Program verification 2019

Lecture 4: Automated reachability

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2019-01-15

Topic 4.1

Concrete model checking - enumerate reachable states

Isn't enumeration impossible?

- Explore the transition graph explicitly, light weight machinery
- If edge labels are guarded commands then finding next values are trivial
- After resolving non-determinism, concrete model checking reduces to program execution
- May be only finitely many states are reachable
- May be impossible to cover all states explicitly, but it may cover a portion of interest
- Useful for learning design principles of computing reachable states

Concrete model checking

Algorithm 4.1: Concrete model checking

Input: $P = (V, L, \ell_0, \ell_e, E)$
Output: $\textsc{Safe}$ if P is safe, $\textsc{Unsafe}$ otherwise

$reach := \emptyset$;
$worklist := \{(\ell_0, v) | v \in \mathbb{Z}^{|V|}\}$;
while $worklist \neq \emptyset$ **do**
  choose $(\ell, v) \in worklist$; *(the choice defines the nature of exploration)*
  $worklist := worklist \setminus \{(\ell, v)\}$;
  if $(\ell, v) \notin reach$ **then**
    $reach := reach \cup \{(\ell, v)\}$;
    foreach $(\ell, F(V, V'), \ell') \in E$ **do**
      $worklist := worklist \cup \{(\ell', v') | F(v, v')\}$;
if $(\ell_e, \_) \in reach$ **then**
  return $\textsc{Unsafe}$
else
  return $\textsc{Safe}$

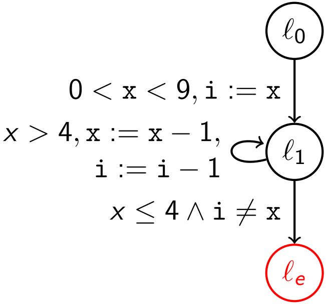
Exercise 4.1

Suggest improvements in the algorithm

Example: concrete model checking

Example 4.1

Consider the following example



Let $V = [\mathtt{x}, \mathtt{i}]$

Initialization:
$reach = \emptyset$, $worklist = \{(\ell_0, v) | v \in \mathbb{Z}^2\}$

Choose a state:
Lets choose $(\ell_0, [8, 0])$

Update worklist:
$worklist := worklist \setminus \{(\ell_0, [8, 8])\}$

Add successors in worklist if state not visited:
$worklist := worklist \cup \{(\ell_1, [8, 8])\}$
$reach := reach \cup \{(\ell_0, [8, 0])\}$

... go back to choosing a new state from worklist

Search strategies

- DFS
- BFS
- A^*
  - worklist is a priority queue,
  - weights are assigned to states based on estimate on possibility of reaching error

Exercise 4.2

Describe A^* search strategy

Optimizations: exploiting structure

- Symmetry reduction
- Assume guarantee
- Partial order reduction (for concurrent systems)

Optimizations: reducing space

- hashed states - reach set contains hash of states (not sound)
- Stateless exploration - no reach set (redundant)

Trade-off among time, space, and soundness

Exercise 4.3

Write concrete model checking using hash tables

Proof and counterexample

Definition 4.1

A proof of a program is an object that allows one to check safety of the program using a low complexity (preferably linear) algorithm in the size of the object.

Example 4.2

In our concrete model checking algorithm, reach set is the proof. The checker needs to find that no more states can be reached from reach.

Definition 4.2

A counterexample of a program is an execution that ends at ℓ_e.

A verification method may produce three possible outcomes for a program

- proof
- counterexample
- unknown or non-termination

Enabling counterexample generation

Algorithm 4.2: Concrete model checking

Input: $P = (V, L, \ell_0, \ell_e, E)$

Output: $\textsc{Safe}$ if P is safe, $\textsc{Unsafe}$ otherwise

$reach := \emptyset$; $parents := \lambda x.\textsc{NaN}$;

$worklist := \{(\ell_0, v) | v \in \mathbb{Z}^{|V|}\}$;

while $worklist \neq \emptyset$ **do**

    choose $(\ell, v) \in worklist$;

    $worklist := worklist \setminus \{(\ell, v)\}$;

    if $(\ell, v) \notin reach$ **then**

        $reach := reach \cup \{(\ell, v)\}$;

        foreach v' *s.t.* $F(v, v')$ *is sat* $\wedge (\ell, F(V, V'), \ell') \in E$ **do**

            $worklist := worklist \cup \{(\ell', v')\}$; $parents((\ell', v')) := (\ell, v)$;

if $(\ell_e, v) \in reach$ **then**

    return $\textsc{Unsafe}$(*traverseToInit*($parents, (\ell_e, v)$))

else

    return $\textsc{Safe}$

Exercise 4.4
add data structure to report counterexample

Topic 4.2

Symbolic methods

Why symbolic?

To avoid, state explosion problem

Symbolic methods

Now, we cover some methods that try/avoid to compute lfp

- Symbolic model checking
- Constraint based invariant generation

Symbolic state

Definition 4.3

A symbolic state s of $P = (V, L, \ell_0, \ell_e, E)$ *is a pair* (ℓ, F)*, where*

- $\ell \in L$
- F *is a formula over variables* V *in a given theory*

Symbolic model checking

Algorithm 4.3: Symbolic model checking

Input: $P = (V, L, \ell_0, \ell_e, E)$
Output: Safe if P is safe, Unsafe otherwise

$reach : L \to \Sigma(V) := \lambda x.\bot;$
$worklist := \{(\ell_0, \top)\};$
while $worklist \neq \emptyset$ **do**
 choose $(\ell, F) \in worklist;$
 $worklist := worklist \setminus \{(\ell, F)\};$
 if $\neg(F \Rightarrow reach(\ell))$ *is sat* **then**
  $reach := reach[\ell \mapsto reach(\ell) \vee F];$
  foreach $(\ell, \rho(V, V'), \ell') \in E$ **do**
   $worklist := worklist \cup \{(\ell', sp(F, \rho))\};$

if $reach(\ell_e) \neq \bot$ **then**
 return Unsafe
else
 return Safe

Note: We need efficient implementations of various logical operators!

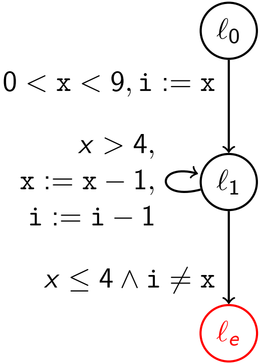
Exercise 4.5

Give a condition for definite termination?

Example: symbolic model checking

Example 4.3

Consider the following example



Let $V = [\mathtt{x}, \mathtt{i}]$

Init: $reach = \lambda x.\bot$, $worklist = \{(\ell_0, \top)\}$

Choose a state: $(\ell_0, \top)$ *(only choice)*

Update worklist: $worklist := \emptyset$

Add successors in worklist:

Since $\neg(\top \Rightarrow reach(\ell_0))$ *is sat,*

$worklist := worklist \cup \{(\ell_1, 0 < x = i < 9)\}$

$reach(\ell_0) := reach(\ell_0) \vee \top := \top$

Again choose a state: $\{(\ell_1, 0 < x = i < 9)\}$

Update worklist: $worklist := \emptyset$

Add successors in worklist:

Since $\neg(0 < x = i < 9 \Rightarrow reach(\ell_1))$ *is sat,*

$worklist := worklist \cup \{(\ell_1, 3 < x = i < 9), (\ell_e, \bot)\}$

$reach(\ell_1) := reach(\ell_1) \vee 0 < x = i < 9$

$reach(\ell_e) := reach(\ell_e) \vee \bot$

Exercise 4.6

complete the run of the algorithm

Proof generation

If the symbolic model checker terminates with the answer SAFE, then it must also report a proof of the safety, which is the *reach* map.

It has implicitly computed a Hoare style proof of $P = (V, L, \ell_0, \ell_e, E)$.

$$(\ell, \rho(V, V'), \ell') \in E \quad \{reach(\ell)\}\rho(V, V')\{reach(\ell')\}$$

If an LTS program has been obtained from a simple language program then one may generate a Hoare style proof system.

Exercise 4.7

Describe the construction for the above translation

Topic 4.3

Constraint based invariant generation

Invariant generation using constraint solving

Invariant generation: find a safe inductive invariant map I

- This is our first method that computes the fixed point automatically without resorting to some kind of enumeration

Templates

Let $L = \{l_0, \ldots, l_n, l_e\}$,
Let $V = \{x_1, \ldots, x_m\}$

We assume the following templates for each invariant in the invariant map.

$$\mathrm{I}(l_0) = 0 \leq 0$$
$$\forall i \in 1..n.\ \mathrm{I}(l_i) = (p_{i1}x_1 + \ldots p_{im}x_m \leq p_{i0})$$
$$\mathrm{I}(l_e) = 0 \leq -1$$

p_{ij} are called parameters to the templates and they define a set of candidate invariants.

Constraint generation

A safe inductive invariant map I must satisfy for all $(l_i, \rho, l_{i'}) \in E$

$$sp(\mathrm{I}(l_i), \rho) \Rightarrow \mathrm{I}(l_{i'}).$$

The above condition translates to

$$\forall V, V'. \, (p_{i1}x_1 + \ldots p_{im}x_m \leq p_{i0}) \wedge \rho(V, V') \Rightarrow (p_{i'1}x'_1 + \ldots p_{i'm}x'_m \leq p_{i'0})$$

Our goal is to find p_{ij}s such that the above constraints are satisfied. Unfortunately there is quantifier alternation in the constraints. Therefore, they are hard to solve.

Constraint solving using Farkas lemma

If all ρs are linear constraints then we can use Farkas lemma to turn the validity question into a "conjunctive satisfiablity question"

Lemma 4.1

For a rational matrix A, vectors a and b, and constant c,
$\forall X.\ AX \leq b \Rightarrow aX \leq c$ *iff*
$\exists \lambda \geq 0.\ \lambda^T A = a$ *and* $\lambda^T b \leq c$

Application of farkas lemma

Consider $(l_i, (AV + A'V \leq b), l_{i'}) \in E$

After applying Farkas lemma on

$$\forall V, V'.\ (p_{i1}x_1 + \ldots p_{im}x_m \leq p_{i0}) \wedge \rho(V, V') \Rightarrow (p_{i'1}x'_1 + \ldots p_{i'm}x'_m \leq p_{i'0}),$$

we obtain

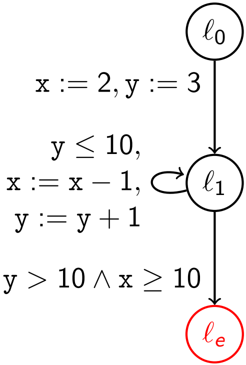
$$\exists \lambda_0, \lambda.\ (\lambda_0[p_{i1}, \ldots, p_{im}] + \lambda^T A) = 0 \wedge \lambda^T A' = [p_{i'1}, \ldots, p_{i'm}] \wedge \\ \lambda_0 p_{i0} + \lambda^T b \leq p_{i'0}$$

All the variables p_{ij}s and λs are existentially quantified, which can be solved by a quadratic constraints solver.

Example: invariant generation

Example 4.4

Consider the following example



Let $V = [\mathrm{x}, \mathrm{y}]$

We assume the following invariant template at ℓ_1*:*
$\mathrm{I}(\ell_1) = (p_1\mathrm{x} + p_2\mathrm{y} \leq p_0)$

We generate the following constraints for program transitions:

For ℓ_0 *to* ℓ_1*,*
$\forall \mathrm{x}', \mathrm{y}'.\ \mathrm{x}' = 2 \land \mathrm{y}' = 3 \Rightarrow (p_1\mathrm{x}' + p_2\mathrm{y}' \leq p_0)$

For ℓ_1 *to* ℓ_1*,*
$$\forall \mathrm{x}, \mathrm{y}, \mathrm{x}', \mathrm{y}'.\ (p_1\mathrm{x} + p_2\mathrm{y} \leq p_0) \land \mathrm{y} \leq 10 \land \mathrm{x}' = \mathrm{x} - 1 \land \mathrm{y}' = \mathrm{y} + 1 \Rightarrow (p_1\mathrm{x}' + p_2\mathrm{y}' \leq p_0)$$

For ℓ_1 *to* ℓ_e*,*
$\forall \mathrm{x}, \mathrm{y}.\ (p_1\mathrm{x} + p_2\mathrm{y} \leq p_0) \land \mathrm{y} > 10 \land \mathrm{x} \geq 10 \Rightarrow \bot$

Example: invariant generation(contd.)

Now consider the second constraint:

$\forall \mathrm{x}, \mathrm{y}, \mathrm{x}', \mathrm{y}'.$

$(p_1\mathrm{x} + p_2\mathrm{y} \leq p_0) \wedge \mathrm{y} \leq 10 \wedge \mathrm{x}' = \mathrm{x} - 1 \wedge \mathrm{y}' = \mathrm{y} + 1 \Rightarrow (p_1\mathrm{x}' + p_2\mathrm{y}' \leq p_0)$

Matrix view of the transition relation $\mathrm{y} \leq 10 \wedge \mathrm{x}' = \mathrm{x} - 1 \wedge \mathrm{y}' = \mathrm{y} + 1$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{x}' \\ \mathrm{y}' \end{bmatrix} \leq \begin{bmatrix} 10 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

Example: invariant generation(contd.)

Applying farkas lemma on the constraint, we obtain

$$\begin{bmatrix} \lambda_0 & \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \end{bmatrix} \begin{bmatrix} p_1 & p_2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & p_1 & p_2 \end{bmatrix}$$

$$\begin{bmatrix} \lambda_0 & \lambda_1 & \lambda_2 & \lambda_3 & \lambda_4 & \lambda_5 \end{bmatrix} \begin{bmatrix} p_0 \\ 10 \\ 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \leq \begin{bmatrix} p_0 \end{bmatrix}$$

Exercise 4.8

Apply farkas lemma on the other two implications

$\forall \mathrm{x}', \mathrm{y}'.\ \mathrm{x}' = 2 \wedge \mathrm{y}' = 3 \Rightarrow (p_1\mathrm{x}' + p_2\mathrm{y}' \leq p_0)$

$\forall \mathrm{x}, \mathrm{y}.\ (p_1\mathrm{x} + p_2\mathrm{y} \leq p_0) \wedge \mathrm{y} > 10 \wedge \mathrm{x} \geq 10 \Rightarrow \bot$

Does this method work?

- Quadratic constraint solving does not scale
- For small tricky problems, this method may prove to be useful

Topic 4.4

Problems

End of Lecture 4