Systematic Acceleration in Regular Model Checking

CAV 2007

Bengt Jonsson and Mayank Saksena
{bengt, mayanks}@it.uu.se

Uppsala University

July 4, 2007
Outline

1 Introduction
   - Parameterized Systems
   - Regular Model Checking
   - Related Work

2 Systematic Acceleration
   - Setting
   - Unary Actions
   - Example
   - Compositions
   - Acceleration Procedure

3 Experiments
   - Implementation
   - Results

4 Conclusion
Parameterized Systems

- **Parameterized system:**
  Family of programs $P(n)$

- **Instance:**
  $n$ processes running same code

- **Uniform verification:**
  Does $P(n)$ satisfy property $\phi$ for all $n$?
  - Undecidable [Apt, Kozen]
Verification

How compute (repeatedly) reachable configurations of a parameterized system?

- Verify instances and generalize
- Induction over system structure
- Finite-state abstraction
- **Symbolic model checking** for infinite sets
Symbolic Model Checking

- **Symbolic representation** of infinite sets of configurations
- (Repeatedly) reachable configurations by **fixpoint**
  - Iteratively compute successors...

Making the fixpoint converge:

- **Acceleration**
  - Effect of arbitrarily many transitions
  - No overapproximation, no spurious counterexample
  - Verification of safety and liveness
Regular Model Checking

- Configuration represented as **word** over finite alphabet
- Regular set represents infinite set of configurations
- Transducers represent actions
  - Automaton accepting $\{(w, w') \mid w \xrightarrow{\alpha} w'\}$ for action $\alpha$
- Suitable for uniform verification of parameterized systems
Inspiration

- Verification of Parameterized Networks with Linear or Ring Structure  [Clarke, Grumberg, Jha], [Emerson, Namjoshi]
- “Use symbolic model checking paradigm with appropriate representation for parameterized and infinite-state systems.” [Kesten, Maler, Pnueli, Shahar 97]
- “Regular sets of words widely applicable.” [Boigelot, Wolper 98]
  - Unbounded FIFO channels
  - Pushdown Systems
  - Regular Hardware Structures
  - Integers and reals
Outline

1 Introduction
   - Parameterized Systems
   - Regular Model Checking
   - Related Work

2 Systematic Acceleration
   - Setting
   - Unary Actions
   - Example
   - Compositions
   - Acceleration Procedure

3 Experiments
   - Implementation
   - Results

4 Conclusion
Fixpoints and Acceleration

- Given initial configurations $\mathcal{I}$ and transition relation $\mathcal{R}$
- Compute reachable configurations $\mathcal{I} \circ \mathcal{R}^*$
  - $\mathcal{R}^+$ not computable
  - $C_0 = \mathcal{I}$, $C_{j+1} = C_j \cup C_j \circ \mathcal{R}$ not terminate
- Exist acceleratable actions
  - Transducers $\alpha$ for which $\alpha^+$ computable
- Standard approach for $\mathcal{R} = \bigcup_i \alpha_i$:
  - Accelerate each $\alpha_i$
  - Accelerated fixpoint: $\mathcal{I} \circ (\bigcup_i \alpha_i^+)^*$
    - Iterations $C_{j+1} = C_j \cup C_j \circ \alpha_i^+$
Unary Actions

- Efficiently acceleratable (and “frequent in protocols”)

- Typical example: one process acts in isolation
Burns’ Algorithm for Mutex

1: \( \text{flag}[i] := 0 \)
2: \( \text{if } \exists j < i : \text{flag}[j] = 1 \text{ then goto 1} \)
3: \( \text{flag}[i] := 1 \)
4: \( \text{if } \exists j < i : \text{flag}[j] = 1 \text{ then goto 1} \)
5: \( \text{await } \forall j > i : \text{flag}[j] \neq 1 \)
6: \( \text{flag}[i] := 0 \)
7: \( \text{goto 1} \)

**Figure:** Pseudo code process \( i \) (of \( n \))

- **Sigma** local state of a process: tuple \((pc, flag)\)
- **Initial configurations**: \( \mathcal{I} = [(pc = 1, flag = 0)]^* \)
- **Actions (unary)**: \( \alpha_j \) represents line \( j \)
  - \( \alpha_5 \) is \( ld(\Sigma)^* \cdot [pc : 5 \rightarrow 6] \cdot ld(flag = 0)^* \)
- **Transition relation**: \( \mathcal{R} = \bigcup_j \alpha_j \)
Fixpoint Termination Problem

2: if $\exists j < i : \text{flag}[j] = 1$ then goto 1
3: $\text{flag}[i] := 1$
4: if $\exists j < i : \text{flag}[j] = 1$ then goto 1
5: await $\forall j > i : \text{flag}[j] \neq 1$

- Executing $\alpha_2 - \alpha_5$ blocks higher-indexed processes
- Many processes can go from 2 to 5, only in order
  - (Higher to lower index)
- Not possible with $\cup_i \alpha_i^+$!
  - Arbitrarily many iterations!
- Need “for $i \in I$ do $\alpha_2(i) \circ \alpha_3(i) \circ \alpha_4(i)$”
Fixpoint Termination Solution

- **Before:** manually added compositions as extra actions
  - Typically feasible for expert verifying safety
  - Liveness more difficult
- **Now:** automatically add acceleratable actions derived from original actions
  - Re-establish previous results with “less expertise”
  - New liveness results
    - Some we cannot, even today, verify by manual choice
### Composition

- \( \alpha \circ \alpha' \) is action “\( \alpha \) followed by \( \alpha' \)”
- Composition of unary not unary
- **Keep it unary** (acceleratable)

<table>
<thead>
<tr>
<th></th>
<th>( \phi_L )</th>
<th>( \tau_0 )</th>
<th>( \phi_R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \phi_L )</td>
<td>( \tau_1 )</td>
<td>( \phi_R )</td>
</tr>
<tr>
<td>2</td>
<td>( \phi'_L )</td>
<td>( \tau'_0 )</td>
<td>( \phi'_R )</td>
</tr>
<tr>
<td>3</td>
<td>( \phi'_L )</td>
<td>( \tau'_1 )</td>
<td>( \phi'_R )</td>
</tr>
</tbody>
</table>
Composition

\[
\begin{array}{ccc}
\phi_L & \tau_0 & \phi_R \\
\phi_L & \tau_1 & \phi_R \\
\phi'_L & \tau'_0 & \phi'_R \\
\phi'_L & \tau'_1 & \phi'_R \\
\end{array}
\]
Composition

\[ \phi_L \quad \tau_0 \quad \phi_R \]

\[ \phi''_L \quad \tau'' \quad \phi''_R \]

\[ \phi''_L \quad \tau'' \quad \phi''_R \]

\[ \phi'_L \quad \tau'_1 \quad \phi'_R \]
## Composition

<table>
<thead>
<tr>
<th>( \phi_L \cap \phi'_L )</th>
<th>( \tau_0 )</th>
<th>( \phi_R \cap \phi'_R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi_L \cap \phi'_L )</td>
<td>( \tau'' )</td>
<td>( \phi_R \cap \phi'_R )</td>
</tr>
<tr>
<td>( \phi_L \cap \phi'_L )</td>
<td>( \tau'' )</td>
<td>( \phi_R \cap \phi'_R )</td>
</tr>
<tr>
<td>( \phi_L \cap \phi'_L )</td>
<td>( \tau'_1 )</td>
<td>( \phi_R \cap \phi'_R )</td>
</tr>
</tbody>
</table>
Composition

\[ \phi_L \cap \phi'_L \quad \tau_0 \quad \phi_R \cap \phi'_R \]

\[ \phi_L \cap \phi'_L \quad \tau'_1 \quad \phi_R \cap \phi'_R \]
 Unary composition:

\[
(\phi_L \cdot \tau \cdot \phi_R) \circ_S (\phi'_L \cdot \tau' \cdot \phi'_R) = (\phi_L \cap \phi'_L \cdot \tau \circ \tau' \cdot \phi_R \cap \phi'_R)
\]
Acceleration Goal

- Generate “sufficiently many” acceleratable actions
  - Derived from original actions
- Can generate actions complete in this sense:

For any unary $\alpha_1, \ldots, \alpha_n$ in $\mathcal{R}$ there exists generated $\alpha$ with

$$(\alpha_1 \circ_s \cdots \circ_s \alpha_n)^+ \subseteq \alpha^+$$
Acceleration Procedure

For any unary $\alpha_1, \ldots, \alpha_n$ in $R$ there exists generated $\alpha$ with $(\alpha_1 \circ_s \cdots \circ_s \alpha_n)^+ \subseteq \alpha^+$

- If above holds for $n \leq k$, call $k$ composition depth
- **Intuition** behind construction:
  1. Extract unary $A_0$ from transducer of $R$ so that $R = \bigcup A_0$
  2. Manipulate $A_0$ to obtain $A_1$ with composition depth 1
  3. Combine actions under **unary union** up to chosen $k$
     - $(\phi_L \cdot \tau \cdot \phi_R) \cup_s (\phi_L' \cdot \tau' \cdot \phi_R') = (\phi_L \cap \phi_L' \cdot \tau \cup \tau' \cdot \phi_R \cap \phi_R')$
     - Build larger $\tau$ from smaller
  4. Known **maximum depth** after step 2
     - Finite number of combinations
Outline

1 Introduction
   • Parameterized Systems
   • Regular Model Checking
   • Related Work

2 Systematic Acceleration
   • Setting
   • Unary Actions
   • Example
   • Compositions
   • Acceleration Procedure

3 Experiments
   • Implementation
   • Results

4 Conclusion
Implementation

- Verified safety and liveness of parameterized mutual-exclusion protocols
- Procedure for generating $A$ with composition depth $k$
  - Known maximum value of $k$
  - Used with RMC model checker
- Fully automatic approaches:
  - Use maximum value for $k$ (possibly suboptimal time)
  - Guess low value of $k$, increase after time-out
Safety Results

- “More automatic” and as fast as our previous work
  - No manually specified compositions
- Exact representation of the invariant
  - Forwards reachability
  - Safety for free when checking liveness
- Abstraction techniques for safety are faster
  - But do not work for liveness
Liveness Results

<table>
<thead>
<tr>
<th>Protocol</th>
<th>This work</th>
<th>[LTL(MSO)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bakery</td>
<td>13</td>
<td>23</td>
</tr>
<tr>
<td>Burns</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>Szymanski</td>
<td>1635</td>
<td></td>
</tr>
<tr>
<td>Dijkstra</td>
<td>244</td>
<td></td>
</tr>
</tbody>
</table>

- Individual starvation freedom under weak fairness
  - Exact computation of reachable loops
  - Most general true property shown; checked others
  - **New results**: Burns and Dijkstra
Outline

1 Introduction
   - Parameterized Systems
   - Regular Model Checking
   - Related Work

2 Systematic Acceleration
   - Setting
   - Unary Actions
   - Example
   - Compositions
   - Acceleration Procedure

3 Experiments
   - Implementation
   - Results

4 Conclusion
Conclusion

- **Automatic systematic acceleration** scheme
  - Less expertise needed
  - Sometimes manual choice not feasible
    - Previously were not able to verify liveness
    - Retrospect: identified compositions sufficient for all protocols, except Dijkstra’s (tricky)

- Extends acceleration based RMC
- Works for safety and liveness
- Principle applicable to other acceleratable actions