooperation scope* of this certain cooperation or hidden combinator.

Recall we have defined the layer in Subsection 2.3 and this could help us easily distinguish the position of different sequential processes and their internal actions in a given model process. Here we borrow the concept of layer to the cooperation and hidden combinator in order to redefine *cooperation scope* much more accurately. The rule for allocating layer to cooperation and hidden combinator is quite similar to the rule that applied to actions in Subsection 2.3. The cooperation scope for either *cooperation combinators* or *hidden combinators* is all sequential processes or actions whose *current layer* is equal or lower than the combinator's(cf. Subsection 2.3, here we underline the *current layer* of each sequential process in the next example).

**Example 5.4** *For the sequential components in Table 5.4, if we add a new model component called case(5) as follows:*

$$\texttt{case(5)} : \underbrace{P}_{\underline{0}0} \underbrace{\underset{\alpha}{\bowtie}}_{0} (\underbrace{P}_{\underline{01}0} \underbrace{\underset{\alpha}{\bowtie}}_{01} (\underbrace{Q}_{\underline{01}1} \underbrace{/\{\alpha\}}_{01}))$$

*It could be easily deduced that for cooperation combinator* $\underbrace{\underset{\alpha}{\bowtie}}_{0}$*, processes* $\underbrace{P}_{\underline{0}0}$, $\underbrace{P}_{\underline{01}0}$ *and* $\underbrace{Q}_{\underline{01}1}$ *are under its cooperation scope. Similarly, for* $\underbrace{\underset{\alpha}{\bowtie}}_{01}$*,* $\underbrace{P}_{\underline{01}0}$ *and* $\underbrace{Q}_{\underline{01}1}$ *are under its cooperation scope. For hidden combinator* $\underbrace{/\{\alpha\}}_{01}$*,* $\underbrace{Q}_{\underline{01}1}$ *is under its cooperation scope.*

**Effective cooperation scope** According to the semantics of the cooperation and hidden combinator, a cooperation combinator's cooperation scope always try to include more processes for cooperating while hidden combinator's cooperation scope always intends to exclude some processes and exempt them from cooperating.

In order to merge these two kinds of *cooperation scope*s, we will introduce *effective cooperation scope*: for any cooperation combinator with a certain type, the effective cooperation scope should contains sequential processes under this cooperation combinator' cooperation scope but excludes the processes that falls into the cooperation scope of *all* hidden combinators(whose layers are lower than the cooperation combinator of this type). Clearly if two processes are under the *same effective cooperation scope* of a cooperation combinator with a certain type, they *must* cooperate with each other when make a transition on that type. If a process is not under *any effective cooperation scope*, it could proceed independently.

**Example 5.5** *When scrutinizing the example 5.4 , for cooperation combinator* $\underbrace{\bowtie_{\alpha}}_{0}$, $\underbrace{P}_{\underline{0}0}$ *and* $\underbrace{P}_{\underline{01}0}$ *are under its effective cooperation scope; for cooperation combinator* $\underbrace{\bowtie_{\alpha}}_{01}$, *its effective cooperation scope covers* $\underbrace{P}_{\underline{01}0}$. *And it could be seen that for process* $\underbrace{P}_{\underline{01}0}$, *it is under both* $\underbrace{\bowtie_{\alpha}}_{0}$ *and* $\underbrace{\bowtie_{\alpha}}_{01}$*'s effective cooperation scope. This means that it must cooperate with either processes that under cooperation scope of these two cooperation combinators. For* $\underbrace{Q}_{\underline{01}1}$, *it is not under any effective cooperation scope, so the action in it could proceed independently.*

Now we will outline the significant steps in function `enabled` (to be presented in Subsection 5.3.1) that calculate all potential transitions from extra extended multiset $E$.

- Given a program, for any action with type $\alpha$, we need to obtain *all* cooperation and hidden combinators whose *cooperation scope* covers this action. This is done using the function $\mathcal{Y}$ describled in Subsection 5.3.2.
- Given a program, when we get the combinators(cooperation or hidden) with the right cooperation scope of *all* actions, we need to get the more useful information of their *effective cooperation scope.* This will leave each action merely to be bounded with cooperation combinators(get rid of the hidden combinators). The algorithm for doing this job is developed in function `removehidd`, which will be presented in Subsection 5.3.3.
- Finally, we are able to find all enabled transitions according to the information from relevant effective cooperation scope. In particular, how to find all actions to be cooperated among different processes is developed in function `matchaction` that would be further discussed in Subsection 5.3.4.

### 5.3.1 The `enabled` function

The main data structure of the `enabled` function are:

- a map $Z \in \mathfrak{Y}$ linking each action with the layer of combinators (either cooperation or hidden) whose *cooperation scope* covers the action itself, where
$$\mathfrak{Y} = \mathbf{Labex} \to (\mathbf{Act} \times \wp(\mathbf{Layer}) \times \wp(\mathbf{Layer}))$$
The first $\wp(\mathbf{Layer})$ describes the set of cooperation combinators that could be on several layers, while the second $\wp(\mathbf{Layer})$ records the similar information for hidden combinators.

- a set $S \in \mathfrak{U}$, this set contains action information and the respective cooperation combinators whose *effective cooperation scope* cover this action.
$$\begin{aligned}\mathfrak{U} &= \wp(\mathbf{Labex} \times \mathbf{Act} \times \wp(\mathbf{Layer})) \\ &= \wp(\mathbf{Lab} \times \mathbf{Layer} \times \mathbf{Act} \times \wp(\mathbf{Layer}))\end{aligned}$$
The $\wp(\mathbf{Layer})$ describes the set of cooperation combinators whose effective cooperation scope covers this action.

- a variable $R \in \mathfrak{R}$ collecting the enabled transitions from both type, where
$$\mathfrak{R} = \wp(\mathbf{Labex}) \times \wp(\mathbf{Labex} \times \mathbf{Labex})$$
$\wp(\mathbf{Labex})$ is the result for transitions involving single action while $\wp(\mathbf{Labex} \times \mathbf{Labex})$ tracks all transitions involving two actions with the same type.

- $E \in \mathfrak{M}_{ex}$ taking the form $\mathrm{E}[q_s]$ is an extra extended multiset that describes *all potential* enabled exposed actions at that state $q_s$.

The overall algorithm for `enabled` function is displayed in Table 5.5 and is explained below. The initialization are performed in line(1) and (2): First Z is calculated by $\mathcal{Y}^p_\star[\![P_0]\!](0, \emptyset, \emptyset)$ and this function is discussed in the following Subsection 5.3.2, here we just give its parameters' meaning: $P$ represents the model component of this program; 0, $\emptyset$, and $\emptyset$ respectively indicate the function will start from the 0 layer(top layer), cooperation and hidden set for combinators is set $\emptyset$ (combinators here should has the right cooperation scope over different actions) and later the function will traverse the program tree and collect necessary information marked with correct layer to the cooperation and hidden set on the fly. The result of the `enabled` function $R$ is initialized to $\emptyset$. Taking $E$ and $Z$ as parameters, line(2) obtain the value of $S$ by `removehidd` function. And this

```
enabled($E$)

(1) $Z := \mathcal{Y}^p_\star[\![P_0]\!](0,\emptyset,\emptyset)$;    $R := \emptyset$;
(2) $S :=$ removehidd$(E, Z)$;
(3) for each $(\ell, \imath, \alpha, C) \in S$ do
(4)     if $C = \emptyset$
(5)     then $R := R \cup \{(\ell, \imath)\}$
(6)     else for each $\imath_c \in C$ do
(7)              $R := R \cup$ matchaction$(\imath_c, (\ell, \imath, \alpha, C), S)$
(8) return $R$;
```

Table 5.5: The enabled($E$) procedure

function will provide information for the following steps in the algorithm that each action links with its cooperation combinators set(here we only record their layers)that has right effective cooperation scope.

Line(3)-(8) will use the information from $S$ and check for each record the action should proceed independently or cooperate with other records in $S$. The inspecting process start from line(3), where each record contains: the label of the action $\ell$, the layer $\imath$ the action belongs to, the action type $\alpha$ and the set $C$ for all cooperation combinators that have the effective cooperation scope covering this action. Line(4)and Line(5) will show that if the action is not under any effective cooperation scope ($C = \emptyset$), this action should be added to the result set $R$ by itself. Line(6) and (7) shows the different cases, if this action has any combinators whose effective cooperation scope covers itself, the function will go through all combinators(line(6)) and check if it could find other actions to cooperate with the default one in $S$ by function `matchaction`(to be presented in the following Subsection 5.3.4), and add all possible pair of actions to the result set $R$. Line(8) will return the final result set $E$ containing both single actions and actions to be proceeded with cooperation.

### 5.3.2 The function $\mathcal{Y}$ and its auxiliary functions

Now we will present the function $\mathcal{Y}$ at line(1) in Table 5.5 that would provide information related to cooperation scopes for all actions in the program.

Recall that we need to get a result from function $\mathcal{Y}$ that belongs to the following domain:

$$\mathfrak{Y} = \mathbf{Labex} \rightarrow (\mathbf{Act} \times \wp(\mathbf{Layer}) \times \wp(\mathbf{Layer}))$$

where the first and second $\wp(\mathbf{Layer})$ represent the set of combinators' layers for cooperation and hidden respectively. The entry in this map **Labex** requires each label must link to some layer information. In other words, **Label** must links to **Layer**, $\wp(\mathbf{Layer})$ and $\wp(\mathbf{Layer})$. The first problem is how to put each label together correspond with layer information.

From the previous chapter, we already know that because of *cooperation environment*, an action in a certain sequential process may behave dissimilarly if it is located at different layers. Here we can conclude that the sequential process itself may also act distinctively when located at different layers. And the layer information of each sequential process (differentiated by sequential process name $C_k$) will be assigned according to the structure of the model component definition.

Thus we come to the point of how to solve the problem above, which should have two steps:
let $C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k$ in $P_0$

1. For every sequential process name $C_k$, collect *all* possible labels(actions) that belong to that process (all possible actions a sequential process might behave as).

2. Put layer information to each sequential process name $C_k$ which is determined by the structure of the model component definition. Subsequently put layer information to *all* possible labels(actions) of different sequential processes separately.

For step1 we have developed a function $\mathcal{Y}^s$, while at step2 a function $\mathcal{Y}^p$ was proposed.

The function $\mathcal{Y}^s_\star$ tries to collect *all* action label-type pairs related for each sequential process name and has the functionality

$$\mathcal{Y}^s_\star : \mathbf{S}_{proc} \rightarrow \wp(\mathbf{Lab} \times \mathbf{Act})$$

The function is defined in Table 5.6 using the overall pattern developed in previous sections. So we have a function

$$\mathcal{Y}^s : \mathbf{S}_{proc} \rightarrow (\mathbf{PN} \rightarrow \wp(\mathbf{Lab} \times \mathbf{Act})) \rightarrow \wp(\mathbf{Lab} \times \mathbf{Act})$$

Exposed actions for let $C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k$ in $P_0$

$$\begin{array}{rcl}
\mathcal{Y}^s[\![(\alpha^\ell, r).S]\!]env & = & (\ell,\alpha) \cup \mathcal{Y}^s[\![S]\!]env \\
\mathcal{Y}^s[\![S_1 + S_2]\!]env & = & \mathcal{Y}^s[\![S_1]\!]env \cup \mathcal{Y}^s[\![S_2]\!]env \\
\mathcal{Y}^s[\![C]\!]env & = & env(C) \\
\mathcal{Y}^s_\star[\![S]\!] & = & \mathcal{Y}^s[\![S]\!]env_{\mathcal{Y}} \\
\text{where}\mathcal{F}_{\mathcal{Y}} & = & [C_1 \mapsto \mathcal{Y}^s[\![S_1]\!]env, \cdots, C_k \mapsto \mathcal{Y}^s[\![S_k]\!]env] \\
\text{and } env_\emptyset & = & [C_1 \mapsto \emptyset, \cdots, C_k \mapsto \emptyset] \\
\text{and } env_{\mathcal{Y}} & = & \sqcup_{j\geq 0}\mathcal{F}^j_{\mathcal{Y}}(env_\emptyset) \\
\mathcal{Y}^p[\![P_1 \underset{L}{\bowtie} P_2]\!](\imath, coop, hidd) & = & \text{let } coop' = \text{addlayers}(coop, L, \imath) \\
& & \text{in } \mathcal{Y}^p[\![P_1]\!](\imath 0, coop', hidd) \cup \mathcal{Y}^p[\![P_2]\!](\imath 1, coop', hidd) \\
\mathcal{Y}^p[\![P/L]\!](\imath, coop, hidd) & = & \text{let } hidd' = \text{addlayers}(hidd, L, i') \\
& & \quad\quad \text{and } \imath = \imath'0 \text{ or } \imath = \imath'1 \\
& & \text{in } \mathcal{Y}^p[\![P]\!](\imath, coop, hidd') \\
\mathcal{Y}^p[\![S]\!](\imath, coop, hidd) & = & \text{let } [(\ell_1,\alpha_1), \cdots, (\ell_n, \alpha_n)] = \mathcal{Y}^s_\star[\![S]\!] \\
& & \text{in } [(\ell_1, \imath) \mapsto (\alpha_1, \partial(coop, \alpha_1), \partial(hidd, \alpha_1)), \cdots, \\
& & (\ell_n, \imath) \mapsto (\alpha_n, \partial(coop, \alpha_n), \partial(hidd, \alpha_n))] \\
\mathcal{Y}^p_\star[\![P]\!](0,\emptyset,\emptyset) & = & \mathcal{Y}^p[\![P]\!](0,\emptyset,\emptyset)
\end{array}$$

Table 5.6: $\mathcal{Y}^s$ and $\mathcal{Y}^p$ function

whose second argument is an environment providing similar information for the process names. The domain $\wp(\mathbf{Lab} \times \mathbf{Act})$ inherits a partial ordering $\sqsubseteq$ from the subset ordering on sets and becomes a complete lattice. The function $\mathcal{F}_{\mathcal{Y}}$ is a monotonic function on a complete lattice and hence has a least fixed point $env_{\mathcal{Y}}$. As the function $\mathcal{F}_{\mathcal{Y}}$ is in fact also continuous the Kleene formulation of the fixed point is permissible. It follows that the overall definition of $\mathcal{Y}^s_\star$ is well-defined.

Now we turn to define the function $\mathcal{Y}^p$

$$\mathcal{Y}^p : \mathbf{P}_{proc} \to \mathfrak{Y} \quad (= \mathbf{P}_{proc} \to (\mathbf{Labex} \to (\mathbf{Act} \times \wp(\mathbf{Layer}) \times \wp(\mathbf{Layer}))))$$

It could be seen clearly that this function will return the affiliation of each action(to be distinguished by label and action type) and its layer information.

For function $\mathcal{Y}^p$, it will always take layer $\imath$, cooperation combinator set *coop* and hidden combinator set *hidd* as parameters and finally append the correct layer information to each $S$ sequential component(Finally to each actions within a $S$

```
addlayers$(B, L, \imath)$

(1) for each $act \in L$ do
(2)         put $(act, \partial(B, act) \cup \imath)$ to $B$;
where
        $B \in \mathfrak{N}($ represents $coop$ or $hidd$ in Table 5.6$); \alpha \in \mathbf{Act}$, and $L \in \wp(\mathbf{Act})$
        $\partial(B, \alpha)$ : return layer information related to $\alpha$ from map B.
```

Table 5.7: function `addlayers`

sequential component). Here $coop, hidd \in \mathfrak{N}$, where

$$\mathfrak{N} = \mathbf{Act} \rightarrow \wp(\mathbf{Layer})$$

which indicates a action might under the cooperation scopes of the same combinator(cooperation or hidden ) that located at different layers at the same time.

Specifically, for cooperation operator, it will update the *coop* set first and then combine the result of two components by $\cup$ operation. One will notice that we utilize a auxiliary function `addlayers`$(coop, L, \imath)$ to update the *coop* set with action types in $L$ at the current layer $\imath$, which is specified in Table 5.7. The clause for the $P/L$ will similarly update the *hidd* set with action type set $L$ by the `addlayers` function. The clause for constant model combinator will borrow the result get from $\mathcal{Y}_{\star}^{s}$ step and set up each action contained by a sequential process $S$ with the *coop* and *hidd* layer information, the current layer of this sequential process $\imath$ as well as the action type of this action $\alpha$.

The `enabled` function in Table 5.5 will simply call $\mathcal{Y}_{\star}^{p}$ to set layer information $(\imath, coop, hidd)$ of each action along the path from the top layer 0 to the layer that the actions locates.

**Example 5.6** *We will take three cases of program from Example 2.1. In addition, we create a new case* $S \underset{\{g,p,h\}}{\bowtie} Q/\{h\}$ *that contains hidd combinator. Then we calculate* $\mathcal{Y}_{\star}^{p}$ *for each of four cases and list the results in the following tables.*

*In case (a), because action h isn't under any cooperation scope, both coop and hidd set of labex (4,01) are empty. However, the counterparts of case (b) are both* $\{0\}$ *and clearly this is the effect of* $\underset{\{g,p,h\}}{\bowtie}$ *and* $/\{h\}$ *combinators. In case (d), the actions(g and p) in S all have coop combinators at the layers* $\{00, 0\}$, *due to the fact they are under the cooperation scopes of* both $\underbrace{\bowtie_{\{g,p\}}}_{0}$ *and* $\underbrace{\bowtie_{\{g,p\}}}_{00}$.

| $\mathcal{Y}^p_\star$ *of* | $\underbrace{S}_{00} \underbrace{\bowtie_{\{g,p\}}}_{0} \underbrace{Q}_{01}$ | | | $\underbrace{S}_{00} \underbrace{\bowtie_{\{g,p,h\}}}_{0} (\underbrace{Q}_{01} \underbrace{/\{h\}}_{0})$ | | |
|---|---|---|---|---|---|---|
| *labex* | *action* | *coop* | *hidd* | *action* | *coop* | *hidd* |
| $(1, 00)$ | $g$ | $\{0\}$ | $\emptyset$ | $g$ | $\{0\}$ | $\emptyset$ |
| $(2, 00)$ | $p$ | $\{0\}$ | $\emptyset$ | $p$ | $\{0\}$ | $\emptyset$ |
| $(3, 01)$ | $g$ | $\{0\}$ | $\emptyset$ | $g$ | $\{0\}$ | $\emptyset$ |
| $(4, 01)$ | $h$ | $\emptyset$ | $\emptyset$ | $h$ | $\{0\}$ | $\{0\}$ |
| $(5, 01)$ | $p$ | $\{0\}$ | $\emptyset$ | $p$ | $\{0\}$ | $\emptyset$ |
| Case | $(a)$ | | | $(b)$ | | |

| $\mathcal{Y}^p_\star$ *of* | $(\underbrace{S}_{000} \underbrace{\bowtie_{\{\}}}_{00} \underbrace{S}_{001}) \underbrace{\bowtie_{\{g,p\}}}_{0} \underbrace{Q}_{01}$ | | | $(\underbrace{S}_{000} \underbrace{\bowtie_{\{g,p\}}}_{00} \underbrace{S}_{001}) \underbrace{\bowtie_{\{g,p\}}}_{0} \underbrace{Q}_{01}$ | | |
|---|---|---|---|---|---|---|
| *labex* | *action* | *coop* | *hidd* | *action* | *coop* | *hidd* |
| $(1, 000)$ | $g$ | $\{0\}$ | $\emptyset$ | $g$ | $\{00, 0\}$ | $\emptyset$ |
| $(2, 000)$ | $p$ | $\{0\}$ | $\emptyset$ | $p$ | $\{00, 0\}$ | $\emptyset$ |
| $(1, 001)$ | $g$ | $\{0\}$ | $\emptyset$ | $g$ | $\{00, 0\}$ | $\emptyset$ |
| $(2, 001)$ | $p$ | $\{0\}$ | $\emptyset$ | $p$ | $\{00, 0\}$ | $\emptyset$ |
| $(3, 01)$ | $g$ | $\{0\}$ | $\emptyset$ | $g$ | $\{0\}$ | $\emptyset$ |
| $(4, 01)$ | $h$ | $\emptyset$ | $\emptyset$ | $h$ | $\emptyset$ | $\emptyset$ |
| $(5, 01)$ | $p$ | $\{0\}$ | $\emptyset$ | $p$ | $\{0\}$ | $\emptyset$ |
| Case | $(c)$ | | | $(d)$ | | |

### 5.3.3 The procedure `removehidd`

In the last subsection, we have discussed how to put the information of cooperation scope to each actions in the program. Now we should talk about the method to merge the information from cooperation scopes and calculate the effective cooperation scope for each cooperation combinator to each action. Put it in another words, we will simplify each entry in map Z (cf. Table 5.5 line(2)) by function `removehidd` that essentially utilizes layers in *hidd* to restrict layers in *coop*, and finally we will get rid of the existing *hidd* and *coop* and instead to replace them with a new *coop* (cooperation combinators set) which has the truly effective cooperation scope to this action.

Our `removehidd` procedure is displayed in Table 5.8. Line(1) is the initialization of the procedure: $S$ is the result set, where each element represents the

*The procedure* `removehidd`$(E, Z)$

```
(1) $S := \emptyset$;
(2) for each $(\ell, \imath) \in$ domEx($E$) do
(3)     let $(\alpha, C, H) = Z((\ell, \imath))$
(4)     in if $\neg(C = \emptyset)$
(5)         then for each $\imath_c \in C$ do
(6)                 if there exists $i_h \in H$
(7)                     with $\aleph(\imath_c) <= \aleph(\imath_h)$
(8)                 then $C := C \setminus \imath_c$;
(9)         $S := S \cup (\ell, \imath, \alpha, C)$;
(10) return $S$;
```

Table 5.8: function `removehidd`

information of one action which has four fields: label $\ell$, layer $\imath$, action type $\alpha$ and cooperation combinators set $C$(represent by several layers) from which all combinators have effective cooperation scope over this action. Line(2) start the loop which checks all elements in `domEx(`$E$`)`(cf. Section 3.1.2 ), meaning that we only consider exposed actions that is described in the extra extended multiset $E$. Line(3) get layer information of each actions to be checked.

Line(4)-(8) describes that some cooperation combinators will be blocked by some hidden combinators for the same action. Line(4) and (5) say we will check each layer in $C$ and find out if any layer should be removed from the set. In another words, we should determine if some cooperation combinators will *lose* their cooperation scope over a certain action when taking the influence caused by some hidden combinators into account. Line(6)-(8) state that under what specific circumstance a layer should be removed from $C$: in *hidd* set check if there exists any layer that is higher than the layer from the cooperation combinator layer set $C$. If the condition is guaranteed, this hidden combinator will hide this cooperation combinator and make the action not cooperate with processes outside the cooperation scope of this hidden combinator even though they are under the cooperation scope of this cooperation combinator. Here function $\aleph(\imath)$ will return the length of the layer $\imath$: the larger the value is, the lower the layer is.

Line(9) will add a new element which contains the updated $C$ to set $S$. And finally the result will be returned by line(10).

**Example 5.7** *Here we will inspect four cases from Example 5.6. First we cal-*

*culate removehidd($\mathcal{E}^p_\star[\![P_0]\!], \mathcal{Y}^p_\star[\![P_0]\!]$) for each of them, based on the result from Example 5.6 and Example 3.1(The result of Case(a) and (b) in Example 3.1 are exactly the same).The table below shows the result.*

*removehidd*$(\mathcal{E}^p_\star[\![P_0]\!], \mathcal{Y}^p_\star[\![P_0]\!])$

| $Case(a)$ | | | $Case(b)$ | | |
|---|---|---|---|---|---|
| $labex$ | $action$ | $coop'$ | $labex$ | $action$ | $coop'$ |
| $(1,00)$ | $g$ | $\{0\}$ | $(1,00)$ | $g$ | $\{0\}$ |
| $(3,01)$ | $g$ | $\{0\}$ | $(3,01)$ | $g$ | $\{0\}$ |
| $(5,01)$ | $p$ | $\{0\}$ | $(5,01)$ | $p$ | $\{0\}$ |
| $Case(c)$ | | | $Case(d)$ | | |
| $labex$ | $action$ | $coop'$ | $labex$ | $action$ | $coop'$ |
| $(1,000)$ | $g$ | $\{0\}$ | $(1,000)$ | $g$ | $\{00,0\}$ |
| $(1,001)$ | $g$ | $\{0\}$ | $(1,001)$ | $g$ | $\{00,0\}$ |
| $(3,01)$ | $g$ | $\{0\}$ | $(3,01)$ | $g$ | $\{0\}$ |
| $(5,01)$ | $p$ | $\{0\}$ | $(5,01)$ | $p$ | $\{0\}$ |

*We find that the coop′ is exactly the same as the coop in Example 5.6. This is because the actions considered in these cases (p,g) are not related to any hidden combinators. Next we show a example that coop in 5.6 is modified by the removehidd function.*

*Imagine the case(b) of the program is in some state $q_x$ that has extra extend multiset $E = \bot_{\mathfrak{M}ex}[(2,00) \mapsto 1, (4,01) \mapsto 1]$, then the result of removehidd (E,$\mathcal{Y}^p_\star[\![P_0]\!]$) is*

| $labex$ | $action$ | $coop'$ |
|---|---|---|
| $(2,00)$ | $p$ | $\{0\}$ |
| $(4,01)$ | $h$ | $\emptyset$ |

*Notice that this time the coop′ of action $(4,01)$ is $\emptyset$ instead of the coop $= \{0\}$ in Example 5.6. And clearly action $(4,01)$ will not cooperate with other actions since it is hidden by $/\{h\}$ combinator.*

### 5.3.4 The procedure `matchaction`

The procedure `matchaction`$(\imath_c, (\ell, \imath, \alpha, C), S)$ tries to find a record in $S$, that will cooperate with action(the default record) $(\ell, \imath, \alpha, C)$ on the cooperation combinator on layer $\imath_c$. In Table 5.9 the function is displayed and we will explain it `matchaction`$(\imath_c, (\ell, \imath, \alpha, C), S)$

```
(1) $T := \emptyset$;
(2) for each $(\ell', \imath', \alpha', C') \in S$ do
(3)     if there exists $\imath'_c \in C'$ with
(4)         $\imath_c = \imath'_c \wedge \alpha = \alpha' \wedge \neg(\rho(\imath, \aleph(\imath_c)) = \rho(\imath', \aleph(\imath_c)))$
(5)     then $T := T \cup \{((\ell, \imath), (\ell', \imath'))\}$;
(6) return $T$;
```

Table 5.9: function `matchaction`

below. Line(1) initialize the variable $T$ for collecting all possible exposed action pairs under the effective cooperation scope of a certain cooperation combinator at layer $\imath_c$.

Line(2)-(5) illustrate the matching procedure: line(2) start the loop that indicates we need to exam all record in $S$; line(3) and (4) states the condition on which two actions should cooperate with each other: if there is a record in S whose cooperation combinator layer set $C'$ contains a layer $\imath'_c$ that meet:

- Two actions are under the same effective cooperation scope: $\imath'_c = \imath_c$.
- Its action type $\alpha'$ is the same as the default record's action type $\alpha$.
- Two actions shouldn't under the same side of cooperation combinator, this is ensured by $\neg(\rho(\imath, \aleph(\imath_c)) = \rho(\imath', \aleph(\imath_c)))$. The function $\rho(\imath, k)$ return the value of the element at position $k+1$ of layer $\imath$. e.g. If $\imath = 001011$, then $\rho(\imath, 3)$ will return 0.

Line(5) and(6) will append a qualified enabled action pair to the result set when the program goes through all the reords in $S$, and finally return the result. One thing worth mentioning here is that: when we try to add a new enabled action pair to the final result set $T$, we assume that $\{(\ell, \imath), (\ell', \imath')\}$ is the same as $\{(\ell', \imath'), (\ell, \imath)\}$, which means we will only add one of them to the result set $T$, not both of them.

**Example 5.8** *Now we are ready to compute of enable$(E)$ function from Table 5.5. Here we inspect four cases from Example 5.6, where $E = \mathcal{E}^p_\star[\![P_0]\!]$. The calculation is based on the result of Example 3.1 and Example 5.7, by utilizing the function matchaction. The results are shown below:*

$\mathtt{enable}(\mathcal{E}^p_\star[\![P_0]\!])$

| *Case* | $(a)$ | $(b)$ | $(c)$ | $(d)$ |
|---|---|---|---|---|
| | $(3,01):(1,00)$ | $(3,01):(1,00)$ | $(3,01):(1,000)$<br>$(3,01):(1,001)$ | $(3,01):(1,000)$<br>$(3,01):(1,001)$<br>$(1,000):(1,001)$ |

**Fact 5.4** *The function* $\mathtt{enabled}$ *is monotonic.*

## 5.4 Correctness result

We now show the correctness of the automaton constructed by Table 5.1. We shall define $P \rhd E$ by

$$P \rhd E \quad \text{iff} \quad \mathcal{E}^p_\star[\![P]\!] \leq_{\mathfrak{M}ex} E$$

and say that a state denoting the extra extended multiset $E$ represents a process $P$ whenever $P \rhd E$.

We now establish the main result which is independent of the choice of the granularity function $H$:

**Theorem 5.5** *First suppose that the algorithm of Table 5.1 terminates and produces a finite automaton* $(\mathrm{Q}, q_0, \delta, \mathrm{E})$. *Second suppose that If* $P \rightarrow_{\widetilde{\ell\imath}} Q$ *then* $\widetilde{\ell\imath} \in$ *enabled*$(\mathcal{E}^p_\star[\![P]\!])$. *Then if*

$$P \rhd \mathrm{E}[q_s] \quad \textit{and} \quad P \rightarrow_{\widetilde{\ell\imath}} Q$$

*then there exists a unique* $q_t \in \mathrm{Q}$ *such that*

$$Q \rhd \mathrm{E}[q_t] \quad \textit{and} \quad (q_s, \widetilde{\ell\imath}, q_t) \in \delta$$

**Proof.** Let us first inspect one specific transition occurred when generate the automaton. This is related to line (4-6) of Table 5.1.

We have assumed $P \rhd \mathrm{E}[q_s]$, which indicates $\mathcal{E}^p_\star[\![P]\!] \leq_{\mathfrak{M}ex} \mathrm{E}[q_s]$.

Since $P \rightarrow_{\widetilde{\ell\imath}} Q$ it follows that $\widetilde{\ell\imath} \in \mathtt{enabled}(\mathrm{E}[q_s])$ using the second assumption and Fact 5.4 and hence that $\widetilde{\ell\imath}$ is selected for consideration in line(4) of Table 5.1. It follows that line (5) of Table 5.1 produces an extra extended multiset $E$ that $E = \mathtt{transfer}_{\widetilde{\ell\imath}}(\mathrm{E}[q_s])$.

By line (6) of Table 5.1 and definition of `update` in Table 5.2, it is immediate that we identify a state $q_t$ in lines(1-4) of Table 5.2 and after the excution of lines (5-8) of Table 5.2 we have

$$(q_s, \widetilde{\ell\imath}, q_t) \in \delta \text{ and } E \leq_{\mathfrak{M}ex} \mathrm{E}[q_t] \tag{5.1}$$

Since $\mathcal{E}^p_\star[\![P]\!] \leq_{\mathfrak{M}ex} \mathrm{E}[q_s]$ and the monotonicity of $\texttt{transfer}_{\widetilde{\ell\imath}}$ function, we have

$$\texttt{transfer}_{\widetilde{\ell\imath}}(\mathcal{E}^p_\star[\![P]\!]) \leq_{\mathfrak{M}ex} \texttt{transfer}_{\widetilde{\ell\imath}}(\mathrm{E}[q_s]) = E \tag{5.2}$$

Since $P \rightarrow_{\widetilde{\ell\imath}} Q$ from the Proposition 4.6, we have

$$\mathcal{E}^p_\star[\![Q]\!] \leq_{\mathfrak{M}ex} \texttt{transfer}_{\widetilde{\ell\imath}}(\mathcal{E}^p_\star[\![P]\!]) \tag{5.3}$$

Combine equation (5.1)(5.2) and (5.3), we have

$$\mathcal{E}^p_\star[\![Q]\!] \leq_{\mathfrak{M}ex} \mathrm{E}[q_t] \text{ and } (q_s, \widetilde{\ell\imath}, q_t) \in \delta$$

this equal to the conclusion of the theorem that

$$Q \rhd \mathrm{E}[q_t] \quad \textit{and} \quad (q_s, \widetilde{\ell\imath}, q_t) \in \delta$$

The proof above try to inspect line (4-6) of Table 5.1 and capture one specific transition $P \rightarrow_{\widetilde{\ell\imath}} Q$ that satisfies the theorem. This could ensure $(q_s, \widetilde{\ell\imath}, q_t)$ is in the growing automaton after one specific transition.

However, if we consider all transitions happen during the constructing of automaton, from line (5-8) of Table 5.2, we know a previously calculated $q_t$($q$ in Table 5.2, which is generated by the previous transition , here we label it with $q_{t1}$ ) might be removed and replaced with an new $q_t$(see line (7) of Table 5.2, we label it with $q_{t2}$). In this case,$(q_s, \widetilde{\ell\imath}, q_{t1})$ will be replaced with $(q_s, \widetilde{\ell\imath}, q_{t2})$. This doesn't contradict our conclusion: we already have

$$Q \rhd \mathrm{E}[q_{t1}] \quad \textit{and} \quad (q_s, \widetilde{\ell\imath}, q_{t1}) \in \delta$$

Because E[.] grows in a non-decreasing manner during the generation of the automaton (that could be seen from line (6) of Table 5.2), $\mathrm{E}[q_{t1}] \leq_{\mathfrak{M}ex} \mathrm{E}[q_{t2}]$, from this we know $Q \rhd \mathrm{E}[q_{t2}]$. We get

$$Q \rhd \mathrm{E}[q_{t2}] \quad \textit{and} \quad (q_s, \widetilde{\ell\imath}, q_{t2}) \in \delta$$

This means even all transitions are considered, the conclusion is still correct.

The uniqueness of $q_t$ is due to the fact that automation is partially deterministic.
$\square$

The first assumption has already been established which is shown in Theory 5.3. However, due to the restricted time schedule of the project, the second assumption: suppose that If $P \rightarrow_{\widetilde{\ell\imath}} Q$ then $\widetilde{\ell\imath} \in \text{enabled}(\mathcal{E}^p_\star[\![P]\!])$, has not been formally proved yet. Here we only give what we have already established and what need further investigation in order to complete the proof:

*In Table 5.5, in the end we need to show: $\widetilde{\ell\imath} \in R$ (**). Since $\widetilde{\ell\imath}$ is formed from the set $S$ that is at line(3) in Table 5.5, then first we need to prove:*

$$\textit{for some } \alpha \textit{ and } C,\ (\ell, \imath, \alpha, C) \in S \textit{ always holds. (*)}$$

*Since $S$ is computed at line (2) in table 5.5, by invoking the function* `removehidd`*, we shall look into removehidd in Table 5.8. At line (3) of Table* `removehidd`*, the $C$ and $\alpha$ could be easily obtained from $Z((\ell, \imath))$ for each $(\ell, \imath)$. Here $\alpha$ is directly chosen to be part of the element of $S$. For $C$, at line(4) in Table 5.8, no matter $C = \emptyset$ is true or not, there will be a subset of $C$ being chosen for forming an element of $S$. Thus, (*) holds.*
*Then line (4)-(5)in Table 5.5) shows if $C = \emptyset$ , $\widetilde{\ell\imath} \in R$ holds immediately, which is the first case for proving (**). For the second case, we need to prove: if $c \neq \emptyset$, $\widetilde{\ell\imath} \in R$. However, this part of proof we have not figured out yet, we guess we should use some invariant comes from $\mathcal{Y}^p_\star[\![P]\!]$, and then go through function* `matchaction` *for completing the proof.*

## 5.5 Termination of the worklist Algorithm

In SubSection 5.2.3, from Fact 5.2 we pinpoint that $k \neq \top$ doesn't guarantee the granularity function to be finitary as well as stable(finitary is the requirement for worklist algorithm to terminate, which is stated in Theorem 5.3). We also state that we still adopt the granularity function with $k = \top$ for our analysis in this thesis. Here we give the reason by discussing under what condition *must* we use the granularity function with $k \neq \top$ to ensure the algorithm terminate while under what condition $k = \top$ would also ensure the program terminate.

Let's see two PEPA programs in Table 5.10 (we already annotated them with labels and layers). They describe the same system behaviors which are written in two styles.

$$\begin{aligned} \text{let } P_1 &\triangleq (f^1, r).P_2 + P_3 \\ P_2 &\triangleq (g^2, r).P_1 + P_3 \\ P_3 &\triangleq (h^3, r).P_1 \\ \text{in } & P_1^0 \end{aligned}$$

$(a)$

$$\begin{aligned} \text{let } P_1 &\triangleq (f^1, r).P_2 + (h^2, r).P_1 \\ P_2 &\triangleq (g^3, r).P_1 + (h^4, r).P_1 \\ \text{in } & P_1^0 \end{aligned}$$

$(b)$

Table 5.10: PEPA programs written in two styles

| $\ell_{ex}$ | $\mathcal{E}^p_\star[\![P_1]\!]$ | $\mathcal{G}^p_\star[\![P_1]\!](\ell_{ex})$ | $\mathcal{K}^p_\star[\![P_1]\!](\ell_{ex})$ |
|---|---|---|---|
| $(1,0)$ | 1 | $\perp_{\mathfrak{M}ex}[(2,0) \mapsto 1, (3,0) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 1, (3,0) \mapsto 1]$ |
| $(2,0)$ | 0 | $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 1, (3,0) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(2,0) \mapsto 1, (3,0) \mapsto 1]$ |
| $(3,0)$ | 1 | $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 1, (3,0) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(3,0) \mapsto 1]$ |

Table 5.11: functions defined for (a) in Table 5.10

The difference of these program is the behavior of $P_3$ in program (a) is directly added to $P_1$ and $P_2$ in program (b). And consequently the program (a) will not terminate while the program (b) will. This could be explained by their kill and gen functions, which are shown in Table 5.11 and Table 5.12 respectively.

First let's see the reason that (a) will not terminate with $k = \top$: Initially action (3,0) is enabled (from function $\mathcal{E}^p_\star[\![P_1]\!]$). If we take this transition, it will kill (3,0) (from function $\mathcal{K}^p_\star[\![P_1]\!]$) and at the same time generate both (1,0) and (3,0) as enable actions (from function $\mathcal{G}^p_\star[\![P_1]\!]$). Now the system will be at state $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 2, (3,0) \mapsto 1]$ and we could take transition (3,0) again! Noticed the occurrence of enable action (1,0) from 1 becomes to 2, while the occurrence of enable action (3,0) remains the same: take transition (3,0), the

| $\ell_{ex}$ | $\mathcal{E}^p_\star[\![P_1]\!]$ | $\mathcal{G}^p_\star[\![P_1]\!](\ell_{ex})$ | $\mathcal{K}^p_\star[\![P_1]\!](\ell_{ex})$ |
|---|---|---|---|
| $(1,0)$ | 1 | $\perp_{\mathfrak{M}ex}[(3,0) \mapsto 1, (4,0) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 1, (2,0) \mapsto 1]$ |
| $(2,0)$ | 1 | $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 1, (2,0) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 1, (2,0) \mapsto 1]$ |
| $(3,0)$ | 0 | $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 1, (2,0) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(3,0) \mapsto 1, (4,0) \mapsto 1]$ |
| $(4,0)$ | 0 | $\perp_{\mathfrak{M}ex}[(1,0) \mapsto 1, (2,0) \mapsto 1]$ | $\perp_{\mathfrak{M}ex}[(3,0) \mapsto 1, (4,0) \mapsto 1]$ |

Table 5.12: functions defined for (b) in Table 5.10

enabled actions being generated always include the actions being killed! If we keep executing action (3,0), the new state will contain a growing number of enabled action (1,0) but always keeps one occurrence of action (3,0). Since in this way all new states generated will be discriminated by granularity function with $k = \top$ and they can't be reused among each other, the program will never stop. Now it is easy to know why Fact 5.2 doesn't hold for $k = \top$: it is not finitary any more: recall in Subsection 5.2.3, the definition of finitary requires to have finite sets $\mathbf{Labex}_{fin}$. Our growing new state doesn't satisfy this condition.

The program (b) will not suffer the problem program (a) encounters. This is because the kill function of (b) is much more accurate than that in program (a). Let's see (2,0) and (4,0), they will kill themselves as well as the actions in other branches of the sequential process: (2,0) will kill (1,0) and (2,0); (4,0) will kill (3,0) and (4,0). However, the action (3,0) in program (a) will only kill itself, even though they stay in both $P_1$ and $P_2$ and *should* kill the branch next to it, like the action (2,0) and (4,0) in program (b) perform (from the semantic point of view). In program (a), just executing (3,0), we don't know weather it comes from $P_1$ or $P_2$, thus we only kill itself: it is safe to kill less!

From this example, we can see that program (b) owns the kill and gen functions that directly correspond to the semantics of the program, while program (a) doesn't.

We can conjecture that if we write PEPA program in the style of program (b): always put constant sequential component into a prefix sequential component, our kill and gen function will always be accurate and they could capture the system behavior exactly. Since there will be no approximation any more, the system will only have a finite number of states.

Our PEPA programs in the thesis are all written in the style of program (b), so it is very safe to use $k = \top$ when specifying the granularity function.

Chapter 6

# Accelerate the Analysis

In the previous chapters, we have defined essential domains and several auxiliary functions. With worklist algorithm we build the automaton that could capture the properties of the interactions among several sequential processes.

Each sequential process is simulated and allocated a layer depending on its position, determined by the model component definition of the program. So we could actually discriminate the same actions which belong to different sequential processes even though they behave exactly the same. For instance, recall two sequential processes $S$ in Example 2.4. Their layers are determined by the model component definition as shown below:

$$(\underbrace{S}_{000} \underset{\{\}}{\bowtie} \underbrace{S}_{001}) \underset{\{g,p\}}{\bowtie} \underbrace{Q}_{01}$$

The actions in $S^{000}$ and $S^{001}$ take different seats in the extra extended multiset when we come to the analysis. Taking the definition of $S$ from Example 2.3, already equipped with label, we annotate the actions of $S^{000}$ as: (1,000) and (2,000) while actions of $S^{001}$ as: (1,001) and (2,001). The entry of extra extended domain for this program will contain these four items. Since the worklist algorithm will work on this domain, we will build an automaton that consists of states and transitions contributed by these four items(cf. Figure 5.1).

However, sometimes we are not that concern about the difference of same actions between $S^{000}$ and $S^{001}$(for example, action labeled 1 will only be noted as (1,00) instead of (1,000), (1,001)). In this way, we could cut down the total number of items in the domain of the analysis. In other words, we are willing to reduce the precision of the analysis by ignoring the difference of sequential process with similar behavior. The motivation for doing so is the benefits from two perspective:

- Generally, lowing the precision (ignore unnecessary details) is always accompanied the speed improvement of analysis. In other words, it will cut down the computing demands.
- In our case, we will build up a more succinct automaton that contains fewer states and transitions. Consequently the graphic representations of the automaton will be illustrated in fewer nodes and edges that enhance the readability of the pictures.

In the following subsections, we will develop techniques to address this issue.

## 6.1 Method 1

The first method is to address the situation mentioned above: given a program, if there exists some sequential processes conjointly cooperating with each other on $\emptyset$ set: $(((S \underset{\{\}}{\bowtie} S) \underset{\{\}}{\bowtie} S) \cdots \underset{\{\}}{\bowtie} S)$ , we could group them and remain only *one* sequential process $S$ to represent *all* of them. Take it one step further, we could group any subparts of program together if they behave exactly the same and conjointly cooperate on $\emptyset$. The subpart of the program could be some sequential processes operating on the hidden or cooperate combinator with *non* empty set or empty set. For instance: $((S \underset{\{\beta\}}{\bowtie} S)/\{\alpha\}) \underset{\{\}}{\bowtie} ((S \underset{\{\beta\}}{\bowtie} S)/\{\alpha\})$ could be simplify to $((S \underset{\{\beta\}}{\bowtie} S)/\{\alpha\})$.

Specifically, we will modify the analysis we have already developed in two steps to accomplish the above goal:

**Step1** First we should simplify the model component definition of the program as much as we can as long as we conform to the idea presented above. Furthermore, we will track the number of subparts(should be same) that participate in grouping event. They will be used in the second step.

Exposed actions for let $C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k$ in $P_0$

$$\begin{aligned}
\mathcal{E}^p[\![P_1 \underset{L}{\bowtie} P_2]\!]^{\imath}_{\lambda} &= \lambda \cdot_{\mathfrak{M}_{ex}} (\mathcal{E}^p[\![P_1]\!]^{\imath 0} +_{\mathfrak{M}_{ex}} \mathcal{E}^p[\![P_2]\!]^{\imath 1}) \\
\mathcal{E}^p[\![P/L]\!]^{\imath}_{\lambda} &= \lambda \cdot_{\mathfrak{M}_{ex}} (\mathcal{E}^p[\![P]\!]^{\imath}) \\
\mathcal{E}^p[\![S]\!]^{\imath}_{\lambda} &= \text{let } (\ell_1 \mapsto n_1, \cdots, \ell_n \mapsto n_n) = \mathcal{E}^s_{\star}[\![S]\!] \\
&\quad\ \text{in } \lambda \cdot_{\mathfrak{M}_{ex}} (\bot_{\mathfrak{M}_{ex}}[(\ell_j, \imath) \mapsto n_j]) \text{ where } \ell_j \in \text{domExlayer}(\bot_{\mathfrak{M}_{ex}}, \imath) \\
&\quad\ \text{ and } j \in \{1, \cdots, n\} \\
\mathcal{E}^p_{\star}[\![P]\!] &= \mathcal{E}^p[\![P]\!]^0
\end{aligned}$$

Table 6.1: The redefined $\mathcal{E}^p$ function

**Example 6.1** *For instance,*

$$(((S \underset{\{\}}{\bowtie} S) \underset{\{\}}{\bowtie} S)/\{\alpha\}) \underset{\{\}}{\bowtie} (((S \underset{\{\}}{\bowtie} S) \underset{\{\}}{\bowtie} S)/\{\alpha\})$$

*could be transformed into*

$$(S^3/\{\alpha\})^2$$

*where 3 is annotated to S to indicate we have three copy of S cooperating* $\emptyset$ *, and 2 is annotated to the hidden combinators to memorize there are two entities of* $(S^3/\{\alpha\})$*.*

**Step2** Second we will calculate the $\mathcal{E}^s_{\star}$ and $\mathcal{E}^p_{\star}$. $\mathcal{E}^s_{\star}$ is defined exactly the same as in Table 3.1. However, $\mathcal{E}^p_{\star}$ is redefined to take into account the number of the same subparts of the program, which is shown in Table 6.1.

In Table 6.1, $\cdot_{\mathfrak{M}_{ex}}$ is scalar multiplication defined by $0 \cdot_{\mathfrak{M}_{ex}} M = \bot_{\mathfrak{M}_{ex}}$ and $(\lambda + 1) \cdot_{\mathfrak{M}_{ex}} M = (\lambda \cdot_{\mathfrak{M}_{ex}} M) +_{\mathfrak{M}_{ex}} M$. Notice we have added a new parameter $\lambda$ to each model component $\mathcal{E}^p[\![]\!]^{\imath}_{\lambda}$, representing the number of same copy of this model component(subpart of program), which is recorded in the first step. The new function scalar multiplies $\lambda$ to the result of old function for each case.

**Example 6.2** *Look into the Example 6.1 again. Once we get* $(S^3/\{\alpha\})^2$*, we apply the* $\mathcal{E}^p[\![]\!]^{\imath}_{\lambda}$ *function to it. Suppose in S we will have one occurrence of each exposed actions, finally we will obtain* $3 * 2 = 6$ *occurrence of the each exposed actions.*

The differences of method1 from the original analysis are presented in Step1 and Step2. Other functions like $\mathcal{G}^s_{\star}$, $\mathcal{K}^s_{\star}$ and worklist algorithm related functions just remain unchanged (However, they will work on the new domain which is caused by the modification in step1). The key point is: the simplification and grouping shouldn't violate the original semantics of the automaton.

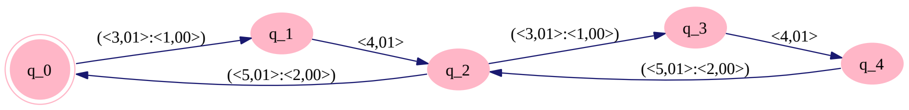

Figure 6.1: Automaton of
$(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ built by method 1

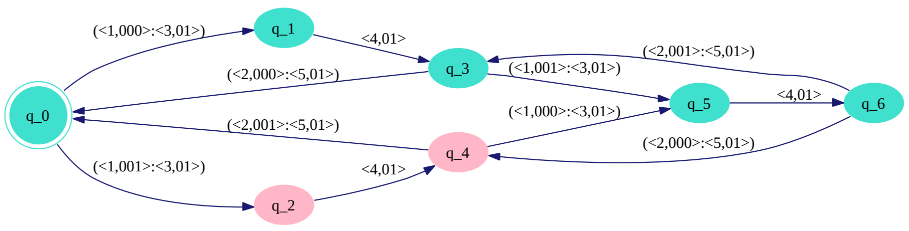

Figure 6.2: Automaton of
$(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ built by original analysis

**Example 6.3** *Let's reconstruct the automaton build in Example 5.1, on the case $(S \underset{\{\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ originally from Example 2.1, this time we obtain the finite automaton shown in Figure 6.1. The initial state is $q_0$ and there are five states in the automaton: $q_0, \cdots, q_4$, instead there was seven states in Example 5.1. The transitions among these states are clearly illustrated in the Figure 6.1. We redrawn the automaton generated by the original analysis in Example 5.1(Figure 5.1), and shown in Figure 6.2. This time, if we only observe the blue nodes $(q_0, q_1, q_3, q_5, q_6)$ in Figure 6.2, it is obviously that the shape of those nodes are exactly the same as shape formed from all nodes in Figure 6.1. Essentially, the Figure 6.2 is symmetric which illustrates the new automaton is able to differentiate between action (1,000) and (1,001) while Figure 6.1 can't do so but just regard them as one action (1,00). This is easily seen by looking into transitions relationship from these two figures.*

*The exposed actions(the extra extended multisets) correspond to each state are listed below*

| $q$ | E$[q]$ |
|---|---|
| $q_0$ | $\bot_{\mathfrak{M}ex}[(1,00) \mapsto 2, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\bot_{\mathfrak{M}ex}[(1,00) \mapsto 1, (2,00) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_2$ | $\bot_{\mathfrak{M}ex}[(1,00) \mapsto 1, (2,00) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_3$ | $\bot_{\mathfrak{M}ex}[(2,00) \mapsto 2, (4,01) \mapsto 1]$ |
| $q_4$ | $\bot_{\mathfrak{M}ex}[(2,00) \mapsto 2, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |

*If we look into $q_0$, we will see the number of exposed actions at (1,00) is 2. It replaces the counterpart in Example 5.1, where (1,000) is 1 and (1,001) is 1.*

## 6.2 Method 2

In the previous section, we are able to speed up the analysis in case the program includes a subpart whose structure in the form: $(((S \underset{\{\}}{\bowtie} S) \underset{\{\}}{\bowtie} S) \cdots \underset{\{\}}{\bowtie} S)$. Now we attempt to go one step further: to simplify a program with a subpart whose structure looks like $(((S \underset{\{L\}}{\bowtie} S) \underset{\{L\}}{\bowtie} S) \cdots \underset{\{L\}}{\bowtie} S)$.

The idea is again to group the subparts of program with the same behavior into one identity(subpart) and let it represent all of them, in order to decrease the total number of actions in the working domain when constructing the automaton. However, the job is not as easy as in method one, because now these subparts themselves cooperating on a non-empty set $L$. If we just do the same job as before, we will *lose* all possible interaction between them, which apparently violate the semantics of the program! Imagine there exist two exposed actions of the same labex from $S$, they might cooperate with themselves!

**Example 6.4** *For example: if we use method one to analysis the case from Example 2.1: $(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$. In step 1, we will simplify the formulae to $(S^2 \underset{\{g,p\}}{\bowtie} Q)^1$. Even though in step 2 we will double the occurrence of all the exposed actions from $S$ in state $q_0$ and let them cooperate with actions from $Q$, we still lose the possible interaction between two $S$: (1,001) and (1,000) or (2,001) and (2,000).*

From the example above, we learn that if we want to group $S$ at this situation and not lose interactions, we shouldn't simply get rid of the cooperation set $L$ without taking further action. We have to track set $L$ because for each action within $S$, set $L$ judges whether it should cooperate with action with the same type in $S$. Then when calculate the enable function, we should treat the actions from $S$ exceptionally. Not only should we find all possible interactions between actions from $S$ and actions from other part of the program, we will also try to evaluate the actions from the same layer(all actions in $S$ will have the same layer, but with various labels), based on the information we already know–the type of actions should cooperate within S.

Now we will modify the analysis again and try to follow the idea above and design a new version of analysis in several steps.

**Step1** First we will simplify the model component definition of program, similar to what we have done at step1 in method one. However this time not only will we track the number of subparts that participate in grouping event, we also append an additional information to $S$(in the program tree): the set $L$. This information would be used in step2 for new version of function $\mathcal{Y}^p_\star$.

**Example 6.5**

$$(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$$

*will be transformed it into*

$$(S^2_{\{p,g\}} \underset{\{p,g\}}{\bowtie} Q)^1$$

*The difference from method one is we annotate another parameter to $S$, shown as the subscript $\{p,g\}$, to depict cooperation set for $S$.*

**Step2** Second step is exactly the same as that in method one, where we choose $\mathcal{E}^p_\star$ defined in Table 6.1.

**Step3** Third step is to modify the function $\mathcal{Y}^p_\star$. First, we will present a new domain $\mathcal{Y}^p_\star$ works on. Instead of the $\mathfrak{Y}$ domain defined on page 56 in section 5.3.1, we modify it to a new domain $\mathfrak{Y}'$ with another **Bool** field.

$$\mathfrak{Y}' = \textbf{Labex} \to (\textbf{Act} \times \wp(\textbf{Layer}) \times \wp(\textbf{Layer}) \times \textbf{Bool}))$$

Value in Bool field tells that if this Labex(action) needs to *cooperate* with other actions situated at the same layer of the same type(include itself). Then the redefined $\mathcal{Y}^p_\star$ is shown in Table 6.2. Only the case for constant component is changed. This constant component has an extra parameter $L$ which is obtained in step 1 ($L = \{p, g\}$ for $S^2_{\{p,g\}}$ in Example 6.5). We will then set the Bool field for each Labex according to whether its action type is in set $L$. and it will be used in our new function `matchgroupaction` which will be demonstrated in step 5.

**Example 6.6** *Here we take (c)(d) cases in Example 5.6 and recalculate the $\mathcal{Y}^p_\star$ for each of them, which could be seen below:*

Exposed actions for let $C_1 \triangleq S_1; \cdots ; C_k \triangleq S_k$ in $P_0$

$$
\begin{aligned}
\mathcal{Y}^p[\![P_1 \underset{L}{\bowtie} P_2]\!](\imath, coop, hidd) &= \text{let } coop' = \text{addlayers}(coop, L, \imath) \\
&\quad \text{in } \mathcal{Y}^p[\![P_1]\!](\imath 0, coop', hidd) \cup \mathcal{Y}^p[\![P_2]\!](\imath 1, coop', hidd) \\
\mathcal{Y}^p[\![P/L]\!](\imath, coop, hidd) &= \text{let } hidd' = \text{addlayers}(hidd, L, i') \\
&\qquad\quad \text{and } \imath = \imath' 0 \text{ or } \imath = \imath' 1 \\
&\quad \text{in } \mathcal{Y}^p[\![P]\!](\imath, coop, hidd') \\
\boldsymbol{\mathcal{Y}^p[\![S]\!](\imath, coop, hidd)_L} &= \text{let } [(\ell_1, \alpha_1), \cdots, (\ell_n, \alpha_n)] = \mathcal{Y}^s_\star[\![S]\!] \\
&\quad \text{in } [(\ell_1, \imath) \mapsto (\alpha_1, \partial(coop, \alpha_1), \partial(hidd, \alpha_1), t_1), \cdots, \\
&\quad (\ell_n, \imath) \mapsto (\alpha_n, \partial(coop, \alpha_n), \partial(hidd, \alpha_n), t_n)] \\
&\quad \text{where } t_i = \begin{cases} true & \text{if } \alpha \in L \\ false & \text{if } \alpha \notin L \end{cases} \text{ Here } i \in \{1, \cdots, n\} \\
\mathcal{Y}^p_\star[\![P]\!](0, \emptyset, \emptyset) &= \mathcal{Y}^p[\![P]\!](0, \emptyset, \emptyset)
\end{aligned}
$$

Table 6.2: The redefined $\mathcal{Y}^p$ function

| $\mathcal{Y}^p_\star$ *of* | $(\underbrace{S}_{000} \underbrace{\bowtie_{\{\}}}_{00} \underbrace{S}_{001}) \underbrace{\bowtie_{\{g,p\}}}_{0} \underbrace{Q}_{01}$ | | | | $(\underbrace{S}_{000} \underbrace{\bowtie_{\{g,p\}}}_{00} \underbrace{S}_{001}) \underbrace{\bowtie_{\{g,p\}}}_{0} \underbrace{Q}_{01}$ | | | |
|---|---|---|---|---|---|---|---|---|
| *labex* | *action* | *coop* | *hidd* | *t* | *action* | *coop* | *hidd* | *t* |
| $(1, 00)$ | $g$ | $\{0\}$ | $\emptyset$ | *false* | $g$ | $\{00, 0\}$ | $\emptyset$ | *true* |
| $(2, 00)$ | $p$ | $\{0\}$ | $\emptyset$ | *false* | $p$ | $\{00, 0\}$ | $\emptyset$ | *true* |
| $(3, 01)$ | $g$ | $\{0\}$ | $\emptyset$ | *false* | $g$ | $\{0\}$ | $\emptyset$ | *false* |
| $(4, 01)$ | $h$ | $\emptyset$ | $\emptyset$ | *false* | $h$ | $\emptyset$ | $\emptyset$ | *false* |
| $(5, 01)$ | $p$ | $\{0\}$ | $\emptyset$ | *false* | $p$ | $\{0\}$ | $\emptyset$ | *false* |
| Case | $(c)$ | | | | $(d)$ | | | |

*In both cases, we get 5 labex in the domain compared with 7 labex in Example 5.6. Case(c) doesn't have any item whose t value is set to true, while Case(d) shows a little different: we set t value of (1,00) and (2,00) to be true, indicating they might further cooperate with actions at layer 00 within S.*

**Step4** The fourth step is to modify the function `removehidd`. First, we will present a new domain that `removehidd` works on. Instead of the $\mathfrak{U}$ domain defined on page 56 in section 5.3.1, we modify it to a new domain $\mathfrak{U}'$ (in Table 6.2,$S \in \mathfrak{U}'$) with new field **Bool** that already discussed in step 3, and another new field $\mathbb{N} \cup \{\top\}$ , which records the number of exposed

*The procedure* `removehidd`$(E, Z)$

```
(1) $S := \emptyset$;
(2) for each $(\ell, \imath) \in \texttt{domEx}(E)$ do
**(3)**    let $(\alpha, C, H, \boldsymbol{t}) = Z((\ell, \imath))$ and $\boldsymbol{n = E((\ell, \imath))}$
(4)    in if $\neg(C = \emptyset)$
(5)       then for each $\imath_c \in C$ do
(6)               if there exists $i_h \in H$
(7)                  with $\aleph(\imath_c) <= \aleph(\imath_h)$
(8)               then $C := C \setminus \imath_c$;
**(9)**        $S := S \cup (\ell, \imath, \alpha, C, \boldsymbol{n}, \boldsymbol{t})$;
(10) return $S$;
```

Table 6.3: The redefined function `removehidd`

actions of this labex.

$$\begin{aligned} \mathfrak{U}' &= \wp(\textbf{Labex} \times \textbf{Act} \times \wp(\textbf{Layer}) \times \mathbb{N} \cup \{\top\} \times \textbf{Bool}) \\ &= \wp(\textbf{Lab} \times \textbf{Layer} \times \textbf{Act} \times \wp(\textbf{Layer}) \times \mathbb{N} \cup \{\top\} \times \textbf{Bool}) \end{aligned}$$

The redefined `removehidd` is shown in Table 6.2, where we emphasize the place we changed with bold font, based on the original function that defined in Table 5.8. Clearly, line (3) generates the t and n, while line (9) put these two additional parameters to variable $S$.

**Step5** In the last step, we have already introduced parameters n,t in $S$($S$ is a set that contains all labex and thier relevant information to be used for performing the analysis)at redefined function `removehidd`. Based on the new information, now we are able to detect: for each labex in $S$, if there exists other labex (include itself) in $S$ to cooperate with. This is done by a new developed function `matchgroupaction`$((\ell, \imath, \alpha, C, n, t), S)$, as shown in Table 6.4, which will then be used in the redefined `enabled` function to be presented in the next step.

Line (3) to (5) of Table 6.4 indicate that under two conditions any two actions might cooperate within a group:

- Two actions are under the same layer($\imath = \imath'$). They are in fact the same actions($\ell = \ell'$), and the number of exposed actions is larger than 2($n \geq 2$), in addition they are ready to cooperate within a group($t = true$).

`matchgroupaction`$((\ell, \imath, \alpha, C, n, t), S)$

```
(1) $T := \emptyset$;
(2) for each $(\ell', \imath', \alpha', C', n', t') \in S$ do
(3)     if $(\imath = \imath' \wedge \ell = \ell' \wedge n \geq 2 \wedge t = true) \vee$
(4)        $(\imath = \imath' \wedge \alpha = \alpha' \wedge t = true \wedge t' = true)$
(5)     then $T := T \cup \{((\ell, \imath), (\ell', \imath'))\}$;
(6) return $T$;
```

Table 6.4: function `matchgroupaction`

- Two actions are under the same layer($\imath = \imath'$). However, even though they are not the same actions, they have the same action type($\alpha = \alpha'$), and they both ready to cooperate within a group($t = true$ and $t' = true$).

**Step6** Now we come to the last change of the analysis: the `enable` function, which is illustrated in Table 6.5. Compared to the old function specified in Table 5.5, we invoke a new defined function `matchgroupaction` to collect all actions that could cooperate at the same layer. It then proceed basic function just as we defined in the old function, except we add an additional condition at Line (6), meaning only the action that is unable to cooperate with other actions at the same layer($t = false$) and needn't cooperate with actions outside this group($C = \emptyset$), will we believe this action could proceed independently and then we add it alone to the $R$ set.

Other than these six main changes, nothing has been changed including the worklist algorithm, and any other parts of the analysis such as $\mathcal{G}^p_\star$,$\mathcal{K}^p_\star$ etc. In this way we further speed up the analysis.

**Example 6.7** *To show the effect of this analysis(method 2), let's look into* $(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ *again. Here we will compare two automatons, generated by the original analysis and the new analysis we developed here(method 1 and the original analysis will have the same result on this program). The automaton built by original analysis is shown in Figure 5.9, and now we redrawn it in Figure 6.3, while the states information is illustrated in Example 5.2 at table "Round 8".*

*The automaton drawn by method2 is shown in Figure 6.4, while the states information is shown below:*

`enabled`$(E)$

```
(1) $Z := \mathcal{Y}^{p}_{\star}[\![P_0]\!](0, \emptyset, \emptyset);\quad R := \emptyset;$
(2) $S :=$ removehidd$(E, Z);$
(3) for each $(\ell, \imath, \alpha, C, \boldsymbol{n}, \boldsymbol{t}) \in S$ do
(4)     $\boldsymbol{R := R \cup}$ matchgroupaction$\boldsymbol{((\ell, \imath, \alpha, C, n, t), S)}$
(5)     if $C = \emptyset$
(6)     then if $t = false$
(7)          then $R := R \cup \{(\ell, \imath)\}$
(8)     else for each $\imath_c \in C$ do
(9)               $R := R \cup$ matchaction$(\imath_c, (\ell, \imath, \alpha, C), S)$
(8) return $R$;
```

Table 6.5: `The redefined enabled(`$E$`) procedure`

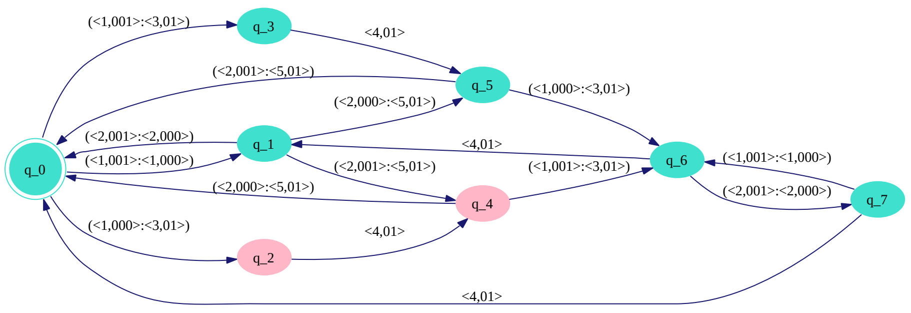

Figure 6.3: Automaton of $(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ built by Original Analysis

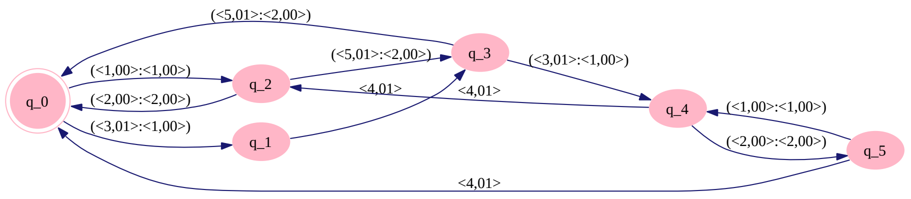

Figure 6.4: Automaton of $(S \underset{\{g,p\}}{\bowtie} S) \underset{\{g,p\}}{\bowtie} Q$ built by Method 1

| $q$ | E[$q$] |
|---|---|
| $q_0$ | $\bot_{\mathfrak{M}ex}[(1,00) \mapsto 2, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_1$ | $\bot_{\mathfrak{M}ex}[(1,00) \mapsto 1, (2,00) \mapsto 1, (4,01) \mapsto 1]$ |
| $q_2$ | $\bot_{\mathfrak{M}ex}[(2,00) \mapsto 2, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_3$ | $\bot_{\mathfrak{M}ex}[(1,00) \mapsto 1, (2,00) \mapsto 1, (3,01) \mapsto 1, (5,01) \mapsto 1]$ |
| $q_4$ | $\bot_{\mathfrak{M}ex}[(2,00) \mapsto 2, (4,01) \mapsto 1]$ |
| $q_5$ | $\bot_{\mathfrak{M}ex}[(1,00) \mapsto 2, (4,01) \mapsto 1]$ |

*Now if we only observe the blue nodes ($q_0, q_1, q_3, q_5, q_6, q_7$) in Figure 6.3, it is obviously that the shape of those nodes are exactly the same as the shape formed from all nodes in Figure 6.4. Essentially, the Figure 6.3 is partly symmetric($q_2$ and $q_3$; $q_4$ and $q_5$) and its automaton could differentiate actions from S with the same label, while Figure 6.4 can't do so but just regard them as one action. This is easily seen by looking into transitions relationship from these two figures.*

## 6.3 Three Approaches of Analysis on Two Examples

In this section we are going to present the effect by adopting two methods for improving the speed of analysis. Here we will consider two variants of Milner's process for a jobshop [31].

The program Jobshop1 is displayed in Table 6.6, processes PoliteWorker0 to PoliteWorker6 and Worker0 to Worker8 describe the behavior of a worker that try to get both tools (hammer and chisel) to work. Normally afterwards it will release these two tools, however the worker could cooperate with other workers to go on strike and then throw away both tools. Processes Hammer_free and Hammer_taken simulate the behavior of one hammer while Chisel_free and Chisel_taken describe the behavior of one Chisel. The program starts with processes involving two workers, three hammers and three chisels.

The program Jobshop2 is displayed in Table 6.7. Other than the processes in Jobshop1, it has another worker with deferent behavior which will simply get two tools in sequence and then release them without intention to go on strike. The program starts with processes involving three workers. Two of them behave as workers in Jobshop1 and the other worker behaves as described just now, four hammers and one chisel.

$$
\begin{aligned}
&\text{PoliteWorker0} &&\stackrel{def}{=} &&(\text{get\_hammer, r}).\text{PoliteWorker1} + (\text{get\_chisel, r}).\text{PoliteWorker2}\\
&\text{PoliteWorker1} &&\stackrel{def}{=} &&(\text{get\_chisel, r}).\text{PoliteWorker3} + (\text{release\_hammer, r}).\text{PoliteWorker0}\\
&\text{PoliteWorker2} &&\stackrel{def}{=} &&(\text{get\_hammer, r}).\text{PoliteWorker3} + (\text{release\_chisel, r}).\text{PoliteWorker0}\\
&\text{PoliteWorker3} &&\stackrel{def}{=} &&(\text{work, r}).\text{PoliteWorker4} + (\text{strike, r}).\text{Worker4}\\
&\text{PoliteWorker4} &&\stackrel{def}{=} &&(\text{release\_hammer, r}).\text{PoliteWorker5} + (\text{release\_chisel, r}).\text{PoliteWorker6}\\
&\text{PoliteWorker5} &&\stackrel{def}{=} &&(\text{release\_chisel, r}).\text{PoliteWorker0}\\
&\text{PoliteWorker6} &&\stackrel{def}{=} &&(\text{release\_hammer, r}).\text{PoliteWorker0}\\
\\
&\text{Worker0} &&\stackrel{def}{=} &&(\text{get\_hammer, r}).\text{Worker1} + (\text{get\_chisel, r}).\text{Worker2}\\
&\text{Worker1} &&\stackrel{def}{=} &&(\text{get\_chisel, r}).\text{Worker3}\\
&\text{Worker2} &&\stackrel{def}{=} &&(\text{get\_hammer, r}).\text{Worker3}\\
&\text{Worker3} &&\stackrel{def}{=} &&(\text{work, r}).\text{Worker4} + (\text{go\_on\_strike, r}).\text{Worker7}\\
&\text{Worker4} &&\stackrel{def}{=} &&(\text{release\_hammer, r}).\text{Worker5} + (\text{release\_chisel, r}).\text{Worker6}\\
&\text{Worker5} &&\stackrel{def}{=} &&(\text{release\_chisel, r}).\text{Worker0}\\
&\text{Worker6} &&\stackrel{def}{=} &&(\text{release\_hammer, r}).\text{Worker0}\\
&\text{Worker7} &&\stackrel{def}{=} &&(\text{throw\_away\_tools, r}).\text{Worker8}\\
&\text{Worker8} &&\stackrel{def}{=} &&(\text{resolve\_strike, r}).\text{Worker0}\\
\\
&\text{Hammer\_free} &&\stackrel{def}{=} &&(\text{get\_hammer, infty}).\text{Hammer\_taken}\\
&\text{Hammer\_taken} &&\stackrel{def}{=} &&(\text{release\_hammer, infty}).\text{Hammer\_free}\\
&\text{Chisel\_free} &&\stackrel{def}{=} &&(\text{get\_chisel, infty}).\text{Chisel\_taken}\\
&\text{Chisel\_taken} &&\stackrel{def}{=} &&(\text{release\_chisel, infty}).\text{Chisel\_free}
\end{aligned}
$$

$$
\begin{aligned}
&(\text{PoliteWorker0} \underset{\{strike\}}{\bowtie} \text{PoliteWorker0})\\
&\quad \bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}}\\
&((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free}) \underset{\{\}}{\bowtie}\\
&\ (\text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free}))
\end{aligned}
$$

Table 6.6: *jobshop_1*

| | | |
|---|---|---|
| PoliteWorker0 | $\overset{def}{=}$ | (get_hammer, r).PoliteWorker1 +(get_chisel, r).PoliteWorker2 |
| PoliteWorker1 | $\overset{def}{=}$ | (get_chisel, r).PoliteWorker3 +(release_hammer, r).PoliteWorker0 |
| PoliteWorker2 | $\overset{def}{=}$ | (get_hammer, r).PoliteWorker3 + (release_chisel, r).PoliteWorker0 |
| PoliteWorker3 | $\overset{def}{=}$ | (work, r).PoliteWorker4 + (strike, r).Worker4 |
| PoliteWorker4 | $\overset{def}{=}$ | (release_hammer, r).PoliteWorker5 + (release_chisel, r).PoliteWorker6 |
| PoliteWorker5 | $\overset{def}{=}$ | (release_chisel, r).PoliteWorker0 |
| PoliteWorker6 | $\overset{def}{=}$ | (release_hammer, r).PoliteWorker0 |
| Worker0 | $\overset{def}{=}$ | (get_hammer, r).Worker1 + (get_chisel, r).Worker2 |
| Worker1 | $\overset{def}{=}$ | (get_chisel, r).Worker3 |
| Worker2 | $\overset{def}{=}$ | (get_hammer, r).Worker3 |
| Worker3 | $\overset{def}{=}$ | (work, r).Worker4 + (go_on_strike, r).Worker7 |
| Worker4 | $\overset{def}{=}$ | (release_hammer, r).Worker5 + (release_chisel, r).Worker6 |
| Worker5 | $\overset{def}{=}$ | (release_chisel, r).Worker0 |
| Worker6 | $\overset{def}{=}$ | (release_hammer, r).Worker0 |
| Worker7 | $\overset{def}{=}$ | (throw_away_tools, r).Worker8 |
| Worker8 | $\overset{def}{=}$ | (resolve_strike, r).Worker0 |
| SimpleWorker0 | $\overset{def}{=}$ | (get_hammer, r).SimpleWorker1 |
| SimpleWorker1 | $\overset{def}{=}$ | (get_chisel, r).SimpleWorker2 |
| SimpleWorker2 | $\overset{def}{=}$ | (work, r).SimpleWorker3 |
| SimpleWorker3 | $\overset{def}{=}$ | (release_chisel, r).SimpleWorker4 |
| SimpleWorker4 | $\overset{def}{=}$ | (release_hammer, r).SimpleWorker0 |
| Hammer_free | $\overset{def}{=}$ | (get_hammer, infty).Hammer_taken |
| Hammer_taken | $\overset{def}{=}$ | (release_hammer, infty).Hammer_free |
| Chisel_free | $\overset{def}{=}$ | (get_chisel, infty).Chisel_taken |
| Chisel_taken | $\overset{def}{=}$ | (release_chisel, infty).Chisel_free |

$$
\begin{aligned}
&(\text{PoliteWorker0} \underset{\{strike\}}{\bowtie} \text{PoliteWorker0} \underset{\{strike\}}{\bowtie} \text{SimpleWorker0})\\
&\quad \bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}}\\
&((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free})\\
&\quad \underset{\{\}}{\bowtie} (\text{Chisel\_free}))
\end{aligned}
$$

Table 6.7: *jobshop_2*

These two programs are much larger than the programs in the previous examples. For each program we performed the original analysis, analysis based on method 1 and analysis based on method 2 and build their corresponding automatons.

The basic configuration of the computer performing the analysis(throughout the whole thesis) is: Intel Pentium M processor 1.60GHz, 1.00GB of RAM and runs on Microsoft Windows XP Professional Service Pack 2.

The results is shown in Table 6.8. It could be seen clearly that in both cases, methods 2 generates the automatons with least states and spends least time, while the original analysis consumes the maximum time among three analysis and also generates the number of states that higher than the others.

The accelerating rates between time used for running jobshop1 by method2 and original analysis is:

$$r_{20} = \frac{\text{4 hours 32 minutes 37 seconds}}{\text{6 seconds}} = 2726$$

While the accelerating rates between time used for running jobshop1 by method1 and original analysis is:

$$r_{10} = \frac{\text{4 hours 32 minutes 37 seconds}}{\text{1 minutes 1 second}} = 268$$

The accelerating rates between time used for running jobshop1 by method2 and method1 is:

$$r_{21} = \frac{\text{1 minutes 1 seconds}}{\text{6 seconds}} = 10$$

Obviously, the acceleration of the analysis works! Method 1 greatly improves the original analysis($r_{10} = 258$), the method 2 slightly improves the method1($r_{21} =$

| Program | Item | Original analysis | Method1 | Method2 |
|---|---|---|---|---|
| jobshop1 | States of Automaton | 1371 | 235 | 131 |
| | Time Cost | 4 hours 32 minutes 37 seconds | 1 minute 1 second | 6 seconds |
| jobshop2 | States of Automaton | 557 | 117 | 66 |
| | Time Cost | 31 minutes 59 seconds | 17 seconds | 2 seconds |

Table 6.8: Analysis results on jobshop1 and jobshop2

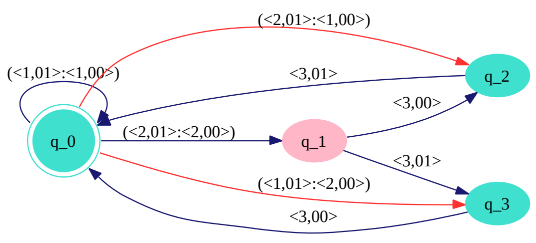

Figure 6.5: Automaton built by original analysis

10) and lastly method2 improves the original analysis by 2726 times($r_{20} = 2726$). The result from jobshop2 shows the similar conclusion and we will not explain it here.

## 6.4 Discussion of method1 and method2

We have seen that both method1 and method2 could increase the speed of the analysis, nevertheless these two techniques hold essentially different characteristics from each other.

Since method2 is based on the development of method1, it inherits all features method1 have: both of them could collapse sequential processes conjointly cooperating on *empty* set. What is more, method 2 goes one step further than method 1 on that it could collapse sequential processes conjointly cooperating on *non-empty* set. Here we pinpoint that this extra feature the method2 owns will potentially add more interactions (some actions are not allowed to cooperate with each other based on the original semantics of the program, but method2 might make them possible) to the automaton built by the original analysis. In contract, method1 will not add more interactions(from the semantics point of view) than the automaton built before.

**Example 6.8** *Let's take the following program*

$$\begin{aligned}&\text{let } S \triangleq (g^1, r_1).S + (g^2, r_2).(p^3, r_3).S \\ &\text{in } \ S^{00} \underset{\{g\}}{\bowtie} S^{01}\end{aligned}$$

*If we use original analysis, we will get an automaton shown in Figure 6.5. The table below shows the exposed actions corresponding to each state.*

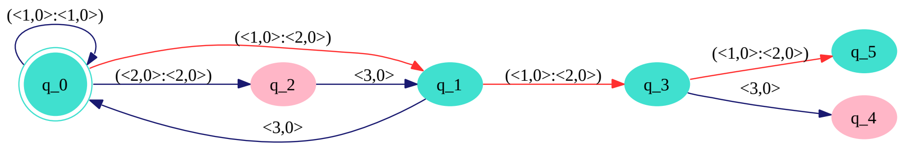

Figure 6.6: Automaton built by method2, round 4

| $q$ | E[$q$] |
|---|---|
| $q_0$ | $\perp_{\mathfrak{M}ex}[(1,00) \mapsto 1, (2,00) \mapsto 1, (1,01) \mapsto 1, (2,01) \mapsto 1]$ |
| $q_1$ | $\perp_{\mathfrak{M}ex}[(3,00) \mapsto 1, (3,01) \mapsto 1]$ |
| $q_2$ | $\perp_{\mathfrak{M}ex}[(1,00) \mapsto 1, (2,00) \mapsto 1, (3,01) \mapsto 1]$ |
| $q_3$ | $\perp_{\mathfrak{M}ex}[(1,01) \mapsto 1, (2,01) \mapsto 1, (3,00) \mapsto 1]$ |

*What we are interested in with this automaton is states $q_2$ and $q_3$, which are generated by interaction $< 1,00 >:< 2,01 >$ from $q_0$, and interaction $< 1,01 >:< 2,00 >$ from $q_0$. Take $q_2$ for instance, the original analysis will ensure that (1,00) and (2,00) will not interact with each other, this is because that both of them are actions from the same sequential process which layered 00 and they should cooperate with actions from other layers but not themselves. The only enabled transition at $q_2$ is (3,01). However, this is not the case in automaton built by method2.*

*If we use the method2 to construct the automaton and keep using granularity function $H_{Labex,\top}$, the automaton will not terminate at all. Figure 6.6 shows the automaton we get after the round 4 of execution of the worklist algorithm in Table 5.1. Notice that red edges marking with transition $< 1,0 >:< 2,0 >$ in Figure 6.6 connect all blue nodes ($q_0, q_1, q_3, q_5$) together. If we compare transitions in Figure 6.5, there is no similar structure, where the transition after $< 1,00 >:< 2,01 >$ ( or $< 1,01 >:< 2,00 >$) will always come with an transition $< 3,01 >$ (or $< 3,00 >$). Lets take a look at the exposed actions of each state after executing round 4 by constructing the automaton with method2:*

| $q$ | E[$q$] |
|---|---|
| $q_0$ | $\perp_{\mathfrak{M}_{ex}}[(1,0) \mapsto 2, (2,0) \mapsto 1]$ |
| $q_1$ | $\perp_{\mathfrak{M}_{ex}}[(1,0) \mapsto 1, (2,0) \mapsto 1, (3,0) \mapsto 1]$ |
| $q_2$ | $\perp_{\mathfrak{M}_{ex}}[(3,0) \mapsto 2]$ |
| $q_3$ | $\perp_{\mathfrak{M}_{ex}}[(1,0) \mapsto 1, (2,0) \mapsto 1, (3,0) \mapsto 2]$ |
| $q_4$ | $\perp_{\mathfrak{M}_{ex}}[(1,0) \mapsto 2, (2,0) \mapsto 2, (3,0) \mapsto 1]$ |
| $q_5$ | $\perp_{\mathfrak{M}_{ex}}[(1,0) \mapsto 1, (2,0) \mapsto 1, (3,0) \mapsto 3]$ |

*If we look closely at $q_1$, we will find exposed actions including (1,0) and (2,0). This state is essentially the states of $q_2$ and $q_3$ in Fugure 6.5. The action (1,0) and (2,0) should come originally from the same sequential process but now method2 can't differentiate between them and will make a* wrong *judgement that they are enabled! Thus the transition $< 1,0 >:< 2,0 >$ that from $q_1$ to $q_3$ shouldn't be there and it is the extra transition caused by method2. The transition from $q_3$ and $q_5$ is generated by the same reason. Furthermore , the occurrence of exposed actions of (1,0) and (2,0) are all to be 1 in $q_1, q_3$ and $q_5$, but the occurrence of exposed actions of (3,0) are gradually increased in these states. The automaton will follow this trend and build more extra states and transitions if we didn't terminate the constructing procedure by hand.*

Method1 will not encounter the problem demonstrated in the above example, this is because all superfluous transitions made by method2 are actions cooperating within the same sequential components(due to the grouping). In method1, only sequential components cooperating on *empty* set will be grouped together and then there is no cooperation within the same sequential components.

In this sense, we say method1 doesn't loose any precision of the original analysis, it just speed up the analysis by hiding information we don't want to differentiate. However, to some extent, method2 will lose some precision from the original analysis. Even though it accelerate the analysis, it might built more interactions and make *over-approximation* of the original analysis.

Anyway, there still exists situation where method2 will not lose any precision from the original analysis: when there are same sequential processes conjointly cooperating on *non-empty* set, and all actions in this set only have maximum one occurrence in the sequential process, then the automaton we build will not encounter problem as shown in the above example and we still get a improvement of the speed without losing any precision.

**Example 6.9** *For example, we define a sequential process $S_1$ below:*

$$\begin{aligned} S_1 &\triangleq (g, r).S_2; \\ S_2 &\triangleq (p, r).(h, r).S_1; \end{aligned}$$

*Sequential process $S_1$ doesn't have two actions with the same type(g,p,h only occur once) and then it is safe to group this kind of sequential process ($S_1$) that cooperating on set containing action $g, p, h$. However, if we define another sequential processes $P_1$ below:*

$$\begin{aligned} P_1 &\triangleq (\boldsymbol{g}, r).P_2; \\ P_2 &\triangleq (p, r).(\boldsymbol{g}, r).P_1; \end{aligned}$$

*Sequential process $P_1$ does have two actions with the same type(g occur twice) and then if we group this kind of sequential process ($P_1$) that cooperating on set containing action $g$, we might lose precision and built more edges than we want.*

The processes in Example 6.7, in Table 6.6 and Table 6.7 are all the cases that could safely use method2 for improving the speed of analysis without losing any precision from the original analysis.

Chapter 7

# Deadlock Verification

In the previous chapters we have developed several static analysis techniques for PEPA. We are able to build up an automaton reflecting the interactions among several PEPA processes. In this chapter, we are going to explore the deadlock property of PEPA program by our analysis, for answering the two questions proposed in Chapter 1.

## 7.1 Deadlock of PEPA

In the operating system area, deadlock can be defined formally as follows [32]:

> A set of processes are deadlocked if *each* process in the set is waiting for an event that only another process in the set can cause.

Remember in PEPA, some actions in a sequential process can only be accomplished when there exists another action in other sequential processes to cooperate with.

Thus we say a program written by PEPA is deadlocked if *each* sequential process in the program is waiting for the actions required to cooperate, and only another sequential process in the same program *may* has the actions required but these actions currently are not enabled (The word *may* means sometimes there is no such actions at all in the other process, in this case, there must be a deadlock). In other words, we say a deadlock occurs if *all* sequential processes execute to a state where there is *no* interaction or single action that could take the system to another state.

## 7.2 Detect the Deadlock by Our Analysis

The analysis we developed in the previous chapters could help us capturing *all* possible transitions that might occur in a program, which is represented by an automation (cf.Theorem 5.5). Theorem 5.5 also indicates that whenever an interaction happens in the semantic of the program, there will be a transition in the automaton. This is because we actually build more transitions and states in our automaton than the semantics of PEPA program ought to have, due to the approximations we made.

The deadlock of the automaton is defined as follows: whenever there exists a state in the automaton with no out-going transition to other states (including transition to itself), we say there is a deadlock in the automaton. In contrast, if for every state in the automaton there exists at least one out-coming transition to other states(including transition to itself), then there is no deadlock in the automaton.

From the above arguments, we can conclude: whenever there exists a deadlock in the automaton, there *may* have a deadlock in the PEPA program. If there is no deadlock in the automaton, then there *must not* be any deadlock in the PEPA program.

Based on the automaton that is built by worklist algorithm displayed in Table

```
(1) $D := \emptyset$;
(2) for each $q_s \in$ Q
(3)     if there doesn't exist $(q_s, \widetilde{\ell\imath}, q) \in \delta$
(4)     then $D := D \cup \{q_s\}$;
(5) return $D$;
```

Table 7.1: The algorithm for collecting all deadlock states from automaton.

5.1, we introduce another algorithm for collecting all deadlock states from the automaton. The algorithm is displayed in Table 7.1. In the algorithm, $D$ is a set that holds all deadlock states which are detected. The other variables just remain the same as before. Line(2)-Line(4) describe that we go through each state of the automaton and check whether the state has any out-going transition. If there doesn't exist any such transition for a certain state, this state will be included in the set $D$.

**Example 7.1** *Let's take the definition of the sequential process from Example 2.1. Now we focus on two model components of the program as shown below in Case1 and Case2.*

$$\begin{array}{rcl} S & \triangleq & (g^1, r_1).(p^2, r_2).S \\ Q & \triangleq & (g^3, r_3).(h^4, r_4).Q + (p^5, r_5).Q \\ Case1: & & S^{00} \underset{\{g,p,h\}}{\bowtie} Q^{01} \\ Case2: & & S^{00} \underset{\{g,p,h\}}{\bowtie} Q^{01}/h \end{array}$$

| $q$ | E[$q$] $\quad D = \{q_1\}$ |
|---|---|
| $q_0$ | $\perp_{\mathfrak{M}ex}[(1,00) \mapsto 1, (3,01) \mapsto 1,$ |
| | $(5,01) \mapsto 1]$ |
| $q_1$ | $\perp_{\mathfrak{M}ex}[(2,00) \mapsto 1, (4,01) \mapsto 1,$ |

| $q$ | E[$q$] $\quad D = \emptyset$ |
|---|---|
| $q_0$ | $\perp_{\mathfrak{M}ex}[(1,00) \mapsto 1, (3,01) \mapsto 1,$ |
| | $(5,01) \mapsto 1]$ |
| $q_1$ | $\perp_{\mathfrak{M}ex}[(2,00) \mapsto 1, (4,01) \mapsto 1,$ |
| $q_2$ | $\perp_{\mathfrak{M}ex}[(2,00) \mapsto 1, (3,01) \mapsto 1,$ |
| | $(5,01) \mapsto 1]$ |

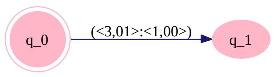

Figure 7.1: The automaton of Case 1

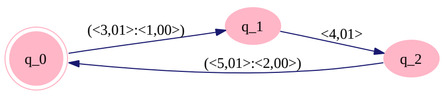

Figure 7.2: The automaton of Case 2

*We construct two automatons shown in Figure 7.1 and Figure 7.2.*

*In Figure 7.1, the state $q_1$ doesn't have any out-going transition($D = \{q_1\}$), this is because all the exposed actions in $q_1$ ($p$ in process $S$ and $h$ in process $Q$) need to cooperate with their corresponding actions in the other sequential process. Since there isn't any required actions available at that time, the program just comes to a deadlock.*

*In Figure 7.2, even though state $q_1$ have the same exposed action as in Figure 7.1, the state $q_1$ has one transition $< 4, 01 >$ to $q_2$. This is because the exposed action $h$ in process $Q$ is hidden by the hidden set $\{h\}$ and doesn't need to cooperate* *with other actions. Furthermore, all states have out-going transitions to other states($D = \emptyset$), so we can guarantee that there is no deadlock in this case.*

*For Case 1, we have found a* possible *deadlock by the deadlock collecting algorithm executing on our automaton where there is* indeed *a deadlock for the semantic execution of the program. For Case 2, we don't detect any deadlock from our automaton and we can* guarantee *that there is no deadlock in the program.*

In the example above, it demonstrate clearly that our automaton could facilitate us finding deadlock or guarantee the absence of deadlocks in a given program.

In the mean time, it also shows us that our analysis is able to answer the question 1 introduced in Chapter 1: our analysis could verify whether the system potentially have chance to go into the deadlock states.

## 7.3 Detect the Deadlock of Jobshop Examples

In this part, we are going to utilize our analysis to detect the deadlock in several variants of Milner's process for a jobshop. Quite different from the simple programs presented above, they are programs far more complex and it is often impossible for people to check manually. However, our analysis shows its competence in this scenario.

### 7.3.1 The deadlock path-finding algorithm

Before analyzing any jobshop examples, we introduce a simple path-finding algorithm, which could be used to find several paths from the initial state $q_0$ to each deadlock state of the automaton.

The algorithm applies Depth-First searching methodology to find the deadlock states and will track each path leading to them. When traverse the graph of the automaton, we will not visit a *non- deadlock* state twice to guarantee the termination of the algorithm. Actually, the algorithm only traverses the program tree once and complete the job.

This algorithm could help us find *some* paths on the way to deadlock state. However, we can't guarantee any property of the paths we find: they are not necessarily the shortest paths to the deadlock states, nor do the paths cover

| | | | |
|---|---|---|---|
| `let` | Worker0 | $\triangleq$ | (get_hammer$^1$, r).Worker1 + (get_chisel$^2$, r).Worker2; |
| | Worker1 | $\triangleq$ | (get_chisel$^3$, r).Worker3; |
| | Worker2 | $\triangleq$ | (get_hammer$^4$, r).Worker3; |
| | Worker3 | $\triangleq$ | (work$^5$,r).Worker4; |
| | Worker4 | $\triangleq$ | (release_hammer$^6$, r).Worker5 +(release_chisel$^7$, r).Worker6; |
| | Worker5 | $\triangleq$ | (release_chisel$^8$,r).Worker0; |
| | Worker6 | $\triangleq$ | (release_hammer$^9$,r).Worker0; |
| | | | |
| | Hammer_free | $\triangleq$ | (get_hammer$^{10}$, infty).Hammer_taken; |
| | Hammer_taken | $\triangleq$ | (release_hammer$^{11}$, infty).Hammer_free; |
| | Chisel_free | $\triangleq$ | (get_chisel$^{12}$, infty).Chisel_taken; |
| | Chisel_taken | $\triangleq$ | (release_chisel$^{13}$, infty).Chisel_free; |
| | | | |
| `in` | | | $(\text{Worker0}^{000} \underset{\{\}}{\bowtie} \text{Worker0}^{001})$ $\bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}}$ $(\text{Hammer\_free}^{010} \underset{\{\}}{\bowtie} \text{Chisel\_free}^{011})$ |

Table 7.2: *jobshop_3*

all possible routes to each state (sometimes finding all possible paths to each deadlock state is impossible because there often exists circle path in the graph of the automaton). The purpose of developing this algorithm is just to help us find *some* paths leading to the deadlock states.

Here, we only presented the basic idea of the algorithm and will not make further discussion in the following subsections.

## 7.3.2 Analysis of the Jobshop3

Firstly, let's take jobshop3 displayed in Table 7.2 as an example, where we add label to all sequential components and add layer to all model components.

#### 7.3.2.1 Analysis by Method 1

From the characteristic of model components($(Worker0 \underset{\{\}}{\bowtie} Worker0)$), we are able to choose method 1 and ignore the difference of them to accelerate analysis speed. Using method 1, we obtain an automaton which is shown in Figure 7.3.

The automaton has 10 states, in which $q_4$ is the deadlock state represented

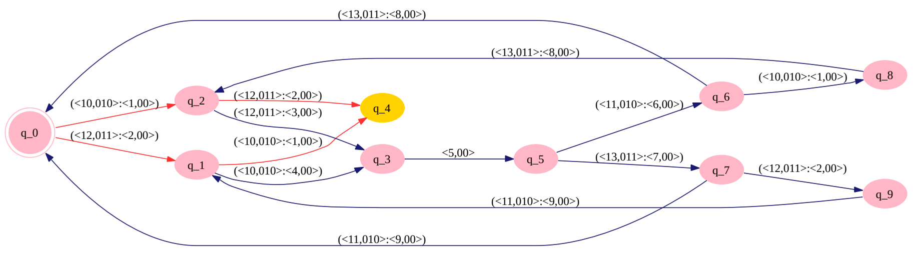

Figure 7.3: Automaton of jobshop3 by method1

by a gold node. The red edges are two paths from the initial state $q_0$ to the deadlock state $q_4$ computed by our path-finding algorithm. The path from $q_0$ via $q_2$ to $q_4$ illustrates the case that if a worker(layered 00) gets a hammer and then a worker(layered 00) gets a chisel, then there might be a deadlock state, while the path from $q_0$ via $q_1$ to $q_4$ illustrates that picking up the tools in an opposite sequence might still lead to a deadlock state. However, just from these oversimplified information it is not straightforward to guess why a deadlock exists. The worker0 defined in 7.3 only tells us a typical worker will always pickup two tools and then release them, in any order. The information obtained from our deadlock paths seems to illustrate a case that the workers behavior regularly!

#### 7.3.2.2 Analysis by Original Method

Since we already know there might be a deadlock state from the result of Method 1, now we will use the original analysis without ignoring any information and try to distinguish the behavior of two workers. We want to find out what *exactly* would happen when we go into a deadlock state!

We built the automaton shown in 7.4, which has 19 states. Two gold states $q_6$ and $q_8$ are deadlock states. As before, deadlock paths found by our path-finding algorithm are represented by red edges in the figure. It can be easily seen from the graph that the deadlock path $q_0$ via $q_2$ to $q_8$ is not figured out by our path-finding algorithm, but some much longer paths are found, for example: $q_0, q_1, q_5, q_9, q_12, q_16, q_2, q_8$. This is due to the fact our algorithm bases on Depth-First searching methodology and will not revisit any non deadlock state.

Now we inspect some deadlock paths in the automaton. Take $q_0$ via $q_1$ to $q_6$ for example, the transitions during this paths explain clearly that worker layered 000 first gets a chisel, and then worker layered 001 gets a hammer. And we start

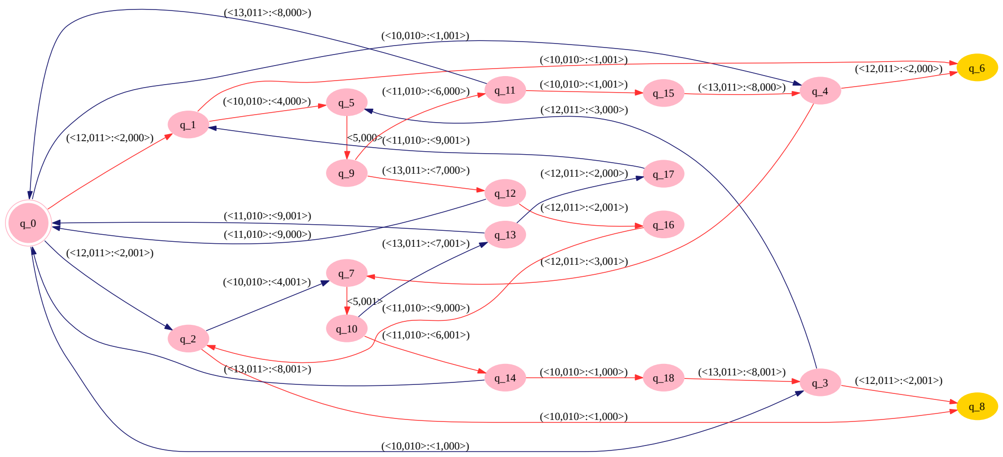

Figure 7.4: Automaton of jobshop3 by original analysis

the system with one hammer and one chisel. Now we could know what *exactly* happen when we go into a deadlock state: each worker gets a tool and waits for the other one to release the tool!

Up to now, we have shown that the two analysis methods indeed could help us study the deadlock property of the program. Method1 is much faster but can't give us all information we want, while the original analysis needs more computing power but is very precise and provides us with more information.

This example also shows us our analysis not only could answer question 1, it also be able to answer the question 2 introduced in Chapter 1: our analysis could tell that if there exist deadlock states, how the system behaves before reaching those states.

## 7.3.3 Analysis of the Jobshop1 and Jobshop2

In Section 6.3, we have already introduced jobshop1(cf. Table 6.6) and jobshop2(cf. Table 6.7) and built automatons in three approaches for each of them. Now we study the deadlock property of these two programs from three approaches.

Table 7.3 lists the results produced by the deadlock states collecting algorithm

| Program | Item | Original analysis | Method1 | Method2 |
|---|---|---|---|---|
| jobshop1 | Overall States | 1371 | 235 | 131 |
| | Time Cost | 4 hours 32 minutes 37 seconds | 1 minute 1 second | 6 seconds |
| | Deadlock States | {q_1359, q_1361, q_1370} | {q_224, q_225, q_234} | {q_126, q_130} |
| jobshop2 | Overall States | 557 | 117 | 66 |
| | Time Cost | 31 minutes 59 seconds | 17 seconds | 2 seconds |
| | Deadlock States | $\emptyset$ | $\emptyset$ | $\emptyset$ |

Table 7.3: Deadlock results on jobshop1 and jobshop2

shown in Table 7.1, and it also includes the result from Table 6.8.

For jobshop1, all three analyses have detected out deadlock states in the automaton. While for jobshop2, nothing has been reported from each analysis. We conclude that to check the existence of deadlock in an automaton, method 1 or method 2 are good choices since they are much faster and still correct. However, to understand how comes the deadlocks, we should try to proceed the original analysis, even though sometimes it costs too much time and becomes impossible.

### 7.3.4 Analysis of Jobshop4-Jobshop11

Now we are going to analyze jobshop4 to jobshop11 shown in Appendix A. Since most of them are very complex and they all meet the condition of method 1, we just use method 1 to analyze and verify these programs. The results are listed in Table 7.4.

#### 7.3.4.1 Jobshop4 and Jobshop5

Our analysis shows that jobshop4 and jobshop5 both *may* have deadlock states, which is easily understood: the worker0 will pick up hammer and chisel to work, but its behavior determines that sometimes he will throw away tools. When it comes to a state that all tools have been thrown away by worker0, the deadlock occurs since they can't pick up tools anymore.

| Program | Overall States | Deadlock States | Time Cost |
|---|---|---|---|
| jobshop4 | 87 | {q_49, q_77} | 2 seconds |
| jobshop5 | 868 | {q_596, q_771, q_841} | 37 minutes 29 seconds |
| jobshop6 | 28 | ∅ | less then 1 second |
| jobshop7 | 10 | ∅ | less then 1 second |
| jobshop8 | 122 | ∅ | 10 seconds |
| jobshop9 | 164 | ∅ | 21 seconds |
| jobshop10 | 1199 | {q_256} | 3 hours and 34 seconds |
| jobshop11 | 1672 | ∅ | 10 hours 51 minutes 38 seconds |

Table 7.4: Deadlock results on jobshop4 to jobshop11 by method1

#### 7.3.4.2 Jobshop6 to Jobshop9

Jobshop6 to Jobshop9 doesn't show any deadlock states by our analysis. They contain two kinds of workers:

- The worker0 who won't throw away tools. If there are enough tools, they will work continually without stopping.(No matter how many tools are available at the start of the program, if a program contains a worker0 that behaves as the one (could throw away tools) in jobshop4 and jobhshop5, the program will definitely go to the deadlock state)
- The politework0 who doesn't necessarily request for two tools at one time. If there exists one tool(no matter it is a hammer or a chisel)in the system, the politework0 will always be satisfied.

From the description above, these two workers will not throw away tools. Thus if we start the system with enough tools, all workers will be satisfied. Since we don't get any deadlock from the report of our analysis, there *must* be no deadlock. Thus we believe that the tools are enough for the workers in these programs.

#### 7.3.4.3 Jobshop10 and Jobshop11

Jobshop10 and jobshop11 are both larger examples. One contains 1199 states and the other contains 1672 states(by method 1)! There are two kinds of workers: LeftHandedWorker0 and RightHandedWorker0, characterized by picking up hammer first and picking up chisel respectively.

One deadlock state has been detected in jobshop10, showing that there *may* be a deadlock. After carefully considering some extreme cases, we believe there should be one deadlock: image that if all four LeftHandedWorker0 pick up all four hammers, while all four RightHandedWorker0 pick up all four chisels, all workers will be stuck, which comes to the deadlock state.

There is no deadlock state in jobshop11, reported from our analysis, and there *must* not have deadlock. The only difference from jobshop11 and jobhshop10 is: in jobhsop11 the system starts with five hammers instead of four. Since the RightHandedWorker0 will still get hammer even if all LeftHandedWorker0 have already picked their hammers. It seems that at least one RightHandedWorker0 will always be satisfied, which should be the reason the program doesn't have deadlock state.

Through jobshop4 to jobshop11, we just verify the deadlock property for each of them by the deadlock state collecting algorithm. Based on the results we got, some speculation on the result are presented, and we try to explain why there exists a deadlock or why not. Instead, we could also use path-finding algorithm to show the path leading to each deadlock state, and based on those information and we can make a much more accurate explanation of the result. We didn't show the result generated by path-finding algorithm since in some of these examples, the number of transitions along these pathes are quite huge and it doesn't worth showing them all. Please refer to the CD attached with this thesis to see all results generated by our analysis from the jobshop1-11.

# Chapter 8

# Conclusion

The initial goal of the project is to develop static analysis and answer the two questions proposed in Chapter 1:

- Does the system potentially have chance to go into the deadlock states?
- If there exist deadlock states, how does the system behave before reaching those states?

## 8.1 Achievement

To solve these questions, we developed our analysis and constructed an automaton to capture the control structure of the PEPA program, which is presented from Chapter 2 to Chapter 5. The automaton could faithfully reflect the interactions by transitions, while track the configuration of their exposed actions by states. Once we get this automaton, the questions above could be easily solved, as shown in Chapter 7. The initial version of the analysis developed are very precise that could be used to proceed exhaustive study of the system behavior. However, sometimes it is not necessary to be that accurate. Thus we developed two methods in Chapter 6 to lower the precision of the analysis, and these modification would generate much more succinct automatons in a shorter time.

Even though our work adopts the approach originally developed for analyzing CCS [16, 15], it owns its unique feature. In [16, 15], the authors annotate *label* to the CCS program. Not only did we annotate label, but also annotate *layer* to PEPA program. Consequently, we could do a much more precise approximation than theirs could do. Simply speaking, their analysis is quite similar to the method 1 and method 2 we proposed and the common characteristic of them is that they don't differentiate sequential processes with the same name. In contrast, our original analysis is able to distinguish those processes. Subsequently, our original analysis is much more accurate and informative, and could capture the *controlstructure* of the program much better.

## 8.2 Limitation

Our analysis could only handle PEPA program in a certain scale, depending on the approach we chose. The original analysis could get the most precise result comparing two other methods, however it could only handle PEPA program in the least scale. Method 2 and Method 1 could handle the program with a much larger scale, as long as the program fulfills certain requirements.

However, from Table 7.4, we could see that even by Method 1, constructing the automaton of jobshop11 costs 10 hours 51 minutes 38 seconds, which is not a short time indeed. It can be imagined that our program can't efficiently handle an arbitrary large program. In other words, our analysis suffers the so-called *state explosion problem*[33].

## 8.3 Future Work

Due to the restricted time schedule of the project, there are some tasks worth doing in the future.

### 8.3.1 Prove the Conjectures

The formal proof for one of the assumption in Theorem 5.5 is not completely done yet, we only give several steps of the proof in this thesis. We believe this assumption could be proved after finding more invariants in the `enabled` function and all its auxiliary functions.

Moreover, under certain condition(the PEPA program is written in the style of program (b) in Subsection 5.5), we hypothesize that our original analysis performs the approximation of system behavior with one hundred percent precision. In other words, we believe our analysis could faithfully capture the system behavior exactly, and there will be no more edges in the automaton we built than the transitions in the semantic world. We believe this conjecture could be formally proven to be true.

### 8.3.2 Optimize the Path-finding Algorithm

In Chapter 7, we have discussed that we developed a path-finding algorithm to find paths leading to deadlock states. We also admit that this algorithm can't guarantee the found paths hold any property. Thus, we could change this algorithm to show much more interesting paths for people to study the system property, e.g. algorithm of finding the shortest path from the initial state to a deadlock state (Dijkstra's algorithm [14]). Basically, different algorithms could generate pathes with different properties. The various algorithms could be freely picked up from the graph theory.

### 8.3.3 Integrate activity rate into the Analysis

In PEPA, an activity $a$ is defined as a pair $(\alpha, r)$ where $\alpha$ is the action type and $r$ is the activity rate that indicates the duration of this activity. In our analysis, we only deal with activity type $\alpha$, which will directly be influenced by cooperation or hiding combinators, telling whether the activity should proceed or not. For the activity rate $r$, we just get rid of it. The automaton we obtained in this way could answer the two questions proposed in Chapter 1, however, it can't answer other questions like: in what probability does the system enter into a certain state? In other words, our analysis could capture all transitions in the system, however, it doesn't keep track of any properties(the rate $r$) of the transitions.

Thus, in order to get a much more comprehensive understanding of the system behavior, we shall integrate the activity rate into the analysis in an appropriate way. For example, we shall redefine a set of functions like : $\mathcal{E}^p_\star$,$\mathcal{G}^p_\star$ etc, and put $r$ into them according to the semantics of PEPA shown in Table 2.1. And this direction definitely worth further investigation.

Appendix A

# The Syntax of Jobshop 4 - 11

| | | |
|---|---|---|
| Worker0 | $\overset{def}{=}$ | (get_hammer, r).Worker1 + (get_chisel, r).Worker2 |
| Worker1 | $\overset{def}{=}$ | (get_chisel, r).Worker3 |
| Worker2 | $\overset{def}{=}$ | (get_hammer, r).Worker3 |
| Worker3 | $\overset{def}{=}$ | (work, r).Worker4 + (go_on_strike, r).Worker7 |
| Worker4 | $\overset{def}{=}$ | (release_hammer, r).Worker5 + (release_chisel, r).Worker6 |
| Worker5 | $\overset{def}{=}$ | (release_chisel, r).Worker0 |
| Worker6 | $\overset{def}{=}$ | (release_hammer, r).Worker0 |
| Worker7 | $\overset{def}{=}$ | (throw_away_tools, r).Worker8 |
| Worker8 | $\overset{def}{=}$ | (resolve_strike, r).Worker0 |
| Hammer_free | $\overset{def}{=}$ | (get_hammer, infty).Hammer_taken |
| Hammer_taken | $\overset{def}{=}$ | (release_hammer, infty).Hammer_free |
| Chisel_free | $\overset{def}{=}$ | (get_chisel, infty).Chisel_taken |
| Chisel_taken | $\overset{def}{=}$ | (release_chisel, infty).Chisel_free |

$$\begin{aligned}&(\text{Worker0} \underset{\{\}}{\bowtie} \text{Worker0} \underset{\{\}}{\bowtie} \text{Worker0})\\ &\quad \bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}}\\ &((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free})\\ &\ \underset{\{\}}{\bowtie} (\text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free}))\end{aligned}$$

Table A.1: *jobshop_4*

$$
\begin{aligned}
&\text{Worker0} && \stackrel{\mathit{def}}{=} && (\text{get\_hammer, r}).\text{Worker1} + (\text{get\_chisel, r}).\text{Worker2}\\
&\text{Worker1} && \stackrel{\mathit{def}}{=} && (\text{get\_chisel, r}).\text{Worker3}\\
&\text{Worker2} && \stackrel{\mathit{def}}{=} && (\text{get\_hammer, r}).\text{Worker3}\\
&\text{Worker3} && \stackrel{\mathit{def}}{=} && (\text{work, r}).\text{Worker4} + (\text{go\_on\_strike, r}).\text{Worker7}\\
&\text{Worker4} && \stackrel{\mathit{def}}{=} && (\text{release\_hammer, r}).\text{Worker5} + (\text{release\_chisel, r}).\text{Worker6}\\
&\text{Worker5} && \stackrel{\mathit{def}}{=} && (\text{release\_chisel, r}).\text{Worker0}\\
&\text{Worker6} && \stackrel{\mathit{def}}{=} && (\text{release\_hammer, r}).\text{Worker0}\\
&\text{Worker7} && \stackrel{\mathit{def}}{=} && (\text{throw\_away\_tools, r}).\text{Worker8}\\
&\text{Worker8} && \stackrel{\mathit{def}}{=} && (\text{resolve\_strike, r}).\text{Worker0}\\[1ex]
&\text{Hammer\_free} && \stackrel{\mathit{def}}{=} && (\text{get\_hammer, infty}).\text{Hammer\_taken}\\
&\text{Hammer\_taken} && \stackrel{\mathit{def}}{=} && (\text{release\_hammer, infty}).\text{Hammer\_free}\\
&\text{Chisel\_free} && \stackrel{\mathit{def}}{=} && (\text{get\_chisel, infty}).\text{Chisel\_taken}\\
&\text{Chisel\_taken} && \stackrel{\mathit{def}}{=} && (\text{release\_chisel, infty}).\text{Chisel\_free}
\end{aligned}
$$

$$
\begin{aligned}
&(\text{Worker0} \underset{\{\}}{\bowtie} \text{Worker0} \underset{\{\}}{\bowtie} \text{Worker0} \underset{\{\}}{\bowtie} \text{Worker0})\\
&\qquad \bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}}\\
&((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free})\\
&\ \underset{\{\}}{\bowtie} (\text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free}))
\end{aligned}
$$

Table A.2: *jobshop_5*

$$
\begin{aligned}
&\text{Worker0} && \stackrel{def}{=} && (\text{get\_hammer, r}).\text{Worker1} + (\text{get\_chisel, r}).\text{Worker2}\\
&\text{Worker1} && \stackrel{def}{=} && (\text{get\_chisel, r}).\text{Worker3}\\
&\text{Worker2} && \stackrel{def}{=} && (\text{get\_hammer, r}).\text{Worker3}\\
&\text{Worker3} && \stackrel{def}{=} && (\text{work,r}).\text{Worker4}\\
&\text{Worker4} && \stackrel{def}{=} && (\text{release\_hammer, r}).\text{Worker5} + (\text{release\_chisel, r}).\text{Worker6}\\
&\text{Worker5} && \stackrel{def}{=} && (\text{release\_chisel,r}).\text{Worker0}\\
&\text{Worker6} && \stackrel{def}{=} && (\text{release\_hammer,r}).\text{Worker0}\\[1ex]
&\text{Hammer\_free} && \stackrel{def}{=} && (\text{get\_hammer, infty}).\text{Hammer\_taken}\\
&\text{Hammer\_taken} && \stackrel{def}{=} && (\text{release\_hammer, infty}).\text{Hammer\_free}\\
&\text{Chisel\_free} && \stackrel{def}{=} && (\text{get\_chisel, infty}).\text{Chisel\_taken}\\
&\text{Chisel\_taken} && \stackrel{def}{=} && (\text{release\_chisel, infty}).\text{Chisel\_free}
\end{aligned}
$$

$$
\begin{aligned}
&(\text{Worker0} \underset{\{\}}{\bowtie} \text{Worker0})\\
&\qquad \bowtie_{\{get\_hammer,release\_hammer,get\_chisel,release\_chisel\}}\\
&((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free})\\
&\quad \underset{\{\}}{\bowtie} (\text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free}))
\end{aligned}
$$

Table A.3: *jobshop_6*

$$
\begin{aligned}
&\text{PoliteWorker0} && \stackrel{def}{=} && (\text{get\_hammer, r}).\text{PoliteWorker1} + (\text{get\_chisel, r}).\text{PoliteWorker2}\\
&\text{PoliteWorker1} && \stackrel{def}{=} && (\text{get\_chisel, r}).\text{PoliteWorker3} + (\text{release\_hammer, r}).\text{PoliteWorker0}\\
&\text{PoliteWorker2} && \stackrel{def}{=} && (\text{get\_hammer, r}).\text{PoliteWorker3} + (\text{release\_chisel, r}).\text{PoliteWorker0}\\
&\text{PoliteWorker3} && \stackrel{def}{=} && (\text{work, r}).\text{PoliteWorker4}\\
&\text{PoliteWorker4} && \stackrel{def}{=} && (\text{release\_hammer, r}).\text{PoliteWorker5} + (\text{release\_chisel, r}).\text{PoliteWorker6}\\
&\text{PoliteWorker5} && \stackrel{def}{=} && (\text{release\_chisel, r}).\text{PoliteWorker0}\\
&\text{PoliteWorker6} && \stackrel{def}{=} && (\text{release\_hammer, r}).\text{PoliteWorker0}\\[1ex]
&\text{Hammer\_free} && \stackrel{def}{=} && (\text{get\_hammer, infty}).\text{Hammer\_taken}\\
&\text{Hammer\_taken} && \stackrel{def}{=} && (\text{release\_hammer, infty}).\text{Hammer\_free}\\
&\text{Chisel\_free} && \stackrel{def}{=} && (\text{get\_chisel, infty}).\text{Chisel\_taken}\\
&\text{Chisel\_taken} && \stackrel{def}{=} && (\text{release\_chisel, infty}).\text{Chisel\_free}
\end{aligned}
$$

$$
\begin{aligned}
&(\text{PoliteWorker0} \underset{\{\}}{\bowtie} \text{PoliteWorker0})\\
&\qquad \bowtie_{\{get\_hammer,release\_hammer,get\_chisel,release\_chisel\}}\\
&((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free})
\end{aligned}
$$

Table A.4: *jobshop_7*

| | | |
|---|---|---|
| Worker0 | $\stackrel{def}{=}$ | (get_hammer, r).Worker1 + (get_chisel, r).Worker2 |
| Worker1 | $\stackrel{def}{=}$ | (get_chisel, r).Worker3 |
| Worker2 | $\stackrel{def}{=}$ | (get_hammer, r).Worker3 |
| Worker3 | $\stackrel{def}{=}$ | (work,r).Worker4 |
| Worker4 | $\stackrel{def}{=}$ | (release_hammer, r).Worker5 +(release_chisel, r).Worker6 |
| Worker5 | $\stackrel{def}{=}$ | (release_chisel,r).Worker0 |
| Worker6 | $\stackrel{def}{=}$ | (release_hammer,r).Worker0 |
| PoliteWorker0 | $\stackrel{def}{=}$ | (get_hammer, r).PoliteWorker1 +(get_chisel, r).PoliteWorker2 |
| PoliteWorker1 | $\stackrel{def}{=}$ | (get_chisel, r).PoliteWorker3 +(release_hammer, r).PoliteWorker0 |
| PoliteWorker2 | $\stackrel{def}{=}$ | (get_hammer, r).PoliteWorker3 + (release_chisel, r).PoliteWorker0 |
| PoliteWorker3 | $\stackrel{def}{=}$ | (work, r).PoliteWorker4 |
| PoliteWorker4 | $\stackrel{def}{=}$ | (release_hammer, r).PoliteWorker5 + (release_chisel, r).PoliteWorker6 |
| PoliteWorker5 | $\stackrel{def}{=}$ | (release_chisel, r).PoliteWorker0 |
| PoliteWorker6 | $\stackrel{def}{=}$ | (release_hammer, r).PoliteWorker0 |
| Hammer_free | $\stackrel{def}{=}$ | (get_hammer, infty).Hammer_taken |
| Hammer_taken | $\stackrel{def}{=}$ | (release_hammer, infty).Hammer_free |
| Chisel_free | $\stackrel{def}{=}$ | (get_chisel, infty).Chisel_taken |
| Chisel_taken | $\stackrel{def}{=}$ | (release_chisel, infty).Chisel_free |

$$(\text{Worker0} \underset{\{\}}{\bowtie} \text{Worker0} \underset{\{\}}{\bowtie} \text{PoliteWorker0}) \bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}} ((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free}) \underset{\{\}}{\bowtie} (\text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free}))$$

Table A.5: *jobshop_8*

$$
\begin{array}{lll}
\mathrm{Worker0} & \overset{def}{=} & (\mathrm{get\_hammer,\ r}).\mathrm{Worker1} + (\mathrm{get\_chisel,\ r}).\mathrm{Worker2} \\
\mathrm{Worker1} & \overset{def}{=} & (\mathrm{get\_chisel,\ r}).\mathrm{Worker3} \\
\mathrm{Worker2} & \overset{def}{=} & (\mathrm{get\_hammer,\ r}).\mathrm{Worker3} \\
\mathrm{Worker3} & \overset{def}{=} & (\mathrm{work,r}).\mathrm{Worker4} \\
\mathrm{Worker4} & \overset{def}{=} & (\mathrm{release\_hammer,\ r}).\mathrm{Worker5} + (\mathrm{release\_chisel,\ r}).\mathrm{Worker6} \\
\mathrm{Worker5} & \overset{def}{=} & (\mathrm{release\_chisel,r}).\mathrm{Worker0} \\
\mathrm{Worker6} & \overset{def}{=} & (\mathrm{release\_hammer,r}).\mathrm{Worker0} \\
\\
\mathrm{PoliteWorker0} & \overset{def}{=} & (\mathrm{get\_hammer,\ r}).\mathrm{PoliteWorker1} + (\mathrm{get\_chisel,\ r}).\mathrm{PoliteWorker2} \\
\mathrm{PoliteWorker1} & \overset{def}{=} & (\mathrm{get\_chisel,\ r}).\mathrm{PoliteWorker3} + (\mathrm{release\_hammer,\ r}).\mathrm{PoliteWorker0} \\
\mathrm{PoliteWorker2} & \overset{def}{=} & (\mathrm{get\_hammer,\ r}).\mathrm{PoliteWorker3} + (\mathrm{release\_chisel,\ r}).\mathrm{PoliteWorker0} \\
\mathrm{PoliteWorker3} & \overset{def}{=} & (\mathrm{work,\ r}).\mathrm{PoliteWorker4} \\
\mathrm{PoliteWorker4} & \overset{def}{=} & (\mathrm{release\_hammer,\ r}).\mathrm{PoliteWorker5} + (\mathrm{release\_chisel,\ r}).\mathrm{PoliteWorker6} \\
\mathrm{PoliteWorker5} & \overset{def}{=} & (\mathrm{release\_chisel,\ r}).\mathrm{PoliteWorker0} \\
\mathrm{PoliteWorker6} & \overset{def}{=} & (\mathrm{release\_hammer,\ r}).\mathrm{PoliteWorker0} \\
\\
\mathrm{Hammer\_free} & \overset{def}{=} & (\mathrm{get\_hammer,\ infty}).\mathrm{Hammer\_taken} \\
\mathrm{Hammer\_taken} & \overset{def}{=} & (\mathrm{release\_hammer,\ infty}).\mathrm{Hammer\_free} \\
\mathrm{Chisel\_free} & \overset{def}{=} & (\mathrm{get\_chisel,\ infty}).\mathrm{Chisel\_taken} \\
\mathrm{Chisel\_taken} & \overset{def}{=} & (\mathrm{release\_chisel,\ infty}).\mathrm{Chisel\_free} \\
\\
 & & (\mathrm{Worker0} \underset{\{\}}{\bowtie} \mathrm{Worker0} \underset{\{\}}{\bowtie} \mathrm{Worker0} \underset{\{\}}{\bowtie} \mathrm{PoliteWorker0}) \\
 & & \quad \bowtie_{\{get\_hammer,release\_hammer,get\_chisel,release\_chisel\}} \\
 & & ((\mathrm{Hammer\_free} \underset{\{\}}{\bowtie} \mathrm{Hammer\_free}) \\
 & & \underset{\{\}}{\bowtie} (\mathrm{Chisel\_free} \underset{\{\}}{\bowtie} \mathrm{Chisel\_free}))
\end{array}
$$

Table A.6: *jobshop_9*

$$
\begin{array}{lcl}
\text{LeftHandedWorker0} & \stackrel{def}{=} & (\text{get\_hammer, r}).\text{LeftHandedWorker1} \\
\text{LeftHandedWorker1} & \stackrel{def}{=} & (\text{get\_chisel, r}).\text{LeftHandedWorker2} \\
\text{LeftHandedWorker2} & \stackrel{def}{=} & (\text{work, r}).\text{LeftHandedWorker3} \\
\text{LeftHandedWorker3} & \stackrel{def}{=} & (\text{release\_chisel, r}).\text{LeftHandedWorker4} \\
\text{LeftHandedWorker4} & \stackrel{def}{=} & (\text{release\_hammer, r}).\text{LeftHandedWorker0} \\
\\
\text{RightHandedWorker0} & \stackrel{def}{=} & (\text{get\_chisel, r}).\text{RightHandedWorker1} \\
\text{RightHandedWorker1} & \stackrel{def}{=} & (\text{get\_hammer, r}).\text{RightHandedWorker2} \\
\text{RightHandedWorker2} & \stackrel{def}{=} & (\text{work, r}).\text{RightHandedWorker3} \\
\text{RightHandedWorker3} & \stackrel{def}{=} & (\text{release\_hammer, r}).\text{RightHandedWorker4} \\
\text{RightHandedWorker4} & \stackrel{def}{=} & (\text{release\_chisel, r}).\text{RightHandedWorker0} \\
\\
\text{Hammer\_free} & \stackrel{def}{=} & (\text{get\_hammer, infty}).\text{Hammer\_taken} \\
\text{Hammer\_taken} & \stackrel{def}{=} & (\text{release\_hammer, infty}).\text{Hammer\_free} \\
\text{Chisel\_free} & \stackrel{def}{=} & (\text{get\_chisel, infty}).\text{Chisel\_taken} \\
\text{Chisel\_taken} & \stackrel{def}{=} & (\text{release\_chisel, infty}).\text{Chisel\_free}
\end{array}
$$

$$
\begin{array}{l}
((\text{LeftHandedWorker0} \underset{\{\}}{\bowtie} \text{LeftHandedWorker0} \underset{\{\}}{\bowtie} \text{LeftHandedWorker0} \\
\quad \underset{\{\}}{\bowtie} \text{LeftHandedWorker0}) \underset{\{\}}{\bowtie} \\
\quad (\text{RightHandedWorker0} \underset{\{\}}{\bowtie} \text{RightHandedWorker0} \underset{\{\}}{\bowtie} \text{RightHandedWorker0} \\
\quad \underset{\{\}}{\bowtie} \text{RightHandedWorker0})) \\
\quad \bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}} \\
((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free}) \\
\quad \underset{\{\}}{\bowtie} (\text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free}))
\end{array}
$$

Table A.7: *jobshop_10*

| | | |
|---|---|---|
| LeftHandedWorker0 | $\overset{def}{=}$ | (get_hammer, r).LeftHandedWorker1 |
| LeftHandedWorker1 | $\overset{def}{=}$ | (get_chisel, r).LeftHandedWorker2 |
| LeftHandedWorker2 | $\overset{def}{=}$ | (work, r).LeftHandedWorker3 |
| LeftHandedWorker3 | $\overset{def}{=}$ | (release_chisel, r).LeftHandedWorker4 |
| LeftHandedWorker4 | $\overset{def}{=}$ | (release_hammer, r).LeftHandedWorker0 |
| RightHandedWorker0 | $\overset{def}{=}$ | (get_chisel, r).RightHandedWorker1 |
| RightHandedWorker1 | $\overset{def}{=}$ | (get_hammer, r).RightHandedWorker2 |
| RightHandedWorker2 | $\overset{def}{=}$ | (work, r).RightHandedWorker3 |
| RightHandedWorker3 | $\overset{def}{=}$ | (release_hammer, r).RightHandedWorker4 |
| RightHandedWorker4 | $\overset{def}{=}$ | (release_chisel, r).RightHandedWorker0 |
| Hammer_free | $\overset{def}{=}$ | (get_hammer, infty).Hammer_taken |
| Hammer_taken | $\overset{def}{=}$ | (release_hammer, infty).Hammer_free |
| Chisel_free | $\overset{def}{=}$ | (get_chisel, infty).Chisel_taken |
| Chisel_taken | $\overset{def}{=}$ | (release_chisel, infty).Chisel_free |

$$\begin{aligned}&((\text{LeftHandedWorker0} \underset{\{\}}{\bowtie} \text{LeftHandedWorker0} \underset{\{\}}{\bowtie} \text{LeftHandedWorker0}\\&\quad \underset{\{\}}{\bowtie} \text{LeftHandedWorker0}) \underset{\{\}}{\bowtie}\\&\ (\text{RightHandedWorker0} \underset{\{\}}{\bowtie} \text{RightHandedWorker0} \underset{\{\}}{\bowtie} \text{RightHandedWorker0}\\&\quad \underset{\{\}}{\bowtie} \text{RightHandedWorker0}))\\&\quad \bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}}\\&((\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Hammer\_free})\\&\ \underset{\{\}}{\bowtie} (\text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free}))\end{aligned}$$

Table A.8: *jobshop*_11

Appendix B

# Design and Demonstration of the Tool

In this part, we will first give a short description of the tool we designed for static analysis of PEPA. Later we will show a concrete example of using the tool.

The overall procedure of utilizing our tool with other auxiliary software is demonstrated in Figure B.1. From the figure, it is seen that our tool takes a plain-text PEPA file and output the result into two files:

- One file contains the important datastructure generated during the analyzing procedure, text representation of the automaton in the form as introduced at start of Chapter 5, as well as the information of all deadlock states that has been found;
- The other one contains the text representation of the automaton which follows the syntax of dot language [17, 4]. The dot format file could be input into the *dot tool*(one part of the the open source graphic visualization software Graphviz [1]), from which a graphic representation of the automaton could be generated.

Our tool is developed under Java language and contains two main parts: the

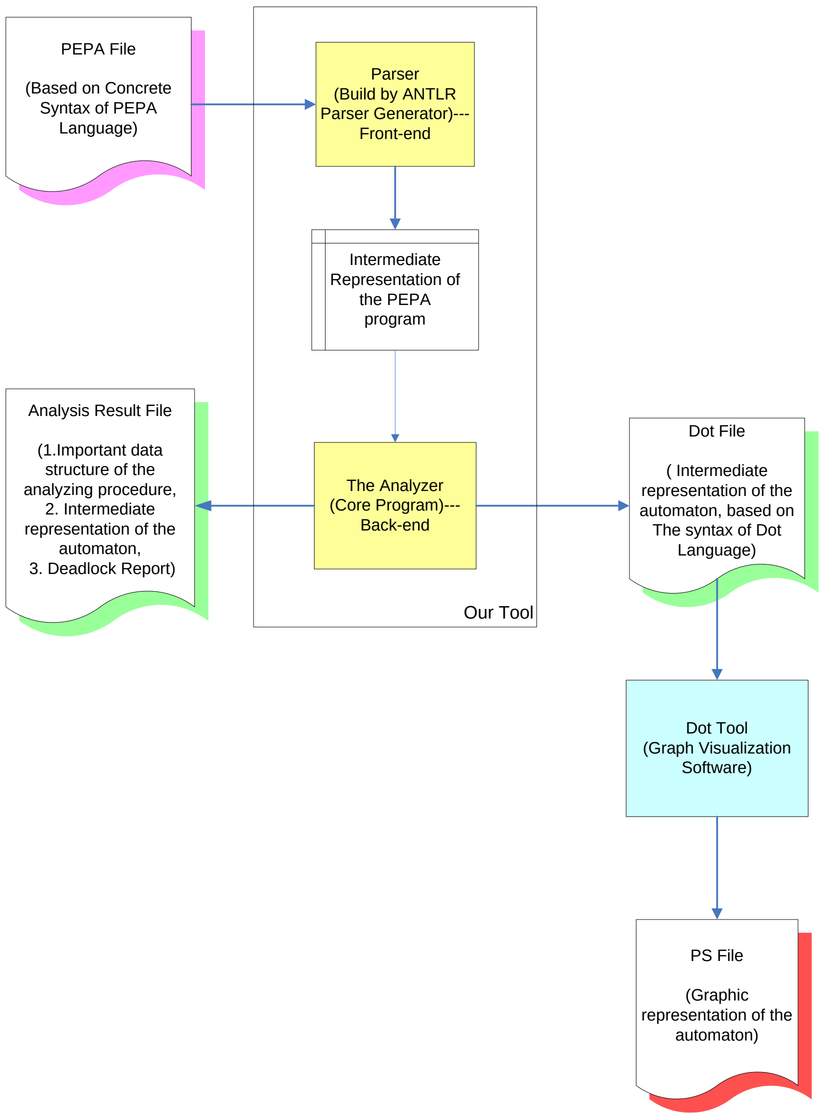
PEPA File
(Based on Concrete Syntax of PEPA Language)
Parser (Build by ANTLR Parser Generator)--- Front-end
Intermediate Representation of the PEPA program
Analysis Result File
(1.Important data structure of the analyzing procedure, 2. Intermediate representation of the automaton, 3. Deadlock Report)
The Analyzer (Core Program)--- Back-end
Dot File
( Intermediate representation of the automaton, based on The syntax of Dot Language)
Our Tool
Dot Tool (Graph Visualization Software)
PS File
(Graphic representation of the automaton)

Figure B.1: The Workflow of the Static Analysis

front-end, a parser; the back-end, the core program. We will introduce them in the following subsections.

## B.1 The Parser

The parser could transform plain-text PEPA programs into the intermediate representations that are ready for the back-end program to work on, implemented using Antlr [2], an open source parser generating tool. Here the plain-text PEPA is written in concrete syntax of PEPA. The reason to introduce this new syntax is that the mathematical notation of PEPA introduced in Chapter 2.1 is really hard to input into the plain-text file.

Refering to some other concrete syntaxes of PEPA [30, 18, 19], we developed our own syntax more or less the same as theirs. The only difference is we get rid of the rate initialization at the front of the program (such as $r_1 = 1; r_2 = 5; \cdots$), since our static analysis doesn't take care of rate. Most of the combinators remain the same as displayed in Chapter 2.1, except for the cooperation combinator: e.g. $S \underset{\{g,p\}}{\bowtie} Q$ would be transformed into $S < g, p > Q$. The program will start with the sequential component definition that contains one or more sequential components, separated by semicolon, with model component definition being the following part for further processing.

**Example B.1** *Let's see how we transform a program written in mathematic syntax into our new concrete syntax. Figure B.1 shows jobshop3 in original mathematic notation. After the transformation, it becomes the program in Figure B.2, which could be directly taken into our parser for further processing.*

## B.2 The Analyzer

The analyzer is the core part of our tool which takes the intermediate representation of the PEPA program as input, and after finishing the computation, it outputs the analysis result into two files. The analyzer implements the worklist algorithm, all its auxiliary functions, and functions related to deadlock detecting and displaying, which are well defined through Chapter 3 to Chapter 7. Here we won't dig into any details, please refer to the source code that included in the Appendix CD.

We thoroughly tested our tool, using Junit[3], an open source unit-testing framework for java. The test covers most significant classes and methods, and will significantly ensure the correctness of our design and implementation.

| | | |
|---|---|---|
| Worker0 | $\stackrel{def}{=}$ | (get_hammer, r).Worker1 + (get_chisel, r).Worker2 |
| Worker1 | $\stackrel{def}{=}$ | (get_chisel, r).Worker3 |
| Worker2 | $\stackrel{def}{=}$ | (get_hammer, r).Worker3 |
| Worker3 | $\stackrel{def}{=}$ | (work,r).Worker4 |
| Worker4 | $\stackrel{def}{=}$ | (release_hammer, r).Worker5 +(release_chisel, r).Worker6 |
| Worker5 | $\stackrel{def}{=}$ | (release_chisel,r).Worker0 |
| Worker6 | $\stackrel{def}{=}$ | (release_hammer,r).Worker0 |
| | | |
| Hammer_free | $\stackrel{def}{=}$ | (get_hammer, infty).Hammer_taken |
| Hammer_taken | $\stackrel{def}{=}$ | (release_hammer, infty).Hammer_free |
| Chisel_free | $\stackrel{def}{=}$ | (get_chisel, infty).Chisel_taken |
| Chisel_taken | $\stackrel{def}{=}$ | (release_chisel, infty).Chisel_free |

$$\begin{array}{l}(\text{Worker0} \underset{\{\}}{\bowtie} \text{Worker0}) \\ \qquad \bowtie_{\{get\_hammer, release\_hammer, get\_chisel, release\_chisel\}} \\ (\text{Hammer\_free} \underset{\{\}}{\bowtie} \text{Chisel\_free})\end{array}$$

Table B.1: *jobshop_3* in mathematic notation of PEPA

```
Worker0        =   (get_hammer, r).Worker1 + (get_chisel, r).Worker2;
Worker1        =   (get_chisel, r).Worker3;
Worker2        =   (get_hammer, r).Worker3;
Worker3        =   (work,r).Worker4;
Worker4        =   (release_hammer, r).Worker5 +(release_chisel, r).Worker6;
Worker5        =   (release_chisel,r).Worker0;
Worker6        =   (release_hammer,r).Worker0;

Hammer_free    =   (get_hammer, infty).Hammer_taken;
Hammer_taken   =   (release_hammer, infty).Hammer_free;
Chisel_free    =   (get_chisel, infty).Chisel_taken;
Chisel_taken   =   (release_chisel, infty).Chisel_free;

(Worker0<>Worker0)
     < get_hammer, release_hammer, get_chisel, release_chisel >
(Hammer_free<>Chisel_free)
```

Table B.2: *jobshop_3* in concrete syntax of PEPA

## B.3 A Guide for Using the Tool through an Example

In order to use the tool, the standard Java Runtime Environment should be set up appropriate. In addition, we also need to put antlr package (antlr.jar) to the right class path. If we want to see the graph representation of the result, the Graphviz package should also be installed. Once we set up the environment, we could use the tool to analyze PEPA program.

**An Example**

Here we demonstrate the tool through an example. Suppose the program we plan to analyze is stored in jobshop3.txt, with the content shown in Table B.2.

***The Input***

To start the program, type

```
java -classpath PEPATool.jar automaton.Automaton jobshop3.txt -v2 -pt -kn
```

In this command, PEPATool.jar contains all classes relevant to our tool, antomaton.Automaton is the main entry of the tool, jobshop3.txt is the PEPA file we are going to analyze. The parameters afterwards could control the tool to behave differently.

- -v: controls the version of the analysis. -v1 means using the original analysis; -v2 means using method 1 for accelerating the analysis; -v3 means using method 2 for accelerating the analysis.
- -p: controls graph output. -pt means to draw the graph while -pf means no.
- -k: controls the granularity function of the analysis. -kn means $k = \top$, -k0 means $k = 0$, -k1 means $k = 1$, etc.

If omit any number of these three parameters, the default setting is: -v2 -pf -kn.

In this example, we choose to use version 2(method1) with $k = \top$. The graph would be automatically drawn by the tool.

***The Output***

We will obtain the follow files: jobshop3_Result.txt (cf.Appendix B.4.1), jobshop3.dot (cf.Appendix B.4.2), and jobshop3.ps (cf.Appendix B.4.3).

Here jobshop3_Result.txt is the analysis result. jobshop3.dot is the automaton with dot format. jobshop3.ps is graphic representation of the automaton drawn from jobshop3.dot.

## B.4 Analysis Results

### B.4.1 jobshop3_Result.txt

```
-----------------------------------------------------------------------
Initially the program is....

Worker0=(<get_hammer,r>_0.Worker1+<get_chisel,r>_0.Worker2);
Worker1=<get_chisel,r>_0.Worker3; Worker2=<get_hammer,r>_0.Worker3;
Worker3=<work,r>_0.Worker4;
Worker4=(<release_hammer,r>_0.Worker5+<release_chisel,r>_0.Worker6);
Worker5=<release_chisel,r>_0.Worker0;
Worker6=<release_hammer,r>_0.Worker0;
Hammer_free=<get_hammer,infty>_0.Hammer_taken;
Hammer_taken=<release_hammer,infty>_0.Hammer_free;
Chisel_free=<get_chisel,infty>_0.Chisel_taken;
Chisel_taken=<release_chisel,infty>_0.Chisel_free;
((Worker0[]^1<>Worker0[]^1)^1<release_hammer,get_chisel,release_chisel,
      get_hammer>(Hammer_free[]^1<>Chisel_free[]^1)^1)^1
-----------------------------------------------------------------------
The Simplified Program is:

Worker0=(<get_hammer,r>_1.Worker1+<get_chisel,r>_2.Worker2);
Worker1=<get_chisel,r>_3.Worker3; Worker2=<get_hammer,r>_4.Worker3;
Worker3=<work,r>_5.Worker4;
Worker4=(<release_hammer,r>_6.Worker5+<release_chisel,r>_7.Worker6);
Worker5=<release_chisel,r>_8.Worker0;
Worker6=<release_hammer,r>_9.Worker0;
Hammer_free=<get_hammer,infty>_10.Hammer_taken;
Hammer_taken=<release_hammer,infty>_11.Hammer_free;
Chisel_free=<get_chisel,infty>_12.Chisel_taken;
Chisel_taken=<release_chisel,infty>_13.Chisel_free;
(Worker0[]^2<release_hammer,get_chisel,release_chisel,get_hammer>
    (Hammer_free[]^1<>Chisel_free[]^1)^1)^1
-----------------------------------------------------------------
Important Data of the program, for constructing Automaton:

YEx: <12,011>={get_chisel,[0],[],null,false,}
<13,011>={release_chisel,[0],[],null,false,}
<10,010>={get_hammer,[0],[],null,false,}
<11,010>={release_hammer,[0],[],null,false,}
<1,00>={get_hammer,[0],[],null,false,}
<2,00>={get_chisel,[0],[],null,false,}
<3,00>={get_chisel,[0],[],null,false,}
<4,00>={get_hammer,[0],[],null,false,}
<5,00>={work,[],[],null,false,}
<6,00>={release_hammer,[0],[],null,false,}
<7,00>={release_chisel,[0],[],null,false,}
<8,00>={release_chisel,[0],[],null,false,}
<9,00>={release_hammer,[0],[],null,false,}

EEx: {<12,011>=1, <13,011>=0, <10,010>=1, <11,010>=0, <1,00>=2,
<2,00>=2, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

GEx: <12,011>={<12,011>=0, <13,011>=1, <10,010>=0, <11,010>=0,
<1,00>=0, <2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0,
<7,00>=0, <8,00>=0, <9,00>=0}

<13,011>={<12,011>=1, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<10,010>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=1, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<11,010>={<12,011>=0, <13,011>=0, <10,010>=1, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<1,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=1, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<2,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=1, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<3,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=1, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<4,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=1, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<5,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=1, <7,00>=1,
<8,00>=0, <9,00>=0}

<6,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=1, <9,00>=0}

<7,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=1}

<8,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<9,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

KEx: <12,011>={<12,011>=1, <13,011>=0, <10,010>=0, <11,010>=0,
<1,00>=0, <2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0,
<7,00>=0, <8,00>=0, <9,00>=0}

<13,011>={<12,011>=0, <13,011>=1, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<10,010>={<12,011>=0, <13,011>=0, <10,010>=1, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<11,010>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=1, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<1,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<2,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<3,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=1, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<4,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=1, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<5,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=1, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

<6,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=1, <7,00>=1,
<8,00>=0, <9,00>=0}

<7,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=1, <7,00>=1,
<8,00>=0, <9,00>=0}

<8,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=1, <9,00>=0}

<9,00>={<12,011>=0, <13,011>=0, <10,010>=0, <11,010>=0, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=1}

----------------------------------------------------------------------
----------------------------------------------------------------------
The Automaton is:

States: [q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9]
Transition: q_0: {(<12,011>:<2,00>)=q_1, (<10,010>:<1,00>)=q_2}
```

```
q_1:{(<10,010>:<1,00>)=q_4, (<10,010>:<4,00>)=q_3}
q_2:{(<12,011>:<2,00>)=q_4, (<12,011>:<3,00>)=q_3}
q_3:{<5,00>=q_5}
q_5:{(<11,010>:<6,00>)=q_6, (<13,011>:<7,00>)=q_7}
q_6:{(<13,011>:<8,00>)=q_0, (<10,010>:<1,00>)=q_8}
q_7:{(<12,011>:<2,00>)=q_9, (<11,010>:<9,00>)=q_0}
q_8:{(<13,011>:<8,00>)=q_2}
q_9: {(<11,010>:<9,00>)=q_1}


E: q_0: {<12,011>=1, <13,011>=0, <10,010>=1, <11,010>=0, <1,00>=2,
<2,00>=2, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

q_1: {<12,011>=0, <13,011>=1, <10,010>=1, <11,010>=0, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=1, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

q_2: {<12,011>=1, <13,011>=0, <10,010>=0, <11,010>=1, <1,00>=1,
<2,00>=1, <3,00>=1, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

q_3: {<12,011>=0, <13,011>=1, <10,010>=0, <11,010>=1, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=0, <5,00>=1, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

q_4: {<12,011>=0, <13,011>=1, <10,010>=0, <11,010>=1, <1,00>=0,
<2,00>=0, <3,00>=1, <4,00>=1, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=0}

q_5: {<12,011>=0, <13,011>=1, <10,010>=0, <11,010>=1, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=1, <7,00>=1,
<8,00>=0, <9,00>=0}

q_6: {<12,011>=0, <13,011>=1, <10,010>=1, <11,010>=0, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=1, <9,00>=0}

q_7: {<12,011>=1, <13,011>=0, <10,010>=0, <11,010>=1, <1,00>=1,
<2,00>=1, <3,00>=0, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=1}

q_8: {<12,011>=0, <13,011>=1, <10,010>=0, <11,010>=1, <1,00>=0,
<2,00>=0, <3,00>=1, <4,00>=0, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=1, <9,00>=0}

q_9: {<12,011>=0, <13,011>=1, <10,010>=0, <11,010>=1, <1,00>=0,
<2,00>=0, <3,00>=0, <4,00>=1, <5,00>=0, <6,00>=0, <7,00>=0,
<8,00>=0, <9,00>=1}

----------------------------------------------------------------------
The deadlock info of the Automaton:

Deadlock states: [q_4]

DeadlockPath:

<<<---q_4--->>>
(1):[q_0,(<12,011>:<2,00>)-->q_1,(<10,010>:<1,00>)-->q_4]
(2):[q_0,(<10,010>:<1,00>)-->q_2, (<12,011>:<2,00>)-->q_4]

----------------------------------------------------------------------
Constructing the Automaton Finished in: 0 hour(s) 0 minute(s) 0
second(s) PEPA tool version 2.0   k=<><>
```

### B.4.2 jobshop3.dot

```
digraph lab1 {
rankdir=LR;
size="7,7";
edge [color="midnightblue"];
node [style=filled,color="pink1"];
q_0 [shape=doublecircle];
q_0 -> q_1 [label="(<12,011>:<2,00>)"];
q_0 -> q_2 [label="(<10,010>:<1,00>)"];
q_1 -> q_4 [label="(<10,010>:<1,00>)"];
q_1 -> q_3 [label="(<10,010>:<4,00>)"];
q_2 -> q_4 [label="(<12,011>:<2,00>)"];
q_2 -> q_3 [label="(<12,011>:<3,00>)"];
q_3 -> q_5 [label="<5,00>"];
q_5 -> q_6 [label="(<11,010>:<6,00>)"];
q_5 -> q_7 [label="(<13,011>:<7,00>)"];
q_6 -> q_0 [label="(<13,011>:<8,00>)"];
q_6 -> q_8 [label="(<10,010>:<1,00>)"];
q_7 -> q_9 [label="(<12,011>:<2,00>)"];
q_7 -> q_0 [label="(<11,010>:<9,00>)"];
q_8 -> q_2 [label="(<13,011>:<8,00>)"];
q_9 -> q_1 [label="(<11,010>:<9,00>)"];
}
```

### B.4.3 jobshop3.ps

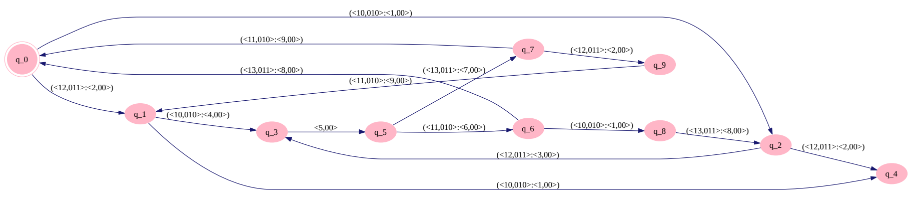

Figure B.2: Automaton of jobshop3

## B.5 Source Code

Please refer to the CD attached with this thesis.

# Bibliography

[1] `http://www.graphviz.org/About.php`.

[2] `http://www.antlr.org/`.

[3] `http://www.Junit.org/`.

[4] The dot language. `http://www.graphviz.org/doc/info/lang.html`.

[5] Drawing graphs with dot. 2002.

[6] *The Not So Short Introduction to Latex2e.* 2005.

[7] J. C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3):131–146, 2005.

[8] J. C. M. Baeten, J. A. Bergstra, and S. A. Smolka. Axiomatizing probabilistic processes: Acp with generative probabilities. *Inf. Comput.*, 121(2):234–255, 1995.

[9] Jan A. Bergstra and Jan Willem Klop. Acttau: A universal axiom system for process specification. In *Algebraic Methods: Theory, Tools and Applications [papers from a workshop in Passau, Germany, June 9-11, 1987]*, pages 447–463, London, UK, 1989. Springer-Verlag.

[10] Bodei, Degano, Nielson, and Nielson. Static analysis for the pi-calculus with application to security. *INFCTRL: Information and Computation (formerly Information and Control)*, 168, 2001.

[11] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Nielson. Automatic validation of protocol narration, 2003.

[12] Chiara Braghin, Agostino Cortesi, Riccardo Focardi, Flaminia L. Luccio, and Carla Piazza. Complexity of nesting analysis in mobile ambients.

[13] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY.

[14] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[15] Hanne Riis Nielson Flemming Nielson. Data flow analysis for ccs. 2006.

[16] Hanne Riis Nielson Flemming Nielson. A monotone framework for ccs. 2006.

[17] Emden Gansner, Eleftherios Koutsofios, and Stephen North. Drawing graphs with dot, 2006.

[18] Stephen Gilmore. The pepa workbench: User's manual. 2001.

[19] Stephen Gilmore and Jane Hillston. The PEPA workbench: A tool to support a process algebra-based approach to performance modelling, 1994.

[20] G.Lowe. Probabilities and priorities in timed csp. 1999.

[21] Flemming Nielson Hanne Riis Nielson. *Semantics with Applications – A Formal Introduction.* Wiley Professional Computing, 1992.

[22] J. Hillston. A compositional approach to performance modelling. *Cambridge University Press*, 1996.

[23] J. Hillston. Process algebras for quantitative analysis. In *LICS '05: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 239–248, Washington, DC, USA, 2005. IEEE Computer Society.

[24] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[25] John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[26] R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[27] M. Neubauer and P. Thiemann. An implementation of session types, 2004.

[28] Flemming Nielson, Hanne R. Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999. Second printing, 2005.

[29] Hanne Riis Nielson and Flemming Nielson. Shape analysis for mobile ambients. In *POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 142–154, New York, NY, USA, 2000. ACM Press.

[30] N.V.Haenel. User guide for the java edition of the pepa workbench. *Tabasco release, version 0.9.4*, 2004.

[31] R.Milner. *Communication and Mobile Systems: The pi-Calculus.* Cambridge University Press, 1st edition,1999.

[32] Andrew S. Tanenbaum. *Modern Operating Systems, 2nd Edition.* Prentice Hall., 2001.

[33] Antti Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets*, pages 429–528, London, UK, 1998. Springer-Verlag.

[34] V. Vasconcelos, A. Ravara, and S. Gay. Session types for functional multithreading, 2004.

[35] Wikipedia. Process calculus. `http://en.wikipedia.org/wiki/Process_calculi`.