



Verification - Lecture 5 Parameterized Programs, Linear Invariants

Bernd Finkbeiner - Sven Schewe
Rayna Dimitrova - Lars Kuhtz - Anne Proetzsch

Wintersemester 2007/2008

Proving Invariance Properties

Review

For assertions $q, \varphi, \chi_1, \dots, \chi_k$

$$\text{I0. } P \models \square \chi_1, \dots, \square \chi_k$$

$$\text{I1. } P \models \left(\bigwedge_{i=1}^k \chi_i \right) \wedge \varphi \rightarrow q$$

$$\text{I2. } P \models \theta \rightarrow \varphi$$

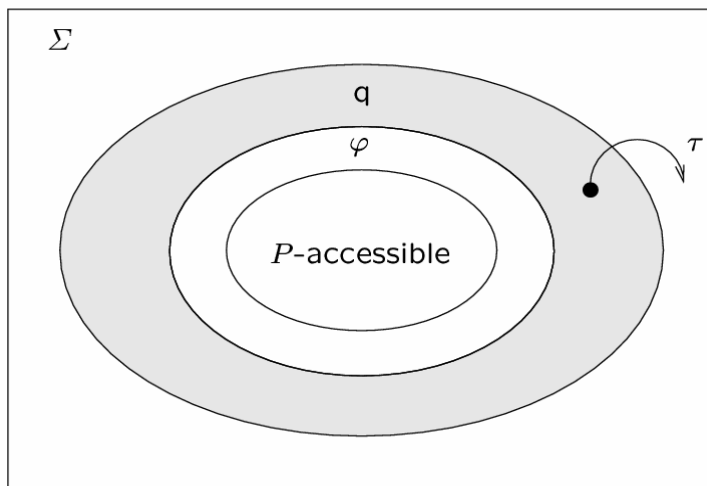
$$\text{I3. } P \models \left\{ \left(\bigwedge_{i=1}^k \chi_i \right) \wedge \varphi \right\} \mathcal{T} \{ \varphi \}$$

$$P \models \square q$$

INC-INV

Strategy 1: Strengthening

Review



Find a stronger assertion φ that is inductive and implies the assertion q we want to prove.

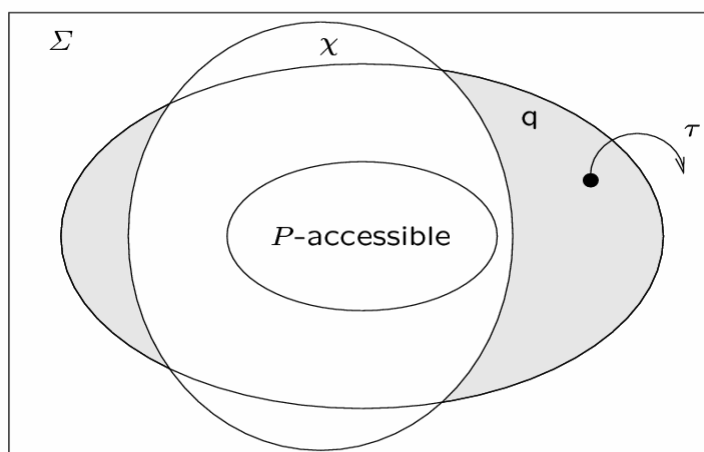
Bernd Finkbeiner

Verification - Lecture 5

3

Strategy 2: Incremental Proofs

Review



Use previously proven invariances χ to exclude parts of the state space from consideration.

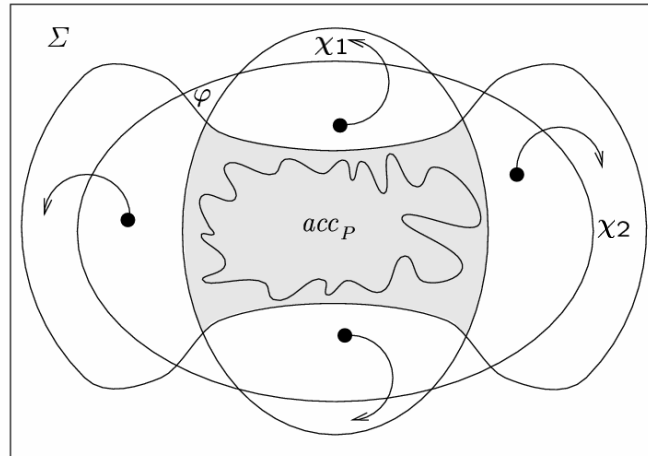
Bernd Finkbeiner

Verification - Lecture 5

4

Discussion

Review



Although the assertion acc_P is inductive and strengthens any P -invariant, it is not very useful in practice.

Finding Inductive Invariants

Review

Construction of inductive assertions by

1. Bottom-up methods:

- Based on program text only
- Algorithmic
- Guaranteed to produce an inductive invariant

2. Top-down methods:

- Guided by the property we want to prove
- Heuristic
- Not guaranteed to produce an inductive invariant

Top-Down Approach

Review

previously proven $\Box \chi$

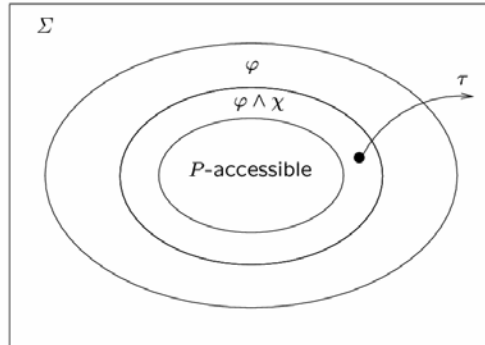
want to prove $\Box \varphi$

but

$$\{\chi \wedge \varphi\} \tau \{\varphi\}$$

not state-valid

for some $\tau \in \mathcal{T}$.

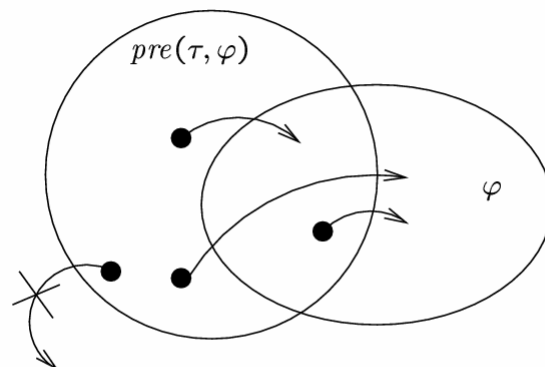


Solution: Take the largest set of states that will result in a φ -state when τ is taken.

Precondition

Review

$$pre(\tau, \varphi) : \forall V'. \rho_{\tau} \rightarrow \varphi'$$



a state s satisfies $pre(\tau, \varphi)$

iff

all τ -successors of s satisfy φ .

Heuristic

Review

If the verification condition

$$\{\chi \wedge \varphi\} \tau \{\varphi\}$$

is not state-valid

- Strengthening approach:
strengthen φ by adding the conjunct $pre(\tau, \varphi)$
- Incremental approach:
prove $\Box pre(\tau, \varphi)$ and add $pre(\tau, \varphi)$ to χ .

Example

Review

local y_1, y_2 : boolean where $y_1 = F, y_2 = F$
 s : integer where $s = 1$

ℓ_0 : loop forever do

P_1 :: $\left[\begin{array}{l} \ell_1 : \text{noncritical} \\ \ell_2 : (y_1, s) := (T, 1) \\ \ell_3 : \text{await } (\neg y_2) \vee (s = 2) \\ \ell_4 : \text{critical} \\ \ell_5 : y_1 := F \end{array} \right]$

||

m_0 : loop forever do

P_2 :: $\left[\begin{array}{l} m_1 : \text{noncritical} \\ m_2 : (y_2, s) := (T, 2) \\ m_3 : \text{await } (\neg y_1) \vee (s = 1) \\ m_4 : \text{critical} \\ m_5 : y_2 := F \end{array} \right]$

Goal:

Mutual Exclusion for
Peterson's algorithm:

$$\Box \underbrace{\neg(at_l_4 \wedge at_m_4)}_{\psi}$$

Bottom-up invariants:

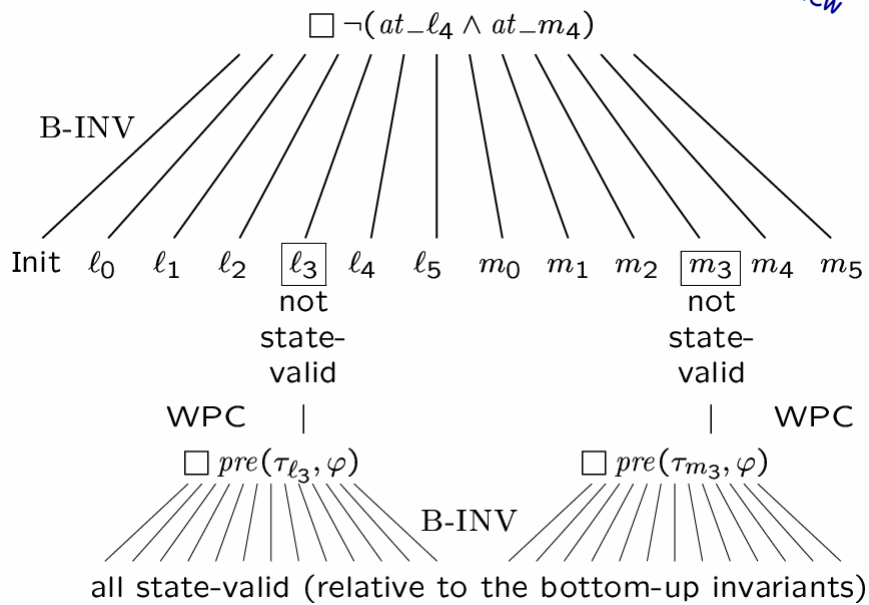
$$\varphi_0: s = 1 \vee s = 2$$

$$\varphi_1: y_1 \leftrightarrow at_l_{3..5}$$

$$\varphi_2: y_2 \leftrightarrow at_m_{3..5}$$

Example (cont'd)

Review



Bernd Finkbeiner

Verification - Lecture 5

11

Goals for Today

1. Generalization to parameterized systems
2. Bottom-up generation of linear invariants

Introduction: Parameterized Programs

$$S:: \left[\begin{array}{l} \ell_0: \text{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1: \text{noncritical} \\ \ell_2: \text{request } y \\ \ell_3: \text{critical} \\ \ell_4: \text{release } y \end{array} \right] \end{array} \right]$$

$P^3::$ [local y : integer where $y = 1$; $[S||S||S]$]
(with some renaming of labels of the S 's.)

$P^4::$ [local y : integer where $y = 1$; $[S||S||S||S]$]

⋮

$P^n::$?

Introduction: Parameterized Specifications

Mutual exclusion:

$$P^3: \square(\neg(at_{\ell_3} \wedge at_{m_3}) \wedge \neg(at_{\ell_3} \wedge at_{k_3}) \wedge \neg(at_{m_3} \wedge at_{k_3}))$$

$$P^4: \square(\neg(\dots) \wedge \dots \wedge \neg(\dots))$$

P^n : ?

We want to deal with these programs, i.e., programs with an arbitrary number of identical components, in a more uniform way.

Syntax

cooperation: $\prod_{j=1}^M S[j] : [S[1] || \dots || S[M]]$

Selection: $\text{OR}_{j=1}^M S[j] : [S[1] \text{ or } \dots \text{ or } S[M]]$

$S[j]$ is a parameterized statement.

Parameterized Statements

- explicit variable in expression
 $\dots := j + \dots$
- explicit subscript in array x
 $\dots := x[j] + \dots$ or $x[j] := \dots$
- implicit subscript of all local variables in $S[j]$
 z stands for $z[j]$
- implicit subscript of all labels in $S[j]$
 ℓ_3 stands for $\ell_3[j]$

Example: Program PAR-SUM

```
in  M: integer where M ≥ 1
    x : array [1..M] of integer
out z : integer where z = 0
```

```


$$\prod_{j=1}^M P[j] :: \left[ \begin{array}{l} \text{local } y: \text{integer} \\ \ell_0: y := x[j] \\ \ell_1: z := z + y \cdot y \\ \ell_2: \end{array} \right]$$

```

Parallel sum of squares:

$$z = x[1]^2 + x[2]^2 + \dots + x[M]^2$$

Parameterized Transition Systems

The number M of processes is not fixed, so there is an unbounded $\#$ of transitions. To finitely represent these, we use parameterization of transition relations.

Example: PAR-SUM

The unbounded $\#$ of transitions associated with ℓ_0 are represented by a single transition relation using parameter j :

$$\rho_{\ell_0}[j]: \text{move}(\ell_0[j], \ell_1[j]) \wedge y'[j] = x[j] \\ j = 1 \dots M$$

Arrays

Arrays (explicit or implicit) are treated as variables that range over functions:

$$[1 \dots M] \mapsto \text{integers}$$

Representation of array operations in transition relations:

- Retrieval: $y[k]$
to retrieve the value of the k th element of array y
- Modification: $update(y, k, e)$
the resulting array agrees with y on all i , $i \neq k$, and $y[k] = e$

Arrays (cont'd)

Properties of $update$

$$update(y, k, e)[k] = e$$

$$update(y, k, e)[j] = y[j] \text{ for } j \neq k$$

Example: PAR-SUM

The proper representation of the transition relation for $\ell_0[j]$ is

$$\begin{aligned} \rho_0[j]: \quad & move(\ell_0[j], \ell_1[j]) \wedge \\ & y' = update(y, j, x[j]) \wedge \\ & pres(\{x, z\}) \end{aligned}$$

Notation in Specifications

- $L_i = \{j \mid \ell_i[j] \in \pi\} \subseteq \{1, \dots, M\}$

The set of indices of processes that currently reside at l_i

- $N_i = |L_i|$

The number of processes currently residing at l_i

Abbreviations

$$L_{i_1, i_2, \dots, i_k} = L_{i_1} \cup L_{i_2} \cup \dots \cup L_{i_k}$$

$$L_{i..j} = L_i \cup L_{i+1} \cup \dots \cup L_j$$

$$N_{i_1, i_2, \dots, i_k} = |L_{i_1, i_2, \dots, i_k}|$$

$$N_{i..j} = |L_{i..j}|$$

Example: MPX-SEM

```

in   M: integer where M ≥ 2
local y : array [1..M] of integer
      where y[1] = 1, y[j] = 0 for 2 ≤ j ≤ M
  
```

$$\prod_{j=1}^M P[j] :: \left[\begin{array}{l} \ell_0: \text{loop forever do} \\ \left[\begin{array}{l} \ell_1: \text{noncritical} \\ \ell_2: \text{request } y[j] \\ \ell_3: \text{critical} \\ \ell_4: \text{release } y[j \oplus_M 1] \end{array} \right] \end{array} \right]$$

Multiple mutual exclusion by semaphors

$$j \oplus_M 1 = (j \bmod M) + 1 = \begin{cases} j + 1 & \text{if } j < M \\ 1 & \text{if } j = M \end{cases}$$

Example: Specification

```

in   M: integer where M ≥ 2
local y : array [1..M] of integer
      where y[1] = 1, y[j] = 0 for 2 ≤ j ≤ M
  
```

$$\prod_{j=1}^M P[j] :: \left[\begin{array}{l} \ell_0: \text{loop forever do} \\ \left[\begin{array}{l} \ell_1: \text{noncritical} \\ \ell_2: \text{request } y[j] \\ \ell_3: \text{critical} \\ \ell_4: \text{release } y[j \oplus_M 1] \end{array} \right] \end{array} \right]$$

mutual exclusion:

$$\square \underbrace{\forall i, j \in [1..M]. i \neq j. \neg (at_{-\ell_3}[i] \wedge at_{-\ell_3}[j])}_{\psi}$$

abbreviated as

$$\square (N_3 \leq 1)$$

Example: Verification

in M : integer where $M \geq 2$
 local y : array $[1..M]$ of integer
 where $y[1] = 1, y[j] = 0$ for $2 \leq j \leq M$

$\square(N_3 \leq 1)$
 φ

$$\prod_{j=1}^M P[j] :: \left[\begin{array}{l} \ell_0: \text{loop forever do} \\ \left[\begin{array}{l} \ell_1: \text{noncritical} \\ \ell_2: \text{request } y[j] \\ \ell_3: \text{critical} \\ \ell_4: \text{release } y[j \oplus_M 1] \end{array} \right] \end{array} \right]$$

The assertion φ is not inductive, therefore we prove the invariance of

$$\varphi_1: \forall j. y[j] \geq 0$$

$$\varphi_2: \left(N_{3,4} + \sum_{j=1}^M y[j] \right) = 1$$

Example (cont'd)

$$\varphi_1: \forall j. y[j] \geq 0$$

$$\varphi_2: \left(N_{3,4} + \sum_{j=1}^M y[j] \right) = 1$$

$\square(N_3 \leq 1)$
 φ

Then φ can be deduced by monotonicity:

$$\varphi_1 \wedge \varphi_2 \rightarrow \underbrace{N_3 \leq 1}_{\varphi}$$

since

$$N_3 \leq \underbrace{N_{3,4}}_{\varphi_2} = 1 - \sum_{j=1}^M y[j] \leq \underbrace{1}_{\varphi_1}$$

Example (cont'd)

- Proof of $\square \underbrace{(\forall j. y[j] \geq 0)}_{\varphi_1}$

B1:

$$\dots \wedge y[1] = 1 \wedge \underbrace{(\forall j. 2 \leq j \leq M. y[j] = 0)}_{\theta} \\ \rightarrow \underbrace{\forall j. y[j] \geq 0}_{\varphi_1}$$

B2:

The only transitions that interfere with φ_1 are τ_{ℓ_2} and τ_{ℓ_4} .

Example (cont'd)

$$\rho_{\ell_2}: \text{move}(\ell_2[i], \ell_3[i]) \wedge y[i] > 0 \wedge \\ y' = \text{update}(y, i, y[i]-1)$$

$$\rho_{\ell_4}: \text{move}(\ell_4[i], \ell_0[i]) \wedge \\ y' = \text{update}(y, i \oplus_M 1, y[i \oplus_M 1]+1)$$

$\rho_{\ell_2}[i]$ implies

$$y[i] > 0 \wedge y'[i] = y[i] - 1 \wedge \forall j. j \neq i. y'[j] = y[j]$$

$\rho_{\ell_4}[i]$ implies

$$y'[i \oplus_M 1] = y[i \oplus_M 1] + 1 \wedge$$

$$\forall j. j \neq i \oplus_M 1. y'[j] = y[j]$$

therefore

$$\underbrace{\forall j. y[j] \geq 0}_{\varphi_1} \wedge \left\{ \begin{array}{l} \rho_{\ell_2}[i] \\ \rho_{\ell_4}[i] \end{array} \right\} \rightarrow \underbrace{\forall j. y'[j] \geq 0}_{\varphi'_1}$$

Example (cont'd)

- Proof of $\square \underbrace{(N_{3,4} + \left(\sum_{j=1}^M y[j]\right))}_{\varphi_2} = 1$

B1:

$$\underbrace{\left(\pi = \{\ell_0[1], \dots, \ell_0[M]\} \wedge \right.}_{\theta}$$

$$\left. y[1] = 1 \wedge (\forall j. 2 \leq j \leq M. y[j] = 0) \right)$$

$$\rightarrow \underbrace{N_{3,4} + \left(\sum_{j=1}^M y[j]\right)}_{\varphi_2} = 1$$

Example (cont'd)

$$\rho_{\ell_2}: \text{move}(\ell_2[i], \ell_3[i]) \wedge y[i] > 0 \wedge$$

$$y' = \text{update}(y, i, y[i]-1)$$

$$\rho_{\ell_4}: \text{move}(\ell_4[i], \ell_0[i]) \wedge$$

$$y' = \text{update}(y, i \oplus_M 1, y[i \oplus_M 1]+1)$$

B2: Verification conditions:

$\rho_{\ell_2}[i]$ implies:

$$N'_{3,4} = N_{3,4} + 1 \wedge \left(\sum_{j=1}^M y'[j]\right) = \left(\sum_{j=1}^M y[j]\right) - 1$$

Example (cont'd)

$$\rho_{\ell_4}: \text{move}(\ell_4[i], \ell_0[i]) \wedge$$

$$y' = \text{update}(y, i \oplus_M 1, y[i \oplus_M 1] + 1)$$

$\rho_{\ell_4}[i]$ implies:

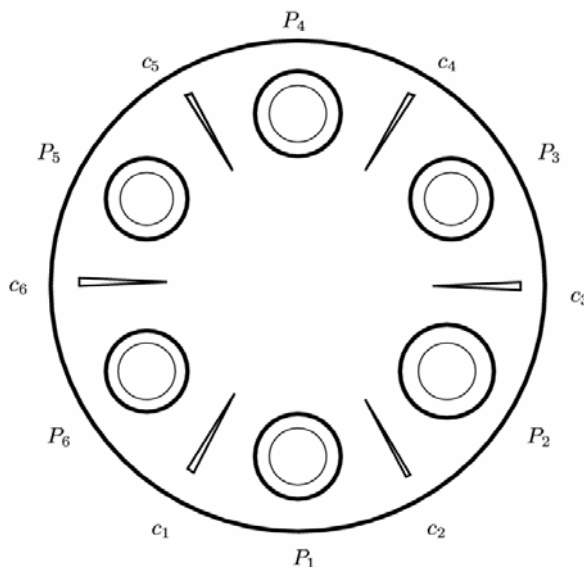
$$N'_{3,4} = N_{3,4} - 1 \wedge \left(\sum_{j=1}^M y'[i] \right) = \left(\sum_{j=1}^M y[i] \right) + 1$$

Therefore

$$\underbrace{N_{3,4} + \left(\sum_{j=1}^M y[i] \right)}_{\varphi_2} = 1 \wedge \left\{ \begin{array}{l} \rho_{\ell_2}[i] \\ \rho_{\ell_4}[i] \end{array} \right\}$$

$$\rightarrow \underbrace{N'_{3,4} + \left(\sum_{j=1}^M y[i] \right)}_{\varphi'_2} = 1$$

The Dining Philosophers



DINE - A Simple Solution

in M : integer where $M \geq 2$
 local c : array $[1..M]$ of integer where $c = 1$

$$\prod_{j=1}^M P[j] :: \left[\begin{array}{l} \ell_0: \text{loop forever do} \\ \left[\begin{array}{l} \ell_1: \text{noncritical} \\ \ell_2: \text{request } c[j] \\ \ell_3: \text{request } c[j \oplus_M 1] \\ \ell_4: \text{critical} \\ \ell_5: \text{release } c[j] \\ \ell_6: \text{release } c[j \oplus_M 1] \end{array} \right] \end{array} \right]$$

„Chopstick Exclusion“

$$\square \underbrace{\forall j \in [1..M]. \neg (at_l_4[j] \wedge at_l_4[j \oplus_M 1])}_{\psi}$$

- φ_0 and φ_1 are inductive

$$\varphi_0: \forall j \in [1..M]. c[j] \geq 0$$

$$\varphi_1: \forall j \in [1..M]. at_l_{4..6}[j] + at_l_{3..5}[j \oplus_M 1] + c[j \oplus_M 1] = 1$$

- Then,

$$at_l_4[j] + at_l_4[j \oplus_M 1]$$

$$\leq 1 - c[j \oplus_M 1] \leq 1$$

$$\varphi_1 \qquad \qquad \qquad \varphi_0$$

Mutual exclusion
 between every two
 adjacent
 philosophers

Program DINE-EXCL

```
in  $M$ : integer where  $M \geq 2$   
local  $c$  : array  $[1..M]$  integer where  $c = 1$   
 $r$  : integer where  $r = M - 1$ 
```

```
 $\prod_{j=1}^M P[j] ::$   
[  $\ell_0$ : loop forever do  
  [  $\ell_1$ : noncritical  
     $\ell_2$ : request  $r$   
     $\ell_3$ : request  $c[j]$   
     $\ell_4$ : request  $c[j \oplus_M 1]$   
     $\ell_5$ : critical  
     $\ell_6$ : release  $c[j]$   
     $\ell_7$ : release  $c[j \oplus_M 1]$   
     $\ell_8$ : release  $r$   
  ]  
]
```

Finding Inductive Invariants

Review

Construction of inductive assertions by

1. Bottom-up methods:

- Based on program text only
- Algorithmic
- Guaranteed to produce an inductive invariant

2. Top-down methods:

- Guided by the property we want to prove
- Heuristic
- Not guaranteed to produce an inductive invariant

Control Invariants

Review

- CONFLICT:
for labels l_i, l_j that are in conflict
(i.e., not \sim_L , not parallel):

$$\square \neg(at_{-l_i} \wedge at_{-l_j})$$

- SOMEWHERE:
for the set of labels \mathcal{L}_i in a
top-level process:

$$\square \bigvee_{\ell \in \mathcal{L}_i} at_{-\ell}$$

Control Invariants

Review

- EQUAL:
for labels l, m , s.t. $l \sim_L m$:

$$\square (at_{-l} \leftrightarrow at_{-m})$$

- PARALLEL:
for substatement $[S_1 || S_2]$:

$$\square (in_{-S_1} \leftrightarrow in_{-S_2})$$

i.e., if control is in S_1 it must also be in S_2
and vice versa.

Transition-Validated Assertions

Review

l_1 : [while c do S]; l_2 : $at_l_2 \rightarrow \neg c$

if no statement parallel to l_1 can
modify variables in c

l_1 : $y := e$; l_2 : $at_l_2 \rightarrow y = e$

if no statement parallel to l_1 can modify y
or variables occurring in e
and if y does not occur in e .

Single-Variable Assertions

Review

$y = 0$

[loop forever do
...
request y
...
release y]

$y \geq 0$

$s = 1$

[...
 $s := 1$] || [...
 $s := 2$]

$s = 1 \vee s = 2$

where no other statement
modifies s

Linear Variables

Definition: integer variable y is linear in P if

$$y' = y + c \quad \text{for every } \rho_\tau$$

where c is some integer constant

Example: semaphore variables are linear

$$\underbrace{y' = y + 1}_{\text{release}}$$

$$\underbrace{y' = y - 1}_{\text{request}}$$

$$\underbrace{y' = y}_{\text{otherwise}}$$

Linear Invariants

A linear invariant is of the form

$$\underbrace{\sum_{i=1}^r a_i \cdot y_i}_{\text{body}} + \underbrace{\sum_{\ell \in \mathcal{L}} b_\ell \cdot at_\ell}_{\text{compensation expression}} = K$$

where

a_i, b_ℓ, K – integer constants.

\mathcal{L} – set of all locations in P

y_1, \dots, y_r – all linear variables in P

Example: Program DOUBLE

$$\begin{array}{c} \text{local } y: \text{ integer where } y = 0 \\ \left[\begin{array}{l} \ell_0: y := y + 1 \\ \ell_1: \end{array} \right] \parallel \left[\begin{array}{l} m_0: y := y + 1 \\ m_1: \end{array} \right] \end{array}$$

linear variable: y

linear invariant:

$$y + at_{\ell_0} + at_{m_0} = 2$$

Assumptions

Program $\ell_0^1: S_1 \parallel \dots \parallel \ell_0^i: S_i \parallel \dots \parallel \ell_0^m: S_m$

- no nested parallel statements. Therefore, all move expressions in all ρ_τ are of the form $move(\ell_i, \ell_j)$
- all linear variables y_i have a single initial value y_i^0
- every transition τ enabled on some P -accessible state

Increments

- $\Delta(y, \tau) = c$ if $\rho_\tau \rightarrow y' = y + c$
therefore $\rho_\tau \rightarrow y' = y + \Delta(y, \tau)$

- $\Delta(at_l, \tau) = \begin{cases} 1 & \text{if } l = l_j \\ -1 & \text{if } l = l_i \\ 0 & \text{otherwise} \end{cases}$
if $\rho_\tau \rightarrow \text{move}(l_i, l_j)$
therefore $\rho_\tau \rightarrow at'_l = at_l + \Delta(at_l, \tau)$

Automatic Invariant Construction

Construct

$$\varphi: \sum_{i=1}^r a_i \cdot y_i + \sum_{l \in \mathcal{L}} b_l \cdot at_l = K$$

Our procedure guarantees that the generated assertions are P -invariants!

Equations

We obtain the values of the coefficients from a set of equations as follows:

(I) The invariant has to hold at the first state of every computation

$$\Theta \quad \text{implies} \quad y_i = y_i^0 \quad (i = 1 \dots r) \\ \text{and} \quad \pi = \{\ell_0^1, \dots, \ell_0^m\}$$

and so we get

$$\sum_{i=1}^r a_i \cdot y_i^0 + (b_{\ell_0^1} + \dots + b_{\ell_0^m}) = K$$

Equations (cont'd)

(T) the assertion has to be preserved by all transitions (we want it to be inductive):

$$\underbrace{\left(\sum_{i=1}^r a_i \cdot y_i + \sum_{\ell \in \mathcal{L}} b_\ell \cdot at_{-\ell} = K \right)}_{\varphi} \wedge \rho_\tau \\ \rightarrow \underbrace{\left(\sum_{i=1}^r a_i \cdot y'_i + \sum_{\ell \in \mathcal{L}} b_\ell \cdot at'_{-\ell} = K \right)}_{\varphi'}$$

Equations (cont'd)

$$\underbrace{\left(\sum_{i=1}^r a_i \cdot y_i + \sum_{\ell \in \mathcal{L}} b_\ell \cdot at_{-\ell} = K \right)}_{\varphi} \wedge \rho_\tau$$

$$\rightarrow \underbrace{\left(\sum_{i=1}^r a_i \cdot y'_i + \sum_{\ell \in \mathcal{L}} b_\ell \cdot at'_{-\ell} = K \right)}_{\varphi'}$$

or $\rho_\tau \rightarrow \sum_{i=1}^r a_i \cdot (y'_i - y_i) + \sum_{\ell \in \mathcal{L}} b_\ell \cdot (at'_{-\ell} - at_{-\ell}) = 0$

resulting in the set of equations

$$\boxed{\sum_{i=1}^r a_i \cdot \Delta(y_i, \tau) + \sum_{\ell \in \mathcal{L}} b_\ell \cdot \Delta(at_{-\ell}, \tau) = 0}$$

for every transition $\tau \in \mathcal{T}$

Example: Program DOUBLE

local y : integer where $y = 0$

$$\left[\begin{array}{l} \ell_0: y := y + 1 \\ \ell_1: \end{array} \right] \parallel \left[\begin{array}{l} m_0: y := y + 1 \\ m_1: \end{array} \right]$$

linear invariant:

$$\varphi: a \cdot y + b_{\ell_0} \cdot at_{-\ell_0} + b_{\ell_1} \cdot at_{-\ell_1} + b_{m_0} \cdot at_{-m_0} + b_{m_1} \cdot at_{-m_1} = K$$

$$(I) \quad a \cdot 0 + b_{\ell_0} \quad + b_{m_0} \quad = K$$

(initial value of y is 0)

$$(T) \quad a \cdot 1 - b_{\ell_0} + b_{\ell_1} \quad = 0 \quad (\text{for } \ell_0)$$

$$a \cdot 1 \quad - b_{m_0} + b_{m_1} = 0 \quad (\text{for } m_0)$$

Example (cont'd)

Possible solutions (basis for all solutions)

| | a | b_{ℓ_0} | b_{ℓ_1} | b_{m_0} | b_{m_1} | K |
|-------|-----|--------------|--------------|-----------|-----------|-----|
| S_1 | 0 | 1 | 1 | 0 | 0 | 1 |
| S_2 | 0 | 0 | 0 | 1 | 1 | 1 |
| S_3 | 1 | 1 | 0 | 1 | 0 | 2 |

Corresponding invariants

$$\varphi_1: at_{-\ell_0} + at_{-\ell_1} = 1 \quad (\text{control invariant})$$

$$\varphi_2: at_{-m_0} + at_{-m_1} = 1 \quad (\text{control invariant})$$

$$\varphi_3: y + at_{-\ell_0} + at_{-m_0} = 2$$

Linear Invariants for Cyclic Programs

Program $\ell_0^1: S_1 \parallel \dots \parallel \ell_0^j: S_j \parallel \dots \parallel \ell_0^m: S_m$

where S_j is of the form

$$\ell_0^j: \text{loop forever do } \underbrace{\ell_1^j, \ell_2^j, \dots, \ell_k^j}_{\text{cycle } C}$$

Define

$$\Delta(y, C) = \Delta(y, \tau_1) + \dots + \Delta(y, \tau_n)$$

Example: PRODUCER-CONSUMER

local r, ne, nf : integer where $r = 1, ne = N, nf = 0$
 b : list of integer where $b = \Lambda$

| | | |
|--|--|--|
| <pre> local x: integer l_0: loop forever do [l_1: produce x l_2: request ne l_3: request r l_4: $b := b \bullet x$ l_5: release r l_6: release nf] </pre> | | <pre> local y: integer m_0: loop forever do [m_1: request nf m_2: request r m_3: $(y, b) := (hd(b), tl(b))$ m_4: release r m_5: release ne m_6: consume y] </pre> |
| (Prod) | | (Cons) |

Invariant Construction

$$\underbrace{\sum_{i=1}^r a_i \cdot y_i}_{\text{body}} + \underbrace{\sum_{\ell \in \mathcal{L}} b_\ell \cdot at_\ell}_{\text{compensation expression}} = K$$

3 Phases:

1. Compute a_i 's
2. Compute b_ℓ 's
3. Compute K

Phase 1: Bodies

For cycle $\underbrace{\ell_1, \ell_2, \dots, \ell_k}_C$

$$\sum_{i=1}^r a_i \cdot \Delta(y_i, \tau_{\ell_1}) - b_{\ell_1} + b_{\ell_2} = 0$$

$$\sum_{i=1}^r a_i \cdot \Delta(y_i, \tau_{\ell_2}) - b_{\ell_2} + b_{\ell_3} = 0$$

⋮

$$\sum_{i=1}^r a_i \cdot \Delta(y_i, \tau_{\ell_k}) + b_{\ell_1} - b_{\ell_k} = 0$$

$$\sum_{i=1}^r a_i \cdot \Delta(y_i, C) = 0$$

Phase 2: Compensation Expressions

$$b_{\ell_0} = 0$$

For $\tau: \ell_j \rightarrow \ell_k$ where $j < k$

$$\sum_{i=1}^r a_i \cdot \Delta(y_i, \tau) - b_{\ell_j} + b_{\ell_k} = 0$$

Assume that for all $j < k$, b_{ℓ_j} is known.

Compute b_{ℓ_k} from

$$b_{\ell_k} = b_{\ell_j} - \sum_{i=1}^r a_i \cdot \Delta(y_i, \tau)$$

(independently for each cycle)

Phase 3: Right Constants

$$K = \sum_{i=1}^r a_i \cdot y_i^0$$

Note: This set of equations has the same solutions as the equations (T) + (I) except for solutions of the form

$$at_{-l_1} + \dots + at_{-l_k} = 1$$

which are produced by (T) + (I), but not by this set.

Example: PRODUCER-CONSUMER

local r, ne, nf : integer where $r = 1, ne = N, nf = 0$
 b : list of integer where $b = \Lambda$

| | | |
|--|--|---|
| <pre> [local x: integer ℓ_0: loop forever do [ℓ_1: produce x ℓ_2: request ne ℓ_3: request r ℓ_4: $b := b \bullet x$ ℓ_5: release r ℓ_6: release nf] </pre> | | <pre> [local y: integer m_0: loop forever do [m_1: request nf m_2: request r m_3: $(y, b) := (hd(b), tl(b))$ m_4: release r m_5: release ne m_6: consume y] </pre> |
|--|--|---|

Increments along each cycle:

| | Prod | Cons |
|-------|------|------|
| r | 0 | 0 |
| ne | -1 | 1 |
| nf | 1 | -1 |
| $ b $ | 1 | -1 |

Example: PRODUCER-CONSUMER

local r, ne, nf : integer where $r = 1, ne = N, nf = 0$
 b : list of integer where $b = \Lambda$

| | | |
|--|--|--|
| <pre> local x: integer ℓ₀: loop forever do [ℓ₁: produce x] [ℓ₂: request ne] [ℓ₃: request r] [ℓ₄: b := b • x] [ℓ₅: release r] [ℓ₆: release nf] </pre> | | <pre> local y: integer m₀: loop forever do [m₁: request nf] [m₂: request r] [m₃: (y, b) := (hd(b), tl(b))] [m₄: release r] [m₅: release ne] [m₆: consume y] </pre> |
|--|--|--|

We look for linear invariants with the body

$$a_r \cdot r + a_e \cdot ne + a_f \cdot nf + a_b \cdot |b|$$

Example (cont'd)

For each cycle: $\sum_{i=1}^r a_i \cdot \Delta(y_i, C) = 0$

Therefore

Prod: $-a_e + a_f + a_b = 0$

Cons: $a_e - a_f - a_b = 0$

Solutions

Bodies

- | | | |
|---------------------|-----------------------|-----------------------------|
| 1. $a_r = 1,$ | $a_e = a_f = a_b = 0$ | B ₁ : r |
| 2. $a_e = a_f = 1,$ | $a_r = a_b = 0$ | B ₂ : $ne + nf$ |
| 3. $a_e = a_b = 1,$ | $a_r = a_f = 0$ | B ₃ : $ne + b $ |

Example (cont'd)

compensation expressions

coefficients of $b_{\ell_1}, \dots, b_{m_6}$
(corresponding to bodies B_1, B_2, B_3)

```

local r, ne, nf: integer where r = 1, ne = N, nf = 0
b : list of integer where b = A

[local x: integer
  ℓ₀: loop forever do
    [ℓ₁: produce x
     ℓ₂: request ne
     ℓ₃: request r
     ℓ₄: b := b • x
     ℓ₅: release r
     ℓ₆: release nf]
] ||
[local y: integer
  m₀: loop forever do
    [m₁: request nf
     m₂: request r
     m₃: (y, b) := (hd(b), tl(b))
     m₄: release r
     m₅: release ne
     m₆: consume y]
]
    
```

| | - Prod - | | | | - Cons - | | |
|--------------|----------|-------|-------|-----------|----------|-------|-------|
| | B_1 | B_2 | B_3 | | B_1 | B_2 | B_3 |
| b_{ℓ_1} | 0 | 0 | 0 | b_{m_1} | 0 | 0 | 0 |
| b_{ℓ_2} | 0 | 0 | 0 | b_{m_2} | 0 | 1 | 0 |
| b_{ℓ_3} | 0 | 1 | 1 | b_{m_3} | 1 | 1 | 0 |
| b_{ℓ_4} | 1 | 1 | 1 | b_{m_4} | 1 | 1 | 1 |
| b_{ℓ_5} | 1 | 1 | 0 | b_{m_5} | 0 | 1 | 1 |
| b_{ℓ_6} | 0 | 1 | 0 | b_{m_6} | 0 | 0 | 0 |

Bodies

$B_1: r$

$B_2: ne + nf$

$B_3: ne + |b|$

Example (cont'd)

Right constants

Initial values $r = 1, ne = N, nf = 0, |b| = 0$

$$K_1 = 1 \cdot \underbrace{1}_r = 1$$

$$K_2 = 1 \cdot \underbrace{N}_{ne} + 1 \cdot \underbrace{0}_{nf} = N$$

$$K_3 = 1 \cdot \underbrace{N}_{ne} + 1 \cdot \underbrace{0}_{|b|} = N$$

The resulting invariants

$\varphi_1: r + at_{\ell_4,5} + at_{m_3,4} = 1$

$\varphi_2: ne + nf + at_{\ell_3,6} + at_{m_2,5} = N$

$\varphi_3: ne + |b| + at_{\ell_3,4} + at_{m_4,5} = N$