Symbolic Methods

Symbolic state-space traversal for finite-state systems

Martin Fränzle

Carl von Ossietzky Universität

Dpt. of CS

Res. Grp. Hybrid Systems

Oldenburg, Germany

What you'll learn

- reduced ordered binary decision diagrams
- symbolic methods for state reachability
  - SAT-based procedures for bounded state reachability
  - full reachability via BDDs
- symbolic CTL model checking

Reduced ordered binary decision diagrams

(RO)BDDs

A decision tree

An Irishman's Philosophy

In life, there are only two things to worry about:
Either you are well or you are sick.
If you are well, there is nothing to worry about,

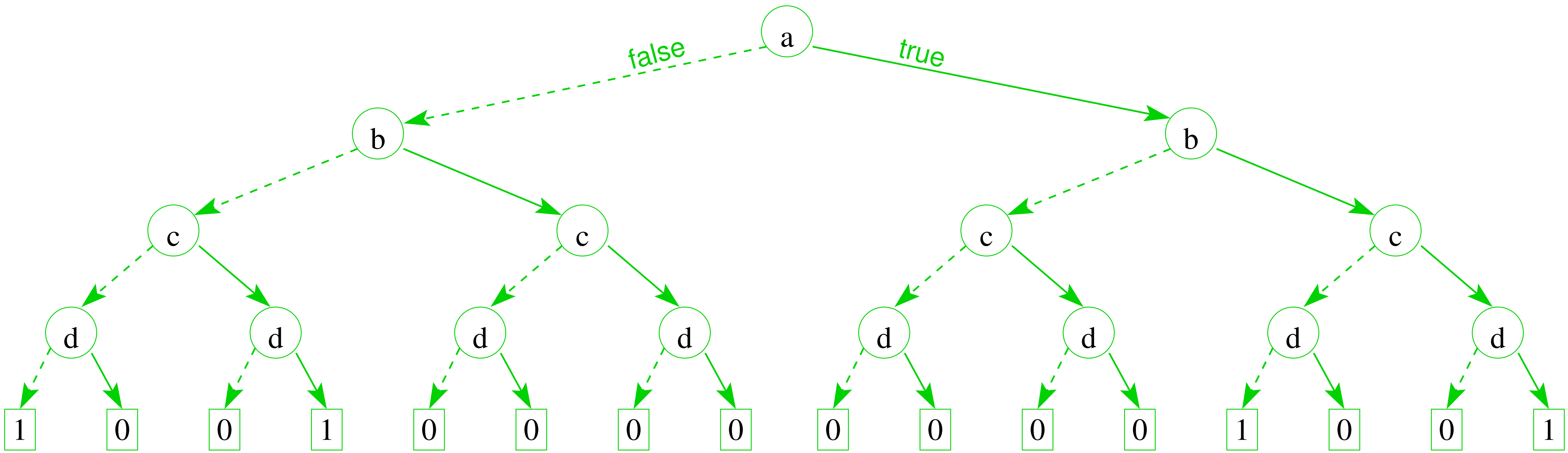
But if you are sick, there are only two things to worry about:
Either you will get well or you will die.
If you get well, there is nothing to worry about,

But if you die, there are only two things to worry about:
Either you will go to heaven or hell.
If you go to heaven, there is nothing to worry about.

And if you go to hell, you'll be so busy shaking hands with all your friends, you won't have time to worry!

Binary decision diagrams

An ordered decision tree for $(a \Leftrightarrow b) \wedge (c \Leftrightarrow d)$:



Size exponential in number of variables!

ROBDDs

Obs.: A lot of the tests in the decision diagram are redundant.

Idea: Combine equivalent sub-cases,
i.e. reduce size of the diagram by

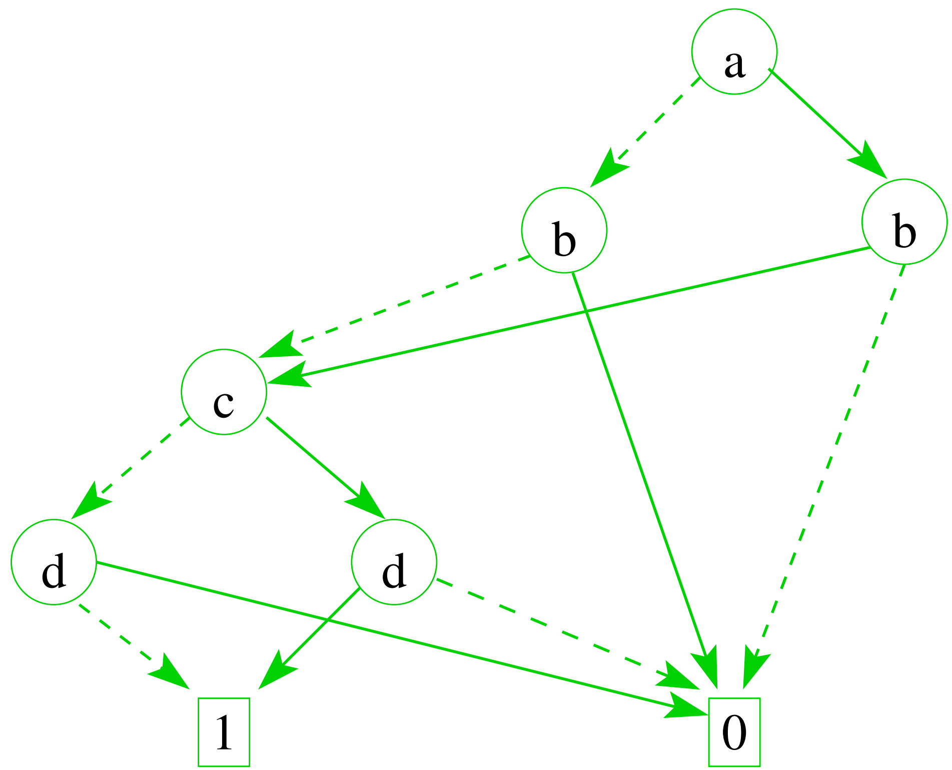
1. omitting nodes that have equivalent left and right sons,
2. sharing common sub-trees:
   - remove duplicate terminal nodes; share instead
   - remove duplicate internal nodes; share instead

Def.: The decision diagrams obtained by above rules are called reduced ordered binary decision diagrams (ROBDDs).

May expect good performance if many substructures are equivalent!

ROBDDs

An ROBDD for $(a \Leftrightarrow b) \wedge (c \Leftrightarrow d)$, using node order $a < b < c < d$:



Note how variable order affects size: Using $a < c < b < d$ would yield a layer with 4 nodes.
For n-bit comparison, we obtain a layer with 2^n nodes if poor order is chosen, yet maximum layer width 2 with appropriate order.

ROBDDS: Some properties

- Given a variable ordering, ROBDDs provide a canonical representation for Boolean functions
  - simple equivalence check, once the ROBDDs have been built:
    - linear in size of BDDs
    - $O(1)$ if sharing across BDDs is used
- Applying a connective to two ROBDDs can be done by simultaneous recursive descent through the two ROBDDs (+acceleration by dynamic programming)
  - $($if x then ϕ_t else $\phi_e) \wedge ($if x then ψ_t else $\psi_e) \equiv$
    $($if x then $\phi_t \wedge \psi_t$ else $\phi_e \wedge \psi_e)$

  $\rightarrow$ efficient

  $\rightarrow$ can construct ROBDDs for non-trivial circuits
- Variable order strongly affects size.
  - need reordering heuristics,
  - even then, some circuits don't permit any good order:
    e.g., multipliers yield exponentially sized BDDs

ROBDD operations

Negation:

Operation: Constructs from an ROBDD B an ROBDD $\mathrm{not}(B)$ with $f_{\mathrm{not}(B)} = \neg f_B$, where f_B is the truth function encoded by B.

Algorithm: Swap the terminal nodes:

- node 0 is replaced with 1
- node 1 is replaced with 0.

Complexity: $O(1)$ w/o sharing across BDDs, $O(|B|)$ w. sharing across BDDs.

ROBDD operations

Boolean junctors:

Operation: Constructs from two ROBDDs B_1, B_2 and a Boolean junctor $\oplus$ an ROBDD $\mathtt{apply}(\oplus, B_1, B_2)$ with $f_{\mathtt{apply}(\oplus,B_1,B_2)} = f_{B_1} \oplus f_{B_2}$.

Algorithm: Recursively proceed as follows:

- If both B_1 and B_2 are terminal nodes then yield terminal node $f_{B_1} \oplus f_{B_2}$.
- If the top nodes of B_1 and B_2 agree on their variable v then
  1. compute $L = \mathtt{apply}(\oplus, \mathtt{left}(B_1), \mathtt{left}(B_2))$,
  2. compute $R = \mathtt{apply}(\oplus, \mathtt{right}(B_1), \mathtt{right}(B_2))$,
  3. build the OBDD (v, L, R),
  4. reduce it.
- If the top nodes of B_1 and B_2 have different variables v_1, v_2 with $v_1 < v_2$ in the variable order then
  1. compute $L = \mathtt{apply}(\oplus, \mathtt{left}(B_1), B_2)$,
  2. compute $R = \mathtt{apply}(\oplus, \mathtt{right}(B_1), B_2)$,
  3. build the OBDD (v, L, R),
  4. reduce it.
- ...

Complexity: $O(|B_1| \cdot |B_2|)$ if memoization is used to save recomputations which may arise due to sharing of subgraphs.

ROBDD operations

Quantification:

Operation: Constructs from an ROBDD B and a variable v an ROBDD $exists(v, B)$ with $f_{exist(v,B)} = \exists v.f_B$.

Algorithm:

1. Replace each sub-BDD of B which has a root node n labeled with v by the ROBDD $apply(\vee, left(n), right(n))$.
2. Reduce the resulting BDD.

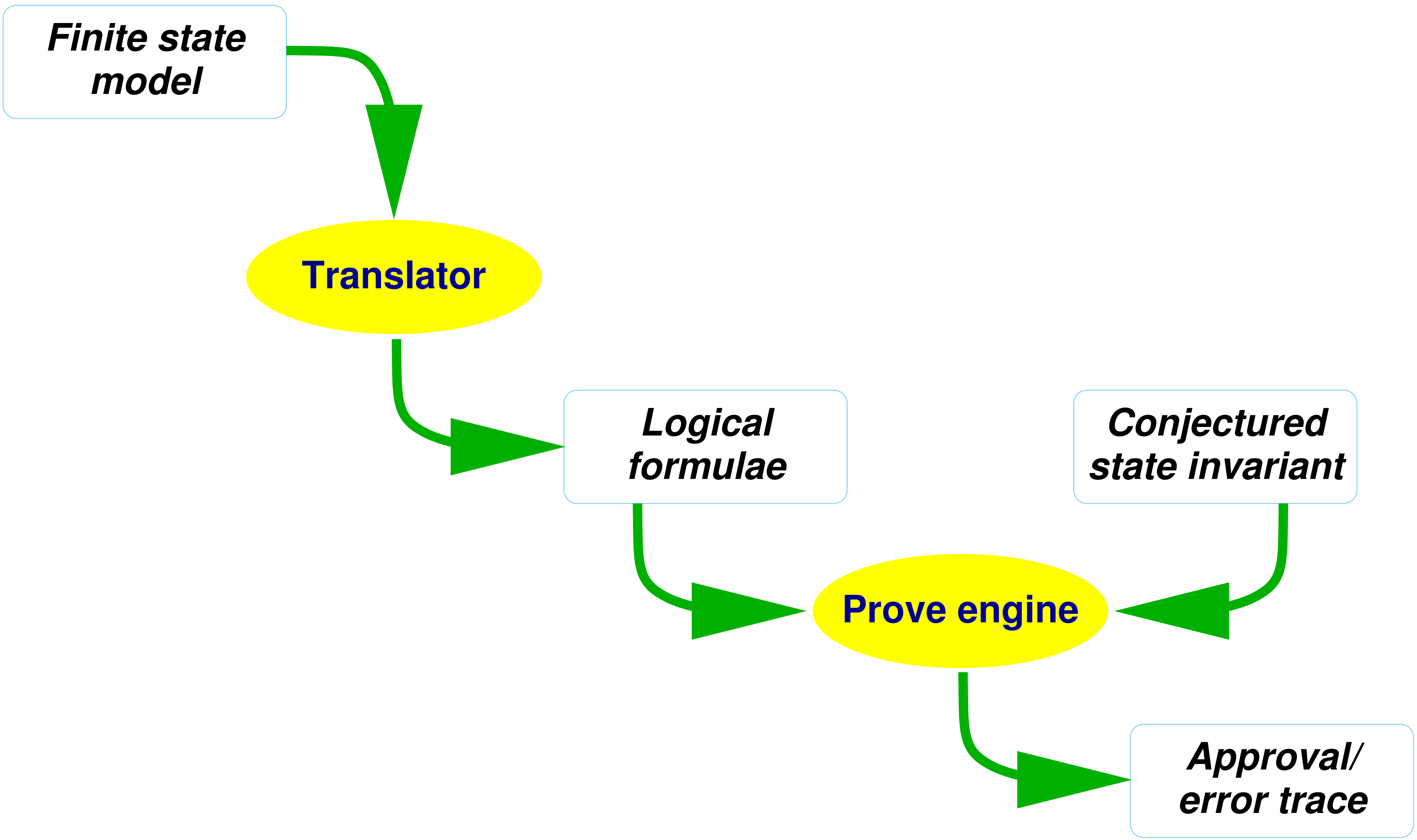
Complexity: $O(|B|^2)$.

Note that BDDs obtained by quantifying *multiple* variables may thus grow exponentially in the number of quantified variables.

Symbolic techniques II:

State reachability in finite-state reactive systems

The general framework



Mapping models to formulae (essence of)

- Each control location s is assigned a proposition p_s;
  each symbolic variable v is assigned $\lceil \log_2 |\mathrm{dom}\, v| \rceil$ propositional variables;
- for describing transitions, propositional variables are duplicated:
  - undecorated version encodes pre-state,
  - primed version encodes post-state,

$$\boxed{s} \xrightarrow{g\,/\,v := e} \boxed{t} \quad \mapsto \quad \phi_{tr} \;\equiv\; p_s \wedge \underbrace{[g]} \wedge \underbrace{[v' = e]} \wedge p'_t$$

proposit. encodings

$$\mathrm{trans}(x, x') \;\equiv\; \bigwedge_{s \text{ state}} \left(p_s \Longrightarrow \bigvee_{tr \text{ transition from } s} \phi_{tr} \right)$$

- similar for describing initial state set, yielding predicate $\mathrm{init}(x)$.

- Translation can be done componentwise, using conjunction for encoding parallel composition.

$\Rightarrow$ This saves computing the automaton product!

Verification/Falsification

Given: Transition pred. $trans(x, x')$, initial state pred. $init(x)$, conj. invar. $\phi(x)$.

QBF-based algorithm:

1. Start with $R_0(x) = init(x)$.
2. Test for satisfiability of $R_i(x) \wedge \neg\phi(x)$. If test succeeds then report violation of goal.
3. Else build $R_{i+1}(x) = R_i(x) \vee \exists\tilde{x}.\,(R_i(\tilde{x}) \wedge trans(\tilde{x}, x))$.
4. Test whether $R_{i+1}(x) \implies R_i(x)$. If so then report satisfaction of goal. Otherwise continue from step 2, with $i + 1$ instead of i.

BF-based algorithm:

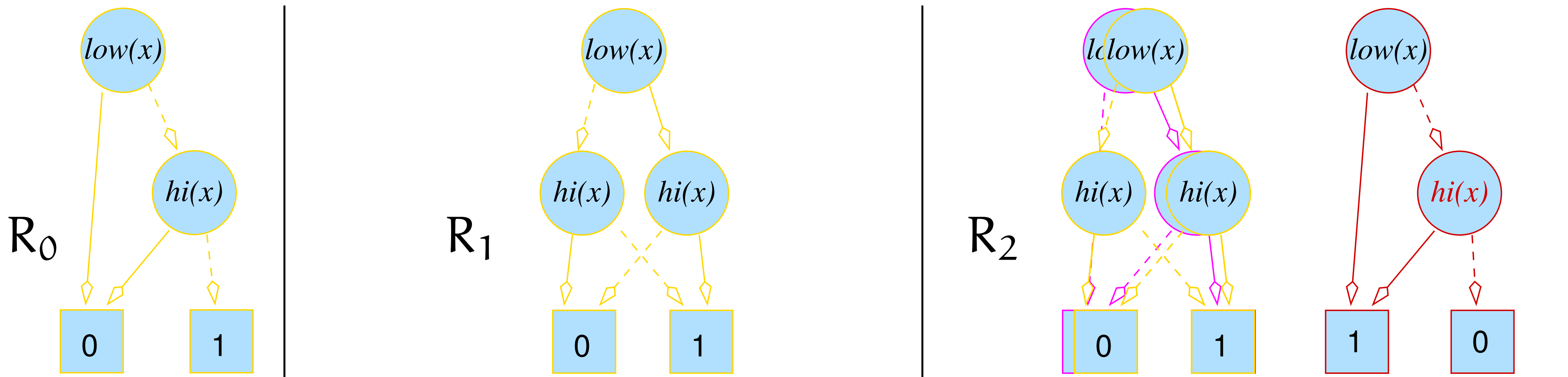
1. For given $i \in \mathbb{N}$ check for satisfiability of
$$\neg\left(\begin{array}{ll} & init(x_0) \wedge trans(x_0, x_1) \wedge \ldots \wedge trans(x_{i-1}, x_i) \\ \Rightarrow & \phi(x_0) \wedge \ldots \wedge \phi(x_i) \end{array}\right).$$
If test succeeds then report violation of goal.
2. Otherwise repeat with larger i.

Algorithms by example

Model: $\mathtt{VAR}\ x : \{0 \ldots 3\};\ \mathtt{INIT}\ x = 0;\ \mathtt{NEXT}\ x := 3 - x$

Conjectured Invar.: $\mathtt{ALWAYS}\ x = 0$

QBF: BDD-based MC



$\overline{l_0} \wedge \overline{h_0} \wedge$	$\overline{l_0} \wedge \overline{h_0} \wedge l_1 = \overline{l_0} \wedge h_1 = \overline{h_0} \wedge$	$\ldots \wedge \ldots \wedge l_2 = \overline{l_1} \wedge h_2 = \overline{h_1} \wedge$
$(l_0 \vee h_0)$	$(l_0 \vee h_0 \vee l_1 \vee h_1)$	$(\ldots \vee \ldots \vee l_2 \vee h_2)$

BF: SAT-based BMC

Comparison

BDD-based model-checking:

- Normalization within each step of graph coloring.

$$\Downarrow$$

1. Keeps size of intermediate representations compact.
2. Detects saturation of graph coloring.

- Tackles ≈ 500 *state bits*

SAT-based model-checking:

- Purely syntactic expansion, followed by satisfiability check.

$$\Downarrow$$

- Size of syntactic expansion grows rapidly. E.g. wrt. number of propositional variables used for characterizing n step reachability:

$$\begin{array}{ccc} & \text{statebits} & \times & (n+1) \\ + & \underbrace{\text{auxbits}}_{>90\%} & \times & n \end{array}$$

- Tackles $\approx 1.000.000$ *propositions*, most of which are auxiliary.

[Use cases: verification of high-level models w. limited arithmetic.]

Symbolic methods III:

Beyond reachability

The *pre* operator

Observation: Given

- a predicative encoding S of a state set (with free variables $\vec{x}$),
- a predicative encoding T of the transition relation (with free variables $\vec{x}, \vec{x}'$),

the set $\mathit{pre}(S)$ of states that have a successor in (i.e., satisfying) S can be expressed symbolicly using QBF operators:

$$\mathit{pre}(S) = \exists \vec{x}'.T \wedge S[\vec{x}'/\vec{x}]$$

This can be used for determining all sequential predecessors of a whole set of states in one sweep, thus implementing predecessor colouring “in parallel”.

Symbolic CTL model checking

Using the *pre* operator, CTL model checking can be performed by any QBF engine, e.g. by BDDs:

Formula	**Algorithm**	**Result**
propos. P	return $[P]$	Formula f_P denoting P-states
EX ϕ	return *pre*(f_ϕ)	Formula $f_{\mathsf{EX}\phi}$ denoting all states satisfying EX ϕ
EG ϕ	Incrementally build $S_0 = f_\phi$, $S_{i+1} = f_\phi \wedge pre(S_i)$ until $(S_n \iff S_{n+1})$ holds	Formula $f_{\mathsf{EG}\phi} = S_n$ denoting all states satisfying EG ϕ
ϕ EU ψ	Incrementally build $S_0 = f_\psi$, $S_{i+1} = f_\psi \vee (f_\phi \wedge pre(S_i))$ until $(S_n \iff S_{n+1})$ holds	Formula $f_{\phi\mathsf{EU}\psi} = S_n$ denoting all states satisfying ϕ EU ψ

If I characterizes initial states then $I \implies f_\phi$ is to be checked finally.