From PSL to NBA: a Modular Symbolic Encoding

A. Cimatti[1] M. Roveri[1] S. Semprini[1] S. Tonetta[2]

[1]ITC-irst Trento, Italy
{cimatti,roveri}@itc.it

[2]University of Lugano, Lugano, Switzerland
tonettas@lu.unisi.ch

Formal Methods in Computer Aided Design, 2006

Outline

- Motivation
- Technical Background
- Monolithic Encoding of PSL into NBA
- Modular encoding of PSL into NBA
  - Suffix Operator Normal Form for PSL
  - Modular Translation into NBA
  - Optimized encoding
- Experimental Evaluation
- Conclusions and Future Work

Outline

- Motivation
- Technical Background
- Monolithic Encoding of PSL into NBA
- Modular encoding of PSL into NBA
  - Suffix Operator Normal Form for PSL
  - Modular Translation into NBA
  - Optimized encoding
- Experimental Evaluation
- Conclusions and Future Work

Motivations

- Assertion Based Verification is becoming increasingly important.
- The Property Specification Language PSL:
  - a means used to capture requirements on behavior of designs.
  - LTL + regular expressions = ω-regular expressiveness.
- Several verification engines efficiently manipulate NBA.
- A lot of research has been done to efficiently translate LTL into NBA.
- Several model checkers for PSL currently accept a subset of the language.
- Converting PSL to symbolic NBA is an important enabling factor.
  - Reuse of large wealth of mature verification tools.

Outline

- Motivation
- Technical Background
- Monolithic Encoding of PSL into NBA
- Modular encoding of PSL into NBA
  - Suffix Operator Normal Form for PSL
  - Modular Translation into NBA
  - Optimized encoding
- Experimental Evaluation
- Conclusions and Future Work

Definition (SEREs syntax)

- if b is propositional, then b is a SERE;
- if r is a SERE, then r**[*]** is a SERE;
- if r_1 and r_2 are SEREs, then the following are SEREs

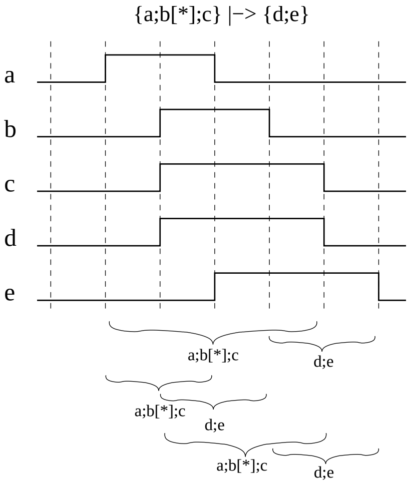
$$r_1 \; \mathbf{;} \; r_2 \qquad r_1 \; \mathbf{:} \; r_2 \qquad r_1 \; \mathbf{|} \; r_2$$

$$r_1 \; \mathbf{\&} \; r_2 \qquad r_1 \; \mathbf{\&\&} \; r_2$$

Definition (PSL syntax)

- if p is propositional, p is a PSL formula;
- if ϕ_1 and ϕ_2 are PSL formulas, then $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ are PSL formulas;
- if ϕ_1 and ϕ_2 are PSL formulas, then $\mathbf{X}\ \phi_1$, $\phi_1\ \mathbf{U}\ \phi_2$, $\phi_1\ \mathbf{R}\ \phi_2$ are PSL formulas;
- if r is a SERE and ϕ is a PSL formulas, then $r \diamond\!\!\rightarrow \phi$ and $r \mapsto \phi$ are PSL formulas;
- if r is a SERE, then r is a PSL formula.

- $\{a ; b[*] ; c\} \mapsto \{d ; e\}$:
  All sequences matching $\{a ; b[*] ; c\}$ should not be followed by a sequence not matching $\{d ; e\}$.
- $\{a ; b[*] ; c\} \diamondsuit\!\!\rightarrow \{d ; e\}$:
  At least one sequence matching $\{a ; b[*] ; c\}$ should not be followed by a sequence not matching $\{d ; e\}$.



Outline

- Motivation
- Technical Background
- **Monolithic Encoding of PSL into NBA**
- Modular encoding of PSL into NBA
  - Suffix Operator Normal Form for PSL
  - Modular Translation into NBA
  - Optimized encoding
- Experimental Evaluation
- Conclusions and Future Work

Monolitic encoding of PSL

φ

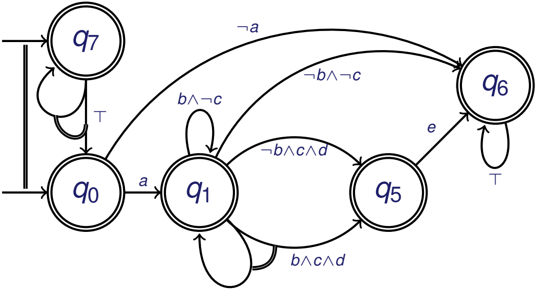
$$\varphi = \textbf{always}\ (\{a\ \textbf{;}\ b\textbf{[*]}\ \textbf{;}\ c\} \mapsto \{d\ \textbf{;}\ e\})$$

Monolitic encoding of PSL

φ

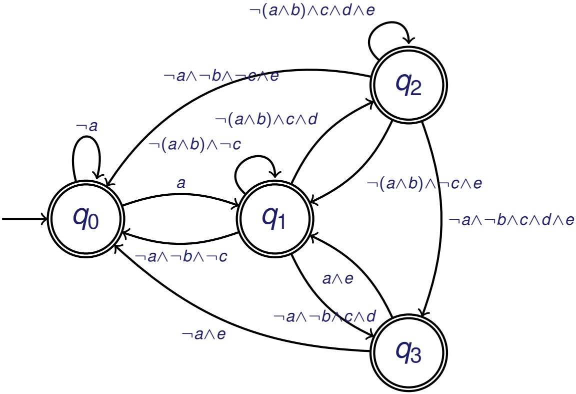
$\downarrow$

$ABA(\varphi)$



Monolitic encoding of PSL

$$\varphi \longrightarrow ABA(\varphi) \xrightarrow{\text{EMH}} NBA(ABA(\varphi))$$

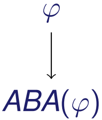


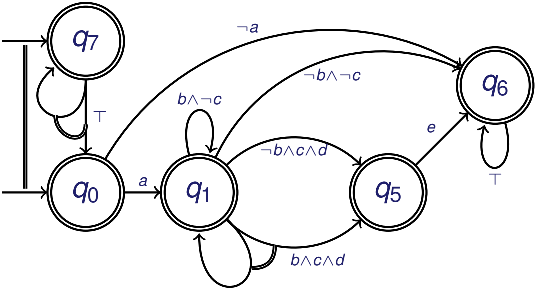
Monolitic encoding of PSL

$$\varphi$$
$$\downarrow$$
$$ABA(\varphi)$$
$$\text{EMH}\downarrow$$
$$NBA(ABA(\varphi))$$
$$\downarrow$$
$$SNBA(NBA(ABA(\varphi)))$$

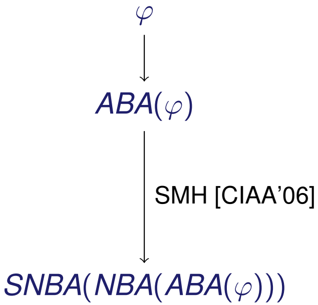
```
VAR
  st :  {qq0, qq1, qq2, qq3};
DEFINE
  q0 := st = qq0; q1 := st = qq1;
  q2 := st = qq2; q3 := st = qq3;
INIT q0
TRANS
  q0 -> ((a & next(q1)) |
         (!a & next(q0))) &
  q1 -> ((!a & !b & !c & next(q0)) |
         (!a & !b & c & d & next(q3)) |
         (!(a & b) & c & d & next(q2)) |
         (!(a & b) & !c & next(q1))) &
.. .. ..
FAIRNESS (q0 | q1 | q2 | q3)
```

Monolitic encoding of PSL





Monolitic encoding of PSL




```
VAR qL0 :  boolean; qL1 :  boolean;
    qL5 :  boolean; qL6 :  boolean;
    qL7 :  boolean;

TRANS
 (qL0 -> ((a & next(qL1)) |
  (!a & next(qL6)))) &
 (qL1 -> ((b & !c & next(qL1)) |
  (!b & c & d & next(qL5)) |
  (b & c & d & next(qL1) & next(qL5)) |
  (!b & !c & next(qL6)))) &
 (qL5 -> (e & next(qL6))) &
 (qL6 -> (next(qL6))) &
 (qL7 -> (next(qL0) & next(qL7)))

INIT qL0 & qL7;

FAIRNESS TRUE
```

Symbolic Encoding of PSL

Pros

- Explicit representation allows for advanced optimization:
  - On average significan reduction of the size of the resulting automata.
  - Very often better performance in search.
  - Applicable both to ABA and to NBA.

Symbolic Encoding of PSL

Cons

- Optimizations are very often expensive.
- The Miyano-Hayashi's construction for an ABA of n states generates an NBA of $O(3^n)$ states.
- Symbolic encoding of Miyano-Hayashi can avoid blowup associated with conversion to NBA.
  - It is postponed to search time.

Outline

- Motivation
- Technical Background
- Monolithic Encoding of PSL into NBA
- Modular encoding of PSL into NBA
  - Suffix Operator Normal Form for PSL
  - Modular Translation into NBA
  - Optimized encoding
- Experimental Evaluation
- Conclusions and Future Work

The modular encoding of PSL into NBA

1. Turn PSL formula into *Suffix Operator Normal Form* (SONF).
   - Separates out SERE components and LTL components.
     - They can be encoded separately.
   - LTL components can leverage on mature techniques.
   - PSL components can be encoded with any standard conversion from ABA to NBA.
   - Final automaton constructed as an implicit symbolic product.
     - Composition delayed at search time.
2. Interface between SERE and LTL is normalized.
   - Only specific PSL patterns are possible.
     - *Suffix Operator Subformulas*.
   - Optimized encoding techniques for such patterns, to improve efficiency of symbolic translation to NBA.

Suffix Operator Normal Form for PSL

Extension of $\mathcal{NNF}$ conversion to PSL:

Definition (NNF)

$$
\begin{array}{lcl}
\mathcal{NNF}(\neg p) & := & \neg p \\
\mathcal{NNF}(\neg(\phi_1 \vee \phi_2)) & := & \mathcal{NNF}(\neg\phi_1) \wedge \mathcal{NNF}(\neg\phi_2) \\
\mathcal{NNF}(\neg(\phi_1 \wedge \phi_2)) & := & \mathcal{NNF}(\neg\phi_1) \vee \mathcal{NNF}(\neg\phi_2) \\
\mathcal{NNF}(\neg(\phi_1 \ \mathbf{U}\ \phi_2)) & := & \mathcal{NNF}(\neg\phi_1) \ \mathbf{R}\ \mathcal{NNF}(\neg\phi_2) \\
\mathcal{NNF}(\neg(\phi_1 \ \mathbf{R}\ \phi_2)) & := & \mathcal{NNF}(\neg\phi_1) \ \mathbf{U}\ \mathcal{NNF}(\neg\phi_2) \\
\mathcal{NNF}(\neg(r \diamond\!\!\rightarrow \phi_1)) & := & r \mapsto \mathcal{NNF}(\neg\phi_1) \\
\mathcal{NNF}(\neg(r \mapsto \phi_1)) & := & r \diamond\!\!\rightarrow \mathcal{NNF}(\neg\phi_1)
\end{array}
$$

Suffix Operator Normal Form for PSL

Let ϕ be the $\mathcal{NNF}(\varphi)$ of a PSL formula φ.

SONF-ization

For every subformula of φ of the form $r \diamond\!\!\rightarrow \psi$ (resp., $r \mapsto \psi$), we introduce two new atoms: $P_{r \diamond\!\!\rightarrow \psi}$ (resp., $P_{r \mapsto \psi}$) and P_ψ

$$\begin{aligned} \varphi[r \diamond\!\!\rightarrow \psi] \quad \Rightarrow \quad & \varphi[P_{r \diamond\!\!\rightarrow \psi} / r \diamond\!\!\rightarrow \psi] \wedge \\ & \mathbf{G}\,(P_{r \diamond\!\!\rightarrow \psi} \rightarrow (r \diamond\!\!\rightarrow P_\psi)) \wedge \\ & \mathbf{G}\,(P_\psi \rightarrow \psi) \end{aligned}$$

$$\begin{aligned} \varphi[r \mapsto \psi] \quad \Rightarrow \quad & \varphi[P_{r \mapsto \psi} / r \mapsto \psi] \wedge \\ & \mathbf{G}\,(P_{r \mapsto \psi} \rightarrow (r \mapsto P_\psi)) \wedge \\ & \mathbf{G}\,(P_\psi \rightarrow \psi) \end{aligned}$$

Suffix Operator Normal Form for PSL

$$\mathtt{Sonf}(\phi) := \overbrace{\bigwedge_i \phi_i}^{\Psi_{LTL}} \wedge \overbrace{\bigwedge_j \mathbf{G}\,(P_j \rightarrow (r_j \star\!\!\longrightarrow P'_j))}^{\Psi_{PSL}}$$

Suffix Operator Normal Form for PSL

$$\texttt{Sonf}(\phi) := \overbrace{\bigwedge_i \phi_i}^{\Psi_{LTL}} \wedge \overbrace{\bigwedge_j \underbrace{\mathbf{G}\,(P_j \rightarrow (r_j \star\!\!\longrightarrow P'_j))}_{\text{Suffix Operator Subformula}}}^{\Psi_{PSL}}$$

Suffix Operator Normal Form for PSL

$$\texttt{Sonf}(\phi) := \overbrace{\bigwedge_i \phi_i}^{\Psi_{LTL}} \wedge \overbrace{\bigwedge_j \underbrace{\mathbf{G}\,(P_j \rightarrow (r_j \star\!\!\longrightarrow P'_j)}_{\text{Suffix Operator Subformula}}}^{\Psi_{PSL}}$$

Theorem

Let ϕ be a PSL formula over $\mathcal{A}$ and ψ a PSL subformula of ϕ that occurs only positively in ϕ. If

$$\phi' := \phi[{}^{P}/_{\psi}] \wedge \mathbf{G}\,(P \rightarrow \psi).$$

then $\mathcal{L}_{\mathcal{A}}(\phi) = \mathcal{L}_{\mathcal{A}}(\phi')$.

Suffix Operator Normal Form for PSL
Example

Example

$$\varphi \;=\; \textbf{always}\ (\{a\ ;\ b\textbf{[*]}\ ;\ c\} \mapsto \{d\ ;\ e\})$$

$$\begin{aligned}\texttt{Sonf}(\varphi) \;=\; & \textbf{always}\ (P_0) \wedge \\ & \textbf{always}\ (P_0 \rightarrow (\{a\ ;\ b\textbf{[*]}\ ;\ c\} \mapsto P_1)) \wedge \\ & \textbf{always}\ (P_1 \rightarrow (\{d\ ;\ e\} \diamond\!\!\rightarrow \mathit{True}))\end{aligned}$$

Modular Translation from PSL to NBA

```
ModPsl2Ba(φ)
input φ the PSL input formula
output a set Q of NBAs;
        the final NBA is the product of all NBAs in Q
begin
  Q := ∅;
  φ' := Sonf(φ);          /* φ' is in the form Ψ_LTL ∧ Ψ_PSL */
  for ψ ∈ Ψ_LTL do
    A := Ltl2Ba(ψ);
    Q := Q ∪ {A};
  end
  for ψ ∈ Ψ_PSL do
    A := Psl2Ba(ψ);
    Q := Q ∪ {A};
  end
  return Q
end
```

Modular Translation from PSL to NBA

```
ModPsl2Ba(φ)
input φ the PSL input formula
output a set Q of NBAs;
        the final NBA is the product of all NBAs in Q
begin
   Q := ∅;
   φ' := Sonf(φ);        /* φ' is in the form Ψ_LTL ∧ Ψ_PSL */
   for ψ ∈ Ψ_LTL do
      A := Ltl2Ba(ψ);
      Q := Q ∪ {A};
   end
   for ψ ∈ Ψ_PSL do
      A := Psl2Ba(ψ);
      Q := Q ∪ {A};
   end
   return Q
end
```

The final NBA is the implicit product of the automata

Modular Translation from PSL to NBA

- For the LTL component we can leverage on highly optimized translations (e.g. `spin`, `ltl2smv`, . . .).
- For the SONF conponent we can leverage on the standard symbolic conversion (SMH).

Modular Translation from PSL to NBA

- For the LTL component we can leverage on highly optimized translations (e.g. `spin`, `ltl2smv`, ...).
- For the SONF conponent we can leverage on the standard symbolic conversion (SMH).

Question

Can we rely on the fact that SONF formulae have a fixed structure and come up with an optimized symbolic encoding?

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \mapsto P_F))$

Standard encoding:

- Build an explicit A_r.
- Complete and determinize A_r and negate it.
- Build the whole ABA automaton by combining previous result with the other operators ($\mathbf{G}$, $\rightarrow$).
- Remove alternation using SMH.

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \mapsto P_F))$

Optimized encoding:

- Build an explicit A_r.
- Build a symbolic completed and deterministic version of A_r.
- Use it to obtain a symbolic version of A_Φ

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \Diamond\!\!\rightarrow P_F))$

Standard encoding:

- Build an explicit A_r.
- Build the whole ABA automaton.
- Remove alternation using symbolic version of MH.

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \diamondsuit\!\!\rightarrow P_F))$

Optimized encoding:

- Build an explicit A_r.
- Build directly the symbolic version of A_Φ without explicitly building the ABA for the whole formula by adapting the symbolic MH encoding to efficiently encode the formula.

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \star\!\!\rightarrow P_F))$

Theorem

$\mathcal{L}_{\mathcal{A}}(A_\phi) = \mathcal{L}_{\mathcal{A}}(S_\phi)$

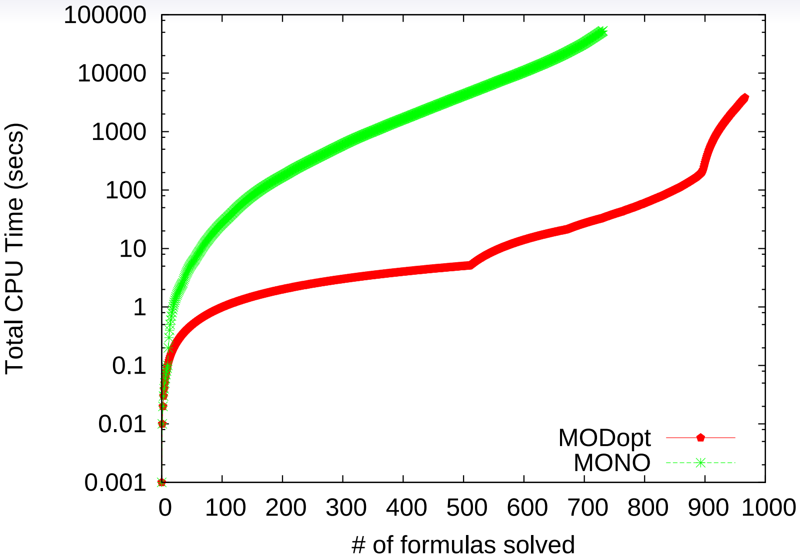
Outline

- Motivation
- Technical Background
- Monolithic Encoding of PSL into NBA
- Modular encoding of PSL into NBA
  - Suffix Operator Normal Form for PSL
  - Modular Translation into NBA
  - Optimized encoding
- **Experimental Evaluation**
- Conclusions and Future Work

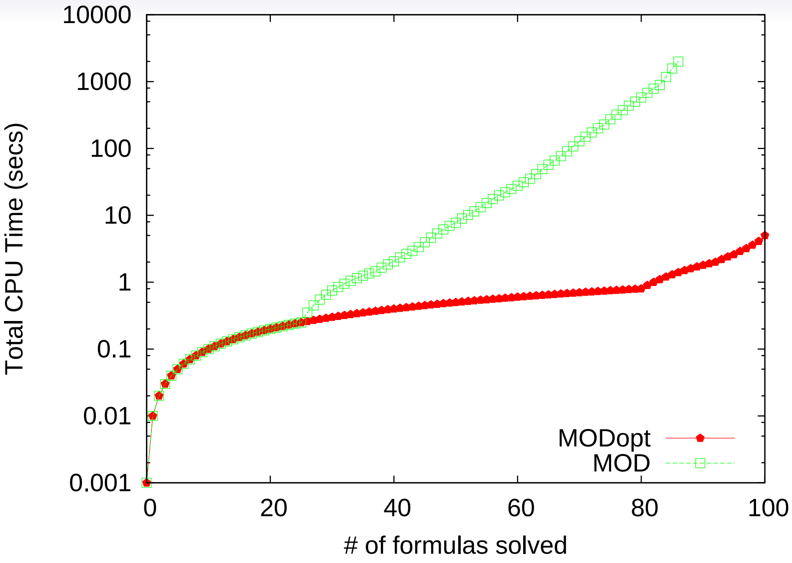
Experimental Evaluation (EE)
Setup

- We implemented the described approach in the NuSMV model checker.
- We compared the monolithic approach (MONO) against the new modular approach (MODopt).
  - Random properties based on patterns coming from industry.
  - Time for symbolic automaton generation.
  - Fair cycle detection (language emptyness) for satisfiability.
  - Fair cycle detection for model checking.
  - Both BDD and SAT based.
    - BDD based Emerson Lei algorithm for language emptyness [EL81].
    - SAT based Simple Bounded Model Checking with induction [HJL05].

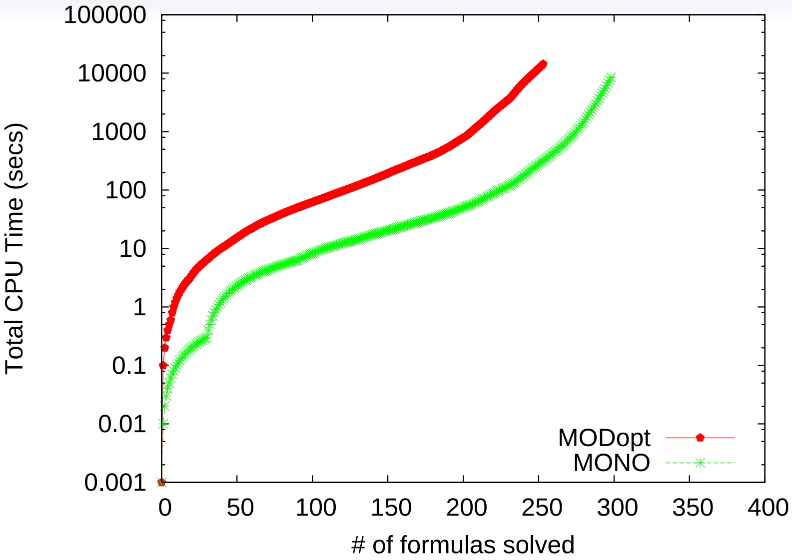
EE: NBA Building time MONO vs MODopt



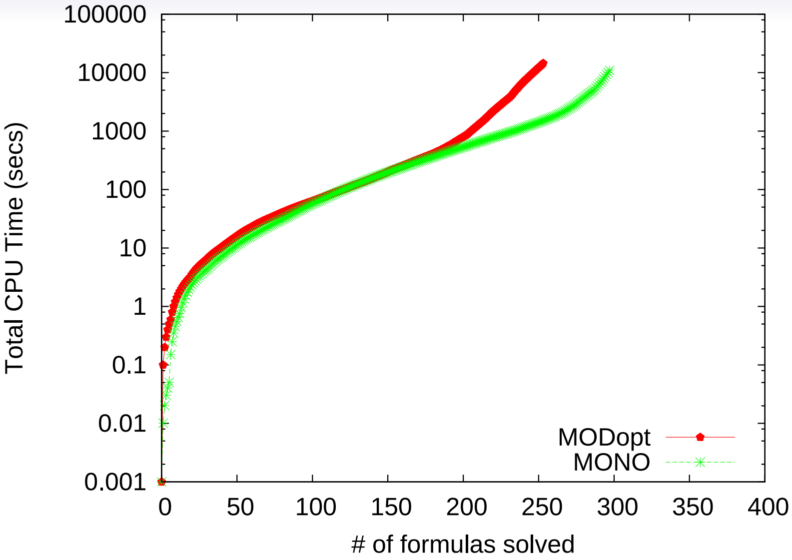
EE: NBA Building time MOD vs MODopt



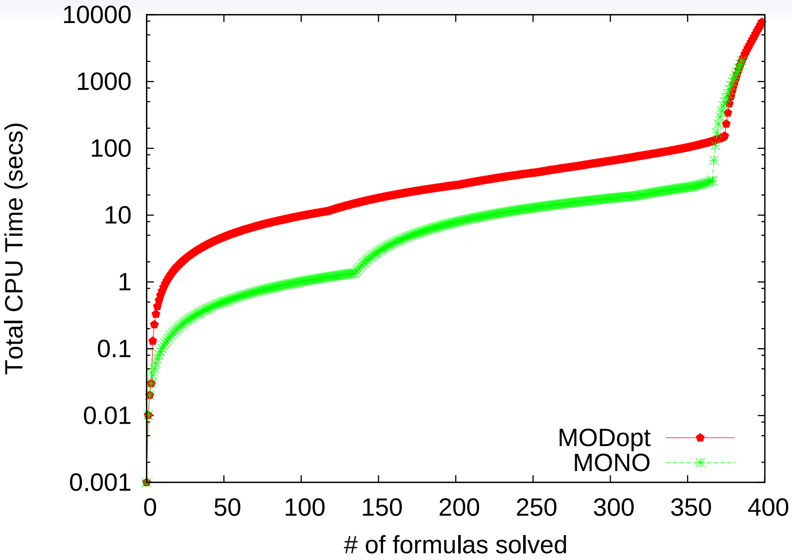
EE: Search Time BDD based language emptyness



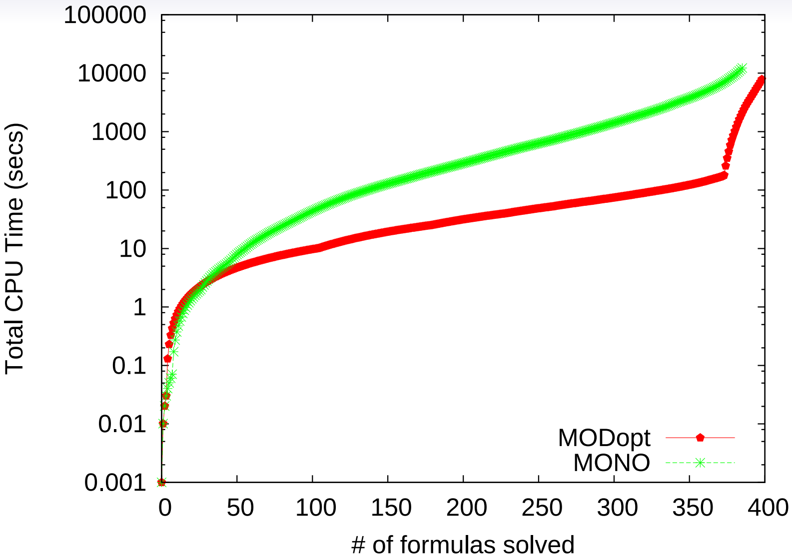
EE: Total Time BDD based language emptyness



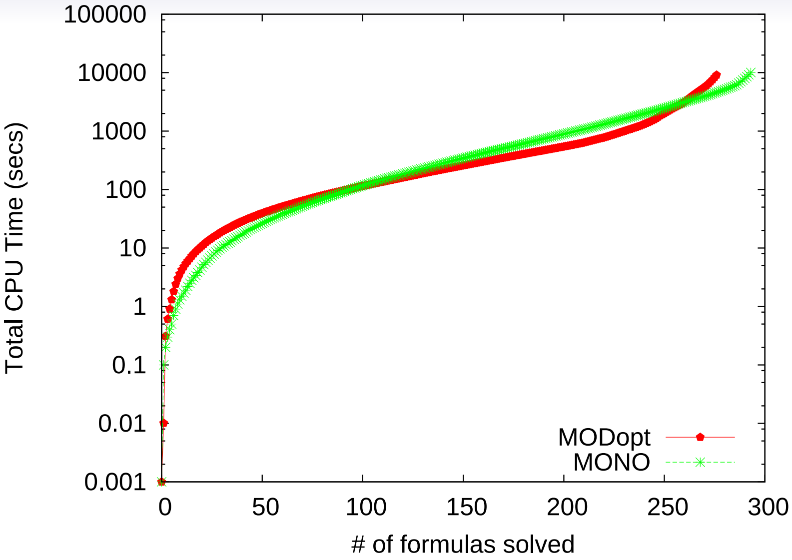
EE: Search Time SBMC based language emptyness



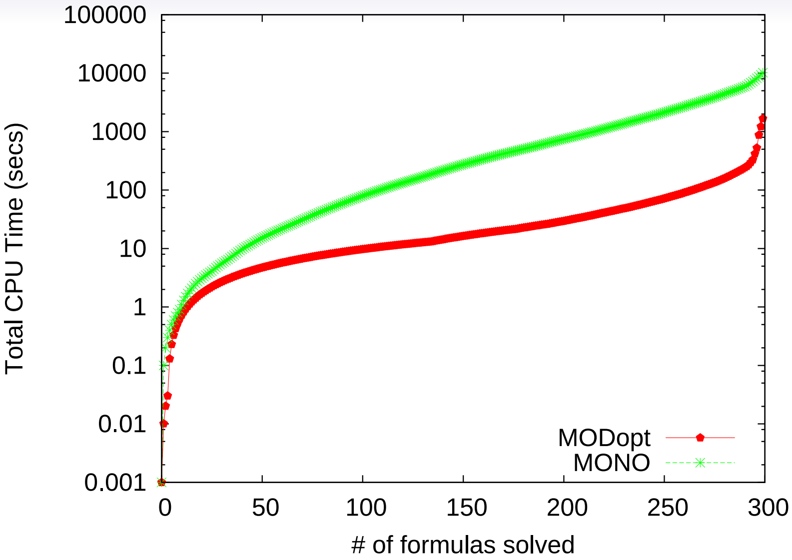
EE: Total Time SBMC based language emptyness



EE: Total Time BDD based model checking



EE: Total Time SBMC based model checking



Outline

- Motivation
- Technical Background
- Monolithic Encoding of PSL into NBA
- Modular encoding of PSL into NBA
  - Suffix Operator Normal Form for PSL
  - Modular Translation into NBA
  - Optimized encoding
- Experimental Evaluation
- Conclusions and Future Work

Conclusions

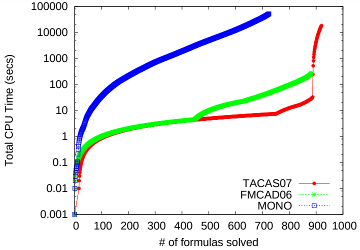
- We presented a new algorithm for the conversion of PSL into a symbolically represented NBA.
  - The approach is based on the decomposition of the PSL specification into a normal form that separates out the LTL part and the SERE part.
  - The various components can be independently symbolically encoded, and they are implicitly conjuncted.
  - Additional optimizations defined by exploiting the specific structure of subformulas involving suffix operators.
- We proved the approach correct.
- We run a thorough experimental evaluation.
  - The new approach consumes less resources than the monolithic encoding.
  - This enables the verification of properties that were previously out of reach.

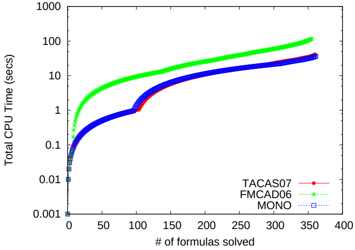
Future Work

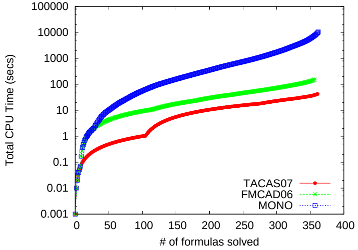
- The main drawback is that generated automata have a redundant structure, which may result in degraded performance.
  - In a new paper submitted to TACAS'07 we propose a new syntactic approach, that by means of rewriting rules, when applied to the SONF-based method result
    - in more compact NBA and then in much faster verification;
    - in a slight improvement in the construction time.
- In the future we work on ways to mitigate this problem along different directions:
  - Use the structure of the automata to devise a better BDD variable ordering.
  - Investigate the application of the reduction of liveness to safety [ShuppanBiere05].
  - Investigate the use of reduction techniques, which may result of smaller automata, thus possibly resulting in a reduction of search time.

Future Work

New results







Questions?

Related Work

- Pnueli's temporal testers for PSL [PZ'06]
  - Finite-state machine that monitors if the suffix of the processed word satisfies the formula.
  - The translation is bottom-up and compositional: each subformula is translated into an automaton and a symbolic variable is used to monitor its satisfiability.
  - We do not build an automaton for every subformula, but we simply separate the LTL part from the SERE part and we leave the freedom to use different translation for each part.
  - We use different optimized compilation for the suffix conjunction and the suffix implication.

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \mapsto P_F))$

1. Build the completed deterministic version of $A_r = \langle \mathcal{A}, Q, q_0, \rho, F \rangle$
   - $V := \{v_q\}_{q \in Q}$
   - $I_r := v_{q_0}$
   - $T_r := \bigwedge_{q \in Q} (v_q \rightarrow (\bigvee_{C \subseteq \rho(q)} (\bigwedge_{(a,q') \in C} (a \wedge v'_{q'}) \wedge \bigwedge_{(a,q') \in \rho(q) \setminus C} \neg a)))$
   - $F_r := \bigvee_{q \in F} v_q$

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \mapsto P_F))$

1. Build the completed deterministic version of $A_r = \langle \mathcal{A}, Q, q_0, \rho, F \rangle$
   - $V := \{v_q\}_{q \in Q}$
   - $I_r := v_{q_0}$
   - $T_r := \bigwedge_{q \in Q} (v_q \rightarrow (\bigvee_{C \subseteq \rho(q)} (\bigwedge_{(a,q') \in C} (a \wedge v'_{q'}) \wedge \bigwedge_{(a,q') \in \rho(q) \setminus C} \neg a)))$
   - $F_r := \bigvee_{q \in F} v_q$
2. Build the FTS $S_\phi = \langle V_\phi, \mathcal{A}, T_\phi, I_\phi, F_\phi \rangle$
   - $V_\phi = V$
   - $I_\phi := \top$
   - $T_\phi := P_I \rightarrow I_r \wedge T_r[v'_q \wedge P_F / v'_q]_{q \in F}$
   - $F_\phi := \top$

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \Diamond\!\!\rightarrow P_F))$

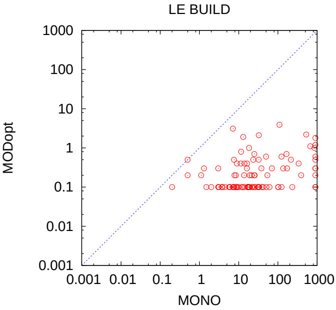
1. Build $A_r = \langle \mathcal{A}, Q, q_0, \rho, F \rangle$, then
   - $V := \{v_q\}_{q \in Q}$
   - $I_r := v_{q_0}$
   - $T_r := \bigwedge_{q \in Q} (v_q \rightarrow (\bigvee_{(q,a,q') \in \rho} (a \wedge v'_{q'})))$
   - $F_r := \bigvee_{q \in F} v_q$

Optimized encoding of $\phi := \mathbf{G}\,(P_I \rightarrow (r \Diamond\!\!\rightarrow P_F))$

1. Build $A_r = \langle \mathcal{A}, Q, q_0, \rho, F \rangle$, then
   - $V := \{v_q\}_{q \in Q}$
   - $I_r := v_{q_0}$
   - $T_r := \bigwedge_{q \in Q} (v_q \rightarrow (\bigvee_{(q,a,q') \in \rho} (a \wedge v'_{q'})))$
   - $F_r := \bigvee_{q \in F} v_q$
2. Build the FTS $S_\phi = \langle V_\phi, \mathcal{A}, T_\phi, I_\phi, F_\phi \rangle$
   - $V_\phi = V_L \cup V_R \qquad V_L := \{v_{qL}\}_{v_q \in V},\ V_R := \{v_{qR}\}_{v_q \in V}$
   - $I_\phi := \top$
   - $$T_\phi := P_I \rightarrow I_r[v_{qL}/v_q]_{q \in Q} \wedge \\ T_{rL}[v'_{qL} \vee P_F / v'_q]_{q \in F} \wedge \quad T_{rR}[v'_{qR} \vee P_F / v'_q]_{q \in F} \wedge \\ ((\bigwedge_{q \in Q} \neg v_{qR}) \rightarrow (\bigwedge_{q \in Q} (v'_{qL} \rightarrow v'_{qR}))) \wedge \quad \bigwedge_{q \in Q} (v_{qR} \rightarrow v_{qL})$$
   - $F_\phi := \bigwedge_{v_q \in V} \neg v_{qR}$

Experimental evaluation

Scatter plots for NBA encoding



Experimental evaluation

Scatter plots for language emptiness and total time