Computer-Aided Program Design

Spring 2015, Rice University

Unit 2

Swarat Chaudhuri

January 27, 2015

Symbolic transition systems

- A *symbolic transition system* (STS) is a representation of programs that makes logical reasoning easier.
- Formally, a structure $\mathcal{M} = (Var, In, T)$, where
  - Var is a set of *variables* (atomic propositions). We let $Var' = \{x' \mid x \in Var\}$.
  - In, a propositional logic formula over Var, is an *initial condition*
  - T is a *symbolic transition relation*, defined by a formula over $(Var \cup Var')$.

Questions

- How do you encode a finite automaton as an STS?

Questions

- How do you encode a finite automaton as an STS?
- How do you compile a C program down to this notation?

```
void foo (bool x1, bool x0) {
  assume (x1 | x0);
L0:  y = x0;
L1:  while (!y) {
L2:     if (x1)
L3:         y = x1 & y;
       }
  }
```

Semantics: Concrete transition systems

- The semantics of an STS is given by a “concrete” transition system $(\mathit{States}, \mathit{InitStates}, \dashrightarrow)$:
  - *States* is a set of states, where a state is an interpretation over atomic propositions *Var*.
  - Each state in *InitStates* is an initial state
  - $\dashrightarrow$ is a binary relation over *States*.
- This transition system defines *exactly* how states change.

Semantics: Concrete transition systems

- The semantics of an STS is given by a "concrete" transition system $(\mathit{States}, \mathit{InitStates}, \dashrightarrow)$:
  - *States* is a set of states, where a state is an interpretation over atomic propositions *Var*.
  - Each state in *InitStates* is an initial state
  - $\dashrightarrow$ is a binary relation over *States*.
- This transition system defines *exactly* how states change.
- How do we define such a transition system?

The verification question

The (safety) verification question for an STS is:

> *Let P be a correctness property written in propositional logic.*
> *Is there a state that: (1) does not satisfy P, and (2) reachable from an initial state in the concrete transition system?*

Bounded model checking

Is there a state that: (1) does not satisfy P, and (2) reachable from an initial state in the concrete transition system in k steps or less?

Bounded model checking

- Let us create $(k+1)$ "versions" of each variable x:
- $T_{i,i+1}$ represents T where for each x, x is replaced by x_i and x' is replaced by x_{i+1}.
- In_0 is In with each x replaced by x_0.

Bounded model checking

- Let us create $(k+1)$ "versions" of each variable x:
- $T_{i,i+1}$ represents T where for each x, x is replaced by x_i and x' is replaced by x_{i+1}.
- In_0 is In with each x replaced by x_0.
- The set of states reachable in k steps is captured by:

$$In_0 \wedge T_{0,1} \wedge T_{1,2} \wedge \cdots \wedge T_{(k-1),k}$$

Bounded model checking

- Let us create $(k+1)$ "versions" of each variable x:
- $T_{i,i+1}$ represents T where for each x, x is replaced by x_i and x' is replaced by x_{i+1}.
- In_0 is In with each x replaced by x_0.
- The set of states reachable in k steps is captured by:

$$In_0 \wedge T_{0,1} \wedge T_{1,2} \wedge \cdots \wedge T_{(k-1),k}$$

- The property P fails in one of the cycles $1 \ldots k$:

$$\neg P_0 \vee \neg P_1 \vee \cdots \vee \neg P_k.$$

Bounded model checking

- To find if the property is violated in k steps or less, check satisfiability of:

$$B(k) = In_0 \wedge \bigwedge_{i=0}^{k-1} T_{i,i+1} \wedge \bigvee_{i=0}^{k} \neg P_i$$

Bounded model checking

- To find if the property is violated in k steps or less, check satisfiability of:

$$B(k) = In_0 \wedge \bigwedge_{i=0}^{k-1} T_{i,i+1} \wedge \bigvee_{i=0}^{k} \neg P_i$$

- Can you extract a concrete error trace using this method?

Bounded model checking

- To find if the property is violated in k steps or less, check satisfiability of:

$$B(k) = In_0 \wedge \bigwedge_{i=0}^{k-1} T_{i,i+1} \wedge \bigvee_{i=0}^{k} \neg P_i$$

- Can you extract a concrete error trace using this method?
- An algorithm for verification: iterate over k

Classic example: two-bit counter

- Two bits: l and r

Classic example: two-bit counter

- Two bits: l and r
- Initial condition: $(\neg l \wedge \neg r)$
- Transition: $l' = (l \neq r) \wedge r' = \neg r$
- Property: $(\neg l \vee \neg r)$
- Use bounded model checking to verify or find bugs!

Bounded model checking: completeness

- Is it true that when a bug exists, we can always find it for some k?

Bounded model checking: completeness

- Is it true that when a bug exists, we can always find it for some k?
- Yes, for finite-state systems (what we have here).

Bounded model checking: completeness

- Is it true that when a bug exists, we can always find it for some k?
- Yes, for finite-state systems (what we have here).
- What is the maximum k up to which we need to iterate?

Bounded model checking: completeness

- Is it true that when a bug exists, we can always find it for some k?
- Yes, for finite-state systems (what we have here).
- What is the maximum k up to which we need to iterate?
- *Diameter*: Longest shortest path between an initial state and a reachable state.

Bounded model checking: resources

1. Original paper by Biere et al: `http://repository.cmu.edu/compsci/451/`.
2. The CBMC system: `http://www.cprover.org/cbmc/`.