Model Checking Parallel Programs with Inputs

Jiří Barnat, Petr Bauch and Vojtěch Havel

12 February 2014
Motivation: Parallel Software Verification

```c
int a, b;
cin >> a >> b;
if ( a > 3 * b )
a = a * b;
```

```
a = a + b;
while ( a > 3 )
b--;
```

```
while ( b <= pow( 2, a ) )
b += a;
a *= a - (b | 14);
```

Thread 1

Thread 2

fork
Motivation: Parallel Software Verification

```plaintext
int a, b;
cin >> a >> b;
if (a > 3 * b)
    a = a * b;

a = a + b;
while (a > 3)
    b--;
while (b <= pow(2, a))
    b += a;
a *= a - (b | 14);
```

- input variables → data-flow nondeterminism

Barnat et. al. (ParaDiSe) Control Explicit—Data Symbolic
Motivation: Parallel Software Verification

```c
int a, b;
cin >> a >> b;
if ( a > 3 * b )
a = a * b;
```

- input variables → data-flow nondeterminism
- Peano arithmetic → undecidable theory of FO
Motivation: Parallel Software Verification

```cpp
int a, b;
cin >> a >> b;
if ( a > 3 * b )
a = a * b;
```

```
a = a + b;
while ( a > 3 )
b--;
while ( b <= pow( 2, a ) )
b += a;
a *= a - (b | 14);
```

- input variables → data-flow nondeterminism
- Peano arithmetic → undecidable theory of FO
- thread interleaving → control-flow nondeterminism
Summary

- **Idea**: Symbolic data, explicit exploration of parallel communication
- **Theory**: represent sets as images of bit-vector functions, translate state-related operations as satisfiability queries
- **Limitation**: non-canonicity of symbolic representation, proving state-equality is difficult
- **Result**: verified a data-intensive model of a work-distribution protocol
Why is Parallelism with Data Difficult

Parallelism
- hard to represent symbolically
- worst-case exponential in the number of threads

Data
- undecidable in worst case
- feasible in practice (if symbolic)

Together they pose conflicting needs on the representation
Better Together

Control Explicit —— Model Checking —— Data Symbolic

DiVinE
- store multi-states explicitly
- parallelise computation

SAT Solver
- successor computation
- state matching

\[ a \oplus b > c \]

\[ b \times a \leq c + 3 \]

\[ pc = 8 \]
Example: Parallel Program with Input

thread $t_0$
1: char $b$;
2: read $b$;

thread $t_1$
1: if ($b > 3$) 
2: while (true) 
3: $b++$;

thread $t_2$
1: if ($b*b <= 16$) 
2: while (true) 
3: $b -= 10$;

Program $P$

Automaton $\mathcal{A}$

Transition system $\mathcal{T}$

$\delta$

$\zeta$
Example: Set-Reduced State Space

1: char b;
2: read b;

thread \( t_0 \)

1: if ( b > 3 )
2: while ( true )
3: b++;

program \( P \)

automaton \( A \)

\[ b > 10 \]

\[ b > 10 \]

\[ \delta \]

\[ b = 0 \]
\[ s = 2,0,0 \]
\[ a = 1 \]

\[ \zeta \]

\[ b = 4 \]
\[ s = 2,1,0 \]
\[ a = 1 \]

transition system \( \mathcal{P} \)

set-reduced transition system \( \mathcal{P}_S \)

\[ d = (b,0) \]
\[ s = 1,0,0 \]
\[ a = 1 \]

\[ d = (b,\{0..255\}) \]
\[ s = 2,1,0 \]
\[ a = 1 \]

\[ d = (b,\{0..4\}) \]
\[ s = 2,0,1 \]
\[ a = 1 \]
Properties of Set Reduction

- read a; while ( a > 10 ) a--;

\[
\begin{align*}
    a = \{0..255\} & \rightarrow a = \{11..255\} & \rightarrow a = \{10..254\} & \rightarrow \cdots \\
\end{align*}
\]

- x = 1; read y; while ( true ) y++;

with specification

\[
\begin{align*}
    x = 0 & \rightarrow x = 1 & \rightarrow x = 1 \\
    y = 0 & \rightarrow y = 0 & \rightarrow y = \{0..255\} \\
\end{align*}
\]
Symbolic Data Representation

Represent a set of variable evaluations

- evaluate Peano arithmetic functions
- decide equality of sets

1. Explicit Set
2. FO formulae

- Peano arithmetic is undecidable
- Presburger arithmetic is not expressive enough
- Bit-vector ($BV$) theory represents computation adequately

\[
PC : a < 10 \land 3 < b \leq 7 \quad a \in \{3, \ldots, 16\} \times \quad b \in \{0, 3, \ldots, 27\} \\
a, b \mapsto a + b, 3 \ast a
\]

- checking path condition (simple SAT)
- state matching (equality of $BV$ function images)
Example: Symbolic Representation

1: read a;
2: if(a > 5)
3:   a++;
4: else
5:   a--;

\[(0, (a, 0), \top)\]
\[(a', \nu_1)\]
\[(1, (a, \nu_1), \top)\]

\[a > 5\]  \[a \leq 5\]

\[(2, (a, \nu_1), \nu_1 > 5)\]
\[(a', a + 1)\]
\[(3, (a, \nu_1 + 1), \nu_1 > 5)\]

\[(4, (a, \nu_1), \nu_1 \leq 5)\]
\[(a', a - 1)\]
\[(5, (a, \nu_1 - 1), \nu_1 \leq 5)\]
Model Checking

detect cycles in the transition system

- a state is reachable if its $pc$ is satisfiable: $\models pc(s)$
- equality of variable evaluations $ve$ (are $s_a$ and $s_b$ the same states?):
  states are sets represented as images of functions over input variables

$$s_a \not\subseteq s_b$$

$$s_a \not\subseteq s_b \lor s_b \not\subseteq s_a$$

$$\models pc(s_a) \land \forall \iota_1, \ldots, \iota_k. pc(s_b)[i_m/\iota_m] \Rightarrow ve(s_a) \neq ve(s_b)[i_m/\iota_m]$$

$$\ldots$$
Control Explicit—Data Symbolic Traversal

storage of seen states $S$, state $s = (s.exp, s.sym)$, transition system $A$

\[
\text{foreach } s.exp \xrightarrow{\varphi} b \text{ in } A
\]

\[
x_i := f
\]
Control Explicit—Data Symbolic Traversal

storage of seen states $S$, state $s = (s.exp, s.sym)$, transition system $A$

foreach $s.exp \xrightarrow{\varphi} b$ in $A$

- $pc(\rho), ve(\rho) := pc(s.sym) \land \varphi, ve(\rho)[x_i/f]$
- if $\neg SAT_{QF}(pc(\rho))$ then continue
storage of seen states \( S \), state \( s = (s.\text{exp}, s.\text{sym}) \), transition system \( A \)

\[
\text{foreach } s.\text{exp} \xrightarrow{\varphi} b \text{ in } A
\]

\[
\begin{align*}
\text{• } pc(\rho), ve(\rho) & := pc(s.\text{sym}) \land \varphi, ve(\rho)[x_i/f] \\
\text{• } \text{if } \neg \text{SAT}_Q(pc(\rho)) \text{ then continue} \\
\text{• } seen & := \text{false} \\
\text{• } \text{foreach } s' \in S \text{ such that } s'.\text{exp} = b \\
\text{• } \rho' & := s'.\text{sym} \\
\text{• } \text{if } SAT_Q(\rho \neq \rho') \text{ then} \\
\text{• } s' \text{ is a successor of } s; seen := true; \text{ break}
\end{align*}
\]
Control Explicit—Data Symbolic Traversal

storage of seen states $S$, state $s = (s.exp, s.sym)$, transition system $A$

\[
\begin{align*}
\text{foreach } s.exp \xrightarrow{\varphi} b & \text{ in } A \\
\quad pc(\rho), ve(\rho) & := pc(s.sym) \land \varphi, ve(\rho)[x_i/f] \\
\quad \text{if } \neg SAT\_QF(pc(\rho)) & \text{ then continue} \\
\quad seen & := \text{false} \\
\quad \text{foreach } s' \in S \text{ such that } s'.exp = b & \\
\quad & \quad \rho' := s'.sym \\
\quad & \quad \text{if } SAT\_Q(\rho \neq \rho') \text{ then} \\
\quad & \quad \quad s' \text{ is a successor of } s; seen := true; \text{ break} \\
\quad & \quad \text{if } seen \text{ then} \\
\quad & \quad \quad s''.sym := \rho; s''.exp := b \\
\quad & \quad \quad \text{store } s'' \text{ into } S \\
\quad & \quad \quad s'' \text{ is a successor of } s
\end{align*}
\]
Experiments: Simulink

- small model ($\sim 20$ states)
- 6 32-bit input variables
- average SAT query: 20s
- complexity grows with \#steps
- scales to arb. bit-width
Improvement 1: Equivalence-Based Comparison

**Problem:** image-based comparison requires solving quantified queries

**Idea:** interpret valuation not as a set but as a function

\[ pc(s_a) \neq pc(s_b) \lor (pc(s_a) \land ve(s_a) \neq ve(s_b)) \lor (\ldots) \]

**Limitations:** no inputs on cycles (infinite state space), not precise:

Yet: model checking remains sound
Improvement 2: Explicating Values

**Problem:** linear search in database is expensive

**Idea:** store single-valued variables explicitly

**Example:**

```c
read d;
if ( d < 10 )
    for ( i = 0; i < 1000; i++ )
        d++;
```

**Implementation:** additional SAT query using previous satis. model
Experiments: Work Distribution Protocol

- server distributes work load using a hash function
- verify that a worker never depletes its load (non-safety)

**TABLE I: Results for the work distribution protocol.**

<table>
<thead>
<tr>
<th>Valuation Representation</th>
<th>Pure</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Matching Technique</td>
<td>Image</td>
<td>Equiv</td>
</tr>
<tr>
<td>Number of states</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>Number of transitions</td>
<td>167</td>
<td></td>
</tr>
<tr>
<td>Number of deadlocks</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Number of control-flow states</td>
<td>5</td>
<td>54</td>
</tr>
<tr>
<td>Average number of collisions</td>
<td>19.4</td>
<td>1.80</td>
</tr>
<tr>
<td>State space distribution</td>
<td>1:2,23:1,24:2</td>
<td>1:42,2:5,3:3,9:4</td>
</tr>
<tr>
<td>Path condition time</td>
<td>~3.2</td>
<td></td>
</tr>
<tr>
<td>Path condition calls</td>
<td>533</td>
<td></td>
</tr>
<tr>
<td>State matching time</td>
<td>78.1</td>
<td>49.6</td>
</tr>
<tr>
<td>State matching calls</td>
<td>2123</td>
<td>2308</td>
</tr>
<tr>
<td>Explicit value time</td>
<td>n/a</td>
<td>13.81</td>
</tr>
<tr>
<td>Explicit value calls</td>
<td>n/a</td>
<td>1680</td>
</tr>
<tr>
<td>Deadlock detection time</td>
<td>~0.55</td>
<td></td>
</tr>
<tr>
<td>Deadlock detection calls</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>Verification time</td>
<td>81.92</td>
<td>59.59</td>
</tr>
<tr>
<td>Memory consumption</td>
<td>37.6</td>
<td>27.9</td>
</tr>
</tbody>
</table>
Comparing SAT Queries

- 100x higher peak complexity (image vs. equiv)
- explicating scales searching

- imprecision (equiv) adds new queries
- even QF queries can be very hard
Improvement 3: Model-Based Matching Resolution

- every state comparison $s_a \neq s_b$ generates a witness $(\nu_I, \nu_S)$

$$ve(s_a)(\nu_I) = \nu_S \neq ve(s_b)(\nu_I)$$

- discovered state $s_c$

$$\begin{align*}
s_c \neq s_b & \quad \text{if} \quad ve(s_a)(\nu_I) = \nu_S \\
s_c \neq s_a & \quad \text{if} \quad ve(s_a)(\nu_I) \neq \nu_S
\end{align*}$$

\{no SAT query\}

- associate $\{s_i\}, \{s'_j\}$ with every witness

$$\forall i, j : ve(s_i)(\nu_I) = \nu_S \neq ve(s'_j)(\nu_I)$$

- build a priority queue of witnesses using their set sizes

- ideally achieve logarithmic search
Experiments: Random Model

![Graph showing time vs. edges for full vs. pruned models. The graph has a y-axis labeled 'Time (s)' and an x-axis labeled 'Edges'. The legend is marked as 'full' in red and 'pruned' in green. The graph displays data points with error bars for each condition.]
Parallelisation

- state space partitioned among threads
- time to handle different states varies
Related Work

- local/global model checking [Biere, Clark, Zhu’99]
- compositional verification [Gries/Owicki, Rely/Guarantee, etc.]
- FO-CTL model checking [Hungar, Grumberg, Damm'95,'98]
- symbolic model checking [BDD-based, backwards, global, CTL]
- abstraction + fair termination [parallelism, LTL, modular arithmetic]
- other reduction techniques [POR, Duret-Lutz, etc.]
Conclusion

verify parallel programs hard
verify data-intensive programs hard \}
together harder still

▶ combine explicit and symbolic approaches to fight the complexity
▶ use explicit model checking to handle thread interleaving (control)
▶ use SMT solvers to handle variable evaluations (data)
▶ heuristics to speed up state space search
  1. explicating values
  2. equivalence-based comparison
  3. witnesses of difference
Future Work

▶ Faster proving of state equality
  ▶ Canonical representation for bit-vector images
  ▶ Rewriting simplifications
▶ Fewer proofs of state equality (CEGAR-like approach)
▶ Investigate solver-based simplifications
▶ Different form of reduction: inductive loops