In some cases, we can identify that all transitions entering location \( s \) are parallel. Then, we can conclude that \( \text{all } 7 \text{ is also an invariant.} \)

Applying the second clause of the transition affixed invariants method to \( 0 < |t| \iff \exists \text{all } 7 \), we can extend this to

Using the method of transition affixed invariants, we can derive the invariant

\[
0 < |t| \iff \exists \text{all } 7
\]

\[
0 < |t| \iff \exists \text{all } 7
\]

Applying these second clauses of the transition affixed invariants method to \( 0 < |t| \iff \exists \text{all } 7 \), we can extend this to

Using the method of transition affixed invariants, we can derive the invariant

\[
0 < |t| \iff \exists \text{all } 7
\]

\[
0 < |t| \iff \exists \text{all } 7
\]

Then, we can conclude that \( \text{all } 7 \) is also an invariant.

No statement parallel to this process can invalidate \( \text{all } 7 \). For example, \( \text{assertion } \exists \text{all } 7 \) preserves \( \text{all } 7 \).

The assignment \( \text{assertion } \exists \text{all } 7 \) does not depend on \( \text{assertion } \exists \text{all } 7 \).

Consider a program segment of the form \( f :: f = : f ; f \) and assume that

\[
f :: f = : f ; f
\]

\[
f :: f = : f ; f
\]

with bottom-up methods.

We will proceed to describe additional methods of each of the classes, starting

The successively strengthening method we have previously described, using the

Top-Down methods. Take into account both the program and the assertion

Bottom-Up methods. Analyze the program independently of the goal assertion

Methods for deriving auxiliary invariants

The methods for deriving auxiliary invariants (which can be used to strengthen a

Non-inductive assertion) can be partitioned into

Methods for deriving auxiliary invariants

The methods for deriving auxiliary invariants (which can be used to strengthen a

Non-inductive assertion) can be partitioned into

Methods for deriving auxiliary invariants

The methods for deriving auxiliary invariants (which can be used to strengthen a

Non-inductive assertion) can be partitioned into

Methods for deriving auxiliary invariants

The methods for deriving auxiliary invariants (which can be used to strengthen a

Non-inductive assertion) can be partitioned into

Methods for deriving auxiliary invariants

The methods for deriving auxiliary invariants (which can be used to strengthen a

Non-inductive assertion) can be partitioned into

Methods for deriving auxiliary invariants

The methods for deriving auxiliary invariants (which can be used to strengthen a

Non-inductive assertion) can be partitioned into

Methods for deriving auxiliary invariants

The methods for deriving auxiliary invariants (which can be used to strengthen a

Non-inductive assertion) can be partitioned into

Methods for deriving auxiliary invariants

The methods for deriving auxiliary invariants (which can be used to strengthen a
This requires showing that no statement parallel to \( y \) can invalidate the assertion. Special attention must be given to cases where modifiable both and \( y \), however, since \( i \) is set to 2 only when \( i \neq 2 \).
For a location predicate \( \xi \) and statement \( S \), we define the execution of statement \( S \) adds the constant \( c \) to \( \xi \). For a location variable \( y \) and statement \( S \), we define the increment of \( S \) at \( y \). Let \( w \) be the integer constants.

\[ \frac{1 - f \cdot c}{1 - f} \neq \frac{0}{1 - f} \]

The execution of statement \( S \) adds the constant \( c \) to \( \xi \). For a location variable \( y \) and statement \( S \), we define the increment of \( S \) at \( y \). Let \( w \) be the integer constants.

### Construction of Linear Invariants

#### Example of Applying the Strategy

**Claim 4.** If the assertion \( \phi \) is an invariant of system \( P \), then so is \( \phi \) for every statement \( \xi \).

The assertion of mutual exclusion.

**Strategy.** If \( (\phi) \xi \) holds then so does \( (\phi)\xi \) for each variable \( x \) such that \( x \) is a set of expression defining \( \phi \). The new value of the variables \( \xi \) for these cases is a set of expression defining \( \phi \). Where \( \phi \) is a set of expression defining \( \phi \) and \( \phi \) is a set of expression defining \( \phi \).

### Logical Methods: Syntactic Strengthening

This claim leads to the following strengthening strategy:

**Premise 1.** If the verification condition condition is \( P \), then so is \( P \).

**Premise 2.** If the verification condition condition is \( P \), then so is \( P \).

**Premise 3.** If the verification condition condition is \( P \), then so is \( P \).
For an expression $E$ and a sequence of consecutive statements $\begin{array}{c}S_1:::S_k\end{array}$, we define the accumulated increment:

$$\begin{array}{c}E::S_1:::S_k\end{array} = \begin{array}{c}\sum_{j=1}^{k} E_{S_j}\end{array}$$

To simplify the presentation, assume that each process has the following structure:

$$\begin{array}{c}S_1:::S_k\end{array} = \begin{array}{c}\{y_i;E_{S_j}\}_{1 \leq i \leq n} + \{y_i;E_{S_j}\}_{1 \leq i \leq n}\end{array}$$

and that there are no nested loops or conditional statements.

For every $f$, let

$$0 = \begin{array}{c}0 = \left(\sum_{i=1}^{n} \sum_{j=1}^{k} E_{S_j} f_{i,j}\right)\end{array}$$

We conclude that the coefficients $a_i$ must satisfy the equations

$$0 = \left(\sum_{i=1}^{n} \sum_{j=1}^{k} E_{S_j} f_{i,j}\right) a_i$$

and that the sequence $f_{i,j}$ can modify $f_{i,j}$ in $f_{j}$ if $f_{i,j} \in f_{j}$ can modify $f_{i,j}$ in $f_{j}$.

We show now that for all $1 \leq i \leq n$ and $1 \leq j \leq k$:

$$0 = \left(\sum_{i=1}^{n} \sum_{j=1}^{k} E_{S_j} f_{i,j}\right) a_i$$

Applying $(\begin{array}{c}0 = \left(\sum_{i=1}^{n} \sum_{j=1}^{k} E_{S_j} f_{i,j}\right) a_i\end{array})$ to both sides of this equation, we obtain

Assume that $f$ is an invariant of a program consisting of the parallel processes $P_j$.

We conclude that the coefficients $a_i$ must satisfy the equations

$$0 = \left(\sum_{i=1}^{n} \sum_{j=1}^{k} E_{S_j} f_{i,j}\right) a_i$$

for every $j$.

Solve and find a basis of independent solutions to the set of linear equations

Computing the Bodies

Necessary Conditions

Linear Invariants Continued

Till now, for an expression $E$ we define the process-accumulated increment to be

$$\begin{array}{c}E::S_1:::S_k\end{array} = \begin{array}{c}\sum_{j=1}^{k} E_{S_j}\end{array}$$
Thus, the left-hand side of the linear invariant for program TWO-SEM has the form
\[ I = (\forall a)(\forall b)(\forall c)(\forall d)q = (\forall a)(\forall b)q = (\forall a)q = (\forall a)q = (\forall a)q = (\forall a)q \]

It follows that

<table>
<thead>
<tr>
<th>( I )</th>
<th>( I - )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 -</td>
<td>1 -</td>
</tr>
<tr>
<td>1 -</td>
<td>1 -</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( (1 - y) \cdot \varphi + (1 - y) \cdot \psi )</td>
<td>( (1 - y) \cdot \varphi + (1 - y) \cdot \psi )</td>
</tr>
</tbody>
</table>

...and

Thus, for program TWO-SEM, the full linear invariant is given by

\[ \text{SEM-TWO} = \text{null} \]

This gives rise to the following system of equations:

\[ \begin{align*}
0 &= \varphi - 1a + 1b - 1y \\
0 &= \psi + 1a - 1b + 1y \\
1 &= 1 + \varphi + \psi
\end{align*} \]

This program has the linear variables \( y, a, b \). Their process-accumulated increments are given by

\[ y_1 = \cdots + y_i + y_i \]

Therefore, any linear invariant for whose solution basis can be given by \( 1 = 1 = 1 = 1 = 1 = 1 = 1 \). Thus, any linear invariant for

\[ \text{MET-SEM} = \text{null} \]

This program will be of the form

\[ \text{MET-SEM} = \text{null} \]

...and

Thus, any linear invariant for whose solution basis can be given by \( 1 = 1 = 1 = 1 = 1 = 1 = 1 \). Thus, any linear invariant for

\[ \text{MET-SEM} = \text{null} \]

This program will be of the form

\[ \text{MET-SEM} = \text{null} \]

...and

Thus, any linear invariant for whose solution basis can be given by \( 1 = 1 = 1 = 1 = 1 = 1 = 1 \). Thus, any linear invariant for

\[ \text{MET-SEM} = \text{null} \]

This program will be of the form

\[ \text{MET-SEM} = \text{null} \]
Example: Producer-Consumer

Consider the following program PROD-CONS:

```
PROD ::
  local x : natural
  `0:
  loop forever do
  `1:
  Produce x
  `2:
  request ne
  `3:
  request r
  `4:
  L := L x
  `5:
  release r
  `6:
  release nf

CONS ::
  local y : natural
  m : 0
  loop forever do
  m : 1
  request nf
  m : 2
  request r
  m : 3
  (y ; L) := (hd (L)); tl (L))
  m : 4
  release r
  m : 5
  release ne
  m : 6
  Consume y
```

Processes `prod` produces values and moves them to process `cons` for consumption. The values are transferred via the buffer L. We wish to guarantee that the size of the buffer never exceeds the constant N. For that purpose, we maintain the semaphores `ne` which counts the number of empty slots within L and the semaphores `nf` which counts the number of occupied slots within L. Locations `r` and `nf` are exclusive.

Formally, the requirements are:

1. Locations `r` and `nf` are exclusive.
2. Never attempt to add a value to an empty buffer.
3. Never attempt to add a value to a full buffer.

leading to the bodies:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Computing Linear Invariants for PROD-CONS

As linear variables we take `r`, `ne`, `nf`, `jLj`. The process-accumulated increments for these four variables are given by

\[
\begin{align*}
v_r &= r_v = 0 \\
v_ne &= ne_v = 1 \\
v_nf &= nf_v = 1 \\
v_jLj &= jLj_v = 1 
\end{align*}
\]

This gives rise to the following set of equations:

\[
\begin{align*}
v_r + v_ne + v_nf + v_jLj &= 0 \\
v_r + v_ne + v_nf + v_jLj &= 0 
\end{align*}
\]

Since we have 4 variables and 1 independent equation, there is a solution basis containing 3 independent solutions. These can be given as

\[
\begin{align*}
\alpha_r &= \alpha_ne & \alpha_nf &= \alpha_jLj \\
0 &= 0 & 0 &= 0 \\
0 &= 0 & 0 &= 0 
\end{align*}
\]

Leading to the invariants:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A.Pnueli

Example: Deductive Verication of Reactive Systems, NYU, Fall, 2007

67

68

69

70
Todetermine the coefficients \( b' \), we compute the accumulated increments 
\[
(B_i; 0::j; 1)
\]
and 
\[
(B_i; 0::j; 1)
\]
as follows:

After computing the right-hand constants, we conclude with the following three invariants:

\[ I_1: r + a t \]
\[ I_2: n + j L j + a t \]
\[ I_3: n f + j L j + a t \]

These three obtained linear invariants imply the main safety properties of program PROD-CONS.

Drawing Conclusions

The three obtained linear invariants imply the main safety properties of program PROD-CONS.

Computation Continued