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

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

- **Top-Down** methods. Take into account both the program and the assertion whose invariance we wish to prove.

The successive strengthening 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.
Transition Affirmed Invariants

In some cases, we can identify that all transitions entering location \( l \), cause an assertion \( \varphi \) to hold in the post-state of the transition. If, in addition, no action of a parallel process can invalidate \( \varphi \) then the assertion

\[ \text{at}_l \rightarrow \varphi \]

is an invariant.

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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Candidate</th>
<th>Provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y := f(\vec{x}); \ l_i : )</td>
<td>( \text{at}_l \rightarrow y = f(\vec{x}) )</td>
<td>( y \not\in \vec{x} )</td>
</tr>
<tr>
<td>\text{await} ( c; \ l_i : )</td>
<td>( \text{at}_l \rightarrow c )</td>
<td></td>
</tr>
</tbody>
</table>
| \text{while} \( c \) \text{ do} \( l_1 : S; \ l_2 : \) | \( \begin{cases} 
\text{at}_l \rightarrow c \\
\land \text{at}_l \rightarrow \neg c 
\end{cases} \) |                                              |
| \( \begin{cases} 
\text{if} \( c \) \text{ then} \( l_1 : S_1 \) \\
\text{else} \ l_2 : S_2 \end{cases} \) | \( \begin{cases} 
\text{at}_l \rightarrow c \\
\land \text{at}_l \rightarrow \neg c 
\end{cases} \) |                                              |

For the first two cases, if \( l_i = l_i^0 \) for some process, we also have to establish \( \Theta \rightarrow \varphi \).
Forward Propagation

Consider a program segment of the form $\ell_1 : y := e; \ell_2$, and assume that

- We previously derived an invariant $at_{\ell_1} \rightarrow \varphi$.
- The assignment $y := e$ preserves the assertion $\varphi$. For example, $\varphi$ does not depend on $y$.
- No statement parallel to this process can invalidate $\varphi$.

Then, we can conclude that $at_{\ell_2} \rightarrow \varphi$ is also an invariant.
**Example: Peterson's Mutual Exclusion for 2 Processes**

Local variables:
- $y_1, y_2$: boolean
- $s$: \{1, 2\} where $s = 1$

Programs $P_1$ and $P_2$:

$P_1 ::
\begin{align*}
l_0 &: \text{loop forever do} \\
l_1 &: \text{Non-Critical} \\
l_2 &: (y_1, s) := (1, 1) \\
l_3 &: \text{await } y_2 = 0 \lor s \neq 1 \\
l_4 &: \text{Critical} \\
l_5 &: y_1 := 0
\end{align*}

\|$ \hspace{1cm}

$P_2 ::
\begin{align*}
m_0 &: \text{loop forever do} \\
m_1 &: \text{Non-Critical} \\
m_2 &: (y_2, s) := (1, 2) \\
m_3 &: \text{await } y_1 = 0 \lor s \neq 2 \\
m_4 &: \text{Critical} \\
m_5 &: y_2 := 0
\end{align*}$

- Using the method of **transition affirmed invariants**, we can derive the invariant $\text{at}_l l_0 \rightarrow y_1 = 0$ \land \text{at}_l l_3 \rightarrow y_1 > 0$

Using **forward propagation**, we can extend this to $\text{at}_l l_3..5 \leftrightarrow y_1 > 0$

- Applying the second clause of the **transition affirmed invariants** method to statement $l_3$, we can derive the invariant $\text{at}_l l_4 \rightarrow y_2 = 0 \lor s \neq 1$

This requires showing that no statement parallel to $l_4$ can invalidate the assertion $y_2 = 0 \lor s \neq 1$. Special attention must be given to $m_2$ which modifies both $y_2$ and $s$. However, since it sets $s$ to $2 \neq 1$, it only revalidates $y_2 = 0 \lor s \neq 1$. 

**Analysis of Reactive Systems, NYU, Spring, 2006**

132
Loop Derived Invariants

Consider the following loop:

\[
\begin{align*}
\ell_j &: \quad i := 1 \\
\ell_{j+1} &: \quad \textbf{while } i \leq n \textbf{ do} \\
& \quad \begin{cases}
\ell_{j+2} : \quad \ldots \\
\ell_k : \quad \ldots \\
\ell_{k+1} : \quad i := i + 1 \\
\ell_{k+2} : \quad \ldots
\end{cases}
\end{align*}
\]

where none of the statements \( \ell_{j+2}, \ldots, \ell_k \) and no statement parallel to this process modifies \( i \).

Then, we can conclude the following invariant:

\[
\text{at}_{\ell_{j+1..k+1}} \rightarrow 1 \leq i \leq n \quad \land \quad \text{at}_{\ell_{j+1}} \quad \land \quad \text{at}_{\ell_{k+2}} \rightarrow i = n + 1
\]

We can draw similar conclusions about the loop

\[
\ell_{j+1} : \textbf{for } i = 1 \textbf{ to } n \textbf{ do } S; \quad \ell_{k+2} : 
\]
Top-Down Derivation Methods: Generalization

Consider the following program:

\begin{align*}
\ell_0 : & \quad sum := 0 \\
\ell_1 : & \quad \textbf{for } i := 1 \text{ to } n \text{ do} \\
\ell_2 : & \quad sum := sum + A[i] \\
\ell_3 : & \quad \ldots
\end{align*}

for which we wish to prove the invariance of the assertion

\[ \varphi : \ \text{at}_{\ell_3} \rightarrow sum = \sum_{r=1}^{n} A[r] \]

Since we know that, at location \( \ell_3 \), \( i = n + 1 \), this can be rewritten as:

\[ \text{at}_{\ell_3} \rightarrow i = n + 1 \land \text{sum} = \sum_{r<i} A[r] \]

It is possible to generalize and conjecture the more general invariant

\[ \text{at}_{\ell_1..3} \rightarrow \text{sum} = \sum_{r<i} A[r] \]

This corresponds to the following insight:

If the purpose of the complete loop is to compute the sum \( A[1] + \cdots + A[n] \) and \( i \) measures the incremental progress, then it seems reasonable that, at an intermediate stage, \( \text{sum} \) should contain the partial sum \( A[1] + \cdots + A[i-1] \).
Top-Down Methods: Systematic Strengthening

Premise I2 of rule INV requires establishing the validity of \( \varphi \land \rho \rightarrow \varphi' \). As \( \rho \) consists of a disjunction \( \bigvee_{\ell} \rho_{\ell} \), where each statement \( \ell \) contributes its own transition relation \( \rho_{\ell} \), this is often established by showing separately

\[
\varphi \land \rho_{\ell} \rightarrow \varphi'
\]

for each statement \( \ell \). Equivalently, this can be written as \( \varphi \rightarrow pre(\ell, \varphi) \), where

\[
pre(\ell, \varphi) = \forall V' : (\rho_{\ell} \rightarrow \varphi').
\]

In our case, all individual transition relations have the form \( \rho_{\ell} : c_{\ell} \land V' = E_{\ell} \), where \( c_{\ell} \) is a boolean expression over \( V \), and \( E_{\ell} \) is a set of expressions defining the new values of the variables \( V \). For these cases, the pre-condition \( pre(\ell, \varphi) \) can be simplified to

\[
pre(\ell, \varphi) : c_{\ell} \rightarrow \varphi(E_{\ell}),
\]

where \( \varphi(E_{\ell}) \) is obtained from \( \varphi \) by substituting the expressions \( E_{\ell} \) for the state variables \( V \).

**Claim 14.** If the assertion \( \varphi \) is an invariant of system \( D \), then so is \( pre(\ell, \varphi) \), for every statement \( \ell \).

This claim leads to the following strengthening strategy:

**Strategy 1.** If the verification condition \( \varphi \land \rho_{\ell} \rightarrow \varphi' \) fails to be \( D \)-valid, strengthen \( \varphi \) by conjuncting it with \( pre(\ell, \varphi) \).
**Example of Applying the Strategy**

Reconsider program PETERSON2. We may start the search for an invariant with the assertion of mutual exclusion

$$\varphi_0 : \pi_1 \neq 4 \lor \pi_2 \neq 4$$

Checking the verification conditions, we find out that this assertion fails to be inductive after execution of the statements $l_3$ and $m_3$. Observing that the enabling condition for $l_3$ is $c_{l_3} : \pi_1 = 3 \land (y_2 = 0 \lor s \neq 1)$ and the variable assignment is $\pi_1 := 4$, we compute $\text{pre}(l_3, \varphi_0)$ and obtain:

$$\varphi_1 : \pi_1 = 3 \land (y_2 = 0 \lor s \neq 1) \rightarrow (4 \neq 4 \lor \pi_2 \neq 4) \sim \text{at}_{l_3} \land \text{at}_{m_4} \rightarrow y_2 \neq 0 \land s = 1$$

In a similar way, $\text{pre}(m_3, \varphi_0)$ yields

$$\varphi_2 : \text{at}_{l_4} \land \text{at}_{m_3} \rightarrow y_1 \neq 0 \land s = 2$$

Together with the bottom-up derived invariants

$$\varphi_3 : \text{at}_{l_3..5} \rightarrow y_1 = 1 \quad \varphi_4 : \text{at}_{m_3..5} \rightarrow y_2 = 1,$$

This set of assertions is inductive and implies $\varphi_0$ which specifies mutual exclusion.
Construction of Linear Invariants

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

We are looking for invariants of the form

$$\sum_{i=1}^{r} a_i \cdot y_i + \sum_{\ell \in \mathcal{L}} b_\ell \cdot at_\ell = K,$$

where $y_1, \ldots, y_r$ are linear variables, $a_i$, $b_j$, and $K$ are integer constants.

For a linear variable $y$ and statement $\ell : S$, we define the increment $\Delta(y, \ell) = c$ if the execution of statement $S$ adds the constant $c$ to $y$.

For a location predicate $\ell_j$ and statement $\ell_i : S$, we define

$$\Delta(at_{\ell_j}, \ell_i) = \begin{cases} +1 & i = j - 1 \\ -1 & i = j \\ 0 & i \notin \{j, j - 1\} \end{cases}$$

For an expression $E$ and a sequence of consecutive statements $\ell_i : S_i; \ldots; \ell_j : S_j$, we define the accumulated increment

$$\Delta(E, \ell_i..j) = \Delta(E, \ell_i) + \cdots + \Delta(E, \ell_j)$$
Linear Invariants Continued

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

\[ P_j :: \ell_0 : \text{loop forever do } [\ell_1 : S_1; \ldots; \ell_k : S_k] \]

and that there are no nested loops or conditional statements.

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

\[ \Delta(E, P_j) = \Delta(E, \ell_0..k). \]
Necessary Conditions

Assume that
\[ \sum_{i=1}^{r} a_i \cdot y_i + \sum_{\ell \in \mathcal{L}} b_{\ell} \cdot at_{-\ell} = K \]
is an invariant of a program consisting of the parallel processes \( P_1, \ldots, P_n \). Applying \( \Delta(\cdot, P_j) \) to both sides of this equality, we obtain
\[ \sum_{i=1}^{r} a_i \cdot \Delta(y_i, P_j) + \sum_{\ell \in \mathcal{L}} b_{\ell} \cdot \Delta(at_{-\ell}, P_j) = 0 \]
We show now that \( \Delta(at_{-\ell_i}, P_j) = 0 \) for all \( \ell_i \) and \( P_j \). If \( \ell_i \not\in \mathcal{L}_j \), then no statement in \( P_j \) can modify \( \ell_i \). If \( \ell_i \in \mathcal{L}_j \), then \( \Delta(at_{-\ell_i}, P_j) \) sums together \( \Delta(at_{-\ell_i}, \ell_{i-1}) = +1 \) and \( \Delta(at_{-\ell_i}, \ell_i) = -1 \), yielding 0.

We conclude that the coefficients \( a_i \) must satisfy the equations
\[ \sum_{i=1}^{r} a_i \cdot \Delta(y_i, P_j) = 0 \]
for every \( j = 1, \ldots, n \).
Computing the Bodies

Solve and find a \textit{basis} of independent solution to the set of linear equations

\[
\sum_{i=1}^{r} a_i \cdot \Delta(y_i, P_j) = 0.
\]

Any such solution provides a possible body.
Example: Mutual Exclusion with Two Semaphores

Consider program **TWO-SEM**:

\[
y_1, y_2 : \text{natural initially } y_1 = 1, y_2 = 0
\]

\[
\begin{align*}
\ell_0 : & \quad \text{loop forever do} \\
\ell_1 : & \quad \text{Non-critical} \\
\ell_2 : & \quad \text{request } y_1 \\
\ell_3 : & \quad \text{Critical} \\
\ell_4 : & \quad \text{release } y_2
\end{align*}
\]

\[
\begin{align*}
m_0 : & \quad \text{loop forever do} \\
m_1 : & \quad \text{Non-critical} \\
m_2 : & \quad \text{request } y_2 \\
m_3 : & \quad \text{Critical} \\
m_4 : & \quad \text{release } y_1
\end{align*}
\]

This program has the linear variables \(y_1, y_2\). Their process-accumulated increments \(\Delta(y_i, P_j)\) are given by

\[
\begin{array}{c|cc}
P_1 & y_1 & y_2 \\ 
P_2 & -1 & +1 \\ 
\end{array}
\]

This gives rise to the set of equations:

\[
\begin{align*}
-a_1 + a_2 & = 0 \\
a_1 - a_2 & = 0
\end{align*}
\]

whose solution basis can be given by \(a_1 = a_2 = 1\). Thus, any linear invariant for this program will be of the form

\[
y_1 + y_2 + \cdots = K
\]
Computing the Compensation Expressions

Let $\ell_{ji}$ be a location within process $P_j$. Assuming that we have already computed a body $B = \sum_{i=1}^{r} a_i \cdot y_i$, then the coefficient $b_i$ is given by

$$b_i = -\Delta(B, \ell_{0..i-1}^j)$$

Going back to program TWO-SEM with the body $B = y_1 + y_2$, we compute the accumulated increments $\Delta(y_1 + y_2, \ell_{0..i-1}^j)$ as follows:

<table>
<thead>
<tr>
<th></th>
<th>$\Delta(y_1 + y_2, \ell_{0..i-1})$</th>
<th>$\Delta(y_1 + y_2, m_{0..i-1})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i = 1$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$i = 2$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$i = 3$</td>
<td>$-1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>$i = 4$</td>
<td>$-1$</td>
<td>$-1$</td>
</tr>
</tbody>
</table>

It follows that

$$b(\ell_0) = b(m_0) = b(\ell_1) = b(m_1) = b(\ell_2) = b(m_2) = 0$$
$$b(\ell_3) = b(m_3) = b(\ell_4) = b(m_4) = 1$$

Thus, the left-hand side of the linear invariant for program $\text{TWO-SEM}$ has the form

$$y_1 + y_2 + at_{\ell_3,4} + at_{m_3,4}$$
Computing the Right-Hand-Side Constant

Assume that the initial values of the linear variables $y_1, \ldots, y_r$ are given, respectively, by $\eta_1, \ldots, \eta_r$. Then, the right-hand-side constant $K$ is given by

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

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

$$y_1 + y_2 + at_{\ell_{3,4}} + at_{m_{3,4}} = 1$$

since the initial values are $\eta_1 = 1$ and $\eta_2 = 0$. This together with the obvious invariants $y_1 \geq 0$ and $y_2 \geq 0$ are sufficient in order to establish mutual exclusion.
Example: Producer-Consumer

Consider the following program $\text{PROD-CONS}$:

$$
\begin{align*}
\text{local} & \quad r, ne, nf : \quad \text{natural where } r = 1, ne = N, nf = 0 \\
L : & \quad \text{list of natural where } L = ()
\end{align*}
$$

$\text{Prod} ::$

- $\ell_0 : \quad \text{loop forever do}$
  - $\ell_1 : \quad \text{Produce } x$
  - $\ell_2 : \quad \text{request } ne$
  - $\ell_3 : \quad \text{request } r$
  - $\ell_4 : \quad L := L \circ x$
  - $\ell_5 : \quad \text{release } r$
  - $\ell_6 : \quad \text{release } nf$

$\text{Cons} ::$

- $m_0 : \quad \text{loop forever do}$
  - $m_1 : \quad \text{request } nf$
  - $m_2 : \quad \text{request } r$
  - $m_3 : \quad (y, L) := (hd(L), tl(L))$
  - $m_4 : \quad \text{release } r$
  - $m_5 : \quad \text{release } ne$
  - $m_6 : \quad \text{Consume } y$

Process $\text{Prod}$ produces values and moves them to process $\text{Cons}$ 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 semaphore $ne$ which counts the number of empty slots within $L$ and the semaphore $nf$ which maintains the number of occupied slots within $L$. Formally, the requirements are...
\( \varphi_1 : \neg (at_{-l_4} \land at_{-m_3}) \) Locations \( l_4 \) and \( m_3 \) are exclusive.

\( \varphi_2 : \, at_{-l_4} \to |L| < N \) Never attempt to add a value to a full buffer.

\( \varphi_3 : \, at_{-m_3} \to |L| > 0 \) Never attempt to dequeue an empty buffer.
### Computing Linear Invariants for PROD-CONS

As linear variables we take \( \{r, ne, nf, |L|\} \). The process-accumulated increments for these four variables are given by

|       | \( v = r \) | \( v = ne \) | \( v = nf \) | \( v = |L| \) |
|-------|-------------|-------------|-------------|-------------|
| \( \Delta(v, P_1) \) | 0           | -1          | +1          | +1          |
| \( \Delta(v, P_2) \) | 0           | +1          | -1          | -1          |

This gives rise to the following set of equations:

\[
0 \cdot a_r - a_{ne} + a_{nf} + a_{|L|} = 0 \\
0 \cdot a_r + a_{ne} - a_{nf} - a_{|L|} = 0
\]

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

|       | \( a_r \) | \( a_{ne} \) | \( a_{nf} \) | \( a_{|L|} \) |
|-------|------------|------------|------------|------------|
| \( \vec{a}_1 \) | 1          | 0          | 0          | 0          |
| \( \vec{a}_2 \) | 0          | 1          | 0          | 1          |
| \( \vec{a}_3 \) | 0          | 0          | -1         | 1          |

Leading to the bodies:

\[
B_1 : \quad r \\\nB_2 : \quad ne + |L| \\\nB_3 : \quad -nf + |L|
\]
Computation Continued

To determine the coefficients $b_\ell$, we compute the accumulated increments $\Delta(B_i, \ell_{0..j-1})$ and $\Delta(B_i, \ell_{0..j-1})$ as follows:

<table>
<thead>
<tr>
<th></th>
<th>$j:2$</th>
<th>$j:3$</th>
<th>$j:4$</th>
<th>$j:5$</th>
<th>$j:6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta(B_1, \ell_{0..j-1})$</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−1</td>
<td>0</td>
</tr>
<tr>
<td>$\Delta(B_2, \ell_{0..j-1})$</td>
<td>0</td>
<td>−1</td>
<td>−1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\Delta(B_3, \ell_{0..j-1})$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$j:2$</th>
<th>$j:3$</th>
<th>$j:4$</th>
<th>$j:5$</th>
<th>$j:6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta(B_1, m_{0..j-1})$</td>
<td>0</td>
<td>−1</td>
<td>−1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\Delta(B_2, m_{0..j-1})$</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>−1</td>
<td>0</td>
</tr>
<tr>
<td>$\Delta(B_3, m_{0..j-1})$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

$$I_1: \quad r + at_{\ell_{4,5}} + at_{m_{3,4}} = 1$$
$$I_2: \quad ne + |L| + at_{\ell_{3,4}} + at_{m_{4,5}} = N$$
$$I_3: \quad -nf + |L| - at_{\ell_{5,6}} - at_{m_{2,3}} = 0$$
Drawing Conclusions

The three obtained linear invariants

\[
I_1 : \quad r + at_{-4,5} + at_{-m_{3,4}} = 1
\]
\[
I_2 : \quad ne + |L| + at_{-3,4} + at_{-m_{4,5}} = N
\]
\[
I_3 : \quad -nf + |L| - at_{-5,6} - at_{-m_{2,3}} = 0
\]

imply the main safety properties of program PROD-CONS.

- Property \( \varphi_1 : \neg(at_{-4} \land at_{-m_3}) \) follows from \( I_1 \), because \( at_{-4} = at_{-m_3} = 1 \) implies \( r = -1 \) which is impossible.

- From \( I_2 \), we obtain

\[
|L| = N - ne - at_{-3,4} - at_{-m_{4,5}} \leq N - at_{-4}
\]

which implies \( \varphi_2 : at_{-4} \rightarrow |L| < N \) since, when \( at_{-4} = 1 \), \( |L| \leq N - 1 \).

- From \( I_3 \), we obtain

\[
|L| = nf + at_{-5,6} + at_{-m_{2,3}} \geq at_{-m_3}
\]

which implies \( \varphi_3 : at_{-m_3} \rightarrow |L| > 0 \) since, when \( at_{-m_3} = 1 \), \( |L| \geq 1 \).