

CS766: Analysis of concurrent programs 2020

Lecture 4: Labeled transition systems and invariants

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2020-01-27

Topic 4.1

Program as labeled transition system

A more convenient program model

- ▶ Simple language has many cases to write an algorithm
 - ▶ or any other language, we may consider
- ▶ automata like program models **allow more succinct description** of verification methods
- ▶ Let us look one of those.

Program as labeled transition system (LTS)

Definition 4.1

A program P is a tuple

$$(V, L, \ell_0, \ell_e, E),$$

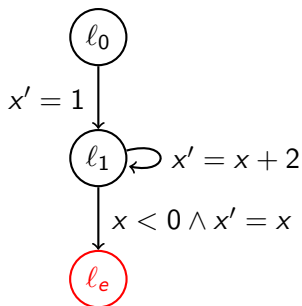
where

- ▶ V is a vector of variables,
- ▶ L be set of program locations,
- ▶ $\ell_0 \in L$ is initial location,
- ▶ $\ell_e \in L$ is error location, and
- ▶ $E \subseteq L \times \Sigma(V, V') \times L$ is a set of labeled transitions between locations.

Example: LTS

Example 4.1

Consider an LTS $P = \{[x], \{l_0, l_1, l_e\}, l_0, l_e, E\}$



$$E = \{ (l_0, x' = 1, l_1), (l_1, x' = x + 2, l_1), (l_1, x < 0 \wedge x' = x, l_e) \}$$

Shorthand notation for handling transitions

If $e = (\ell, \rho(V, V'), \ell') \in E$, then let us define

$$e(V, V') \triangleq \rho(V, V'), \quad e(\text{loc}) \triangleq \ell, \text{ and} \quad e(\text{loc}') \triangleq \ell'.$$

Example 4.2

Let $e = (\ell_1, \mathbf{x}' = \mathbf{x} + 2, \ell_2) \in E$.

$e(V, V')$ denotes $\mathbf{x}' = \mathbf{x} + 2$.

$e(\text{loc})$ denotes ℓ_1 .

$e(\text{loc}')$ denotes ℓ_2 .

Cumbersome labels

The labels in LTS are cumbersome to write.

Example 4.3

Let $V = [x, y, z]$.

For statement $x := 1$, we have to add the following label in LTS.

$$x' = 1 \wedge y' = y \wedge z' = z.$$

Guarded command

Definition 4.2

A *guarded command* is a pair of

- ▶ a formula in $\Sigma(V)$ and (called guard)
- ▶ a sequence of update constraints (including havoc) of variables in V .
(called command)

Example 4.4

Let $V = [x, y, z]$.

$(x > y, [x := x + 1, z := \text{havoc}()])$ is a guarded command.

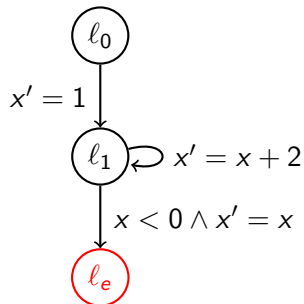
The formula represented by the guarded command is

$$x > y \wedge x' = x + 1 \wedge y' = y.$$

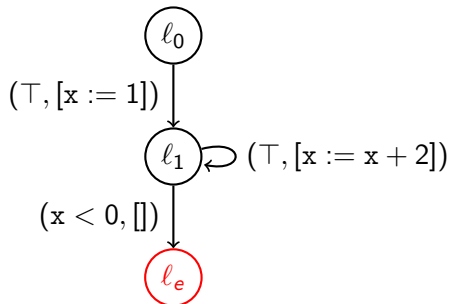
Guarded command is a convenient way of writing transitions.

Example: guarded command

Example 4.5



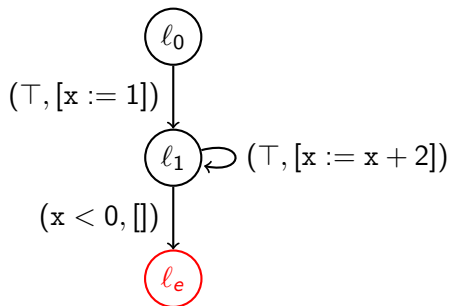
LTS with formulas



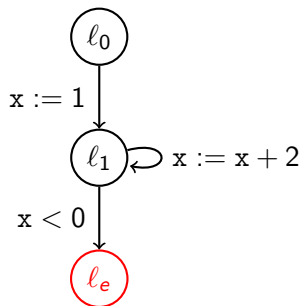
LTS with guarded commands

Further shorthanded view

Example 4.6



Guarded command



Simplified guarded commands

Trivial, guards and updates need not be explicitly written.

Semantics: state of LTS

Consider program $P = (V, L, \ell_0, \ell_e, E)$.

Definition 4.3

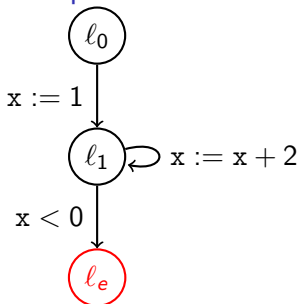
A **state** $s = (\ell, v)$ of a program is program location ℓ and a valuation v of V .

Notation:

Let $v(x) \triangleq$ value of variable x in v .

For state $s = (\ell, v)$, let $s(x) \triangleq v(x)$ and $s(\text{loc}) \triangleq \ell$.

Example 4.7



$(\ell_1, [2])$ is a state.

$s = (\ell_e, [19])$ is a state.

We will write $s(x) = 19$ and $s(\text{loc}) = \ell_e$.

Path

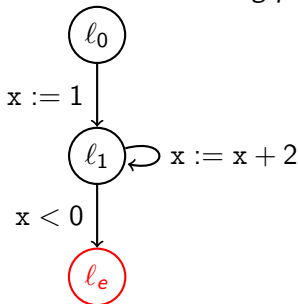
Definition 4.4

A *path* $\pi = e_1, \dots, e_n$ in P is a sequence of transitions such that, for each $0 < i < n$,

$$e_i = (\ell_{i-1}, -, \ell_i) \quad \text{and} \quad e_{i+1} = (\ell_i, -, \ell_{i+1}).$$

Example 4.8

Consider the following program P .



$(\ell_0, x := 1, \ell_1), (\ell_1, x := x + 2, \ell_1)$ is a path.
 $(\ell_1, x < 0, \ell_e)$ is a path.

Exercise 4.1

Is the following a path of P ?

- ▶ $(\ell_0, x < 0, \ell_e)$
- ▶ $(\ell_1, x := x + 2, \ell_1), (\ell_0, x := 1, \ell_1)$

Execution of paths

Definition 4.5

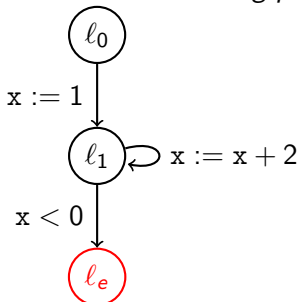
An *execution* corresponding to path $\pi = e_1, \dots, e_n$ is a sequence of states

$$(\ell_0, v_0), \dots, (\ell_n, v_n)$$

such that $\forall i \in 1..n, e_i(v_{i-1}, v_i)$ holds true.

Example 4.9

Consider the following program P .



Path $(\ell_0, x := 1, \ell_1), (\ell_1, x := x + 2, \ell_1)$ has the following execution.

$$(\ell_0, [-234]), (\ell_1, [1]), (\ell_1, [3])$$

Exercise 4.2

Give an execution for a path that reaches l_e .

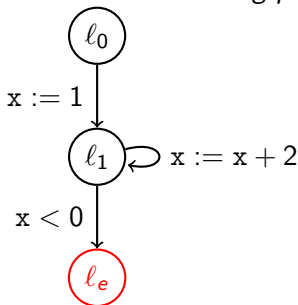
Feasibility of paths

Definition 4.6

A path $\pi = e_1, \dots, e_n$ is *feasible* if there is an execution corresponding to the path.

Example 4.10

Consider the following program P .



Path $(l_0, x := 1, l_1), (l_1, x := x + 2, l_1)$ is feasible, since we have seen an execution along the path.

Exercise 4.3

Give an infeasible path?

Execution of program

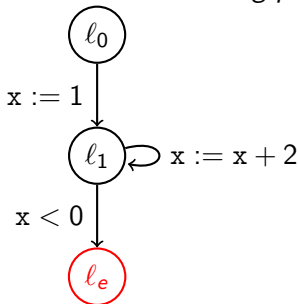
Definition 4.7

An execution s_0, \dots, s_n *belongs to* P if

- ▶ $s_0(\text{loc}) = \ell_0$ and
- ▶ there is a corresponding path in P .

Example 4.11

Consider the following program P .



$(\ell_0, [-234]), (\ell_1, [1]), (\ell_1, [3])$ is an execution of P and the corresponding path is

$(\ell_0, x := 1, \ell_1), (\ell_1, x := x + 2, \ell_1)$.

Exercise 4.4

Give an execution of P that reaches ℓ_e .

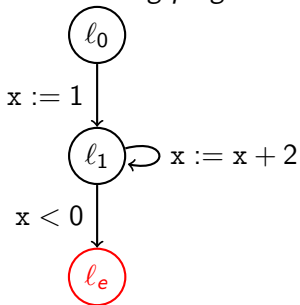
Safety in LTS

Definition 4.8

P is **safe** if there is no execution of P that reaches to l_e .

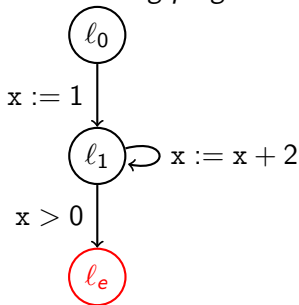
Example 4.12

The following program is **safe**



Example 4.13

The following program is **not safe**



Path constraints

$V_i \triangleq$ variable vector obtained by adding subscript i after each variable in V .

Definition 4.9

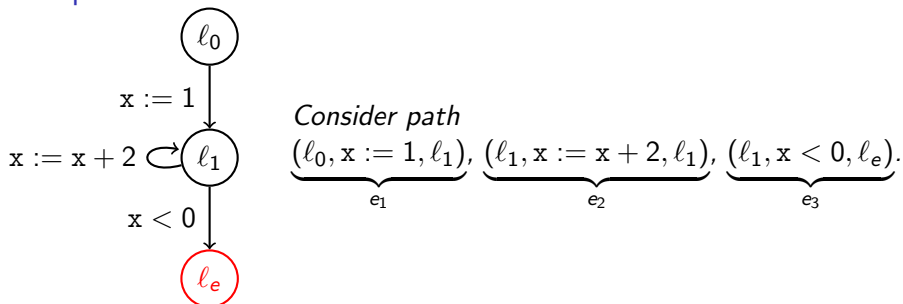
For a path $\pi = e_1, \dots, e_n$, *path constraints* $\rho(\pi)$ is

$$\bigwedge_{i \in 1..n} e_i(V_{i-1}, V_i).$$

Path constraints are also known as “SSA formulas”

Example: path constraints

Example 4.14



Path constraint for the path is

$$\rho(e_1 e_2 e_3) = (x_1 = 1 \wedge x_2 = x_1 + 2 \wedge x_2 < 0 \wedge x_3 = x_2).$$

Since F is unsat, there is no execution along the path.

Exercise 4.5

Give $\rho(e_1 e_2 e_2)$

Path constraints feasible

Theorem 4.1

If path constraints of a path is satisfiable, then there is an execution that corresponds to the path.

Proof.

We can easily generate the execution from the satisfying assignment. □

Example 4.15

Consider path constraints for $\rho(e_1 e_2 e_2)$ in our running example

$$\rho(e_1 e_2 e_2) = (x_1 = 1 \wedge x_2 = x_1 + 2 \wedge x_3 = x_2 + 2).$$

A satisfying assignment to $\rho(e_1 e_2 e_2)$ is

$$\{x_0 \rightarrow -12030, x_1 \rightarrow 1, x_2 \rightarrow 3, x_3 \rightarrow 5\}.$$

symbolic strongest post over edges

Recall,

$$sp : \Sigma(V) \times \Sigma(V, V') \rightarrow \Sigma(V)$$

We define symbolic post over labels of P as follows.

$$sp(F, \rho) \triangleq (\exists V. F(V) \wedge \rho(V, V'))[V/V']$$

Using polymorphism, we also define sp over edges of LTSs.

Definition 4.10

$$sp(\underbrace{(\ell, F)}_{\text{symbolic state}}, \underbrace{(\ell, \rho, \ell')}_\text{edge}) \triangleq \underbrace{(\ell', sp(F, \rho))}_{\text{Nextsymbolicstate}} .$$

Symbolic strongest post over paths

Definition 4.11

For path $\pi = e_1, \dots, e_n$ of P ,

$$sp((\ell, F), \pi) \triangleq sp(\dots sp((\ell, F), e_1), \dots e_n).$$

Let us expand out $sp((\ell, F), \pi)$

$$(\exists V. \dots (\exists V. (\exists V. F(V) \wedge e_1(V, V')) [V/V'] \wedge e_2(V, V')) [V/V'] \dots) [V/V']$$

We get away with the renaming if use different name in quantifier everytime

$$(\exists V_{n-1}. \dots (\exists V_1. (\exists V_0. F(V_0) \wedge e_1(V_0, V_1)) \wedge e_2(V_1, V_2)) \dots) [V/V_n]$$

Symbolic strongest post over paths

If we pull all the quantifiers in front

$$(\exists V_{n-1} \dots V_0. F(V_0) \wedge \underbrace{e_1(V_0, V_1) \wedge e_2(V_1, V_2) \dots}_{\text{Path constraints of } \pi})[V/V_n]$$

Therefore,

$$sp((\ell, F), \pi) = (\exists V_{n-1} \dots V_0. F(V_0) \wedge \rho(\pi))[V/V_n]$$

Strongest post and implication

For a path $\pi = e_1, \dots, e_n$, let us suppose we want to check Hoare triple $\{P\}\pi\{Q\}$.

We need to implement

$$\forall V. sp(P, \pi) \Rightarrow Q.$$

Let us expand sp .

$$\forall V. (\exists V_{n-1} \dots V_0. P(V_0) \wedge \rho(\pi))[V/V_n] \Rightarrow Q.$$

Again by renaming quantifiers, we get rid of explicit renamings.

$$\forall V_n. (\exists V_{n-1} \dots V_0. P(V_0) \wedge \rho(\pi)) \Rightarrow Q(V_n).$$

To prove the above is true, we can prove the following negation false.

$$\exists V_n. (\exists V_{n-1} \dots V_0. P(V_0) \wedge \rho(\pi)) \wedge \neg Q(V_n).$$

Statement post and implement II

After flattening the quantifiers, we obtain

$$\exists V_n \dots V_0. P(V_0) \wedge \rho(\pi) \wedge \neg Q(V_n).$$

All we need to show that the following formula is unsatisfiable.

$$P(V_0) \wedge \rho(\pi) \wedge \neg Q(V_n).$$

We only need a **satisfiability solver** to check validity of a Hoare triple over a straight line program.

From simple language to labelled transition system

Theorem 4.2

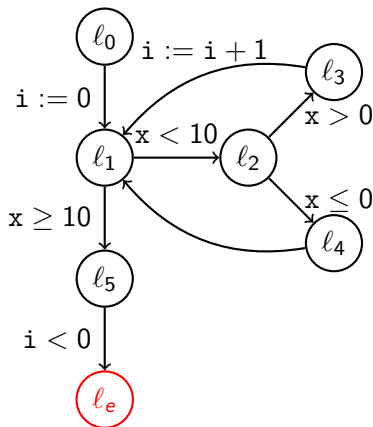
Simple programming language is isomorphic to the labelled transition systems

Proof.

We show it by an example. □

Example 4.16

```
L0: i = 0;  
L1: while( x < 10 ) {  
L2:   if x > 0 then  
L3:     i := i + 1  
       else  
L4:     skip  
       }  
L5: assert( i >= 0 )
```



Cut-points

Definition 4.12

For a program $P = (V, L, \ell_0, \ell_e, E)$, $\text{CUTPOINTS}(P)$ is the a minimal subset of L such that every path of P containing a loop passes through one of the location in $\text{CUTPOINTS}(P)$.

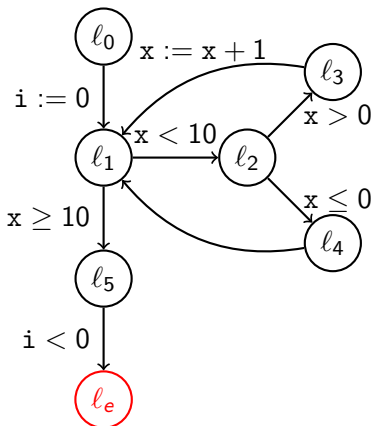
Typically, $\text{CUTPOINTS}(P)$ in LTS are loop heads in simple language.

There may not be a unique cutpoint set.

Example: cut-points

Example 4.17

Consider the following program P .

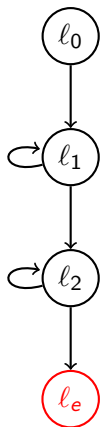


$$\text{CUTPOINTS}(P) = \{l_1\}$$

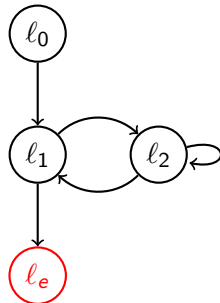
Exercise: cut-points

Exercise 4.6

Give a set of cut-points for the following programs.



Sequential loops



Nested loops

Topic 4.2

Loop invariants

Invariants

Definition 4.13

For P , a map $I : L \rightarrow \Sigma(V)$ is called *invariant map* if, for each $\ell \in L$, all reachable states at ℓ satisfy $I(\ell)$.

Definition 4.14

For P , a map $I : L \rightarrow \Sigma(V)$ is called *inductive* if, for each $(\ell, \rho, \ell') \in E$,

$$sp(I(\ell), \rho) \Rightarrow I(\ell').$$

Definition 4.15

For P , a map $I : L \rightarrow \Sigma(V)$ is called *safe* if $I(\ell_0) = \top$ and $I(\ell_e) = \perp$

Theorem 4.3

For P , if I is inductive and safe then I is an invariant and P is safe.

Invariant checking: is I a safe inductive invariant map?

Exercise 4.7

What is the algorithm for invariant checking?

Cut-point invariant maps

Let P be a program and $C = \text{CUTPOINTS}(P) \cup \{\ell_0, \ell_e\}$.

Definition 4.16

A map $I : C \rightarrow \Sigma(V)$ is called *cut-point invariant map* if, for each $\ell \in C$, all reachable states at ℓ satisfy $I(\ell)$.

Definition 4.17

A map $I : C \rightarrow \Sigma(V)$ is called *inductive* if, for each $\ell, \ell' \in C$ and $\pi \in \text{LOOPFREEPATHS}(P, \ell, \ell')$, $sp(I(\ell), \pi) \Rightarrow I(\ell')$.

Definition 4.18

A map $I : C \rightarrow \Sigma(V)$ is called *safe* if $I(\ell_0) = \top$ and $I(\ell_e) = \perp$.

Theorem 4.4

If I is inductive and safe then I is an cut-point invariant map and P is safe.

Proof.

Every path from ℓ_0 to ℓ_e can be segmented into loop free paths between cut-points. Therefore, no such path is feasible. □

Annotated verification: VCC demo

<http://rise4fun.com/Vcc>

Exercise 4.8

Complete the following program such that Vcc proves it correct

```
#include <vcc.h>
int main()
{
    int x, y;
    _(assume x > y +3 && x < 3000 )
    while( 0 < y ) _(invariant ....) {
        x = x + 1;
        y = y -1;
    }
    _(assert x >= y)
    return 0;
}
```


Exercise: Invariants guess and check

Example 4.18

Fill the annotations to prove following program correct via Vcc

```
#include <vcc.h>
int main()
{
    int x = 0, y = 2;
    _(assume 1==1 )
    while( x < 3 ) _(invariant ... ) {
        x = x + 1;
        y = 3;
    }
    _(assert y == 3)
    return 0;
}
```

Annotated verification

- ▶ There are many tools like VCC that require user to write invariants at the loop heads and function boundaries
- ▶ Rest of the verification is done as discussed in earlier slides
- ▶ User needs to do a lot of work, **not a very desirable method**

What if we want to compute the invariants automatically?

Topic 4.3

Problems

Exercise: bubble sort

Exercise 4.9

Write inductive invariants at the loop heads in the bubble sort such that they prove that at the end array is sorted and the content is preserved.

```
procedure bubbleSort( A : list of sortable items )
  n = length(A)
  repeat
    swapped = false
    for i = 1 to n-1 inclusive do
      if A[i-1] > A[i] then
        swap( A[i-1], A[i] )
        swapped = true
      end if
    end for
  until not swapped
end procedure
```

Exercise: merge sort

Exercise 4.10

Write inductive invariants at the loop heads in the merge sort such that they prove that at the end array is sorted and the content is preserved.

```
function merge_sort(list m)
  if length of m <= 1 then
    return m
  var left := empty list
  var right := empty list
  for each x with index i in m do
    if i <= (length of m)/2 then
      add x to left
    else
      add x to right
  left := merge_sort(left)
  right := merge_sort(right)
  return merge(left, right)
```

```
function merge(left, right)
  var result := empty list
  while left is not empty and right is not empty do
    if first(left) <= first(right) then
      append first(left) to result
      left := rest(left)
    else
      append first(right) to result
      right := rest(right)
  while left is not empty do
    append first(left) to result
    left := rest(left)
  while right is not empty do
    append first(right) to result
    right := rest(right)
  return result
```

Exercise: strange array properties

Exercise 4.11

Write inductive loop invariants for the following program that prove the following property.

```
int main ( int A[ N ] , int B[ N ] , int C[ N ] ) {
    int i;
    int j = 0;
    for (i = 0; i < N ; i++) {
        if ( A[i] == B[i] ) {
            C[j] = i;
            j = j + 1;
        }
    }

    assert( forall x: ( 0 <= x < j ) ==> ( C[x] <= x + i - j ) );
    assert( forall x: ( 0 <= x < j ) ==> ( C[x] >= x ) );
}
```

Topic 4.4

Bonus slides: 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 $L = \{l_0, \dots, l_n, l_e\}$,

Let $V = \{x_1, \dots, x_m\}$

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

$$I(l_0) = 0 \leq 0$$

$$\forall i \in 1..n. I(l_i) = (p_{i1}x_1 + \dots + p_{im}x_m \leq p_{i0})$$

$$I(l_e) = 0 \leq -1$$

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 $(l_i, \rho, l_{i'}) \in E$

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

The above condition translates to

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

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 ρ s are linear constraints then we can use Farkas lemma to turn the validity question into a “conjunctive satisfiability 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

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

we obtain

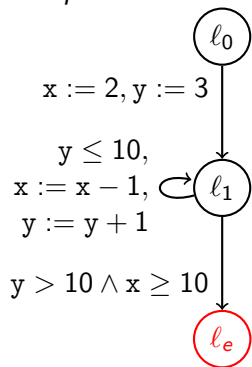
$$\begin{aligned} \exists \lambda_0, \lambda. (\lambda_0[p_{i1}, \dots, p_{im}] + \lambda^T A) = 0 \wedge \lambda^T A' = [p_{i'1}, \dots, p_{i'm}] \wedge \\ \lambda_0 p_{i0} + \lambda^T b \leq p_{i'0} \end{aligned}$$

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

Example: invariant generation

Example 4.19

Consider the following example



We assume the following invariant template at l_1 :
 $I(l_1) = (p_1x + p_2y \leq p_0)$

We generate the following constraints for program transitions:

For l_0 to l_1 ,
 $\forall x', y'. x' = 2 \wedge y' = 3 \Rightarrow (p_1x' + p_2y' \leq p_0)$

For l_1 to l_1 ,
 $\forall x, y, x', y'. (p_1x + p_2y \leq p_0) \wedge y \leq 10 \wedge x' = x - 1 \wedge y' = y + 1 \Rightarrow (p_1x' + p_2y' \leq p_0)$

Let $V = [x, y]$

For l_1 to l_e ,
 $\forall x, y. (p_1x + p_2y \leq p_0) \wedge y > 10 \wedge x \geq 10 \Rightarrow \perp$

Example: invariant generation(contd.)

Now consider the second constraint:

$\forall x, y, x', y'$.

$$(p_1x + p_2y \leq p_0) \wedge y \leq 10 \wedge x' = x - 1 \wedge y' = y + 1 \Rightarrow (p_1x' + p_2y' \leq p_0)$$

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

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

Example: invariant generation(contd.)

Applying farkas lemma on the constraint, we obtain

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

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

Exercise 4.12

Apply farkas lemma on the other two implications

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

$$\forall x, y. (p_1 x + p_2 y \leq p_0) \wedge y > 10 \wedge x \geq 10 \Rightarrow \perp$$

Does this method work?

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

End of Lecture 4