Generalization Strategies for the Verification of Infinite State Systems

Fabio Fioravanti
Dip. Scienze, University of Chieti-Pescara, Italy

joint work with

Alberto Pettorossi, Valerio Senni
DISP, University of Rome Tor Vergata, Italy

and

Maurizio Proietti
IASI-CNR, Rome, Italy
Outline

- Verification of infinite state systems
  - Computational tree logic
  - Constraint logic programming
- Two-phase Verification method
  - Rule-based program specialization
    - Generalization strategies
    - Perfect model computation
- Experimental evaluation
Infinite state systems

- The behaviour of a concurrent system can be represented as a state transition system which generates infinite computation paths:
Computational Tree Logic

- Properties are expressed in CTL, a propositional logic augmented with:
  - quantifiers over paths: E (Exists), A (All), and
  - temporal operators along paths: X (Next, in the next state in the path), F (Future, there exists a state in the path), G (Globally, for all states of the path).

- CTL Model Checking: decide whether or not $K,s \models \varphi$
  - decidable in polynomial time for finite state systems
  - undecidable for infinite state systems
Computational Tree Logic

Let $K$ be a Kripke structure $(S, I, R, L)$, $s$ a state, and $\text{Elem}$ a set of element properties, where $S$ is the set of states, $I \subseteq S$ is the set of initial states, $R \subseteq S \times S$ is the transition relation, $L: S \to \mathcal{P}(\text{Elem})$ is the labeling function.

Let $\pi$ be an infinite list $[s_0, ..., s_k, ...]$ of states and $d, \phi, \psi$ be CTL formulas

\[
K, s \models d \quad \text{iff} \quad d \in L(s)
\]

\[
K, s \models \neg \phi \quad \text{iff} \quad K, s \models \phi \quad \text{does not hold}
\]

\[
K, s \models \phi \land \psi \quad \text{iff} \quad K, s \models \phi \quad \text{and} \quad K, s \models \psi
\]

\[
K, s \models \text{EX} \phi \quad \text{iff} \quad \exists \pi = [s_0, s_1, ...], s = s_0, \text{ and } K, s_1 \models \phi
\]

\[
K, s \models \text{EU}(\phi, \psi) \quad \text{iff} \quad \exists \pi = [s_0, s_1, ...] \text{ s.t. } s = s_0 \text{ and } \exists n \geq 0
\]

\[
((\forall k, 0 \leq k < n, K, s_k \models \phi) \text{ and } K, s_n \models \psi)
\]

\[
K, s \models \text{AF} \phi \quad \text{iff} \quad \forall \pi = [s_0, s_1, ...] \text{ if } s = s_0 \text{ then } \exists n \geq 0 \text{ s.t. } K, s_n \models \psi
\]
The Bakery Protocol (Lamport)

Each process has: control state: \( s \in \{\text{think, wait, use}\} \) and counter: \( n \in \mathbb{N} \)

Process \( A \)

- \( \text{think} \)
  - \( n_A := n_B + 1 \)
- \( \text{wait} \)
  - \( n_A := 0 \)
- \( \text{use} \)
  - If \( n_A < n_B \) or \( n_B = 0 \)

Process \( B \)

- \( \text{think} \)
  - \( n_B := n_A + 1 \)
- \( \text{wait} \)
  - \( n_B := 0 \)
- \( \text{use} \)
  - If \( n_B < n_A \) or \( n_A = 0 \)

System: \( A || B \)

Path: \(<\text{think},0,\text{think},0> \rightarrow <\text{wait},1,\text{think},0> \rightarrow <\text{wait},1,\text{wait},2> \rightarrow <\text{use},1,\text{wait},2> \rightarrow \cdots\)

Mutual Exclusion: \(<\text{think},0,\text{think},0> \mid= \neg \ EF \ unsafe\)

where, for all \( n_A, n_B : \ <\text{use},n_A,\text{use},n_B> \mid= \ unsafe\)
Temporal Properties as Constraint Logic Programs

A system $S$ and the temporal logic are encoded by a CLP program.

- the transition relation is encoded by a binary predicate $tr$, like f.e.:

  $\text{tr}(<\text{think}, A, S, B>, <\text{wait}, A_1, S, B>) :- A_1=B+1.$
  
  $\text{tr}(<\text{wait}, A, S, B>, <\text{use}, A, S, B>) :- A<B.$
  
  $\text{tr}(<\text{wait}, A, S, B>, <\text{use}, A, S, B>) :- B=0.$
  
  $\text{tr}(<\text{use}, A, S, B>, <\text{think}, A_1, S, B>) :- A_1=0.$

  + similar clauses for process $B$

- the initial states:

  $\text{initial}(<\text{think}, A, \text{think}, B>) :- A=0, B=0.$

- the elementary properties:

  $\text{elem}(<\text{use}, A, \text{use}, B>, \text{unsafe}).$
Temporal Properties as Constraint Logic Programs

The satisfaction relation $\models$ is encoded by a binary predicate \texttt{sat}

\begin{verbatim}
  sat(X, F) :- elem(X,F)
  sat(X, not(F)) :- \+ sat(X,F)
  sat(X, and(F1,F2))   :-  sat(X,F1), sat(X,F2)
  sat(X, ex(F)) :- tr(X,Y), sat (Y,F)
  sat(X, eu(F1,F2))  :-  sat(X,F2)
  sat(X, eu(F1,F2)) :-  sat(X,F1), tr(X,Y), sat(Y,eu(F1,F2))
  sat(X, af(F)) :- sat(X,F)
  sat(X, af(F)) :-  ts(X,Ys), sat_all(Ys,af(F))
  sat_all([],F).
  sat_all([X|Xs],F) :- sat(X,F), sat_all(Xs,F)
\end{verbatim}

where \texttt{ts(X,Ys)} holds iff \texttt{Ys} is a list of all the successor states of \texttt{X}
The property to be verified is defined by a predicate prop.

s.t. \[ \text{prop} \equiv \forall X (\text{initial}(X) \rightarrow \text{sat}(X, \varphi)) \]

\[ \neg \exists X (\text{initial}(X) \land \neg \text{sat}(X, \varphi)) \]

encoded as follows

\[ g1 : \text{prop} :- \neg \text{negprop} \]

\[ g2 : \text{negprop} :- \text{initial}(X), \neg \text{sat}(X, \varphi) \]
Let $P_s$ be the set of clauses defining predicates sat, tr, ts, sat_all, prop, negprop. $P_s$ is locally stratified, and thus it has a unique perfect model.

Theorem 1. Let $K$ be a Kripke structure, let $I$ be the set of initial states of $K$, and let $\phi$ be a CTL formula. Then,

$$(\text{for all states } s \in I, \ K,s \models \phi) \iff \text{prop} \in M(P_s).$$

But...

- **Bottom-up** construction of $M(P_s)$ from facts may not terminate because $M(P_s)$ is infinite.
- **Top-down** evaluation of $P_s$ from $prop$ may not terminate due to infinite computation paths.
Two-phase Verification Method

- **Phase 1**: specialize $P_S$ w.r.t. the query $prop$:
  \[ P_S \rightarrow \ldots \rightarrow SpP_S \text{ s.t. } \text{prop} \in M(P_S) \iff \text{prop} \in M(SpP_S) \]
  and keep only the clauses on which the predicate $prop$ depends. $SpP_S$ is a stratified program.

  Specialization is performed by using the rules + strategies program transformation approach
  - '$\rightarrow'$ is an application of a transformation rule.

- **Phase 2**: construct bottom-up the perfect model of $M(SpP_S)$
  (may not terminate)
Specialization strategy

- Input: The program $P_S$
  Output: A stratified program $SpP_S$ such that
  $\text{prop} \in M(P_S)$ iff $\text{prop} \in M(SpP_S)$.

- $SpPs := \{g1\}; \text{InDefs} := \{g2\}; \text{Defs} := \{\}$;
- while (there exists a clause $\gamma$ in InDefs) do
  - Unfold($\gamma, \Gamma$);
  - Generalize&Fold(Defs, $\Gamma$, NewDefs, $\Phi$);
  - $SpPs := SpPs \cup \Phi$; InDefs := (InDefs $\setminus \{\gamma\}) \cup$ NewDefs;
  end-while
Termination of specialization (Phase 1)

- Local control
  - Termination of the Unfold procedure
- Global control
  - Termination of the while loop
  - We use constraint generalization techniques
Generalization

- For limiting the number of clauses introduced by definition, sometimes we introduce definitions containing a generalized constraint
- Well quasi orderings: generalization is eventually applied
- Generalization operators: each definition can be generalized a finite number of times only
- Selecting a good generalization strategy is not trivial
  - Too coarse -> unable to prove property
  - Too fine-grained -> high verification times
The constraint domain $\text{Lin}_k$

- $\text{Lin}_k$ are linear inequations over $k$ distinct variables $X_1,\ldots,X_k$
- Constraints of $\text{Lin}_k$ are conjunctions of atomic constraints of the form
  - $p \leq 0$ or $p < 0$
where $p$ is a polynomial of the form
- $q_0 + q_1X_1 + \ldots + q_kX_k$
and $q_i$'s are integers
Well-quasi orderings

- A well-quasi ordering on a set $S$ is a reflexive, transitive, binary relation $\leq$ such that, for every infinite sequence $e_0, e_1, \ldots$ of elements of $S$, there exist $i$ and $j$ such that $i < j$ and $e_i \leq e_j$. 
HomeoCoeff compares sequences of absolute values of integer coefficients occurring in polynomials

(i) \( q_0 + q_1 x_1 + \ldots + q_k x_k \leq r_0 + r_1 x_1 + \ldots + r_k x_k \)

iff there exist a permutation \( h \) of the indexes \( \langle 0, \ldots, k \rangle \) such that, for \( i=0,\ldots,k \), \( |q_i| \leq |r_{h(i)}| \)

- Extended to atomic constraints and constraints
  - for example \( q<0 \preceq r<0 \) iff (i) holds
MaxCoeff and SumCoeff wqo's

- **MaxCoeff** compares the maximum absolute value of coefficients occurring in polynomials for any two atomic constraints $q$ and $r$, we have that $q \preceq r$ iff $\max\{|q_0|, \ldots, |q_k|\} \leq \max\{|r_0|, \ldots, |r_k|\}$

- **SumCoeff** compares the sum of the absolute value of coefficients occurring in polynomials
  Similarly $q \preceq r$ iff $|q_0| + \ldots + |q_k| \leq |r_0| + \ldots + |r_k|$
Generalization operators

- Given a wqo \( \preceq \), the generalization of a constraint \( c \) w.r.t. a constraint \( d \) is a constraint \( c \ominus d \) such that
  - \( d \sqsubseteq c \ominus d \)
  - \( c \ominus d \preceq c \)
- \( c \ominus d \) can replace \( d \) in a candidate definition for folding
- Every infinite sequence of constraints constructed by using the generalization operator eventually stabilizes (similar to the widening operator in abstract interpretation)
- In general, \( \ominus \) is not commutative
Generalization operators

Let $c = a_1, ..., a_m$ and $d = b_1, ..., b_n$

- **Top**: $c \ominus d$ is the constraint *true*
- **Widen**: $c \ominus d$ is the conjunction of all $a_i$'s such that $d \sqsubseteq a_i$
- **WidenPlus**: $c \ominus d$ is the conjunction of all $a_i$'s such that $d \sqsubseteq a_i$ and of all $b_j$'s such that $b_j \preceq c$
- **CHWiden and CHWidenPlus** obtained by applying the Convex Hull operator
Experimental evaluation

- Experiments performed using the MAP transformation system
  - http://www.iasi.cnr.it/~proietti/system.html
- **Mutual exclusion protocols:**
  - bakery2 (safety and liveness)
  - bakery3 (safety)
  - Mutast (safety)
  - Peterson (safety for N processes)
  - Ticket (safety and liveness)
Experimental evaluation

- Parameterized cache coherence protocols
  - Berkeley RISC, DEC Firefly, IEEE Futurebus+, Illinois University, MESI, MOESI, Synapse N+1, and Xerox PARC Dragon.

- Used in shared-memory multiprocessing systems for guaranteeing data consistency of the local cache associated with every CPU
Experimental evaluation

- Other systems
  - Parameterized barber problem with N customers
  - Producer-consumer via Bounded and Unbounded buffer
  - CSM a central server model
  - Insertion and selection sort: check array bounds
  - Office light control
  - Reset Petri nets
<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>wquery W:</th>
<th>Generalization G.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CHWiden</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HC</td>
</tr>
<tr>
<td>Bakery 2 (safety)</td>
<td>20</td>
<td>70</td>
</tr>
<tr>
<td>Bakery 2 (liveness)</td>
<td>60</td>
<td>120</td>
</tr>
<tr>
<td>Bakery 3 (safety)</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>Mut.Ast</td>
<td>160</td>
<td>800</td>
</tr>
<tr>
<td>Peterson N</td>
<td>230</td>
<td>440</td>
</tr>
<tr>
<td>Ticket (safety)</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Ticket (liveness)</td>
<td>90</td>
<td>120</td>
</tr>
<tr>
<td>Berkeley RISC</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>DEC Firefly</td>
<td>190</td>
<td>120</td>
</tr>
<tr>
<td>IEEE Futurebus+</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>Illinois University</td>
<td>50</td>
<td>80</td>
</tr>
<tr>
<td>M68000</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>M1</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>MOESI</td>
<td>80</td>
<td>40</td>
</tr>
<tr>
<td>Synapse N+1</td>
<td>980</td>
<td>160</td>
</tr>
<tr>
<td>Synapse N+2</td>
<td>950</td>
<td>60</td>
</tr>
<tr>
<td>XeroxPARC Dragon</td>
<td>1230</td>
<td>80</td>
</tr>
<tr>
<td>Barber</td>
<td>41380</td>
<td>30150</td>
</tr>
<tr>
<td>Bounded Buffer</td>
<td>73990</td>
<td>370</td>
</tr>
<tr>
<td>Unbounded Buffer</td>
<td>73190</td>
<td>170</td>
</tr>
<tr>
<td>CSM</td>
<td>310</td>
<td>130</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>80</td>
<td>60</td>
</tr>
<tr>
<td>Office Light Control</td>
<td>380</td>
<td>80</td>
</tr>
<tr>
<td>Reset Petri Nets</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
Analysis

- Precision (number of properties proved) and average verification time
  - SumCoeff &WidenPlus 23/23 (820 ms)
  - MaxCoeff &WidenPlus 22/23 (730 ms)
  - SumCoeff &CHWidenPlus 22/23 (2990 ms)

- Top and Widen are fast but not accurate
  - Information about the call can be lost
Comparison with other systems

- **Action Language Verifier (Bultan 01)**
  - combines BDD-based symbolic manipulation for boolean and enumerated types, with a solver for linear constraints on integers

- **DMC (Delzanno 01)**
  - computes (approximated) least and greatest models of CLP(R) programs

- **HyTech (Henzinger 97)**
  - model checker for hybrid systems
<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>MAP ( SC&amp;WidenPlus )</th>
<th>ALV ( \text{default} )</th>
<th>ALV ( A )</th>
<th>ALV ( F )</th>
<th>ALV ( L )</th>
<th>DMC ( \text{noAbs} )</th>
<th>DMC ( \text{Abs} )</th>
<th>HyTech ( Fw )</th>
<th>HyTech ( Bw )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bakery 2 (safety)</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>90</td>
<td>30</td>
<td>10</td>
<td>30</td>
<td>( \infty )</td>
<td>20</td>
</tr>
<tr>
<td>Bakery 2 (liveness)</td>
<td>70</td>
<td>30</td>
<td>30</td>
<td>90</td>
<td>30</td>
<td>60</td>
<td>70</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Bakery 3 (safety)</td>
<td>160</td>
<td>580</td>
<td>570</td>
<td>( \infty )</td>
<td>600</td>
<td>460</td>
<td>3090</td>
<td>( \infty )</td>
<td>360</td>
</tr>
<tr>
<td>MutAst</td>
<td>140</td>
<td>( \perp )</td>
<td>( \perp )</td>
<td>910</td>
<td>( \perp )</td>
<td>150</td>
<td>1370</td>
<td>70</td>
<td>130</td>
</tr>
<tr>
<td>Peterson N</td>
<td>230</td>
<td>71690</td>
<td>( \perp )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>70</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Ticket (safety)</td>
<td>40</td>
<td>( \infty )</td>
<td>80</td>
<td>30</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>60</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Ticket (liveness)</td>
<td>110</td>
<td>( \infty )</td>
<td>230</td>
<td>40</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>220</td>
<td>( \times )</td>
<td>( \times )</td>
</tr>
<tr>
<td>Berkeley RISC</td>
<td>30</td>
<td>10</td>
<td>( \perp )</td>
<td>20</td>
<td>60</td>
<td>30</td>
<td>30</td>
<td>( \infty )</td>
<td>20</td>
</tr>
<tr>
<td>DEC Firefly</td>
<td>20</td>
<td>10</td>
<td>( \perp )</td>
<td>20</td>
<td>80</td>
<td>50</td>
<td>80</td>
<td>( \infty )</td>
<td>20</td>
</tr>
<tr>
<td>IEEE Futurebus+</td>
<td>2460</td>
<td>320</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>670</td>
<td>4670</td>
<td>9890</td>
<td>( \infty )</td>
<td>380</td>
</tr>
<tr>
<td>Illinois University</td>
<td>20</td>
<td>10</td>
<td>( \perp )</td>
<td>( \infty )</td>
<td>140</td>
<td>70</td>
<td>110</td>
<td>( \infty )</td>
<td>20</td>
</tr>
<tr>
<td>MESI</td>
<td>30</td>
<td>10</td>
<td>( \perp )</td>
<td>20</td>
<td>60</td>
<td>40</td>
<td>60</td>
<td>( \infty )</td>
<td>20</td>
</tr>
<tr>
<td>MOESI</td>
<td>60</td>
<td>10</td>
<td>( \perp )</td>
<td>40</td>
<td>100</td>
<td>50</td>
<td>90</td>
<td>( \infty )</td>
<td>10</td>
</tr>
<tr>
<td>Synapse N+1</td>
<td>10</td>
<td>10</td>
<td>( \perp )</td>
<td>10</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>( \infty )</td>
<td>0</td>
</tr>
<tr>
<td>Xerox PARC Dragon</td>
<td>40</td>
<td>20</td>
<td>( \perp )</td>
<td>40</td>
<td>340</td>
<td>70</td>
<td>120</td>
<td>( \infty )</td>
<td>20</td>
</tr>
<tr>
<td>Barber</td>
<td>1170</td>
<td>340</td>
<td>( \perp )</td>
<td>90</td>
<td>360</td>
<td>140</td>
<td>230</td>
<td>( \infty )</td>
<td>90</td>
</tr>
<tr>
<td>Bounded Buffer</td>
<td>3540</td>
<td>0</td>
<td>10</td>
<td>( \infty )</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>( \infty )</td>
<td>10</td>
</tr>
<tr>
<td>Unbonded Buffer</td>
<td>3890</td>
<td>10</td>
<td>10</td>
<td>40</td>
<td>40</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>CSM</td>
<td>6580</td>
<td>79490</td>
<td>( \perp )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>100</td>
<td>40</td>
<td>60</td>
<td>( \infty )</td>
<td>70</td>
<td>30</td>
<td>80</td>
<td>( \infty )</td>
<td>10</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>190</td>
<td>( \infty )</td>
<td>390</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Office Light Control</td>
<td>50</td>
<td>20</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>Reset Petri Nets</td>
<td>0</td>
<td>( \infty )</td>
<td>( \perp )</td>
<td>( \infty )</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>( \infty )</td>
<td>10</td>
</tr>
</tbody>
</table>
Analysis

- Precision (number of properties proved) and average verification time
  - MAP 23/23 (820 ms)
  - DMC (with abstraction) 19/23 (820 ms)
  - ALV (default option) 18/23 (8480 ms)
  - HyTech (backwards) 17/23 (70 ms)
Analysis

- Bounded and Unbounded Buffer can be easily verified by backward reachability
  - The specialization phase is redundant
  - MAP slower than other systems
- Peterson and CSM examples
  - The specialization phase pays off
  - MAP much more efficient than other systems
Future work

- Use approximation methods during the bottom-up computation of the perfect model (Phase 2)
- Apply specialization to concurrent systems specified in different languages, not necessarily (C)LP based
The end
Transformation rules

- Unfolding
  - basically a resolution step
  - From
    
    \[
    \begin{align*}
    p(X,Y) & : - Y=0, q(X) \\
    q(X) & : - X>2, r \\
    q(X) & : - X<1, s \\
    \end{align*}
    \]
  - To
    
    \[
    \begin{align*}
    p(X,Y) & : - Y=0, X>2, r \\
    p(X,Y) & : - Y=0, X<1, s \\
    q(X) & : - X>2, r \\
    q(X) & : - X<1, s \\
    \end{align*}
    \]
Transformation rules

- Constrained atomic definition
- We add a new clause to the current program
  - newpred(X) :- e(X), sat(X, φ)

where newpred is a fresh predicate symbol
Transformation rules

- Constrained atomic folding
  - Inverse of unfolding
  - From
    \[
    p(X) :- X=2, \ q(X) \\
    newq(X) :- X>1, \ q(X)
    \]
  - To
    \[
    p(X) :- X=2, \ newq(X) \\
    newq(X) :- X>1, \ q(X)
    \]
  - Notice that \( X=2 \) implies \( X>1 \)
Transformation rules

- **Clause removal**
- Remove clauses with unsatisfiable constraints
  - p(X) :- X=0, X=1.
- Remove clauses subsumed by other clauses of the form H :- c where c is a constraint
  - For example q(Y) :- Y>2, p(X,Y) is subsumed by q(Y) :- Y>0.
Unfold procedure

- Unfold once, then unfold as long as in the body of a clause obtained by unfolding there is an atom of one of the following forms:
  - $t(s_1, s_2), ts(s, ss)$
  - $sat(s, e)$, where $e$ is an elementary property,
  - $sat(s, not(\psi))$, $sat(s, and(\psi_1, \psi_2))$, $sat(s, ex(\psi_1))$
  - $sat_all(ss, \psi_1)$, where $ss$ is a non-variable list

- Clause removal
- We do not repeatedly unfold atoms $sat(s, eu(\psi))$ and $sat(s, af(\psi))$
- $Unfold(\gamma, \Gamma)$ terminates for any clause $\gamma$ with a ground CTL formula