SS 2017
Software Verification
Bounded Model Checking, Outlook

Prof. Dr. Holger Schlingloff ¹,²
Dr. Esteban Pavese ¹

(1) Institut für Informatik der Humboldt Universität
(2) Fraunhofer Institut für offene Kommunikationssysteme FOKUS
Recap

- What does SMT stand for?
- Is SMT hard to solve?
- How can a SMT solver be used for SMC?
- What is the main problem?
- How to generate invariants?
- What is partial order reduction in SMC?
- What is an ample set?
- Conditions on ample sets?
Bounded Model Checking

- Back to the beginning: treat model checking as a boolean satisfiability problem
- Inductive reasoning: find invariant \( P \) such that
  - \( I[x] \rightarrow P[x] \), and
  - \( P[x] \land T[x,x'] \rightarrow P[x'] \)
- Now: try to satisfy
  - \( I[x] \land T[x,x'] \land T[x',x''] \land T[x'',x'''] \land \ldots \)
  - \( \text{FOL}(\varphi) \)
- Recall that \( \text{FOL}(\varphi) \) translates temporal formulas into first-order:
  - \( \text{FOL}(\bot) = \bot \), \( \text{FOL}(p) = p(x) \), \( \text{FOL}(\langle \varphi \rightarrow \psi \rangle) = (\text{FOL}(\varphi)(x) \rightarrow \text{FOL}(\psi)(x)) \)
  - \( (\varphi U^+ \psi) = X(\psi \lor \varphi \land X(\psi \lor \varphi \land X(\psi \lor \varphi \land \ldots) \ldots)) \)
  - \( \text{FOL}^i(\varphi U^+ \psi) = T[x,x'] \land (\text{FOL}^i(\psi)(x') \lor \text{FOL}^i(\varphi)(x') \land (T[x',x''] \land (\text{FOL}^i(\psi)(x'') \lor \text{FOL}^i(\varphi)(x'') \land \ldots) \ldots)) \)
Basic Idea

- Instead of exploring all states exhaustively, unroll the transition relation up to a certain fixed bound and search for violations of the property within that bound.
- Transform this search to a Boolean satisfiability problem and solve it using a SMT solver.
- We can define a bounded semantics for temporal logic properties so that if a path satisfies a property based on the bounded semantics, then it satisfies the property based on the unbounded semantics.
  - Using this observation, we can show that a counter-example found on a bounded path is guaranteed to be a real counter-example.
  - However, not finding a violation using bounded model checking does not guarantee correctness.
• Formally:

\[
\begin{align*}
[M]_k & \triangleq I(w^0) \land \bigwedge_{i=1}^{k} R(w^{i-1}, w^i) \land \left( T(w^k) \lor \bigvee_{l=0}^{k} R(w^k, w^l) \right) \\
& \quad \text{this formula describes that } (w^0, w^1, \ldots, w^k) \text{ is a maximal path in } M.
\end{align*}
\]

\[
\begin{align*}
[(\varphi U^+ \psi)]_k & \triangleq \bigvee_{j=i+1}^{k} ([\psi]_k^{j} \land \bigwedge_{m=i+1}^{j-1} [\varphi]_k^{m}) \lor \\
& \quad \bigvee_{l=0}^{k} \left( \bigwedge_{m=i+1}^{k} [\varphi]_k^{m} \land R(w^k, w^l) \lor \bigvee_{j=l}^{i} ([\psi]_k^{j} \land \bigwedge_{m=l}^{j-1} [\varphi]_k^{m}) \right)
\end{align*}
\]

• this formula describes that \((\varphi U^+ \psi)\) holds along this path
Resettable Counter Revisited

- Consider the following program
  - `{c=1; while (T) {if (r ∨ (c==n)) then {c=1} else {c++}}}`
- Transition relation $[[M]]_3$: 

\[
[M]_3 = \begin{align*}
&c_0 = 1 \land \neg m_0 = n_0 \land \text{ite}(r_0 \lor c_0 = n_0, c_1 = 1, c_1 = c_0 + 1) \\
&\land \neg m_0 = n_0 \land \text{ite}(r_0 \lor c_0 = n_0, c_2 = 1, c_2 = c_0 + 1) \\
&\land \neg m_0 = n_0 \land \text{ite}(r_0 \lor c_0 = n_0, c_3 = 1, c_3 = c_0 + 1) \\
&\land (\neg m_3 = n_3 \land \text{ite}(r_3 \lor c_3 = n_3, c_0 = 1, c_0 = c_3 + 1) \\
&\land \text{ite}(r_3 \lor c_3 = n_3, c_1 = 1, c_1 = c_3 + 1) \\
&\land \text{ite}(r_3 \lor c_3 = n_3, c_2 = 1, c_2 = c_3 + 1) \\
&\text{ite}(r_3 \lor c_3 = n_3, c_3 = 1, c_3 = c_3 + 1)
\end{align*}
\]
Resettable Counter Revisited

- Check the following properties
  - \((G^*(c \leq n) \land F^*(c = n))\)
  - \((n > 1 \land G^*(c \leq n) \land F^*(c = n))\)
  - \(G^*(-r \land c < n)\)

- Satisfiability:

\[
G^*(c \leq n) \land F^*(c = n) \rightarrow c_0 \leq n_0 \land c_1 \leq n_1 \land c_2 \leq n_2 \land c_3 \leq n_3 \land (c_0 = n_0 \lor c_1 = n_1 \lor c_2 = n_2 \lor c_3 = n_3) = [y]_3
\]
Correctness

12.1. Theorem. There exists a maximal path of length $k$ generated by $M$ which initially validates $\psi$ iff $([M]_k \land [\psi]^0_k)$ is propositionally satisfiable. In other words, $\psi$ is sequence-valid in $M$ iff $([M]_k \rightarrow [\psi]^0_k)$ is propositionally valid for all $k \geq 0$.

- How to determine length $k$?
  - start with “reasonably small” $k$
  - while no satisfying assignment (counterexample) is found, increase $k$
  - stop if “diameter” of $M$ is reached
    (again, this is in general undecidable for infinite-state programs)
  - there is no efficient algorithm for identifying the diameter in general
  - again, heuristics can be used
Properties of BMC

- Bounded model checking finds counterexamples fast. This is due to depth first nature of SAT search procedures.
- Counterexamples of minimal length. This feature helps user understand counterexample more easily.
- Often, less space than BDD based approaches; no need for variable ordering heuristics.
- Can be combined with partial order methods.
Implementations

- CBMC: bounded model checker for ANSI-C programs (sequential)
- ESBMC: Efficient SMT-Based Context-Bounded Model Checker (multi-threaded)
  - new version (2016) for CUDA Programs (Nvidia)
- Benchmarking competitions
CBMC Example

Original code

```c
x=0;
while (x < 2) {
    y=y+x;
    x++;
}
```

Unwinding the loop 3 times

```c
x=0;
if (x < 2) {
    y=y+x;
    x++;
}
if (x < 2) {
    y=y+x;
    x++;
}
if (x < 2) {
    y=y+x;
    x++;
}
assert (! (x < 2))
```

Unwinding assertion:

Thanks to Tevzik Bultan, UCSB, https://www.cs.ucsb.edu/~bultan/courses/272/lectures/SMT-BMC.ppt
From Code to SAT

- After eliminating loops and recursion, CBMC converts the input program to the static single assignment (SSA) form
  - In SSA each variable appears at the left hand side of an assignment only once
  - This is a standard program transformation that is performed by creating new variables
- In the resulting program each variable is assigned a value only once and all the branches are forward branches (there is no backward edge in the control flow graph)
- CBMC generates a Boolean logic formula from the program using bit vectors to represent variables
(Another) CBMC-Example

1. Original code
   \[
   \begin{align*}
   x &= x + y; \\
   \text{if } (x! = 1) & \quad x = 2; \\
   \text{else} & \quad x++;
   \end{align*}
   \]
   \[
   \text{assert } (x \leq 3);
   \]

2. Convert to static single assignment
   \[
   \begin{align*}
   x_1 &= x_0 + y_0; \\
   \text{if } (x_1! = 1) & \quad x_2 = 2; \\
   \text{else} & \quad x_3 = x_1 + 1;
   \end{align*}
   \]
   \[
   x_4 = (x_1! = 1) ? x_2 : x_3; \\
   \text{assert } (x_4 \leq 3);
   \]

3. Generate constraints
   \[
   \begin{align*}
   C \equiv x_1 &= x_0 + y_0 \land x_2 = 2 \land x_3 = x_1 + 1 \land (x_1! = 1 \land x_4 = x_2 \lor x_1 = 1 \land x_4 = x_3) \\
   P \equiv x_4 \leq 3
   \end{align*}
   \]

4. Check if \( C \land \neg P \) is satisfiable
   (if it is then the assertion is violated)
   - \( C \land \neg P \) is converted to boolean logic using a bit vector representation for the integer variables \( y_0, x_0, x_1, x_2, x_3, x_4 \)
Other Software-related Issues

- Arrays, pointers
- Structured data types, lists, sets and bags
- Methods, parameters
- Classes, inheritance
- Type systems
Further Topics

- **Abstraction**
  - Counter-Example Guided Abstraction Refinement
  - over- and underapproximations

- **Testing by model checking**
  - construction of optimal counterexamples

- **Hybrid systems**
  - real-valued variables evolving according to diff. equations

- **Strategic Logics**
  - multi-player games, concurrent game structures
  - application to collaborative systems

- **Synthesis**
  - finding strategies = automatically constructing programs