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();
}

l := 0;
l := 1;

Inlining procedure calls might work sometimes

Beyond Finite-State Systems

bool l; /* global variable */

void lock() {
    if (l) ERROR;
    l := l;
    ... /* 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)
- \((P, \Gamma, \Delta, c_0)\)
  - \(P\) \(\rightarrow\) finite set of states (control locations)
  - \(\Gamma\) \(\rightarrow\) finite stack alphabet
    - \(\varepsilon\) 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 \(\Delta\) 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 $\Gamma$?

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 $\phi$?
  – 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 \( M = (S, \Sigma, R, S_0, F) \)
  - \( S \) → set of states
  - \( \Sigma \) → finite alphabet
  - \( R \subseteq S \times \Sigma \times S \) → transition relation
  - \( S_0 \) → set of initial states
  - \( F \) → 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 \((P, \Gamma, \Delta, c_0)\)
• A set of configurations is represented by a finite automaton \((S, \Sigma, R, S_0, F)\) 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”)  
    • \(\text{Post}(S) = \text{set of states reachable from } S \text{ in one step of the transition relation} \)
  – **Backward** (next state operation = “pre”)  
    • \(\text{Pre}(S) = \text{set of states that can reach } S \text{ in one step} \)
Backward Reachability

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

Backward Reachability for Pushdown Systems

- One step of $\text{Pre}(C)$:
  - 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 $c_1 = (p', w w')$ 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 \((p, \gamma) \rightarrow (p', w)\)
    • 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: \(C = \{p_0, \gamma_0 \gamma_0\}\)

\[\Delta = \{r_1, r_2, r_3, r_4\}\]

\[r_1 = \langle p_0, \gamma_0 \rangle \rightarrow \langle p_1, \gamma_1 \gamma_0 \rangle\]

\[r_2 = \langle p_1, \gamma_1 \rangle \rightarrow \langle p_2, \gamma_2 \gamma_0 \rangle\]

\[r_3 = \langle p_2, \gamma_2 \rangle \rightarrow \langle p_0, \gamma_1 \rangle\]

\[r_4 = \langle p_0, \gamma_1 \rangle \rightarrow \langle p_0, \varepsilon \rangle\]
**Pre*(C) for the Example**

**Rules in pre* computation**

- 3 kinds of rules:
  - $(p, \gamma) \rightarrow (q, \varepsilon)$
    - Add edge $(p, \gamma, q)$
  - $(p, \gamma) \rightarrow (q, \gamma')$
    - Add edge $(p, \gamma, q')$ for each $(q, \gamma', q')$
  - $(p, \gamma) \rightarrow (q, \gamma_1 \gamma_2)$
    - Add edge $(p, \gamma, q'')$ 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:
  – \((p, \gamma) \rightarrow (q, \varepsilon)\) \(\rightarrow \) JUST ONCE
    • Add edge \((p, \gamma, q)\)
  – \((p, \gamma) \rightarrow (q, \gamma')\) \(\rightarrow \) POSSIBLY MANY TIMES
    • Add edge \((p, \gamma, q')\) for each \((q, \gamma', q')\)
  – \((p, \gamma) \rightarrow (q, \gamma_1 \gamma_2)\)
    • Add edge \((p, \gamma, q'')\) for each \\{(q, \gamma_1, q'), (q', \gamma_2, q'')\}\)

• How many times do we need to process each kind of rule?

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?
Complexity of Pre*

- Turns out we can do better if we iterate over edges rather than rules
- $O(N^2M)$
- 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 $\mathcal{P} = (P, \Gamma, \Delta, q_0)$; a $\mathcal{P}$-Automaton $A = (\Gamma, Q, \rightarrow_0, P, F)$ without transitions into $P$

**Output:** the set of transitions of $A_{pre*}$

1. $\text{rel} := \emptyset$; $\text{trans} := \emptyset$; $\Delta' := \emptyset$
2. for all $(p, \gamma) \rightarrow (p', \epsilon) \in \Delta$ do $\text{trans} := \text{trans} \cup \{(p, \gamma, p')\}$
3. while $\text{trans} \neq \emptyset$ do
4. $\text{pop} t = (q, \gamma, q')$ from $\text{trans}$
5. if $t \notin \text{rel}$ then
6. $\text{rel} := \text{rel} \cup \{t\}$
7. for all $(p_1, \gamma_1) \rightarrow (q, \gamma) \in (\Delta \cup \Delta')$ do $\text{trans} := \text{trans} \cup \{(p_1, \gamma_1, q')\}$
8. for all $(p_1, \gamma_1) \rightarrow (q, \gamma, q_2) \in \Delta$ do $\Delta' := \Delta' \cup \{(p_1, \gamma_1) \rightarrow (q', \gamma_2)\}$
9. for all $(q', \gamma_2, q'') \in \text{rel}$ do $\text{trans} := \text{trans} \cup \{(p_1, \gamma_1, q'')\}$
10. return $\text{rel}$

Figure 3.3: An algorithm for computing $\text{pre}^*$. 
Forward Reachability Analysis

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

Computing \(\text{Post}^*(C)\)

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

- **One step of $\text{Post}(C)$:**
  - Given
    - Rule $(p, \gamma) \rightarrow (p', w)$
    - Path $p \xrightarrow{\gamma} q$ in $C$ (path because of $\varepsilon$-moves)
  - If $w = \varepsilon$ add edge $(p', \varepsilon, q)$
  - If $w = \gamma'$ add edge $(p', \gamma', q)$
  - 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 $\text{Post}^*(C)$ 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 $(pc, g)$
    - $pc = \text{prog counter, } g = \text{global variables}$
  - $\gamma \in \Gamma$ is a pair $(proc, l)$
    - $proc = \text{procedure calls/returns, } l = \text{local variables}$
  - 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 $\text{Pre}^*$

If $\langle p, \gamma \rangle \xrightarrow{\delta} \langle p', \gamma' \rangle$ and $p' \xrightarrow{\delta'} q$, then add $p \xrightarrow{\gamma} q$.

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

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

(iii) If $\langle p, \gamma \rangle \xrightarrow{[R]} \langle p', \gamma' \gamma'' \rangle$ and $p' \xrightarrow{[\Gamma]} q'$, then add $p \xrightarrow{\gamma} q$ where $R' = \{ (g, l, g_2) \mid \exists g_0, l_1, g_1, l_2 : (g, l, g_0, l_1) \in R \land (g_0, l_1, g_1) \in R_1 \land (g_1, l_2, g_2) \in R_2 \}$. 
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: $O(|P|^2 \cdot |B|^3 \cdot |\Delta|)$

Next class

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