SAT-based Summarization for Boolean Programs

Gérard Basler  Daniel Kroening  Georg Weissenbacher

ETH Zurich

14th International SPIN Workshop on Model Checking Software, Berlin
Abstract-Verify-Refine Paradigm (CEGAR)

Basler, Kroening, Weissenbacher (ETHZ)
Abstract-Verify-Refine Paradigm (CEGAR)
Abstract-Verify-Refine Paradigm (CEGAR)
Abstract-Verify-Refine Paradigm (CEGAR)

Basler, Kroening, Weissenbacher (ETHZ)
Summarization using SAT & QBF

SPIN 2007
Abstract-Verify-Refine Paradigm (CEGAR)

...several iterations later...
Abstract-Verify-Refine Paradigm (CEGAR)

...several iterations later...
Predicate Abstraction

- Tracks facts in the program using *predicates* (e.g., $i < 5$, $i > 10$)
- Preserves control flow structure
- Generates *Boolean Programs*

```
L1
L2
L3
L4
L5
Error
[i > 10]
[i ≤ 10]
i++;
[i ≥ 5]
i=0;
[i < 5]
```

Basler, Kroening, Weissenbacher (ETHZ)  Summarization using SAT & QBF  SPIN 2007 3 / 1
Predicate Abstraction

- Tracks facts in the program using *predicates* (e.g., \(i < 5\), \(i > 10\))
- Preserves control flow structure
- Generates *Boolean Programs*
Predicate Abstraction

- Tracks facts in the program using *predicates* (e.g., \(i < 5\), \(i > 10\))
- Preserves control flow structure
- Generates *Boolean Programs*

\[
\begin{align*}
L1 & : \quad \text{i=0;} \\
L2 & : \quad [i > 10] \quad \text{ERROR} \\
L3 & : \quad [i \leq 10] \\
L4 & : \quad \text{i++;} \\
L5 & : \quad [i \geq 5] \\
\end{align*}
\]

\[
\begin{align*}
L1 & : \quad b_1 = F; \\
L2 & : \quad \text{ERROR} \\
L3 & : \quad [\neg b_1] \\
L4 & : \quad b_1 = b_1 ? T : *; \\
L5 & : \quad \text{ERROR} \\
\end{align*}
\]

\[
\begin{align*}
L1 & : \quad b_1, b_2 = F, T; \\
L2 & : \quad \text{ERROR} \\
L3 & : \quad [\neg b_1] \\
L4 & : \quad b_1 = b_1 ? T : (b_2 ? F : *), \\
L5 & : \quad b_2 ? * : F; \\
\end{align*}
\]
Reachability of locations in Boolean Programs decidable

- BDD based symbolic model checkers Bebop, Moped

So why bother to work on a “solved” problem?
Reachability of locations in Boolean Programs decidable
- BDD based symbolic model checkers **Bebop, Moped**

So why bother to work on a “solved” problem?
- **SatAbs**: >70% of runtime spent verifying Boolean Programs
- BDD-based techniques don’t scale for large number of variables
Model Checking Boolean Programs

- Reachability of locations in Boolean Programs decidable
  - BDD based symbolic model checkers Bebop, Moped

- So why bother to work on a “solved” problem?
  - SatAbs: >70% of runtime spent verifying Boolean Programs
  - BDD-based techniques don’t scale for large number of variables

- But is there something faster than BDDs?
  - SAT-solvers can solve instances with a huge number of variables
  - QBF-solvers are improving steadily
What can a Boolean Program do?

- Finite number of variables, all of them Boolean
- They have a global state and a stack
Transitions: Neutrations

\[ \langle p, \gamma_1 \rangle \hookrightarrow \langle q, \gamma_2 \rangle \]
Modify the control state $p$
Transitions: Neutrations

\[ \langle p, \gamma_1 \rangle \leftrightarrow \langle q, \gamma_2 \rangle \]

- Modify the control state \( p \)
- Modify the topmost stack element \( \gamma_1 \)
Transitions: Neutrations

\[ \langle p, \gamma_1 \rangle \hookrightarrow \langle q, \gamma_2 \rangle \]

- Modify the control state \( p \)
- Modify the topmost stack element \( \gamma_1 \)
- Do not modify the elements below \( \gamma_1 \)
Transitions: *Expansions*

\[
\langle p, \gamma \rangle \leftrightarrow \langle q, \gamma_1 \gamma_2 \rangle
\]

- Modify the control state \( p \)
- Modify the topmost stack element \( \gamma \)
Transitions: Expansions

\[ \langle p, \gamma \rangle \leftrightarrow \langle q, \gamma_1 \gamma_2 \rangle \]

- Modify the control state \( p \)
- Modify the topmost stack element \( \gamma \)
- Push a new element on the stack
- Corresponds to “call”
Modify the control state $p$
- Pop the topmost stack element $\gamma$
- Corresponds to “return”
- Use symbolic representation of transitions
Use symbolic representation of transitions
- Relates first and last state of path
A Transition Sequence

- Use symbolic representation of transitions
- Relates first and last state of path

\[ \bar{a}_1 \bar{a}_0, \bar{b}_1 \bar{b}_0, \bar{a}_1' \bar{a}_0', \bar{b}_1' \bar{b}_0', \bar{a}_1'' \bar{a}_0'', \bar{b}_1'' \bar{b}_0'', \bar{a}_1''' \bar{a}_0''', b_1' b_0''', \bar{p}, \gamma, \bar{p}', \gamma' \]

\[ R(\bar{p}, \gamma, \bar{p}', \gamma') \]
Symbolic summary for path

- Can represent more than one explicit path

\[
R(\langle a_0, a_1, b_0, b_1 \rangle, \langle a'_0, a'_1, b'_0, b'_1 \rangle) = \\
(\overline{a}_1 + a_0) \cdot (b_1 \cdot b_0) \cdot (\overline{a}'_1 \cdot (a'_0 = a_1)) \cdot (b'_1 \cdot \overline{b}'_0)
\]
Can represent more than one explicit path

\[ R(\langle a_0, a_1, b_0, b_1 \rangle, \langle a'_0, a'_1, b'_0, b'_1 \rangle) = (\bar{a}_1 + a_0) \cdot (b_1 \cdot b_0) \cdot (\bar{a}'_1 \cdot (a'_0 = a_1)) \cdot (b'_1 \cdot \bar{b}'_0) \]

\[ R(\langle p_0, \gamma_3 \rangle, \langle p_2, \gamma_2 \rangle) \]
Can represent more than one explicit path

\[ R(\langle a_0, a_1, b_0, b_1 \rangle, \langle a'_0, a'_1, b'_0, b'_1 \rangle) = (\overline{a}_1 + a_0) \cdot (b_1 \cdot b_0) \cdot (\overline{a}'_1 \cdot (a'_0 = a_1)) \cdot (b'_1 \cdot \overline{b}'_0) \]

\[ R(\langle p_0, \gamma_3 \rangle, \langle p_2, \gamma_2 \rangle), R(\langle p_1, \gamma_3 \rangle, \langle p_2, \gamma_2 \rangle) \]
Can represent more than one explicit path

\[ R(\langle a_0, a_1, b_0, b_1 \rangle, \langle a'_0, a'_1, b'_0, b'_1 \rangle) = (\overline{a}_1 + a_0) \cdot (b_1 \cdot b_0) \cdot (\overline{a}'_1 \cdot (a'_0 = a_1)) \cdot (b'_1 \cdot \overline{b}'_0) \]

\[ R(\langle p_0, \gamma_3 \rangle, \langle p_2, \gamma_2 \rangle), R(\langle p_1, \gamma_3 \rangle, \langle p_2, \gamma_2 \rangle), \text{ and } R(\langle p_3, \gamma_3 \rangle, \langle p_3, \gamma_2 \rangle) \]
Prefix of path constrains entry state
Applying Summaries

- Prefix of path constrains entry state
- Search encounters new “calling context”
Prefix of path constrains entry state
Search encounters new “calling context”
QBF to determine whether entry state and calling context “compatible”:
$$\forall s_c \exists s_i. s_c = s_i$$
Summarization: Why does it work?

- Only a finite number of possible input/output pairs
Symbolic Search & Summarization: When are we done?

- Maintain worklist of path formulas incident to nodes
- Remove from worklist if already *covered* by other formula
- Maintain set of summaries

\[ R_{NEW} \subseteq R_{OLD}? \]
Symbolic Search & Summarization: When are we done?

- Maintain worklist of path formulas incident to nodes
- Remove from worklist if already covered by other formula
- Maintain set of summaries

\[ R_{NEW} \subseteq R_{OLD} ? \]

\[ \forall \langle p_0, \gamma_0 \rangle, \langle p'_0, \gamma'_0 \rangle . \exists \langle p_1, \gamma_1 \rangle, \langle p'_1, \gamma'_1 \rangle . \]

\[ R_{NEW}(p_0 \gamma_0, p'_0 \gamma'_0) = R_{OLD}(p_1 \gamma_1, p'_1 \gamma'_1) \]
Universal Summaries

- *Universal Summary* provides a summary for *any arbitrary* entry state
- “calling context” is unconstrained
Universal Summary provides a summary for any arbitrary entry state

“calling context” is unconstrained

\[
\Sigma_U(\langle p, \gamma \rangle, \langle p', \gamma' \rangle) \\
\iff
\]

\[
\forall \langle p, \gamma w \rangle. \ (\exists \langle p_1, w_1 \rangle, \ldots, \langle p_n, w_n \rangle. \\
\langle p, \gamma \rangle \rightarrow \langle p_1, w_1 \rangle \rightarrow \ldots \rightarrow \langle p_n, w_n \rangle \rightarrow \langle p', \gamma' \rangle \wedge \\
\forall i \in \{1..n\}. |w_i| \geq 2)
\]
Constructing Universal Summaries using BMC

\[ R_{n_3, n_0} \]

\[ R_{n_0, n_1} \]

\[ R_{n_1, n_3} \]

\[ R_{n_0, n_2} \]

\[ R_{n_2, n_3} \]

\[ n_0 \]

\[ n_1 \]

\[ n_2 \]

\[ n_3 \]
Constructing Universal Summaries using BMC

![Diagram of nodes and edges with labels](image)
Constructing Universal Summaries using BMC

\[ R_{n_3,n_0} \]

\[ R_{n_0,n_1} \]

\[ R_{n_1,n_3} \]

\[ R_{n_0,n_2} \]

\[ R_{n_2,n_3} \]

\[ R_{n_3,n_0} \]
Constructing Universal Summaries using BMC

Basler, Kroening, Weissenbacher (ETHZ) Summarization using SAT & QBF
BMC: When to stop unrolling?

- unroll up to *longest loop-free path*
- all states in the path are *pairwise* different
- \( \exists S0, S1, S2, S3 \ . \ S0 \neq S1 \land S0 \neq S2 \land S0 \neq S3 \land S1 \neq S2 \ldots \)
Start with *innermost* functions (no call to other function)

Construct summaries in *top down* manner

*Merge* all summaries obtained by unrolling

\[ \bigvee_{i=1}^{k} \Sigma_i (p, \gamma, p', \gamma') \]
• Start with *innermost* functions (no call to other function)
• Construct summaries in *top down* manner
• *Merge* all summaries obtained by unrolling

\[ \bigvee_{i=1}^{k} \sum_i (p, \gamma, p', \gamma') \]

• Applicable in *all* calling contexts!
## Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#vars</th>
<th>BEBOP</th>
<th>QBF-summaries</th>
<th>univ. summ.</th>
<th>violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>adddevice</td>
<td>434</td>
<td>4m37.4s</td>
<td>0m0.6s</td>
<td>0m1.8s</td>
<td>yes</td>
</tr>
<tr>
<td>nulldevice</td>
<td>434</td>
<td>4m34.0s</td>
<td>0m8.6s</td>
<td>0m1.4s</td>
<td>yes</td>
</tr>
<tr>
<td>pendedcompletedreq</td>
<td>86</td>
<td>0m30.9s</td>
<td>timeout</td>
<td>0m13.5s</td>
<td>yes</td>
</tr>
<tr>
<td>targetrelationneedsref</td>
<td>37</td>
<td>0m0.4s</td>
<td>0m0.5s</td>
<td>0m2.74s</td>
<td>no</td>
</tr>
<tr>
<td>markirppending</td>
<td>11</td>
<td>0m0.4s</td>
<td>0m3.0s</td>
<td>0m18.5s</td>
<td>no</td>
</tr>
<tr>
<td>wmiforward</td>
<td>15</td>
<td>0m0.7s</td>
<td>0m2.0s</td>
<td>0m15.3s</td>
<td>no</td>
</tr>
<tr>
<td>TERMINATOR 1</td>
<td>74</td>
<td>timeout</td>
<td>1m55.9s</td>
<td>1m55.9s</td>
<td>yes</td>
</tr>
<tr>
<td>TERMINATOR 2</td>
<td>60</td>
<td>88m22.6s</td>
<td>timeout</td>
<td>timeout</td>
<td>yes</td>
</tr>
</tbody>
</table>
Conclusion

- **Advantages:**
  - Universal Summaries good for bug finding
    - In CEGAR, for $n$ iterations, $\geq n - 1$ of the abstractions have a “bug”
  - Eliminates many calls to QBF solver

- **Disadvantages:**
  - Does not work for programs with recursion: Fall back to QBF
  - Large universal summaries combined with QBF too hard for solver
  - Does not scale very well for “bug-free” programs