Truly Symbolic Model Checking

Enumerative methods with the explicit-state representation can handle systems with sizes up to $10^{7}$ (i.e., $2^{24}$) states. The situation has greatly improved with the introduction of symbolic model-checking methods which can standardly handle systems with up to $2^{150}$ states.

In addition to efficient representation of boolean assertions by the Ordered Binary Decision Diagrams (OBDD) data structure, symbolic methods call for set-based algorithms in which all the immediate successors of a given set of states can be computed in a single step. In symbolic analysis, we use assertions to represent sets of states. Thus, the boolean assertions encode sets of states.

For finite-state systems, we can always use boolean assertions to encode sets of states. For infinite-state systems, we can always use truly symbolic analysis, we use assertions to represent sets of states. Thus, the boolean assertions encode sets of states.

Truly Symbolic Model Checking

Enumerative methods with the explicit-state representation can handle systems with sizes up to $10^{7}$ (i.e., $2^{24}$) states. The situation has greatly improved with the introduction of symbolic model-checking methods which can standardly handle systems with up to $2^{150}$ states. In addition to efficient representation of boolean assertions by the Ordered Binary Decision Diagrams (OBDD) data structure, symbolic methods call for set-based algorithms in which all the immediate successors of a given set of states can be computed in a single step. In symbolic analysis, we use assertions to represent sets of states. Thus, the boolean assertions encode sets of states.

For finite-state systems, we can always use boolean assertions to encode sets of states. For infinite-state systems, we can always use truly symbolic analysis, we use assertions to represent sets of states. Thus, the boolean assertions encode sets of states.
Notations

Definethe predecessor predicate transformer:

\[
\ldots \land (\phi \diamond d) \diamond d \land (\phi \diamond d) \diamond d \land \phi \diamond d \land \phi
\]

\[
= \phi \diamond d \land \phi
\]

\[
\phi \diamond d \Rightarrow s
\]

Obviously, for an assertion, \( \phi \diamond d \Rightarrow s \).

\[
(\_ \Lambda \phi) \lor (\_ \Lambda, \Lambda) d : \_ \Lambda = \phi \diamond d
\]

Define the predecessor predicate transformer:

\[
\text{Notations}
\]
verify parameterized systems. We will use the adequate language of regular expressions (equivalently, WSTS) to

• The predecessor transformer $p$. This usually requires quantifier elimination.

• Equivalence between $I$-formulas.

• Negation, conjunction, and disjunction of $I$-formulas.

An assertional language (logic) $I$ is called adequate for symbolic model checking if it possesses algorithms for performing all the operations required in the algorithm above. Namely, we must have algorithms for computing:

$\text{Check } = ?\phi \lor \Theta$

$\text{until } (0 \neq \Theta \lor ?\phi) \land I^{+} ?\phi = ?\phi$

$(?\phi \diamond d) \land ?\phi =: I^{+} ?\phi$

for $0, I$, do

$d_0 =: 0 ?\phi$

as:

The symbolic model checking algorithm for verifying $\square$ can be presented:

Languages Adequate for Symbolic Model Checking

A. Pnueli

Advanced Topics in Reactive Verification, NYU, Spring, 2004
A Parameterized System consists of a family of systems $(N)S$ for every natural $N > 0$.

A Parameterized System of Parameterized Systems

Goal: Uniform Verification

The System $(N)S$ may consist of finite-state processes regularly connected by synchronous channels.

Uniform Verification

$\phi = (N)S NA$
Regular Model Checking

Application of symbolic model checking, where regular expressions are used as the (symbolic) assertional language for describing sets of states.

Local state: a letter in $\Sigma$
Global state: a word in $\Sigma^+$
Initial Condition: a regular expression over $\Sigma$
Transitions: Given by rewrite rules over $\Sigma$
or by a regular expression over $\Sigma \times \Sigma$

**TOKEN-STRING EXAMPLE**

Local state: 0 / 1 i.e. without / with a token
Global state: e.g. 0100
Initial Condition: $10^*$
Transitions: $X \ 1 \ 0 \ Y \rightarrow X \ 0 \ 1 \ Y$

\[
\left( \left[ \begin{array}{c} 0 \\ 0' \end{array} \right] + \left[ \begin{array}{c} 1 \\ 1' \end{array} \right] \right)^* \left[ \begin{array}{c} 1 \\ 0' \end{array} \right] \left[ \begin{array}{c} 0 \\ 1' \end{array} \right] \left( \left[ \begin{array}{c} 0 \\ 0' \end{array} \right] + \left[ \begin{array}{c} 1 \\ 1' \end{array} \right] \right)^* 
\]
Lecture 9: Regular Model Checking

Performing Model Checking

- regular expression representing the transition relation
- regular expression representing a property
- regular expression representing the initial condition

Verifying an invariant of the system:

\[ \emptyset = \Theta \cup (\phi \diamond \ast \cdot d) \iff \phi = (N)SNA \]

\[ \emptyset = \phi \cup (\ast \cdot d \diamond \Theta) \iff \phi = (N)SNA \]

Verifying an invariant of the system:

\[ \cdots + ((\phi \diamond d) \diamond d) \diamond d + (\phi \diamond d) \diamond d + \phi \diamond d + \phi = \phi \diamond \ast \cdot d \]

Backward exploration

\[ \cdots + d \diamond (d \diamond (d \diamond \phi)) + d \diamond (d \diamond \phi) + d \diamond \phi + \phi = \ast \cdot d \diamond \phi \]

Forward exploration

-(\Lambda) \phi \cup (\Lambda \cdot \Lambda \cdot d) : \Lambda E

-(\Lambda \cdot \Lambda \cdot d) \cup (\Lambda \cdot \phi) : \Lambda E \text{predecessor of} \quad \phi \diamond d \quad \phi \diamond d

-\text{successor of} \quad d \diamond \phi \quad \phi \diamond d

Verifying Model Checking
Computation doesn't converge:

\[
\begin{align*}
*0 & + 1\varnothing : d \diamond (d \diamond \Theta) + d \diamond \Theta + \Theta = 2\varnothing \\
*0 & + 0\varnothing : d \diamond \Theta + \Theta = 1\varnothing \\
*1 & : \Theta = 0\varnothing
\end{align*}
\]

Forward exploration:

Computation terminates in one iteration step:

\[
\begin{align*}
* (I + 0) & I * (I + 0) I * (I + 0) + *0 : \varnothing \diamond d + \varnothing = 1\varnothing \\
* (I + 0) & I * (I + 0) I * (I + 0) + *0 : \varnothing = 0\varnothing
\end{align*}
\]

Backward exploration:

\[
\begin{align*}
\lambda I 0 X \leftarrow \lambda 0 I X & \varnothing \\
\lambda I 0 X \leftarrow \lambda 0 I X & d \\
\lambda I 0 X \leftarrow \lambda 0 I X & \Theta
\end{align*}
\]

Property Transition Initial Condition

TOKEN-STRING EXAMPLE
Performance Model Checking -
A Richer Example
Regular Model Checking

A. Pnueli

Representation as a Regular FDS

Global state (configuration):

Local state:

Initial condition

Given by rewrite rules:

Transitions

A word in \( \mathbb{Z}^+ \):

\{N, L, C, N, L, C\}:

Unary:

\begin{align*}
\Delta N & \leftarrow \Delta N \\
\Delta N & \leftarrow \Delta N
\end{align*}

Binary:

\begin{align*}
\Delta N & \leftarrow \Delta C \\
\Delta C & \leftarrow \Delta L \\
\Delta L & \leftarrow \Delta N \\
\Delta C & \leftarrow \Delta L \\
\Delta L & \leftarrow \Delta N
\end{align*}
The property \( \square \text{Mutual-Exclusion} \equiv \text{PROC-ARRAY} \) can be checked using the following steps:

1. \( \emptyset = \emptyset \cup \Theta \)
2. \( \emptyset \cup \Theta \)
3. \( \emptyset \cup \emptyset \)
4. \( \emptyset \cup \emptyset \)
5. \( \emptyset \cup \emptyset \)
6. \( \emptyset \cup \emptyset \)
7. \( \emptyset \cup \emptyset \)

Iterate:

The property \( \square \text{Mutual-Exclusion} \equiv \text{PROC-ARRAY} \) can be checked using the following steps:
Acceleration

The Problem: [ABJN99]

The transition relation describes one action of a single process.

The Remedy:

Acceleration

Accelarations of Unary Transitions

Local Acceleration – a single process is allowed to take several actions in the same step.

Global Acceleration – each of a set of processes takes a single action.

Acceleration of Binary Transitions

Binary Acceleration – a subset of (not necessarily disjoint) pairs of contiguous processes take a joint action.

Acceleration of Binary Transitions

• In the same step.

Forming an accelerated transition which allows many processes to take an action.

A. Pnueli

Advanced Topics in Reactive Verification, NYU, Spring, 2004
There are three languages of equal expressive power, which are relevant to regular model checking.

The Relevant Languages

- Regular Expressions
- Regular Model Checking
- WSTS
- TLV
- DFA
Regular Expressions

We have seen how regular expressions can specify the initial condition and the property. How about the transition relation?

The transition relation for TOKEN-RING can be expressed as

\[
\begin{align*}
\text{\texttt{\{tCtC\} + \{tLL\} + \{tNNN\} + \{tCtC\} + \{tLL\} + \{tNN\} = p_i}}
\end{align*}
\]

where

\[
\begin{align*}
\text{\texttt{\ast p_i( (\{tLL\}\{tNNN\}) + (\{tNNN\}\{tNNN\}) )\ast p_i +}}
\end{align*}
\]

\[
\begin{align*}
\text{\texttt{\ast p_i( \{tNNNN\} + \{tCtC\} + \{tLLN\} + \{tNN\} + \{tLN\} )\ast p_i}}
\end{align*}
\]

The transition relation for TOKEN-RING can be expressed as

**Regular Expressions**
Computing a Successor

Given a RE describing a state-set and a RE describing a transition relation, we can compute the $d$-successor of by $\phi$. For example, taking $\phi = \{1,0\}, \pi = \emptyset$ and $I_0 = \emptyset$, we compute

$$(d \cup \phi : \{0\}) \cup \pi \cup I_0 \cup \pi \cup I_0 \cup \pi = d$$

and

$$(d \cup \phi : \emptyset) \cup \pi \cup I_0 \cup \pi \cup I_0 \cup \pi = d$$

Given a RE describing a state-set, $\phi$, and $d$-successing a transition relation, computing a Successor
A. Pnueli

The Logic

We use the logic WS1S (weak second order logic of one successor) [Buc60],

\[ \text{The Logic WS1S} \]
Translating WSTS into Regular Expressions (DFA)

1. Encode every \( \exists^2 \)-array boolean array by \( \left\lfloor \log_2 \right\rfloor \) boolean array \( B, \ldots, B \).

2. Represent every position variable by a boolean array \( p \) satisfying the constraint
\[ \forall d \in \mathbb{D}_p \]

| \[ d \] \[ d \] \[ d \] | \[ d \] \[ d \] | \[ d \] \[ d \] \[ d \] \[ d \] \[ d \] | \[ d \] \[ d \] \[ d \] \[ d \] \[ d \] \[ d \] \[ d \] \[ d \] \[ d \] |
|---|---|---|
| \[ \bigcirc d \] \[ \bigcirc d \] \[ \bigcirc d \] \[ \bigcirc d \] \[ \bigcirc d \] | \[ d \] > \[ b \] | \[ d \] > \[ b \] | \[ d \] > \[ b \] | \[ d \] > \[ b \] | \[ d \] > \[ b \] | \[ d \] > \[ b \] | \[ d \] > \[ b \] |

3. Translational Quantification
\[ \forall d \in \mathbb{D}_p \]

Atomic formulas are translated as follows:
\[ \bigcirc \left( B \right) \left( d \right) \rightleftharpoons \bigcirc \left( B \right) \left( d \right) \]

For example:
\[ \left[ [0 \leftarrow B] \phi \land [1 \leftarrow B] \phi \right] \]

is translated into
\[ \left[ d \right] B \cdot d \in E \]
Using \textbf{WS}$_1$S, we can express the components of \textsc{Token-Ring} as follows:

\begin{align*}
\langle \text{state} = \text{record of} \{ N, T, C \} : \text{tok} \rangle \quad \text{II} & \\
\theta & \\
\end{align*}

\textbf{The State Variables:} We define the type \texttt{state}.

\textbf{The Initial Condition:} The initial condition can be given by

\begin{align*}
\exists \theta : \text{I} \neq \theta & \\
\exists \theta : (N = \theta) & \\
\end{align*}

\textbf{The State Variables:} We define the type \texttt{state}.

\textbf{Using \textbf{WS}$_1$S, we can express the components of \textsc{Token-Ring} as follows:}

\begin{align*}
\forall \theta & \\
\end{align*}
Expressing The Transition Relation

Finally,

\[ ((\exists \text{ seq}) \cdot X \cdot X)^{\exists d} \land ((\exists \text{ seq}) \cdot X \cdot X)^{\exists d} : \exists \text{ seq} \land \text{ idle} = (\exists \text{ seq}) \cdot X \cdot X^{\exists d} \]

Expressed as:

\[ [\exists \text{ seq}] X = [\exists \text{ seq}] X : ([\exists \text{ seq}] X)^{\text{pres} \text{ idle}} \]

The transition relation can be formed as the disjunction of three elementary transitions. Using the abbreviation \( \text{idle} \), these can be expanded as follows:

\[
\begin{align*}
([I \oplus?] \cdot \text{seq}) & \lor ([I \oplus?] \cdot \text{seq}) \\
& \lor ([I \oplus?] \cdot \text{seq}) \\
& \lor ([I \oplus?] \cdot \text{seq}) \\
& \lor ([I \oplus?] \cdot \text{seq}) \\
& \lor ([I \oplus?] \cdot \text{seq}) \\
& \lor ([I \oplus?] \cdot \text{seq}) \\
& \lor ((f) X)^{\text{seq} \text{ idle}} : \exists \text{ seq} : (\exists \text{ seq}) \cdot X \cdot X^{\exists d}
\end{align*}
\]
Local Acceleration

In this mode of acceleration, we allow several actions to be taken in succession by the same process. Given a unary transition relation \( [\mathcal{P}] \), we can compute its locally accelerated version by the repeated composition process.

For example, applying local acceleration to the unary transition relation

\[
\left( ([\mathcal{P}] \mathcal{X} \mathcal{A}, \mathcal{X})^q \mathcal{d} \land ([\mathcal{P}], \mathcal{X})^p \mathcal{d} : \Lambda \mathcal{E} \right) = ([\mathcal{P}], \mathcal{X})^{q \odot p} \mathcal{d}
\]

where the composition \( \odot \) is defined by

\[
\ldots \land \mathcal{I} \mathcal{d} \odot (\mathcal{I} \mathcal{d} \odot (\mathcal{I} \mathcal{d} \odot \mathcal{I} \mathcal{d})) \land \mathcal{I} \mathcal{d} \odot (\mathcal{I} \mathcal{d} \odot \mathcal{I} \mathcal{d}) \land \mathcal{I} \mathcal{d} \odot \mathcal{I} \mathcal{d} \land \mathcal{I} \mathcal{d} = \mathcal{I} \mathcal{d}
\]

compute its locally accelerated version by the repeated composition process. Given a unary transition relation \( ([\mathcal{P}], \mathcal{X}) \mathcal{I} \mathcal{d} \) by the same process, we can allow several actions to be taken in succession in this mode of acceleration.
Next, we consider the acceleration of a unary transition on which each of a set of processes takes a single action. Assume as before that the unary transition relation of process is given by \([\bar{i}] \mathcal{F} \mathcal{G} \). The following acceleration formula uses the boolean array \( \text{state-array variables and } \Lambda \).

### Global Acceleration of Unary Transitions

\[
\begin{align*}
(1 &= [\bar{i}] \mathcal{H} \mathcal{O} \mathcal{O} = [\bar{i}] \mathcal{H} \mathcal{O} \mathcal{O}) \lor \\
\left( \{ \mathcal{C}, \mathcal{L}, \mathcal{N} \} \ni [\bar{i}] \Pi \right) \lor (\{ \mathcal{C}, \mathcal{L}, \mathcal{N} \} \ni [\bar{i}] \Pi) \land \\
(0 &= [\bar{i}] \mathcal{H} \mathcal{O} \mathcal{O} = [\bar{i}] \mathcal{H} \mathcal{O} \mathcal{O}) \lor \\
(\{ \mathcal{L}, \mathcal{N} \} \ni [\bar{i}] \Pi) \lor (\{ \mathcal{C}, \mathcal{L}, \mathcal{N} \} \ni [\bar{i}] \Pi)
\right) \\
\land (\mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A}) \\
= (\mathcal{F} \mathcal{X}, X)_{\mathcal{I} \mathcal{d}}
\end{align*}
\]

Applied to program \text{TOKEN-RING}, this yields

\[
\begin{align*}
\left( \begin{array}{c}
\text{if } \mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A} \\
\text{then } i \geq \mathcal{F} \\
\text{else } (\mathcal{F} \mathcal{X}, X)
\end{array} \right) \\
\left( \begin{array}{c}
\text{if } \mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A} \\
\text{then } i > \mathcal{F} \\
\text{else } (\mathcal{F} \mathcal{X}, X)
\end{array} \right)
\end{align*}
\]

\[
\begin{align*}
\left( \begin{array}{c}
\mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A} \\
\mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A}
\end{array} \right) \\
\left( \begin{array}{c}
\mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A} \\
\mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A}
\end{array} \right)
\end{align*}
\]

\[
\begin{align*}
\left( \mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A} \\
\mathcal{F} \mathcal{X} \mathcal{E} \mathcal{P} \mathcal{R} \mathcal{E} \mathcal{S} : \mathcal{F} \mathcal{A}
\end{array} \right)
\end{align*}
\]
Finally, consider the acceleration of a binary transition, such as $\Delta X' = d$, previously presented for program $\text{TOKEN-RING}$.

Acceleration of Binary Transitions

The sequence of intermediate local states for these processes must accommodate this phenomenon. We employ an additional state-array \( \mathcal{Y} \) to save the state twice. For example, once on receiving and then on sending the token. To accommodate this phenomenon, we employ an additional state-array \( \mathcal{Y} \) to save these sequence of intermediate local states for these processes.

In the case of binary acceleration, some processes may change their local state twice.
The formula for the global acceleration is given by

\[ \langle X, X \rangle^\omega d \]
Computing the forward fix-point it converges in 1 step yielding \( \emptyset \cdot 10 \) as the set of reachable states. For **TOKEN-RING**, it took 4 steps to converge.

<table>
<thead>
<tr>
<th>( X )</th>
<th>( M )</th>
<th>( X )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>I</td>
<td>( \emptyset \cdot 10 )</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>( I + )</td>
<td>( m \cdot )</td>
</tr>
<tr>
<td>( \emptyset \cdot )</td>
<td>( I + )</td>
<td>( ? \cdot )</td>
</tr>
<tr>
<td>( I \cdot )</td>
<td>( ? \cdot )</td>
<td>( \emptyset \cdot )</td>
</tr>
</tbody>
</table>

Applying the forward fix-point yields:

\[
*p_0 \cdot [10] + \cdot [00] [10] \cdot p_1 = (p_0 X X)_0^d
\]

As follows:

**TOKEN-STRING**

**Application to TOKEN-STRING**
Model Checking Liveness Properties

The difficulty in verifying liveness properties of parameterized systems is the need to take into account an unbounded number of fairness assumptions, several for each process, and that these requirements are also parameterized.

For program TOKEN-RING, for example, the justice requirements are given by:

\[
\begin{align*}
&\forall i.\forall L,N \in [I \oplus \mathbb{I}] \forall \mathcal{O} \in \mathcal{I} \exists F \\
&\left( \exists i.\forall L,N \in [I \oplus \mathbb{I}] \exists \mathcal{O} \in \mathcal{I} \right) \\
&\left( \forall i.\forall L,N \in [I \oplus \mathbb{I}] \right)
\end{align*}
\]

Each \( i \) \in \{1, \ldots, n\} and

\[
\left( \exists i.\forall L,N \in [I \oplus \mathbb{I}] \exists \mathcal{O} \in \mathcal{I} \right)
\]
The core of the algorithm for verifying the general liveness property

Coping with Liveness
A sound and complete algorithm but may easily lead to state (and BDD) explosion.

- States on a fair pending cycle:

\[ ([\exists \phi : \neg E]) =: \exists \phi \]

- States having, for each \( \exists \phi \) a cycle visiting \( \exists \phi \) states:

\[ (((\exists \phi : \neg \mathcal{A}) \lor \text{pend} \lor ([\exists \phi : \neg E]) \lor \mathcal{A}) \lor \exists \phi =: \exists \phi \]

- States on a non-empty cycle:

\[ X = \bigcap \exists \phi \quad \forall \mathcal{A}, \exists \phi, \text{pend} =: \exists \phi \]

- States at which restricting to pending steps preserving

\[ ([\exists \phi : \neg E] =: \exists \phi \lor \exists \phi) \lor \exists \phi =: \exists \phi \]

- States on a fair pending cycle:

\[ (\exists \phi : \neg E) \lor \exists \phi \lor \exists \phi \lor (\exists \phi : \neg \mathcal{A}) \lor \exists \phi =: \exists \phi \]

Given \( (X, X)^d \) a fair \((\text{pend}, \exists\phi)\)-cycle:
PseudoCycles

Let $\forall \exists \forall$. We say that $\forall \exists \forall$ induces a PseudoCycles.

Cycles relative to $\exists$, then there exists no fair $\exists$-cycle.

Obviously, if we established that, for every $\exists \forall \exists$, there exists no fair $\exists$-pseudo-cycle.

However, coarser covers and partitions are more efficient.

For example, the cover $\forall \exists \forall$ partitions the state space into sets, each containing a single state. Therefore, an $\forall \exists$-pseudo-cycle is a true cycle.

For some $\exists \forall \exists$, for some $\forall \exists \forall$-state $\exists$, to a (possibly different) $\forall \exists \forall$-pseudo-cycle, is a path leading from an $\forall \exists \forall$-state $\forall$ to a $\forall \exists \forall$-state $\forall$.

Let $\forall \exists \forall$ be an assertion parameterized by $\forall \exists \forall$. We say that $\forall \exists \forall$
Given the following algorithm computes the set of states participating in a fair pseudo-cycle relative to a cover $(\bar{b} \land d \lor \bar{b}) \diamond (\bar{b} \land d \lor (d \land \Theta)) =: \text{pend}$.

Given $(X, X)^d$.

### Computing Bad Pseudo Cycles

Computing bad pseudo cycles
Liveness Using Pseudo-Cycles -

Verification of the liveness property accessibility:

\[ [W \in I] \subseteq I \quad \text{where} \quad [I]_{\forall} = (I) \bar{E} \]

The partition used in the pseudo-cycle algorithm was terminated in 9.2 seconds and 32 iterations.

- Pseudo-cycles algorithm —
  
  terminated in 9.2 seconds and 32 iterations.

- Exact cycles algorithm —
  
  terminated in 53 seconds and 40 iterations.

\( (C = [I]_{\forall}) \diamond \iff (N = [I]_{\forall}) \): Verifiability of the liveness property accessibility:

**TOKEN-RING**

- Liveness Using Pseudo Cycles -

A. Pnueli
Verification of the liveness property communal accessibility:

\[ ([\ell]_{1}^{3} \land [\ell]_{1}^{3}) \land (\neg \ell_{1} \land [\ell]_{1}^{3}) \land (\neg \ell_{1} \land [\ell]_{1}^{3}) \land (\neg \ell_{1} \land [\ell]_{1}^{3}) \land (\neg \ell_{1} \land [\ell]_{1}^{3}) \land (\neg \ell_{1} \land [\ell]_{1}^{3}) \]

We prove it by proving six lemmas:

\[ ([\ell]_{1}^{3} : \ell_{1} \land \ell_{1}) \leftarrow \neg \ell_{1} \land \ell_{1} \]

A. Pnueli

Liveness Using Pseudo Cycles -

SZYMANSKI’S ALGORITHM

Liveness Using Pseudo Cycles -

for \( i = 1, \ldots, 6 \)
Consider for example \( \varphi \in \mathrm{AF} \) where \( \varphi \equiv \varphi_5 \). 

The partition used in the pseudo-cycle algorithm was 

\[
\begin{align*}
\exists \xi & = I \left( [w]^{\exists} & \not\in & \mathrm{at} \land [w]^{\exists} & \mathrm{at} \right) & \in & \forall [t]^{\exists} & \mathrm{at} \\
\exists \eta & = I \left[ [w]^{\exists} & \not\in & \mathrm{at} \lor [t]^{\exists} & \mathrm{at} \right) \\
\exists \zeta & = I \left[ [w]^{\exists} & \not\in & \mathrm{at} \lor [t]^{\exists} & \mathrm{at} \right) \\
\end{align*}
\]

terminated in 3025 seconds and 223 iterations.

- Pseudo-cycles algorithm —
- Exact cycles algorithm —
- Computation exploded

The partition used in the pseudo-cycle algorithm was
## Experimental Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token Ring</td>
<td>0.2</td>
</tr>
<tr>
<td>Szymanski</td>
<td>7</td>
</tr>
<tr>
<td>Term. Det.</td>
<td>6</td>
</tr>
<tr>
<td>Din. Phil.</td>
<td>5.6</td>
</tr>
<tr>
<td>Improved Liveness</td>
<td>3</td>
</tr>
<tr>
<td>Safety</td>
<td></td>
</tr>
<tr>
<td>Time Iers.</td>
<td></td>
</tr>
<tr>
<td>Time Iers.</td>
<td></td>
</tr>
<tr>
<td>Time Iers.</td>
<td></td>
</tr>
<tr>
<td>Time Iers.</td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td></td>
</tr>
<tr>
<td>Liveness</td>
<td></td>
</tr>
<tr>
<td>Improved Liveness</td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>Time in seconds</td>
</tr>
<tr>
<td>Token Ring</td>
<td>0.2</td>
</tr>
<tr>
<td>Szymanski</td>
<td>7</td>
</tr>
<tr>
<td>Term. Det.</td>
<td>6</td>
</tr>
<tr>
<td>Din. Phil.</td>
<td>5.6</td>
</tr>
<tr>
<td>Improved Liveness</td>
<td>3</td>
</tr>
<tr>
<td>Safety</td>
<td></td>
</tr>
<tr>
<td>Liveness</td>
<td></td>
</tr>
<tr>
<td>Improved Liveness</td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td></td>
</tr>
</tbody>
</table>

* time in seconds
Conclusions

Liveness properties can be verified by Regular Model Checking for parameterized systems.

WS1S is a convenient language for expressing various acceleration schemes.

Future Research

Finding additional efficient methods (or heuristics) for verifying liveness properties.

Parametrized systems.

Liveness properties can be verified by Regular Model Checking for