Theory Refinement in HiFrog

Seminar on Concurrency, 13. 10. 2017

HiFrog

HiFrog

- Bounded Model Checker for safety properties
- Uses function summaries to speed-up verification
- Summaries are stored in a database for future runs
- Uses SMT for encoding programs and function summaries

SMT Theories

These quantifier-free theories can be used for encoding programs:

- Unintepreted functions
- Linear real arithmetic
- Propositional logic (bit-vectors)
- Uninterpreted functions for programs
- Bit-vectors for programs

Program Encoding

Given an assertion:

1. Loops in the program are unwound
2. Program is symbolically executed function-per-function
3. The result is a SSA representation of each involved function

Program Encoding

Given an assertion:

1. Loops in the program are unwound
2. Program is symbolically executed function-per-function
3. The result is a SSA representation of each involved function

$$
\begin{aligned}
x_0 &= \texttt{nondet\_a} \\
y_0 &= x_0 + \texttt{nondet\_b} \\
x_1 &= y_0 \\
z_0 &= \ldots
\end{aligned}
$$

Program Encoding (cont.)

Given an assertion:

4. SSA representation is sliced w.r.t the given assertion
5. Pruned SSA is transformed into a SMT formula, using summaries instead functions where possible

Program Encoding (cont.)

Given an assertion:

4. SSA representation is sliced w.r.t the given assertion
5. Pruned SSA is transformed into a SMT formula, using summaries instead functions where possible

$$
\begin{array}{ll}
x_0 = a & (x_0 = a) \wedge \\
y_0 = x_0 + b & (y_0 = (x_0 + b)) \wedge \\
x_1 = y_0 & (x_1 = x_1) \wedge \\
z_0 = \ldots & (z_0 = \ldots) \wedge \ldots
\end{array}
$$

Function Summaries

- Assertions are verified one-by-one
- Interpolants derived from successfully verified assertions are stored as summaries
- Summaries are generally an over-approximation $\rightarrow$ spurious errors
- Summaries involved in a spurious error are refined (using interpolation) and replaced

Theory Refinement

Function Summaries

- Summaries are refined up to a pre-determined logic
- If the logic is insufficient, we need to change the logic
- Idea: change logic not only in summaries, but on-demand in any part of SMT formula representing the program

Theory Refinement

- Use the simplest theory as possible for program encoding
- On spurious error, refine parts of programs that need more precise theory
- Needed theories are identified based on counter-examples (CEGAR)

Theory Refinement – Idea

- Use the simplest theory as possible for program encoding
- On spurious error, refine parts of programs that need more precise theory
- Needed theories are identified based on counter-examples (CEGAR)

Theory Refinement – Practice

- Set of theories partially ordered w.r.t their precision
- Needs support in SMT solver
- In HiFrog – two new theories:
  - uninterpreted functions for programs (UFP)
  - bit-vectors for programs (BVP)

Uninterpreted Functions for Programs

- Based on theory of quantifier free uninterpreted functions with equality
- Adds integer and real constants
- Adds commutativity of some known functions ($*$, $+$, &, ...)
- Terms encoded in UFP are indexed by u, e.g.: t^u

Bit-vectors for Programs

- Based on theory of quantifier free bit-vectors
- We denote b_i the ith bit of bitvector b, b_1 being the *least significant bit*.
- Terms encoded in BVP are indexed by b, e.g.: t^b

Combining UFP and BVP

- For a term t we say that it is bound in formula F, if both t^u and t^b are present in F.

Combining UFP and BVP

- For a term t we say that it is bound in formula F, if both t^u and t^b are present in F.

```
c = ((a % 2) +
     (b % 2)) % 2
c' = (a + b) % 2
d = f * e * c
d' = e * f * c'
```

$$
\begin{aligned}
&(\texttt{c}^b = ((\texttt{a}^b \ \%\ 2^b) \\
&\qquad + (\texttt{b}^b \ \%\ 2^b))\ \%\ 2^b) \wedge \\
&(\texttt{c'}^b = (\texttt{a}^b + \texttt{b}^b)\ \%\ 2^b) \wedge \\
&(\texttt{d}^u = \texttt{f}^u * \texttt{e}^u * \texttt{c}^u) \wedge \\
&(\texttt{d'}^u = \texttt{e}^u * \texttt{f}^u * \texttt{c'}^u)
\end{aligned}
$$

Combining UFP and BVP (cont.)

- Given a set of all bound statements from a formula F, we define the binding formula F_B as follows:

$$F_B = \bigwedge_{t,t' \in B} (t^u = t'^u) \leftrightarrow ((t_1^b \leftrightarrow t_1'^b) \wedge \cdots \wedge (t_n^b \leftrightarrow t_n'^b))$$

  where n is the bitwidth of t.
- Use $F \wedge F_B$ when mixing theories

Combining UFP and BVP (cont.)

```
c = ((a % 2) +
     (b % 2)) % 2
c' = (a + b) % 2
d = f * e * c
d' = e * f * c'
```

$$
\begin{aligned}
&(\mathtt{c}^{b} = ((\mathtt{a}^{b} \;\%\; 2^{b}) \\
&\qquad + (\mathtt{b}^{b} \;\%\; 2^{b})) \;\%\; 2^{b}) \wedge \\
&(\mathtt{c'}^{b} = (\mathtt{a}^{b} + \mathtt{b}^{b}) \;\%\; 2^{b}) \wedge \\
&(\mathtt{d}^{u} = \mathtt{f}^{u} * \mathtt{e}^{u} * \mathtt{c}^{u}) \wedge \\
&(\mathtt{d'}^{u} = \mathtt{e}^{u} * \mathtt{f}^{u} * \mathtt{c'}^{u}) \wedge
\end{aligned}
$$

Combining UFP and BVP (cont.)

```
c = ((a % 2) +
     (b % 2)) % 2
c' = (a + b) % 2
d = f * e * c
d' = e * f * c'
```

$$
\begin{aligned}
&(c^b = ((a^b \ \% \ 2^b) \\
&\qquad + (b^b \ \% \ 2^b)) \ \% \ 2^b) \ \wedge \\
&(c'^b = (a^b + b^b) \ \% \ 2^b) \ \wedge \\
&(d^u = f^u * e^u * c^u) \ \wedge \\
&(d'^u = e^u * f^u * c'^u) \ \wedge \\
&\\
&(c^u = c'^u \leftrightarrow \\
&\qquad ((c_1^b \leftrightarrow c'^b_1) \ \wedge \\
&\qquad \ldots \\
&\qquad \wedge \ (c_n^b \leftrightarrow c'^b_n)))
\end{aligned}
$$

Counter-Example-Guided Theory Refinement

1. Encode program in UFP (get formula F), set $F_B = \top$
2. If $F \wedge F_B$ is unsat, return **safe**, else get counter-example CE^b
3. Take first/some/subset of/every clause c from F which is encoded in UFP and if $c^b \wedge CE^b$ is unsat, refine c^u to c^b, update F_B and goto 2
4. If F is all in BVP, return **unsafe**

Experiments

In the paper ;)