On Model-Checking Concurrent Recursive Programs

Tayssir Touili
LIAFA, CNRS & Univ. Paris 7

Sagar Chaki, Ed Clarke
Carnegie Mellon University

Nick Kidd, Tom Reps
University of Wisconsin
Concurrent Software Model-Checking

• A major challenge in the community

• Various complex features:
  – Data over unbounded (very large) domains
  – Presence of recursive procedure calls
  – Concurrency and synchronisation
  – Dynamism

• Reachability of a control point is undecidable [Ramalingam 2000]

• Any analysis technique is incomplete
Existing Works

- Large data: Predicate abstraction [Graf, Saidi’97]
  - \( x:\text{int}; \quad (x>5) \text{ and } (x\leq5) \)

- Model precision and complexity of the analysis: number of predicates

- Discover a small number of predicates needed to prove the property

- Counter Example Guided Abstraction Refinement (CEGAR)
CEGAR

1. Program → Predicate Abstraction
2. Model → Verification
   - Yes → System OK
   - No
3. Verification
   - No → Counterexample
4. Counterexample
   - Yes
5. Counterexample Valid?
   - No
6. Spurious Counterexample
   - Add new predicates
   - Abstraction Refinement
7. Abstraction Refinement
   - No
8. No
Existing Works: CEGAR

• Implemented in several tools:

  • **SLAM**: no concurrency
  • **MAGIC**: no recursion
  • **BLAST**: no recursion + concurrency
  • etc
Existing Works (recursion+concurrency)

- Pushdown systems: Sequential recursive programs  [Esparza, Knoop’99], [Esparza, Schwoon’01]

- Model Recursion + Represent infinite configurations of recursive programs by regular languages  [Bouajjani, Esparza, Maler’97],

- Tool: MOPED

- Compositions of PDSs: Concurrent recursive programs  [Bouajjani, Esparza, T., Qadeer, Rehof, ......]

Data assumed to have a small domain
This Work

• Handle
  • Large data
  • Recursion
  • Concurrency
Our Approach

• **Large Data**: CEGAR on predicate abstraction

• **Concurrency and recursion**: Communicating PDSs
Our Approach

- **C Program**
  - Predicate Abstraction
    - Model: CPDS
    - Verification
      - Yes → System OK
      - No → Counterexample
      - Counterexample Valid?
        - Yes
        - No → Spurious Counterexample
        - No → Abstraction Refinement
          - Add new predicates

Abstraction Refinement
Pushdown System: Definition

\[
\text{Pushdown System } S = (Q, \text{Act}, \Gamma, c_0, \Delta):
\]

- \(Q\) is a finite set of states
- \(\text{Act}\) is a finite set of actions
- \(\Gamma\) is a finite stack alphabet
- \(c_0\) is the initial configuration \((q, v), q \in Q, v \in \Gamma^*\)
- \(\Delta\) is a finite set of rules of the form \((q, \gamma) \xrightarrow{a} (q', w), q, q' \in Q, a \in \text{Act}, \gamma \in \Gamma, w \in \Gamma^*\)

Transition relation: \((q, \gamma u) \xrightarrow{a} (q', w u)\)
From a Sequential Program to a Pushdown System
[Esparza, Schwoon’01]

- States: Global variables
- Symbols of the stack: Local variables + control points

Predicates?
Predicates?

• Subset $C$ of the conditions of the program

• Close it by computing weakest preconditions w.r.t. statements of the program

\[
S : x := e
\]

\[
WP_s(p) : p[x \leftarrow e]
\]
Predicates?

• Subset $C$ of the conditions of the program

• Close it by computing weakest preconditions w.r.t. statements of the program

Initially $C$ empty

Conditions are added using CEGAR
Compute the predicates?

- Associate to each control point $n$ a set of predicates $P(n) = P(n)_{loc} + P(n)_{glob}$

- $P(n)$: set of predicates needed at point $n$

- Initially, $P(n) = \emptyset$ for all $n$

- $P(n)$: updated by computing weakest preconditions
Compute $P(n)$

\[
\begin{align*}
  s &\colon n \rightarrow m \\
  \text{• } s &\colon \text{assignment; add } WP_s(P(m)) \text{ to } P(n)
\end{align*}
\]
Compute $P(n)$

$s: n \rightarrow m$

$s$: goto or synchronisation statement:
add $P(m)$ to $P(n)$

$s$: (if $c$ then) add $P(m)$ to $P(n)$
c in $C$ : add $c$ to $P(n)$

$s$: call to a procedure $q$
add $P(m)_{loc}$ and $P(init-q)_{glob}$ to $P(n)$
How to compute the PushDown System?
The PDS rules

\[ s: n \rightarrow m : \text{goto} \]

\[(\text{glob},(n,\text{loc})) \rightarrow (\text{glob}',(m,\text{loc}'))\]

\[\text{loc} \in P(n)_{\text{loc}} \quad \text{loc}' \in P(m)_{\text{loc}}\]

\[\text{glob} \in P(n)_{\text{glob}} \quad \text{glob}' \in P(m)_{\text{glob}}\]

\[(\text{loc} \uparrow \text{loc}') \text{ satisfiable} \quad (\text{glob} \uparrow \text{glob}') \text{ satisfiable}\]

\[\text{Undecidable} \text{ for first order formulas over the integers}\]

SIMPLIFY: sound theorem prover that answers true, false, or unknown
The PDS rules

\[ s: n \rightarrow m : \text{assignment} \]

\[(\text{glob}, (n, \text{loc})) \rightarrow (\text{glob}', (m, \text{loc}'))\]

\[\text{loc} \in P(n)_{\text{loc}} \quad \text{loc}' \in P(m)_{\text{loc}}\]

\[\text{glob} \in P(n)_{\text{glob}} \quad \text{glob}' \in P(m)_{\text{glob}}\]

\[(\text{loc} ^ \text{WP}_s(\text{loc}')) \text{ satisfiable}\]

\[(\text{glob} ^ \text{WP}_s(\text{glob}')) \text{ satisfiable}\]
Predicates?

- Subset $C$ of the conditions of the program
- Close it by computing weakest preconditions w.r.t. statements of the program

Initially $C$ empty

Conditions are added using CEGAR
Example

C is empty: Error reachable

\((x>8)\) in C : Error non reachable
What about Concurrency?

$n$ sequential components running in parallel, communicating via rendez-vous through blocking synchronizing actions

[Communicating Pushdown System (CPDS)]

[Bouajjani, Esparza, Touili’03]
Communicating Pushdown Systems

Internal actions

\[
PDS_1 \quad \cdots \quad PDS_i \quad \cdots \quad PDS_n
\]

\[
c_1 \quad \cdots \quad c_i \quad \cdots \quad c_n
\]

\[
c'_1 \quad \cdots \quad c'_i \quad \cdots \quad c_n
\]
Communicating Pushdown Systems

Synchronizing actions

\[ \text{PDS 1} \quad \cdots \quad \text{PDS i} \quad \cdots \quad \text{PDS j} \quad \cdots \quad \text{PDS n} \]

\[ \text{c}_1 \quad \cdots \quad \text{c}_i \quad \cdots \quad \text{c}_j \quad \cdots \quad \text{c}_n \]

\[ \downarrow \text{a} \quad \downarrow \text{a} \]

\[ \text{c}_1 \quad \cdots \quad \text{c'}_i \quad \cdots \quad \text{c'}_j \quad \cdots \quad \text{c}_n \]
What about Concurrency?

\( n \) sequential components running in parallel, communicating via rendez-vous through synchronizing actions

Communicating Pushdown System (CPDS)

[Bouajjani, Esparza, Touili’03]
Our Approach

C Program → Predicate Abstraction → Model: CPDS → Verification

- Yes: System OK
- No: Counterexample

Abstraction Refinement

- Add new predicates
- No: Counterexample
- Yes: Spurious Counterexample

Counterexample Valid?

- Yes: Spurious Counterexample
- No: Counterexample
Reachability Analysis of CPDSs

\[ L_1 \cap L_2 \neq \emptyset \]

\[ L_i : \text{paths of PDS}_i \text{ leading from } c_i \text{ to } c'_i \]

: Context - free language

Undecidable!

No exact solution
Solution: A CEGAR Scheme
[Clarke, Chaki, Kidd, Reps, Touili’06]

\[ L_1 \cap L_2 = \phi? \]

Compute over-approximations \( A_1 \supseteq L_1 \) and \( A_2 \supseteq L_2 \)

\[ I = A_1 \cap A_2 = \phi? \]

Can we extract a real path?

Compute refinable over-approximations

Refine approximation
Computing refinable over-approximations

\[ \alpha_k(L) = \text{prefix}_k(L) \cdot \Sigma^* \]

**Example:**

\[ L = abababc^* \]

\[ \alpha_3(L) = aba(a + b + c)^* \]

\[ \alpha_4(L) = abab(a + b + c)^* \]

Refinable abstractions: \( \alpha_1, \alpha_2, \alpha_3, \ldots \)

**Theorem:** [Bouajjani, Esparza, Touili’03]

\( L : \text{context free language} \Rightarrow \alpha_k(L) \) can be effectively computed
Solution: A CEGAR Scheme

\[ L_1 \cap L_2 = \phi \]

Compute over-approximations \( A_1 \supseteq L_1 \) and \( A_2 \supseteq L_2 \)

\[ I = A_1 \cap A_2 \supseteq \emptyset \]

**Theorem:** Termination if \( L_1 \cap L_2 \neq \emptyset \)

**Find errors**

**Prove correctness**

Can we extract a real path?

Refine approximation

YES

NO
Our Approach

C Program → Predicate Abstraction → Model: CPDS → Verification

- Yes → System OK
- No → Counterexample

Counterexample → Abstraction Refinement

Add new predicates

Spurious Counterexample → Counterexample Valid?

- No → Spurious Counterexample
- Yes → Predicate Abstraction

Predicate Abstraction → Abstraction Refinement
CounterExample Validation

Program 1 || Program 2

PDS 1 || PDS 2

$\ldots r_n$ in PDS1
$r'_1 \ldots r'_m$ in PDS2

$\ldots s_n$ in Program1?
$s'_1 \ldots s'_m$ in Program2?
CounterExample Validation

$s_1 ... s_n$ in Program1?

• Initially, $\varphi=\text{glob}_0 \wedge \text{loc}_0$

• For $i=1$ to $n$ do:

• If $s_i$ assignment, compute the strongest postcondition of $\varphi$ w.r.t. $s_i$

• $s_i : x := x + 5$; $\varphi : 1 < x < 4$; $\varphi' : 6 < x < 9$

• If $s_i : \text{if}$ statement with condition $c$; $\varphi' : \varphi \wedge c$

• If $\varphi$ satisfiable, the counterexample is valid
Our Approach

C Program → Predicate Abstraction → Model: CPDS → Verification

Verification:
- Yes → System OK
- No → Counterexample

Counterexample:
- Yes → Valid? (Yes)
- No → Abstraction Refinement

Abstraction Refinement:
- No → Counterexample
- Yes → Add new predicates

Spurious Counterexample

Predicate Abstraction

Refinement

Valid?
Abstraction Refinement

$s_1...s_n$ not in Program 1

Process 1 || Process 2

PDS 1 || PDS 2

Refine PDS 1

Add new conditions in $C$
Implementation
MAGIC

- Modular Analysis of proGrams In C
  [http://www.cs.cmu.edu/~chaki/magic](http://www.cs.cmu.edu/~chaki/magic)

- A Model checker for Concurrent C-programs developed at CMU

- Counter-Example Guided Abstraction Refinement (CEGAR)
Experiments:
Windows NT Bluetooth driver

• Discovery of a new unknown bug in a corrected version (20 seconds)

• Re-discovery of a bug in an old version (5 seconds)
Experiments:
Concurrent Insertions in Binary trees

• Find a bug in an uncorrect version

<table>
<thead>
<tr>
<th>#processes</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>0.8</td>
</tr>
<tr>
<td>5</td>
<td>1.1</td>
</tr>
<tr>
<td>6</td>
<td>2.7</td>
</tr>
<tr>
<td>7</td>
<td>12.9</td>
</tr>
</tbody>
</table>
Our technique behaves better than inlining

<table>
<thead>
<tr>
<th></th>
<th>CPDS</th>
<th>INLINING</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL Server1</td>
<td>2.82</td>
<td></td>
</tr>
<tr>
<td>OpenSSL Server2</td>
<td>3.83</td>
<td></td>
</tr>
<tr>
<td>OpenSSL Server3</td>
<td>19.2</td>
<td></td>
</tr>
<tr>
<td>OpenSSL Server4</td>
<td>2.76</td>
<td></td>
</tr>
<tr>
<td>OpenSSL Server5</td>
<td>3.02</td>
<td></td>
</tr>
<tr>
<td>OpenSSL Server6</td>
<td>2.93</td>
<td></td>
</tr>
<tr>
<td>OpenSSL Server7</td>
<td>3.34</td>
<td></td>
</tr>
<tr>
<td>2MicroC</td>
<td>1.324</td>
<td></td>
</tr>
<tr>
<td>3MicroC</td>
<td>2.144</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• Techniques+tool for the verification of concurrent recursive programs with unbounded data
• CEGAR in 2 levels: Abstraction level and the verification level
• Technique compositional: scalable to large programs
• Encouraging experimental results (discovery of an unknown bug in Windows NT driver+ better execution times)
Questions?