EECS 219C: Computer-Aided Verification

Model Checking Pushdown Systems

Sanjit A. Seshia
EECS, UC Berkeley

Acknowledgments:
S. Rajamani, S. Schwoon

Today's Lecture

- What are Pushdown Systems?
  - Formal model
- Model Checking Algorithms
  - Reachability Analysis
- Symbolic representation
- LTL Model Checking
- Details in a thesis posted on the webpage

- R. Jhala guest lecture: Application to Software Model Checking

Beyond Finite-State Systems

```
void m() {
    if (?) {
        s(); right();
        if (?) m();
    } else {
        up(); m(); down();
    }
}
```

```
void s() {
    if (?) return;
    up(); m(); down();
}

main() {
  s();
}
```

Need to handle procedure calls and recursion

Beyond Finite-State Systems

```
bool l; /* global variable */

void lock() {
     if (l) ERROR;
        l := 1;
     ... /* acquire a lock */
}

void unlock() {
     if (!l) ERROR;
     ... /* release the lock */
       l := 0;
}
```

```
bool g (bool x) {
     return !x;
}

void main() {
     bool a,b;
     l,a := 0,0;
     ...
     lock();
     b := g(a);
     unlock();
     ...
}
```

Inlining procedure calls might work sometimes

Pushdown Automaton

- Finite set of states plus one stack
  - Stack can grow unbounded
- Instead of states, we talk of configurations
  - Configuration = (State, Stack contents)
- (P, Γ, Δ, c_0)
  - $P \rightarrow$ finite set of states (control locations)
  - $\Gamma \rightarrow$ finite stack alphabet
    - ε denotes empty stack, other symbols: $\gamma_1, \gamma_2, \ldots$
  - $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*) \rightarrow$ transition relation
  - $c_0 \in P \times \Gamma^* \rightarrow$ initial configuration

Transition Relation

- $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$
  - (Current state, top stack symbol) $\rightarrow$ (Next state, new top symbols)
  - In practice, we can think of Δ comprising the following kinds of 'rules' / 'actions':
    - R1: $(p, \gamma) \rightarrow (p', \varepsilon)$ [POP]
    - R2: $(p, \gamma_1) \rightarrow (p', \gamma_2\, \gamma_1)$ [PUSH]
    - R3: $(p, \gamma_1) \rightarrow (p', \gamma_2)$ [SWAP or NOP]
    - R4: $(p, \gamma_1) \rightarrow (p', \gamma_2\, \gamma_3)$ [SWAP + PUSH]
  - In theory: The right hand side can have any finite number of stack symbols (but these are not needed in practice)

Programs as Pushdown Systems

- Given a single-threaded program with variables of finite datatypes (global and local) and procedure calls [no pointers/dynamic memory allocation]
- What are the states P? Stack alphabet Γ?

Infinite-State Systems?

- Pushdown automata are said to be "infinite-state".

Why? Is this true in practice?

Model Checking

- Given a pushdown system, does it satisfy an LTL formula ϕ?
  - We will consider the simple case of reachability analysis
  - $\phi = G\ p$
  - Suppose we want to do explicit-state model checking. What's the challenge?

Representation Issues

- In finite state model checking, we needed to represent (finite sets of) states and transitions
- For pushdown model checking, we need to represent
  - Configurations
  - Transitions
  - (potentially infinite) Sets of them

Need for Symbolic Repn.

- Pushdown model checking inherently needs to be symbolic
  - to be complete (i.e., find all bugs)
    - Representing infinitely many configs.
- Observation: The part that's infinite is the stack
  - View the stack as a word in the language of some finite automaton
  - The set of possible stacks is the language (but we need to define the role of P, too)

Recap of Finite Automata

- A Finite Automaton is a 5-tuple $M = (S, \Sigma, R, S_0, F)$
  - $S \rightarrow$ set of states
  - $\Sigma \rightarrow$ finite alphabet
  - $R \subseteq S \times \Sigma \times S \rightarrow$ transition relation
  - $S_0 \rightarrow$ set of initial states
  - $F \rightarrow$ set of accepting (final) states
- A word $w \in \Sigma^*$ is accepted by M if there's a path $s_0 \xrightarrow{w} f$ with $s_0 \in S_0$ and $f \in F$

Symbolic Representation

- Given pushdown system (P, Γ, Δ, c_0)
- A set of configurations is represented by a finite automaton (S, Σ, R, S_0, F) where
  - $S_0 = P$
  - $S \supseteq P$
  - $\Sigma = \Gamma$
  - Stack configuration (p, w) is represented as a path from initial state p to a final state f with edges labeled with the sequence of symbols in w

Reachability Analysis

- Start with (set of) initial / error state(s)
- Repeatedly compute set of next states, going either
  - Forward (next state operation = “post”)
    - Post(S) = set of states reachable from S in one step of the transition relation
  - Backward (next state operation = “pre”)
    - Pre(S) = set of states that can reach S in one step

Backward Reachability

- C = set of configurations
  - Identified with its finite automaton repn.
- Pre(C) = set of configs that can reach C by applying one rule in transition relation R
- We want to compute Pre*(C)
  - Iteratively compute Pre(C) until no new configurations added
  - Then check if the initial configuration is in Pre*(C)
- Example: C = err config {p, lock() z* lock() z*}

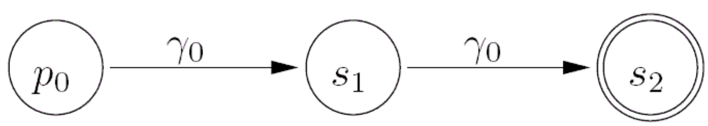
Backward Reachability for Pushdown Systems

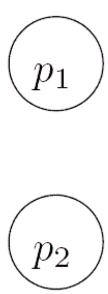
- One step of Pre(C) :
  - Given
    - Rule $(p, \gamma) \rightarrow (p', w)$
    - Path $p' \xrightarrow{w} q$ in C
  - Add an edge $p \xrightarrow{\gamma} q$ to C
- Intuition:
  - If config $c_1 = (p', ww')$ is in C, then given above rule, $c_2 = (p, \gamma w')$ is c_1's predecessor and should be in C

Backward Reachability for Pushdown Systems

- One step of Pre(C) :
  - Given
    - Rule (p, γ) → (p', w)
    - Path $p' \xrightarrow{w} q$ in C
  - Add an edge $p \xrightarrow{\gamma} q$ to C
- Observe: no new states are added!
  - Apart from initial states which are states of the pushdown system (and possibly some other pre-existing states)

Example: C = {p_0, $\gamma_0 \gamma_0$ }

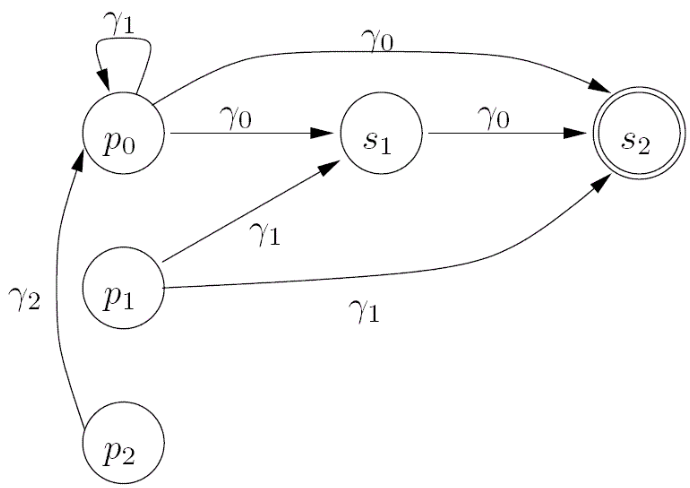




$$\Delta = \{r_1, r_2, r_3, r_4\}$$

$$r_1 = \langle p_0, \gamma_0 \rangle \hookrightarrow \langle p_1, \gamma_1\gamma_0 \rangle$$
$$r_2 = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_2, \gamma_2\gamma_0 \rangle$$
$$r_3 = \langle p_2, \gamma_2 \rangle \hookrightarrow \langle p_0, \gamma_1 \rangle$$
$$r_4 = \langle p_0, \gamma_1 \rangle \hookrightarrow \langle p_0, \varepsilon \rangle$$

Pre*(C) for the Example



Rules in pre* computation

- 3 kinds of rules:
  - $(p, \gamma) \rightarrow (q, \varepsilon)$
    - Add edge (p, γ, q)
  - $(p, \gamma) \rightarrow (q, \gamma')$
    - Add edge (p, γ, q') for each (q, γ', q')
  - $(p, \gamma) \rightarrow (q, \gamma_1 \gamma_2)$
    - Add edge (p, γ, q'') for each $\{(q, \gamma_1, q'),\ (q', \gamma_2, q'')\}$
- How many times do we need to process each kind of rule?

Rules in pre* computation

- 3 kinds of rules:
  - $(p, \gamma) \rightarrow (q, \varepsilon)$ JUST ONCE
    - Add edge (p, γ, q)
  - $(p, \gamma) \rightarrow (q, \gamma')$ POSSIBLY MANY TIMES
    - Add edge (p, γ, q') for each (q, γ', q')
  - $(p, \gamma) \rightarrow (q, \gamma_1 \gamma_2)$
    - Add edge (p, γ, q'') for each $\{(q, \gamma_1, q'), (q', \gamma_2, q'')\}$
- How many times do we need to process each kind of rule?

S. A. Seshia 21

Complexity of Pre*(C)

- N = number of states in C
- K = size of stack alphabet
- M = number of rules for pushdown system
- Assume we cycle through the rules on each iteration, adding edges if any match
- What's the asymptotic running time of the Pre*(C) computation?

S. A. Seshia 22

Complexity of Pre*

- Turns out we can do better if we iterate over edges rather than rules
- O(N^2 M)
- Key is to process each edge just once
  - Iterate through all rules that match that edge
  - Add new 1-symbol RHS rules that correspond to 2-symbol RHS rules matching that edge
  - Details in Schwoon's PhD thesis (posted online)

Schwoon's Pre* Algorithm

Algorithm 1

Input: a pushdown system $\mathcal{P} = (P, \Gamma, \Delta, c_0)$;
a $\mathcal{P}$-Automaton $\mathcal{A} = (\Gamma, Q, \rightarrow_0, P, F)$ without transitions into P

Output: the set of transitions of $\mathcal{A}_{pre^*}$

$rel := \emptyset;\ trans := \rightarrow_0;\ \Delta' := \emptyset;$
for all $\langle p, \gamma \rangle \hookrightarrow \langle p', \varepsilon \rangle \in \Delta$ **do** $trans := trans \cup \{(p, \gamma, p')\};$
while $trans \neq \emptyset$ **do**
  pop $t = (q, \gamma, q')$ from $trans$;
  if $t \notin rel$ **then**
    $rel := rel \cup \{t\};$
    for all $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma \rangle \in (\Delta \cup \Delta')$ **do**
      $trans := trans \cup \{(p_1, \gamma_1, q')\};$
    for all $\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma\gamma_2 \rangle \in \Delta$ **do**
      $\Delta' := \Delta' \cup \{\langle p_1, \gamma_1 \rangle \hookrightarrow \langle q', \gamma_2 \rangle\};$
      for all $(q', \gamma_2, q'') \in rel$ **do**
        $trans := trans \cup \{(p_1, \gamma_1, q'')\};$
return rel

Figure 3.3: An algorithm for computing pre^*.

Forward Reachability Analysis

- Start with initial config (c_0, ε)
  - Single state finite automaton representation
- Post(C) = set of configs reached from C by applying one rule in transition relation R
- We want to compute Post*(C)
  - Iteratively compute Post(C) until no new configurations added
  - Then check if the error configuration is in Post*(C)

Computing Post*(C)

- One step of Post(C) :
  - Given
    - Rule (p, γ) → (p', w)
    - Path $p \xrightarrow{\gamma} q$ in C (*path* because of ε-moves)
  - If w = ε add edge (p', ε, q)
  - If w = γ' add edge (p', γ', q)
  - If w = $\gamma' \gamma''$
    - add a *new state* $s_{p'\gamma'}$
    - add (p', γ', $s_{p'\gamma'}$) and ($s_{p'\gamma'}$, γ'',q)

Computing Post*(C)

- One step of Post(C) :
  - Given
    - Rule $(p, \gamma) \rightarrow (p', w)$
    - Path $p \xrightarrow{\gamma} q$ in C (path because of ε-moves)
  - If $w = \varepsilon$ add edge (p', ε, q)
  - If $w = \gamma'$ add edge (p', γ', q)
  - If $w = \gamma' \gamma''$
    - add a *new state* $s_{p'\gamma'}$
    - add $(p', \gamma', s_{p'\gamma'})$ and $(s_{p'\gamma'}, \gamma'', q)$
- How many new states might we add?

Exercise: Compute Post*(C) for previous example

More Symbolic Representation

- Notice that the rules are "explicit-state"
- Typically these can be represented symbolically
  - $p \in P$ is a pair (pc, g)
    - pc = prog counter, g – global variables
  - $\gamma \in \Gamma$ is a pair (proc, l)
    - proc – procedure calls/returns, l – local variables
  - Rule's behavior on global/local variables can be represented as a relation R(<g,l>,<g',l'>) by a Boolean function

More Symbolic Representation

- Rules can be represented symbolically
  - $p \in P$ is a pair (pc, g)
  - $\gamma \in \Gamma$ is a pair (proc, l)
  - Rule's behavior on global/local variables can be represented as a relation R(<g,l>,<g',l'>) by a Boolean function
- Set of configs encoded by a finite automaton with expanded alphabet
  - Edges are labeled with these Boolean functions (BDDs) representing next-state relations

Symbolically Computing Pre*

If $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle$ and $p' \xrightarrow{w}{}^{*} q$, then add $p \xrightarrow{\gamma} q$.

(i) If $\langle p, \gamma \rangle \underset{[R]}{\hookrightarrow} \langle p', \varepsilon \rangle$, then add $p \xrightarrow[{[R]}]{\gamma} p'$.

(ii) If $\langle p, \gamma \rangle \underset{[R]}{\hookrightarrow} \langle p', \gamma' \rangle$ and $p' \xrightarrow[{[R_1]}]{\gamma'} q$, then add $p \xrightarrow[{[R']}]{\gamma} q$ where
$R' = \{ (g, l, g_1) \mid \exists g_0, l_1 \colon (g, l, g_0, l_1) \in R \wedge (g_0, l_1, g_1) \in R_1 \}$.

(iii) If $\langle p, \gamma \rangle \underset{[R]}{\hookrightarrow} \langle p', \gamma' \gamma'' \rangle$ and $p' \xrightarrow[{[R_1]}]{\gamma'} q' \xrightarrow[{[R_2]}]{\gamma''} q$, then add $p \xrightarrow[{[R']}]{\gamma} q$ where
$R' = \{ (g, l, g_2) \mid \exists g_0, l_1, g_1, l_2 \colon (g, l, g_0, l_1) \in R \wedge (g_0, l_1, g_1) \in R_1 \wedge (g_1, l_2, g_2) \in R_2 \}$.

LTL Model Checking

- Similar strategy to finite-state systems
- Convert negation of LTL formula into Buchi automaton
- Construct product of Pushdown system P and Buchi automaton B
  - Transitions of both are synchronized
  - Accepting state of product has control part of P's configuration as accepting state of B
    - Check if such a config. occurs infinitely often
- Run-time: $O(|P|^2 \,.\, |B|^3 \,.\, |\Delta|)$

Next class

- Game Theory and Verification
  - Modeling open systems
  - Controller synthesis