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

$$at_\cdot l \rightarrow \varphi$$

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>$l_i : \quad \ y := f(\vec{x})$</td>
<td>$at_\cdot l_i \rightarrow y = f(\vec{x})$</td>
<td>$y \notin \vec{x}$</td>
</tr>
</tbody>
</table>
| \[ \begin{align*} l_i : \ 
\text{await } c \\
\text{while } c \text{ do } l_1 : S \\
l_2 : \end{align*} \right|$ | \( \begin{align*}
\text{at}_\cdot l_i \rightarrow c \\
\land \text{at}_\cdot l_1 \rightarrow \neg c \\
\land \text{at}_\cdot l_2 \rightarrow \neg c 
\end{align*} \right|$ |
| \[ \begin{align*} l_1 : S_1 \\
\text{else } l_2 : S_2 \end{align*} \right|$ | \( \begin{align*}
\text{at}_\cdot l_1 \rightarrow c \\
\land \text{at}_\cdot l_2 \rightarrow \neg c 
\end{align*} \right|$ |

For the first two cases, if $l_i = l_0^i$ for some process, we also have to establish $\Theta \rightarrow \varphi$. 

\[\text{Analysis of Reactive Systems, NYU, Fall, 2002}\]
Forward Propagation

Consider a program segment of the form $l_1 : y := e; l_2$, and assume that

- We previously derived an invariant $at_l_1 \rightarrow \varphi$.
- 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 $at_l_2 \rightarrow \varphi$ is also an invariant.
Example: Peterson’s Mutual Exclusion for 2 Processes

\[\begin{align*}
\text{local} & \quad y_1, y_2 & : & \text{boolean where } y_1 = y_2 = 0 \\
\text{s} & : & \{1, 2\} & \text{where } s = 1
\end{align*}\]

\[
P_1:: \\
\begin{align*}
\ell_0 & : \text{loop forever do} \\
\ell_1 & : \text{Non-Critical} \\
\ell_2 & : (y_1, s) := (1, 1) \\
\ell_3 & : \text{await } y_2 = 0 \lor s \neq 1 \\
\ell_4 & : \text{Critical} \\
\ell_5 & : y_1 := 0
\end{align*}
\]

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

- Using the method of transition affirmed invariants, we can derive the invariant
  \[\text{at}_{\ell_0} \rightarrow y_1 = 0 \quad \land \quad \text{at}_{\ell_3} \rightarrow y_1 > 0\]

Using forward propagation, we can extend this to

\[\text{at}_{\ell_3..5} \iff y_1 > 0\]

- Applying the second clause of the transition affirmed invariants method to statement \(\ell_3\), we can derive the invariant

\[\text{at}_{\ell_4} \rightarrow y_2 = 0 \lor s \neq 1\]

This requires showing that no statement parallel to \(\ell_4\) 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 \(y_2 = 0 \lor s \neq 1\).
Loop Derived Invariants

Consider the following loop:

\[ \ell_j : ~ i := 1 \]
\[ \ell_{j+1} : \begin{array}{l}
\text{while } i \leq n \text{ do} \\
\quad \ell_{j+2} : \cdots \\
\quad \quad \cdots \\
\quad \ell_k : \cdots \\
\quad \ell_{k+1} : i := i + 1
\end{array} \]
\[ \ell_{k+2} : \cdots \]

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:

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

We can draw similar conclusions about the loop

\[ \ell_{j+1} : \begin{array}{l}
\text{for } i = 1 \text{ to } n \text{ do } S; \\
\ell_{k+2} : \end{array} \]
**Top-Down Derivation Methods: Generalization**

Consider the following program:

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

for which we wish to prove the invariance of the assertion

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

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

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

It is possible to generalize and conjecture the more general invariant

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

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, \(\textit{sum}\) should contain the partial sum \(A[1] + \cdots + A[i-1]\).
Top-Down Methods: Systematic Strengthening

Premise 12 of rule \( \text{INV} \) requires establishing the validity of \( \varphi \land \rho \rightarrow \varphi' \). As \( \rho \) consists of a disjunction \( \bigvee \rho \ell \), 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 \( \varphi \rightarrow \text{pre}(\ell, \varphi) \), where \( \text{pre}(\ell, \varphi) = \forall V': (\rho \ell \rightarrow \varphi') \).

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 \( \text{pre}(\ell, \varphi) \) can be simplified to

\[
\text{pre}(\ell, \varphi) : c \ell \rightarrow \varphi(E \ell),
\]

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

Claim 4. If the assertion \( \varphi \) is an invariant of system \( D \), then so is \( \text{pre}(\ell, \varphi) \), 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 \( \text{pre}(\ell, \varphi) \).
Example of Applying the Strategy

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

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

Checking the verification conditions, we find out that this assertion fails to be
inductive after execution of the statements $\ell_3$ and $m_3$. Observing that the enabling
condition for $\ell_3$ is $c_\ell_3 : \pi_1 = 3 \land (y_2 = 0 \lor s \neq 1)$ and the variable assignment
is $\pi_1 := 4$, we compute $\text{pre}(\ell_3, \varphi_0)$ and obtain:

$$\varphi_1 : \quad \pi_1 = 3 \land (y_2 = 0 \lor s \neq 1) \rightarrow (4 \neq 4 \lor \pi_2 \neq 4) \sim
\at_{\ell_3} \land \at_{m_4} \rightarrow y_2 \neq 0 \land s = 1$$

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

$$\varphi_2 : \quad \at_{\ell_4} \land \at_{m_3} \rightarrow y_1 \neq 0 \land s = 2$$

Together with the bottom-up derived invariants

$$\varphi_3 : \quad \at_{\ell_3..5} \rightarrow y_1 = 1 \quad \varphi_4 : \quad \at_{m_3..5} \rightarrow y_2 = 1,$$

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 $\ell_i \notin \mathcal{L}_j$, 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 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**:

\[
\begin{align*}
  &y_1, y_2 : \text{natural initially } y_1 = 1, y_2 = 0 \\
  &\begin{cases}
    \ell_0 : \text{loop forever do} \\
    \ell_1 : \text{Non-critical} \\
    \ell_2 : \text{request } y_1 \\
    \ell_3 : \text{Critical} \\
    \ell_4 : \text{release } y_2
  \end{cases} \\
  \parallel
  \begin{cases}
    m_0 : \text{loop forever do} \\
    m_1 : \text{Non-critical} \\
    m_2 : \text{request } y_2 \\
    m_3 : \text{Critical} \\
    m_4 : \text{release } y_1
  \end{cases}
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>P_1</th>
<th>P_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
</tr>
</tbody>
</table>

This gives rise to the following 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 $\ell^j_i$ 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

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

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

<table>
<thead>
<tr>
<th>$i$</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 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 PROD-CONS:

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

\[
\begin{align*}
\text{Prod} & :: \\
& \begin{cases}
\text{local } x : \text{natural} \\
\ell_0 : \text{loop forever do} \\
\ell_1 : \text{Produce } x \\
\ell_2 : \text{request } ne \\
\ell_3 : \text{request } r \\
\ell_4 : L := L \circ x \\
\ell_5 : \text{release } r \\
\ell_6 : \text{release } nf
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\text{Cons} & :: \\
& \begin{cases}
\text{local } y : \text{natural} \\
\ell_0 : \text{loop forever do} \\
\ell_1 : \text{request } nf \\
\ell_2 : \text{request } r \\
\ell_3 : (y, L) := (hd(L), tl(L)) \\
\ell_4 : \text{release } r \\
\ell_5 : \text{release } ne \\
\ell_6 : \text{Consume } y
\end{cases}
\end{align*}
\]

Process \textit{Prod} produces values and moves them to process \textit{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

\[
\begin{align*}
\varphi_1 & : \quad \neg(at_{\ell_4} \land at_{m_3}) & \text{Locations } \ell_4 \text{ and } m_3 \text{ are exclusive.} \\
\varphi_2 & : \quad at_{\ell_4} \rightarrow |L| < N & \text{Never attempt to add a value to a full buffer.} \\
\varphi_3 & : \quad at_{m_3} \rightarrow |L| > 0 & \text{Never attempt to dequeue an empty buffer.}
\end{align*}
\]
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:

\[
\begin{align*}
0 \cdot a_r - a_{ne} + a_{nf} + a_{|L|} &= 0 \\
0 \cdot a_r + a_{ne} - a_{nf} - a_{|L|} &= 0
\end{align*}
\]

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

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

Leading to the bodies:

\[
\begin{align*}
B_1 : & \quad r \\
B_2 : & \quad ne + |L| \\
B_3 : & \quad -nf + |L|
\end{align*}
\]
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>$\ell$</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>$\ell$</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

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

imply the main safety properties of program PROD-CONS.

- Property \( \varphi_1 : \neg(at_{\ell_{4}} \land at_{m_{3}}) \) follows from \( I_1 \), because \( at_{\ell_{4}} = at_{m_{3}} = 1 \) implies \( r = -1 \) which is impossible.

- From \( I_2 \), we obtain

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

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

- From \( I_3 \), we obtain

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

which implies \( \varphi_3 : at_{m_{3}} \rightarrow |L| > 0 \) since, when \( at_{m_{3}} = 1 \), \( |L| \geq 1 \).