From PSL to NBA: a Modular Symbolic Encoding

A. Cimatti\textsuperscript{1} M. Roveri\textsuperscript{1} S. Semprini\textsuperscript{1} S. Tonetta\textsuperscript{2}

\textsuperscript{1}ITC-irst Trento, Italy
\{cimatti,roveri\}@itc.it

\textsuperscript{2}University of Lugano, Lugano, Switzerland
tonettas@lu.unisi.ch

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.
  - $\text{LTL} + \text{regular expressions} = \omega$-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:
  
  \[
  r_1 ; r_2 \quad r_1 : r_2 \quad r_1 \mid r_2 \\
  r_1 \& r_2 \quad r_1 \&\& r_2
  \]
Definition (PSL syntax)

- if $p$ is propositional, $p$ is a PSL formula;
- if $\phi_1$ and $\phi_2$ are PSL formulas, then $\neg\phi_1$, $\phi_1 \land \phi_2$, $\phi_1 \lor \phi_2$ are PSL formulas;
- if $\phi_1$ and $\phi_2$ are PSL formulas, then $X \phi_1$, $\phi_1 U \phi_2$, $\phi_1 R \phi_2$ are PSL formulas;
- if $r$ is a SERE and $\phi$ is a PSL formulas, then $r \diamondrightarrow \phi$ and $r \rightharpoonup \phi$ are PSL formulas;
- if $r$ is a SERE, then $r$ is a PSL formula.
The Property Specification Language PSL

Property Specification Language: PSL

- \{a ; b[*] ; c\} \rightarrow \{d ; e\}:
  All sequences matching \{a ; b[*] ; c\} should not be
  followed by a sequence not matching \{d ; e\}.

- \{a ; b[*] ; c\} \&\Rightarrow \{d ; e\}:
  At least one sequence
  matching \{a ; b[*] ; c\} should
  not be followed by a sequence not matching \{d ; e\}.

\{a;b[*];c\} \rightarrow \{d;e\}

\{a;b[*];c\} \rightarrow \{d;e\}

M. Roveri (ITC-irst)
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
Monolithic encoding of PSL

\[ \varphi = \text{always} (\{a ; b[*] ; c\} \mapsto \{d ; e\}) \]
Monolithic encoding of PSL

$\varphi$

$ABA(\varphi)$

$q_0 \rightarrow q_1 \rightarrow q_5 \rightarrow q_6$

$a$, $\neg a$, $b \land \neg c$, $\neg b \land \neg c$, $\neg b \land c \land d$, $b \land c \land d$, $e$, $T$
Monolithic encoding of PSL

\[ \varphi \]

\[ \text{ABA}(\varphi) \]

EMH\[ \text{NBA}(\text{ABA}(\varphi)) \]

\[ \neg(a \land b) \land c \land d \land e \]

\[ \neg(a \land b) \land \neg c \land d \land e \]

\[ \neg(a \land b) \land \neg c \land \neg e \]

\[ \neg a \land \neg b \land c \land d \land e \]

\[ \neg a \land \neg b \land c \land d \land e \]
Monolithic encoding of PSL

\[ \varphi \]
\[ \xrightarrow{\text{ABA}(\varphi)} \]
\[ \xrightarrow{\text{EMH}} \]
\[ \text{NBA}(\text{ABA}(\varphi)) \]
\[ \xrightarrow{\text{SNBA}(\text{NBA}(\text{ABA}(\varphi)))} \]

VAR
\[
\begin{align*}
st : & \{qq0, qq1, qq2, qq3\}; \\
\text{DEFINE} & \\
q0 := & \quad st = qq0; \quad q1 := \quad st = qq1; \\
q2 := & \quad st = qq2; \quad q3 := \quad st = qq3; \\
\text{INIT} q0 \\
\text{TRANS} & \\
q0 \rightarrow & (\(a \& \text{next}(q1)\)) \mid \\
& (\neg a \& \text{next}(q0)) \& \\
q1 \rightarrow & (\neg a \& \neg b \& \neg c \& \text{next}(q0)) \mid \\
& (\neg a \& \neg b \& c \& d \& \text{next}(q3)) \mid \\
& (\neg (a \& b) \& c \& d \& \text{next}(q2)) \mid \\
& (\neg (a \& b) \& \neg c \& \text{next}(q1)) \& \\
\ldots \ldots \ldots \\
\text{FAIRNESS} & (q0 \mid q1 \mid q2 \mid q3)
\end{align*}
\]
Monolithic encoding of PSL

ABA(\(\varphi\))
Monolitic encoding of PSL

\[ \varphi \]

\[ ABA(\varphi) \]

\[ SNBA(NBA(ABA(\varphi))) \]

VAR \( qL0 : \text{boolean; } qL1 : \text{boolean; } qL5 : \text{boolean; } qL6 : \text{boolean; } qL7 : \text{boolean; } \)

TRANS
\[
(qL0 \rightarrow ((a \& \text{next}(qL1)) \mid \\
(b \& !c \& \text{next}(qL1)) \mid \\
(b \& c \& d \& \text{next}(qL5)) \mid \\
(b \& c \& d \& \text{next}(qL1) \& \text{next}(qL5)) \mid \\
(!b \& !c \& \text{next}(qL6))) \}\) &
\[
(qL1 \rightarrow ((b \& !c \& \text{next}(qL1)) \mid \\
(b \& c \& d \& \text{next}(qL1) \& \text{next}(qL5)) \mid \\
(!b \& !c \& \text{next}(qL6))) \}\) &
\[
(qL5 \rightarrow (e \& \text{next}(qL6))) \}\) &
\[
(qL6 \rightarrow (\text{next}(qL6))) \}\) &
\[
(qL7 \rightarrow (\text{next}(qL0) \& \text{next}(qL7))) \}\)

INIT \( qL0 \& qL7; \)

FAIRNESS TRUE
Pros

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

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.
Symbolic Encoding of PSL

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

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

**Definition (NNF)**

\[
\begin{align*}
\mathcal{NNF}(\neg p) & := \neg p \\
\mathcal{NNF}(\neg (\phi_1 \lor \phi_2)) & := \mathcal{NNF}(\neg \phi_1) \land \mathcal{NNF}(\neg \phi_2) \\
\mathcal{NNF}(\neg (\phi_1 \land \phi_2)) & := \mathcal{NNF}(\neg \phi_1) \lor \mathcal{NNF}(\neg \phi_2) \\
\mathcal{NNF}(\neg (\phi_1 \cup \phi_2)) & := \mathcal{NNF}(\neg \phi_1) \mathcal{R} \mathcal{NNF}(\neg \phi_2) \\
\mathcal{NNF}(\neg (\phi_1 \mathcal{R} \phi_2)) & := \mathcal{NNF}(\neg \phi_1) \mathcal{U} \mathcal{NNF}(\neg \phi_2) \\
\mathcal{NNF}(\neg (r \Diamond \phi_1)) & := r \rightarrow \mathcal{NNF}(\neg \phi_1) \\
\mathcal{NNF}(\neg (r \rightarrow \phi_1)) & := r \Diamond \mathcal{NNF}(\neg \phi_1)
\end{align*}
\]
Suffix Operator Normal Form for PSL

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

SONF-ization

For every subformula of $\phi$ of the form $r \diamond \psi$ (resp., $r \rightarrow \psi$), we introduce two new atoms: $P_r \diamond \psi$ (resp., $P_r \rightarrow \psi$) and $P_\psi$

\[
\phi[r \diamond \psi] \Rightarrow \phi[P_r \diamond \psi/r \diamond \psi] \land \\
\mathbf{G} (P_r \diamond \psi \rightarrow (r \diamond \rightarrow P_\psi)) \land \\
\mathbf{G} (P_\psi \rightarrow \psi)
\]

\[
\phi[r \rightarrow \psi] \Rightarrow \phi[P_r \rightarrow \psi/r \rightarrow \psi] \land \\
\mathbf{G} (P_r \rightarrow \psi \rightarrow (r \rightarrow \rightarrow P_\psi)) \land \\
\mathbf{G} (P_\psi \rightarrow \psi)
\]
Suffix Operator Normal Form for PSL

\[ \text{Sonf}(\phi) := \bigwedge_{i} \phi_i \wedge \bigwedge_{j} \psi_{\text{PSL}} \psi_{\text{LTL}} \wedge G(P_j \rightarrow (r_j \leftrightarrow P'_j)) \]
Suffix Operator Normal Form for PSL

\[ \text{Sonf}(\phi) := \bigwedge_i \phi_i \land \bigwedge_j \text{G} (P_j \rightarrow (r_j \leftrightarrow P'_j)) \]

\( \Psi_{LTL} \quad \Psi_{PSL} \)

Suffix Operator Subformula
Suffix Operator Normal Form for PSL

\[ \text{Sonf}(\phi) := \bigwedge_i \phi_i \land \bigwedge_j \Psi_{\text{PSL}} \land G(P_j \rightarrow (r_j \leftrightarrow P_j')) \]

**Theorem**

Let \( \phi \) be a PSL formula over \( A \) and \( \psi \) a PSL subformula of \( \phi \) that occurs only positively in \( \phi \). If

\[ \phi' := \phi[^P/\psi] \land G(P \rightarrow \psi) \]

then \( \mathcal{L}_A(\phi) = \mathcal{L}_A(\phi') \).
Example

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

\[ \text{Sonf}(\varphi) = \textbf{always} (P_0) \land \\
\text{always} (P_0 \rightarrow (\{a; b[*]; c\} \rightarrow P_1)) \land \\
\text{always} (P_1 \rightarrow (\{d; e\} \Diamond \rightarrow \text{True})) \]
ModPsl2Ba(\(\phi\))

input \(\phi\) the PSL input formula

output a set \(Q\) of NBAs;

the final NBA is the product of all NBAs in \(Q\)

begin

\(Q := \emptyset;\)

\(\phi' := \text{Sonf}(\phi);\) /* \(\phi'\) is in the form \(\psi_{\text{LTL}} \land \psi_{\text{PSL}} \) */

for \(\psi \in \Psi_{\text{LTL}}\) do

\(A := \text{Lt12Ba}(\psi);\)

\(Q := Q \cup \{A\};\)

end

for \(\psi \in \Psi_{\text{PSL}}\) do

\(A := \text{Psl2Ba}(\psi);\)

\(Q := Q \cup \{A\};\)

end

return \(Q\)

end
Modular Translation from PSL to NBA

ModPs12Ba(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  The final NBA is the implicit product of the automata

end
For the LTL component we can leverage on highly optimized translations (e.g. \textit{spin}, \textit{ltl2smv}, ...).

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

For the SONF component 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 $\phi := G(P_I \rightarrow (r \mid\rightarrow P_F))$

### 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 ($G$, $\rightarrow$).
- Remove alternation using SMH.
Optimized encoding of $\phi := G (P_I \rightarrow (r \rightarrow P_F))$

- Build an explicit $A_r$.
- Build a symbolic completed and deterministic version of $A_r$.
- Use it to obtain a symbolic version of $A_\phi$.
Optimized encoding of $\phi : = G (P_I \rightarrow (r \diamond \rightarrow P_F))$

### Standard encoding:
- Build an explicit $A_r$.
- Build the whole ABA automaton.
- Remove alternation using symbolic version of MH.

M. Roveri (ITC-irst)
Optimized encoding of $\phi := G \left( P_I \rightarrow (r \diamondrightarrow P_F) \right)$

Optimized encoding:

- Build an explicit $A_r$.
- Build directly the symbolic version of $A_{\phi}$ 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**

$\mathcal{L}_A(A_\phi) = \mathcal{L}_A(S_\phi)$
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 emptiness) for satisfiability.
  - Fair cycle detection for model checking.
  - Both BDD and SAT based.
    - BDD based Emerson Lei algorithm for language emptiness [EL81].
    - SAT based Simple Bounded Model Checking with induction [HJL05].
EE: NBA Building time MONO vs MODopt
EE: NBA Building time MOD vs MODopt

![Graph showing the comparison of total CPU time (in seconds) for MODopt and MOD with the number of formulas solved. The y-axis represents total CPU time in logarithmic scale, ranging from 0.001 to 10000, and the x-axis represents the number of formulas solved, ranging from 0 to 100. The graph compares MODopt (red line) and MOD (green line).]
EE: Search Time BDD based language emptyness

![Graph showing total CPU time vs. number of formulas solved for MODopt and MONO.

- MODopt curve is red.
- MONO curve is green.
- The x-axis represents the number of formulas solved, ranging from 0 to 400.
- The y-axis represents total CPU time in seconds, ranging from 0.0001 to 100000.

Data points indicate the performance of MODopt and MONO approaches across different formula counts.]
EE: Total Time BDD based language emptyness

Total CPU Time (secs)

# of formulas solved

MODopt
MONO

M. Roveri (ITC-irst)

PSL2NBA: a Modular Symbolic Encoding

FMCAD'06 27
EE: Search Time SBMC based language emptiness

![Graph showing Total CPU Time (secs) vs. # of formulas solved for MODopt and MONO]

- MODopt
- MONO

M. Roveri (ITC-irst)  PSL2NBA: a Modular Symbolic Encoding  FMCAD'06  28
EE: Total Time SBMC based language emptiness

Total CPU Time (secs)

# of formulas solved

MODopt

MONO

M. Roveri (ITC-irst)
PSL2NBA: a Modular Symbolic Encoding
FMCAD’06
EE: Total Time BDD based model checking

![Graph showing Total CPU Time (secs) vs. # of formulas solved for MODopt and MONO.](image)

- **MODopt**
- **MONO**

M. Roveri (ITC-irst)  PSL2NBA: a Modular Symbolic Encoding  FMCAD'06 30
EE: Total Time SBMC based model checking

![Graph showing the total CPU time in seconds against the number of formulas solved for MODopt and MONO.

- The x-axis represents the number of formulas solved, ranging from 0 to 300.
- The y-axis shows the total CPU time in seconds, ranging from 0.001 to 100000.
- Two lines are plotted: one for MODopt (red solid line) and one for MONO (green dashed line).

The graph demonstrates how the total CPU time increases as the number of formulas solved increases, with MODopt generally showing a lower CPU time compared to MONO.]

M. Roveri (ITC-irst)
PSL2NBA: a Modular Symbolic Encoding
FMCAD'06
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?
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 $\phi := G (P_I \rightarrow (r \mid\rightarrow P_F))$

1. Build the completed deterministic version of $A_r = \langle A, Q, q_0, \rho, F \rangle$
   - $V := \{v_q\}_{q \in Q}$
   - $I_r := v_{q_0}$
   - $T_r := \bigwedge_{q \in Q} (v_q \rightarrow (\bigvee C \subseteq \rho(q) (a, q') \in C (a \land v_{q'})) \land \bigwedge_{(a, q') \in \rho(q) \setminus C} \neg a))$
   - $F_r := \bigvee_{q \in F} v_q$
Optimized encoding of $\phi := \mathsf{G} \ (P_I \rightarrow (r \mid \rightarrow P_F))$

1. Build the completed deterministic version of $A_r = \langle A, Q, q_0, \rho, F \rangle$
   - $V := \{v_q\}_{q \in Q}$
   - $I_r := v_{q_0}$
   - $T_r := \bigwedge (v_q \rightarrow (\bigvee (\bigwedge (a \land v_{q'}') \land \bigwedge (a, q') \in C) (a, q') \in \rho(q) \setminus C))$
   - $F_r := \bigvee v_q$  
     
2. Build the FTS $S_\phi = \langle V_\phi, A, T_\phi, I_\phi, F_\phi \rangle$
   - $V_\phi = V$
   - $I_\phi := \top$
   - $T_\phi := P_I \rightarrow I_r \land T_r[v_{q'}' \land P_F/v_q']_{q \in F}$
   - $F_\phi := \top$
Optimized encoding of $\phi := G (P_I \rightarrow (r \diamondrightarrow P_F))$ 

1. Build $A_r = \langle A, Q, q_0, \rho, F \rangle$, then

- $V := \{v_q\}_{q \in Q}$
- $I_r := v_{q_0}$
- $T_r := \bigwedge_{q \in Q} (v_q \rightarrow (\bigvee_{(q,a,q') \in \rho} (a \land v_{q'})))$
- $F_r := \bigvee_{q \in F} v_q$
Optimized encoding of $\phi := G (P_I \rightarrow (r \Diamond \rightarrow P_F))$

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

2. Build the FTS $S_\phi = \langle V_\phi, A, T_\phi, I_\phi, F_\phi \rangle$
   - $V_\phi = V_L \cup V_R$
     - $V_L := \{v_q\}_{q \in V}$, $V_R := \{v_q\}_{q \in V}$
   - $I_\phi := \top$
   - $T_\phi := P_I \rightarrow I_r[v_qL/v_q]_{q \in Q} \wedge
dl
   - $T_{rL}[v'_{qL}/v']_{q \in F} \wedge
dl
   - $T_{rR}[v'_{qR}/v']_{q \in F} \wedge
dl
   - ((\bigwedge_{q \in Q} \neg v_qR) \rightarrow (\bigwedge_{q \in Q} (v'_{qL} \rightarrow v'_{qR}))) \wedge \bigwedge_{q \in Q} (v_qR \rightarrow v_qL)$
   - $F_\phi := \bigwedge_{v_q \in V} \neg v_qR$
Experimental evaluation
Scatter plots for NBA encoding
Experimental evaluation
Scatter plots for language emptiness and total time

LE BDD search

LE BDD total

LE SBMC search

LE SBMC total

MODopt
MONO

MODopt
MONO

MODopt
MONO

MODopt
MONO

M. Roveri (ITC-irst)
PSL2NBA: a Modular Symbolic Encoding
FMCAD'06 42