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:

- **Top-Down methods:** Analyze the program independently of the goal assertion to be proven.

- **Bottom-Up methods:** Analyze the program independently of the goal assertion to be proven.

The successivestrengthening method we have previously described, using the TLV tool, is a typical top-down method.

We will proceed to describe additional methods of each of the classes, starting with bottom-up methods.

A.Pnueli

Deductive Verification of Reactive Systems, NYU, Fall, 2007
For the first two cases, if \( i_0 \neq i \) for some process, we need to establish \( \varphi \). For the first two cases, if \( i_0 = i \) for some process, we need to establish \( \Theta \).

<table>
<thead>
<tr>
<th>Provided</th>
<th>Candidate</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi \leftarrow \Theta ) \lor ( \varphi \leftarrow \Theta )</td>
<td>[ (x)f = i \leftarrow i ]</td>
<td>For the first two cases, if ( i_0 \neq i ) for some process, we need to establish ( \varphi ).</td>
</tr>
<tr>
<td>( \varphi \leftarrow \Theta ) \lor ( \varphi \leftarrow \Theta )</td>
<td>[ (x)f = i \leftarrow i ]</td>
<td>For the first two cases, if ( i_0 = i ) for some process, we need to establish ( \Theta ).</td>
</tr>
</tbody>
</table>

Following are some configurations of statements and the candidate assertions corresponding to them:

- **provided**: \( \varphi \leftarrow \Theta \)
- **candidate**: \( (x)f = \leftarrow i \)
- **configuration**: if \( c \) then \( S_1 \) else \( S_2 \)

Deductive Verification of Reactive Systems, NYU, Fall, 2007
Consider a program segment of the form \( y := e \). For example, \( y := e \) does not depend on \( y \). Therefore, \( y := e \) preserves the assertion \( \phi \). No statement parallel to this process can invalidate \( \phi \). We previously derived an invariant at \( I \), and assume that

\[
\phi \Rightarrow I
\]

is also an invariant.

Then, we can conclude that at \( 2 \), \( \phi \Rightarrow I \) is an invariant.
Example: Peterson's Mutual Exclusion for 2 Processes

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

\[
I \not= s \land 0 = \bar{z}_f \iff \text{at-}z_4
\]

Applying the second clause of the transition affirmed invariants method to

\[
0 < \bar{z}_f \iff \text{at-}z_3.5
\]

Using forward propagation, we can extend this to

\[
0 < \bar{z}_f \iff \text{at-}z_3 \lor 0 = \bar{z}_f \iff \text{at-}z_0
\]

Applying the method of transition affirmed invariants, we can derive the invariant

```
local y_1, y_2:

P_1 ::
\begin{align*}
0 =: \bar{z}_f : \text{wait} (\text{I}', \text{I}) =: (s, \bar{z}_f) : \text{wait} \\
0 =: \bar{z}_f : \text{Non-Critical}
\end{align*}
```

Local boolean where \( s = \bar{z}_f = \bar{z}_f \) where \( \{ \text{I}', \text{I} \} \) : \( s \)

```
P_2 ::
\begin{align*}
0 =: \bar{z}_f : \text{wait} (\text{I}', \text{I}) =: (s, \bar{z}_f) : \text{wait} \\
0 =: \bar{z}_f : \text{Non-Critical}
\end{align*}
```

Loop forever do

\[
I \not= s \land 0 = \bar{z}_f
\]

A. Pnueli
This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and

This requires showing that no statement parallel to $\phi_2$ can invalidate the assertion

s. However, since if sets $s$ to 2, $s \neq I$, it only revalidates $\phi_2$ and
Consider the following loop:

\[
\begin{align*}
&j := 1 \\
&\text{while } i \geq n \\
&\quad j := j + 1
\end{align*}
\]

Then, we can conclude the following invariant:

\[
\begin{align*}
&\forall i \in \mathbb{N} : j + 1 \geq i \\
&\text{while } i \geq n \\
&\quad j := j + 1
\end{align*}
\]

We can draw similar conclusions about the loop:

\[
\begin{align*}
&i + u = ? \\
&\text{while } i \geq n \\
&\quad j := j + 1
\end{align*}
\]

Consider the following loop:

\[
\begin{align*}
&\text{while } i \geq n \\
&\quad j := j + 1
\end{align*}
\]

Loop Derived Invariants
Top-Down Derivation Methods: Generalization

Consider the following program:

```
0: sum := 0
1: for i := 1 to n do
   2: sum := sum + A[i]

::: for which we wish to prove the invariance of the assertion

\[ \forall i \geq 1 \exists s \forall u \geq 1 \sum_{r=1}^{i} A[r] = i + u = s \]
```

It is possible to generalize and conjecture the more general invariant:

\[ \exists s \forall u \geq 1 \sum_{r=1}^{i} A[r] = s \]

Since we know that, at location \(3\), \( i + u = s \) can be rewritten as:

\[ \sum_{r=1}^{i} A[r] = s \]

For which we wish to prove the invariance of the assertion:

\[ \sum_{r=1}^{i} A[r] + n = : s \]

\[ \forall i \exists s \forall u \geq 1 \sum_{r=1}^{i} A[r] + n = : s \]

Consider the following program:
This corresponds to the following insight:

If the purpose of the complete loop is to compute the sum $\sum_{i}^n A[i]$, and $i$ measures the incremental progress, then it seems reasonable that, at an intermediate stage, the partial sum $\sum_{i=1}^{i-1} A[i] + A[i]$ should contain the partial sum at stage $i$. This insight corresponds to the following:

\[ [w]A + \cdots + [1]A \]
Claim 4. If the assertion $\phi$ is an invariant of system $\mathcal{D}$, then so is $\text{pre}(\phi)$. For the state $\mathcal{E}$ where $\mathcal{E} \phi$ is defined from $\phi$ by substituting the expressions $\mathcal{E}$ for the state variables $\forall i$, the pre-condition $\text{pre}(\phi)$ can be simplified to $\forall i, \phi \leftarrow \phi \wedge \phi \leftarrow d$, where $d$ is obtained from $\phi$ by substituting the expressions $\mathcal{E}$ for the state variables $\forall i$. For these cases, the pre-condition $\text{pre}(\phi)$ is a disjunction of boolean expressions over $\lambda i$ and is a set of expressions defining $\mathcal{E} = \lambda i, \text{c} \leftarrow \mathcal{E}$. In our case, all individual transition relations have the form $\phi \leftarrow \phi \wedge \phi \leftarrow d$. As this can be written as $\phi \leftarrow \phi \wedge \phi \leftarrow d \vee \phi$, this is often established by showing separately relation $\mathcal{I} \phi$ that is often established by showing separately relation $\mathcal{I} \phi$, where each statement contributes its own transition $\mathcal{E}$ of a disjunction $\wedge$, where each statement contributes its own transition $\mathcal{E}$. As consists $\phi \leftarrow d \vee \phi$.
Strategy 1. If the verification condition fails to be $D$-valid, strengthen it by conjunction with $\text{pre}(\psi')$.

This claim leads to the following strengthening strategy:

A. Pnueli

\[ \phi \leftarrow \phi \land \text{pre}(\psi') \]
Example of Applying the Strategy

This set of assertions is inductive and implies which specifies mutual exclusion.

\[ I, \; I \rightarrow \Phi \]

Together with the bottom-up derived invariants

\[ \Phi \]

In a similar way, \( \text{pre}(m_3, \Phi^0) \) yields

\[ \Phi \]

and the variable assignment is

\[ \text{compute pre}(m_3, \Phi^0) \]

Observe that the enabling condition after execution of the statements \( e_3 \) and \( m_3 \), checking the verification conditions, we find out that this assertion fails to be inductive.

We may start the search for an invariant with

Reconsider program Peterson2. We may start the search for an invariant with
A. Pnueli

Construction of Linear Invariants

An integer variable $y$ is called linear if the modification of $y$ in each statement has the form $y' = y + c$ for some constant $c$ (possibly 0).

We are looking for invariants of the form $\forall i.\ldots\forall j.\exists q.\forall y_i$ and $K$ are integer constants.

We define:

- Increment $(y;\ell)$ = $c$ if the execution of statement $S$ adds the constant $c$ to $y$.
- Location predicate $(\ell;\ell)$ if $\ell = c$.

For a location predicate $\ell$ and statement $S$, we define the compensation expression:

$$\text{Compensation Expression} = K \quad \text{Body} \quad \text{Right Constant}$$

$$\text{Compensation Expression} = \exists \ell \in Q \cdot q \ell - a \ell \cdot q \ell + i \cdot q \ell$$

We are looking for invariants of the form $\{ I - y, f \} = \{ 0, I - y, I + y \}$.
For an expression $E$ and a sequence of consecutive statements $\vdash S_i; \vdash S_{i+1}; \cdots; \vdash S_j$, we define the accumulated increment $(\vdash E \land \Box) \downarrow + \cdots + (\vdash E \land \Box) \downarrow = (\vdash E \land \Box) \downarrow$.
To simplify the presentation, assume that each process has the following structure:

\[ P_j::[\forall \gamma : \forall \gamma . \gamma \cdots \gamma_j . S_1 \cdots S_l . \text{loop forever} \qquad \text{do} \quad 0 . \gamma_j \cdots \gamma] \]

Then, for an expression \( E \), we define the process-accumulated increment to be:

\[ (E; P_j) \downarrow = (E; \forall \gamma . \gamma_0 . \gamma \cdots \gamma_j) \downarrow \]
A Pnueli

Lecture 3: Deriving Auxiliary Invariants

For every \( i \in \{1, \ldots, n\} \), we have

\[
0 = (\bigtriangledown \cdot a_i)^{a_i} \bigtriangledown \cdot \sum_{j=1}^{\infty} \]

We conclude that the coefficients \( a_i \) must satisfy the equations

\[
0 = \bigtriangledown \cdot (\sum_{j=1}^{\infty} a_i) \bigtriangledown
\]

Applying the equation \( I + 1 = (I - 1)^{-1} \) yields

\[
I + 1 = (I - 1)^{-1} \bigtriangledown \]

We show now that if \( \exists j \), then no statement in \( P_j \) can modify \( \alpha_i \). If \( \not\exists j \), then \( \forall j \), then \( \forall j \) for all \( j \) and \( P_j \). If \( \not\exists j \), then \( \not\exists j \) and \( P_j \)’s, we obtain

\[
0 = (\bigtriangledown \cdot a_i)^{a_i} \bigtriangledown \cdot \sum_{j=1}^{\infty} (\bigtriangledown \cdot a_i)^{a_i} \bigtriangledown \cdot \sum_{j=1}^{\infty}
\]

Assume that

\[
K = \bigtriangledown a_i \cdot \sum_{j=1}^{\infty} \]

Necessary Conditions
Computing the Bodies

Any such solution provides a possible body:

\[ 0 = (\nabla \cdot D^{ij} f_i) \cdot \sum_{j=1}^{I=I} a_j \]

Solve and find a basis of independent solution to the set of linear equations.
Consider program `TWO-SEM`:

```plaintext
```
```
```}

This gives rise to the following set of equations:

\[
\begin{align*}
0 &= p y_2 - y_1 \\
0 &= p y_2 + y_1 - 1
\end{align*}
\]

\[
\begin{array}{c|c|c|c}
\text{y}_1 & \text{y}_2 & \text{P}_1 & \text{P}_2 \\
\hline
1 & 1 & + & - \\
1 & - & + & - \\
- & 1 & + & - \\
- & - & + & - \\
\end{array}
\]

Their process-accumulated increments \((y_i,P_j)\) are given by:

\[
\begin{align*}
\text{y}_1 &= a_1 + a_2 \\
\text{y}_2 &= a_1 + a_2
\end{align*}
\]

This program has the linear variables \(y_1, y_2\). Their process-accumulated increments

\[
\begin{align*}
\text{y}_1 &= a_1 + a_2 \\
\text{y}_2 &= a_1 + a_2
\end{align*}
\]

Deductive Verification of Reactive Systems, NYU, Fall, 2007

---

Example: Mutual Exclusion with Two Semaphores

A. Pnueli
A. Pnueli

Thus, any linear invariant for whose solution basis can be given by $a_2 = a_1 = \ldots + \bar{a} + \bar{a}$.
Let $j_i$ be a location within process $P_j$. Assuming that we have already computed a body $B = \sum_{i=1}^{r} \alpha_i \gamma_i$, the coefficient $b_i$ is given by:

$$b_i = (B; j_0 :: i_1)$$

Going back to program TWO-SEM with the body $B$,

$$\gamma_i = \begin{cases} \sum_{i=1}^{r} \alpha_i \gamma_i + 1 \gamma_i & \text{if } i < \gamma_i \\ B & \text{if } i = \gamma_i \\ 0 & \text{otherwise} \end{cases}$$

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>

The accumulated increments are:

$$\gamma_i = \begin{cases} \sum_{i=1}^{r} \alpha_i \gamma_i + 1 \gamma_i & \text{if } i < \gamma_i \\ B & \text{if } i = \gamma_i \\ 0 & \text{otherwise} \end{cases}$$

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>

Thus, the left-hand side of the linear invariant for program TWO-SEM has the form:

$$I = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q$$

$$0 = (\forall w)q = (\forall j)q = (\exists w)q = (\exists j)q = (0 w)q = (0 j)q$$

It follows that:

<table>
<thead>
<tr>
<th>I-</th>
<th>I-</th>
<th>\forall = ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>I-</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\exists = ?</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I = ?</td>
</tr>
</tbody>
</table>
Computing the Right-Hand-Side Constant

A.Pnueli

Computing the Right-Hand-Side Constant

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

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

\[ I = 1 = \{ i \mid 0 \leq x_i \leq 1 \} \]

Assume that the initial values of the linear variables \( y_1, \ldots, y_r \) are given, respectively, by \( y_1 = 1 \) and \( y_2 = 0 \). This together with the obvious invariants \( y_1 \leq 0 \) and \( y_2 \leq 0 \) are sufficient in order to establish mutual exclusion.

Assume that the initial values of the linear variables are given, respectively, by \( 0 \leq x_i \leq 1 \). Then, the right-hand-side constant \( K \) is given by

Assume that the initial values of the linear variables \( y_1, \ldots, y_r \) are given, respectively, by \( y_1 = 1 \) and \( y_2 = 0 \). This together with the obvious invariants \( y_1 \leq 0 \) and \( y_2 \leq 0 \) are sufficient in order to establish mutual exclusion.

Assume that the initial values of the linear variables are given, respectively, by \( 0 \leq x_i \leq 1 \). Then, the right-hand-side constant \( K \) is given by

Computing the Right-Hand-Side Constant

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

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

\[ I = 1 = \{ i \mid 0 \leq x_i \leq 1 \} \]
Formally, the requirements are:

1. The semaphore which maintains the number of occupied slots within $T$.
2. The semaphore which counts the number of empty slots within $T$.

For that purpose, we maintain the semaphore $ne$ which counts the number of empty slots within $T$. We wish to guarantee that the size of the buffer never exceeds the constant $N$. The values are transferred via the buffer $T$. We wish to guarantee this consumption.

Consider the following program:

```plaintext
Producer
::
  local x : natural
  local i : natural
  loop forever do
    Produce x
  end loop

Consu mer
::
  local y : natural
  loop forever do
    Consume y
  end loop
```

Process $Pro d$ produces values and moves them to process $Cons$ for consumption. The values are transferred via the buffer $T$. We wish to guarantee that the size of the buffer never exceeds the constant $N$. For that purpose, we maintain the semaphore $ne$ which counts the number of empty slots within $T$.
Locations $\ell_4$ and $m_3$ are exclusive.

Never attempt to dequeue an empty buffer.

Never attempt to add a value to a full buffer.

$0 < |T| \leftarrow \ell_4 - m_3 : \exists \varphi$

$N > |T| \leftarrow \ell_4 - \ell_4 \land \ell_4 - \ell_4 \lor \ell_4 - \ell_4 : \exists \varphi$

$Lecture 3: Deriving Auxiliary Invariants
Computing Linear Invariants for PROD-CONS

As linear variables we take \( r, ne, nf, jLg \). The process-accumulated increments are given by:

\[
\begin{align*}
|T| &= r, fn = ne, \forall u = nf, \forall jLg
\end{align*}
\]

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

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

\[
\begin{align*}
0 &= |T| - fn - fn + fn - jLg \cdot 0 \\
0 &= |T| + fn + fn - fn - jLg \cdot 0
\end{align*}
\]

This gives rise to the following set of equations:

\[
\begin{align*}
& I- & I- & I+ & 0 & (\exists p, q) \\
& I+ & I+ & I- & 0 & (\exists p, q) \\
& |T| &= p, fn = q, \forall u = r, \forall jLg
\end{align*}
\]

Leading to the bodies:

\[
\begin{array}{|c|c|c|c|}
\hline
I & I- & 0 & 0 \\
I & 0 & I & 0 \\
0 & 0 & 0 & 1 \\
|T| & fn & \forall u & jLg \\
\hline
\end{array}
\]
A. Pnueli

Deductive Verification of Reactive Systems, NYU, Fall, 2007

\begin{align*}
|T| + fu - : & B_3 \\
|T| + ne - : & B_2 \\
\ldots & : B_1
\end{align*}
To determine the coefficients \( q_i \), we compute the accumulated increments.

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

\[
I_1: r + al - m_{2,3} + \frac{\mu}{\lambda} + |I| + \mu - \varepsilon \quad : I_3
\]

\[
N = \begin{cases} 
0 & \text{if } m_{2,3} \neq 0 \\
\frac{\mu}{\lambda} & \text{if } m_{2,3} = 0
\end{cases}
\quad : I_2
\]

\[
I = \begin{cases} 
0 & \text{if } m_{2,3} \neq 0 \\
\varepsilon \quad & \text{if } m_{2,3} = 0
\end{cases}
\quad : I_1
\]

Computation Continued
which implies $\exists \in I', \text{ when } at_{m^3} = I', |T| < 0$ since $|T| = al_{m^3} at_{m^3} al_{m^3} al_{m^6} + al_{m^4} + fu = |T|$

From $I_3$, we obtain

which implies $\forall \in I$, we obtain

which implies $\in I$, which is impossible.

From $I_2$, we obtain

which implies $|T| + fu - : I_3$

$N = |T| + au - : I_2$

$I = : I_1$

The three obtained linear invariants

Drawing Conclusions