CS615: Formal Specification and Verification of Programs 2019

Lecture 4: Labeled transition systems and invariants

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2019-08-17

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

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

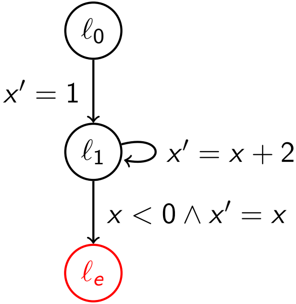
where

- 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 $P = \{[\mathtt{x}], \{\ell_0, \ell_1, \ell_e\}, \ell_0, \ell_e, E\}$



$$E = \{ \quad (\ell_0, \mathtt{x}' = 1, \ell_1), \quad (\ell_1, \mathtt{x}' = \mathtt{x} + 2, \ell_1), \quad (\ell_1, \mathtt{x} < 0 \wedge \mathtt{x}' = \mathtt{x}, \ell_e) \quad \}$$

Shorthand notation for handling transitions

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

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

Example 4.2

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

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

$e(loc)$ *denotes* ℓ_1*.*

$e(loc')$ *denotes* ℓ_2*.*

Cumbersome labels

The labels in LTS are cumbersome to write.

Example 4.3

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

For statement $\mathtt{x} := 1$*, we have to add the following label in LTS.*

$$\mathrm{x}' = 1 \wedge \mathrm{y}' = \mathrm{y} \wedge \mathrm{z}' = \mathrm{z}.$$

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

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

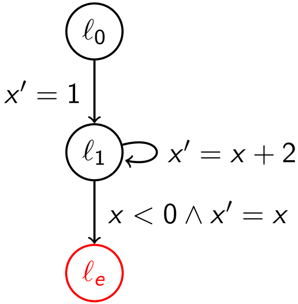
The formula represented by the guarded command is

$$\mathrm{x} > \mathrm{y} \wedge \mathrm{x}' = \mathrm{x} + 1 \wedge \mathrm{y}' = \mathrm{y}.$$

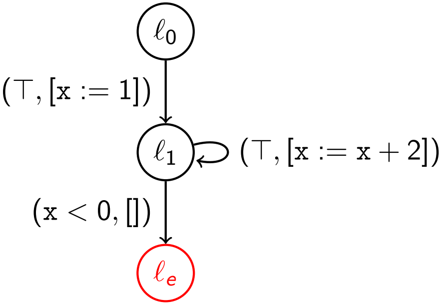
Guarded command is a convenient way of writing transitions.

Example: guarded command

Example 4.5



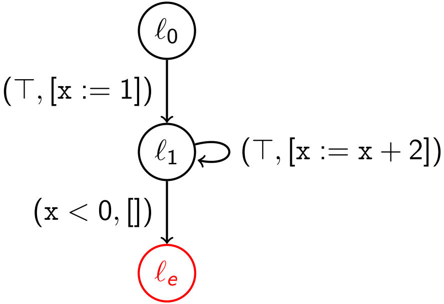
LTS with formulas



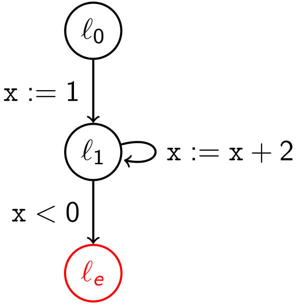
LTS with guarded commands

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

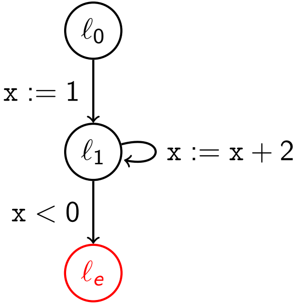
A state $s = (\ell, v)$ of a program is program location ℓ and a valuation v of V.

Notation:

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

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

Example 4.7



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

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

We will write $s(x) = 19$ *and* $s(loc) = \ell_e$.

Path

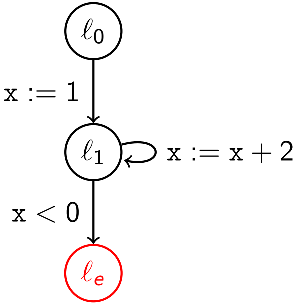
Definition 4.4

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

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

Example 4.8

Consider the following program P.



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

Exercise 4.1

Is the following a path of P*?*

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

Execution of paths

Definition 4.5

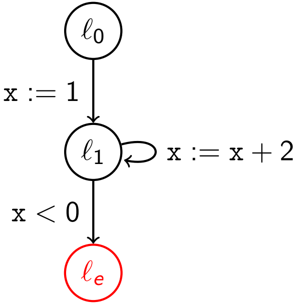
An execution *corresponding to path* $\pi = e_1, \ldots, e_n$ *is a sequence of states*

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

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 $(\ell_0, \mathtt{x} := 1, \ell_1), (\ell_1, \mathtt{x} := \mathtt{x} + 2, \ell_1)$ *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*.*

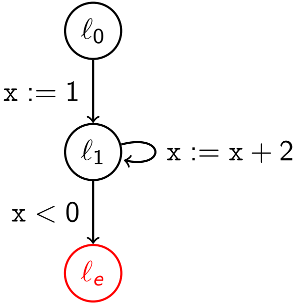
Feasibility of paths

Definition 4.6

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

Example 4.10

Consider the following program P.



Path $(\ell_0, \mathtt{x} := 1, \ell_1), (\ell_1, \mathtt{x} := \mathtt{x} + 2, \ell_1)$ is feasible, since we have seen an execution along the path.

Exercise 4.3

Give an infeasible path?

Execution of program

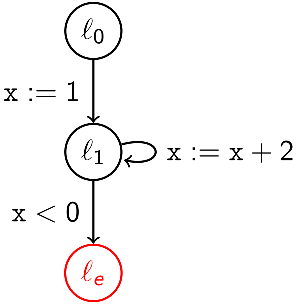
Definition 4.7

An execution $s_0, ..., s_n$ belongs to P if

- $s_0(loc) = \ell_0$ *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*

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

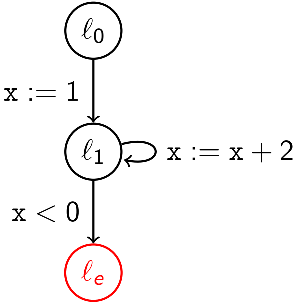
Exercise 4.4

Give an execution of P that reaches ℓ_e.

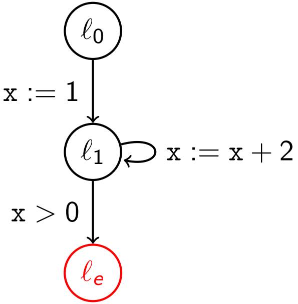
Safety in LTS

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

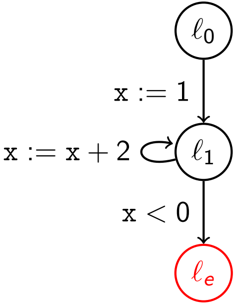
For a path $\pi = e_1, \dots, e_n$, path constraints $\rho(\pi)$ is

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

Path constraints are also known as "SSA formulas"

Example: path constraints

Example 4.14



Consider path

$$\underbrace{(\ell_0, \mathtt{x} := 1, \ell_1)}_{e_1}, \underbrace{(\ell_1, \mathtt{x} := \mathtt{x} + 2, \ell_1)}_{e_2}, \underbrace{(\ell_1, \mathtt{x} < 0, \ell_e)}_{e_3}.$$

Path constraint for the path is

$$\rho(e_1 e_2 e_3) = (\mathtt{x}_1 = 1 \land \mathtt{x}_2 = \mathtt{x}_1 + 2 \land \mathtt{x}_2 < 0 \land \mathtt{x}_3 = \mathtt{x}_2).$$

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

$$\rho(e_1 e_2 e_2) = (\mathrm{x}_1 = 1 \wedge \mathrm{x}_2 = \mathrm{x}_1 + 2 \wedge \mathrm{x}_3 = \mathrm{x}_2 + 2).$$

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

$$\{\mathrm{x}_0 \to -12030, \mathrm{x}_1 \to 1, \mathrm{x}_2 \to 3, \mathrm{x}_3 \to 5\}.$$

symbolic strongest post over edges

Recall,

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

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

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

Symbolic strongest post over paths

Definition 4.11

For path $\pi = e_1, .., e_n$ *of* P,

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

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

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

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

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

Symbolic strongest post over paths

If we pull all the quantifiers in front

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

Therefore,

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

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*.

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

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

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

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

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

Statement post and implement II

After flattening the quantifiers, we obtain

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

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

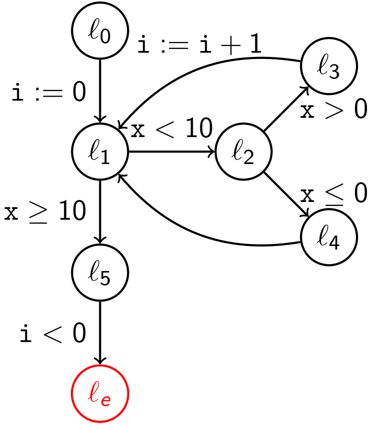
Simple programming language is isomorphic to the labelled transition systems

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

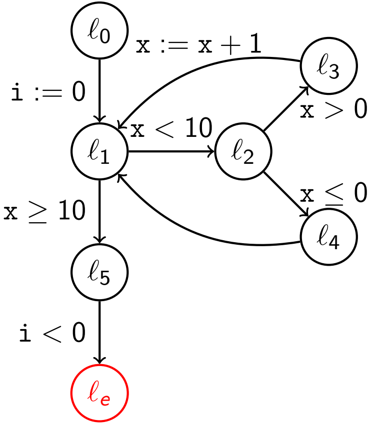
Typically, $\textsc{CutPoints}(P)$ in LTS are loop heads in simple language.

There may not be a unique cutpoint set.

Example: cut-points

Example 4.17

Consider the following program P.

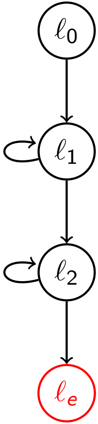


$$\textsc{CutPoints}(P) = \{\ell_1\}$$

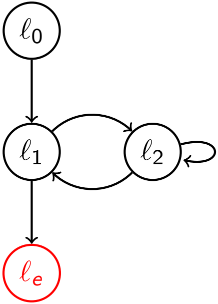
Exercise: cut-points

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 $\mathrm{I} : L \to \Sigma(V)$ is called invariant map *if, for each $\ell \in L$, all reachable states at ℓ satisfy $\mathrm{I}(\ell)$.*

Definition 4.14
For P, a map $\mathrm{I} : L \to \Sigma(V)$ is called inductive *if, for each $(\ell, \rho, \ell') \in E$,*

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

Definition 4.15
For P, a map $\mathrm{I} : L \to \Sigma(V)$ is called safe *if $\mathrm{I}(\ell_0) = \top$ and $\mathrm{I}(\ell_e) = \bot$*

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 $C = \textsc{CutPoints}(P) \cup \{\ell_0, \ell_e\}$.

Definition 4.16

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

Definition 4.17

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

Definition 4.18

A map $\mathrm{I} : C \to \Sigma(V)$ *is called safe if* $\mathrm{I}(\ell_0) = \top$ *and* $\mathrm{I}(\ell_e) = \bot$

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

$\boxed{\textbf{Invariant generation:} \text{ find a safe inductive invariant map } \mathrm{I}}$

- 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.

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

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

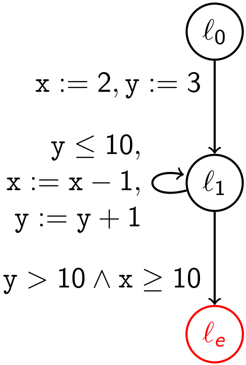
$$\exists \lambda_0, \lambda. \, (\lambda_0[p_{i1}, \ldots, p_{im}] + \lambda^T A) = 0 \wedge \lambda^T A' = [p_{i'1}, \ldots, p_{i'm}] \wedge \\ \lambda_0 p_{i0} + \lambda^T b \leq p_{i'0}$$

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

Example: invariant generation

Example 4.19

Consider the following example



Let $V = [x, y]$

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

We generate the following constraints for program transitions:

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

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

For ℓ_1 *to* ℓ_e,
$$\forall x, y.\ (p_1 x + p_2 y \leq p_0) \wedge y > 10 \wedge x \geq 10 \Rightarrow \perp$$

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 4.12

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