References for further reading:

- Volume III of Manna & Pnueli, Chapter 1

References are available from Zohar Manna’s web page, [http://theory.stanford.edu/~zm/](http://theory.stanford.edu/~zm/); look at the class web site for a link to the initial chapters of Volume III.

Response under Justice
(Chapter 1)

**Progress Properties**

We will consider deductive methods to prove response properties (which are also applicable to obligation and guarantee properties since these are subclasses)

Response properties are those properties that can be expressed by a formula of the form

\[ \Box \Diamond p \]

for a past formula \( p \).

Progress properties:
Temporal logic plays a more prominent role and **fairness** becomes important.

Property hierarchy:

![Property hierarchy diagram](image)

Response formulas

The verification rules presented assume that the response property is expressed by a response formula

\[ p \Rightarrow \Box q \]

for past formulas \( p \) and \( q \).

Note:
- Response formula expresses a response property because of the equivalence

\[ p \Rightarrow \Box q \sim \Box (\neg p B q) \]

- Every response property can be expressed by a response formula due to the equivalence

\[ \Box q \sim T \Rightarrow \Box q \]
Overview

We consider the simple case where \( p, q \) are assertions.

The proof of a response property

\[
p \Rightarrow \Diamond q
\]

often relies on the identification of one or more so-called helpful transitions. We consider three cases:

1. Rule **RESP-J**
   (single-step response under justice)
   A single helpful transition \( \tau_h \) suffices to establish the property

\[
p \quad \tau_h \quad q
\]

2. Rule **CHAIN-J**
   (chain rule under justice)
   A fixed number of helpful transitions (independent of the value of variables) suffices to establish the property

\[
q \quad \tau_{h_n} \quad \varphi_{n-1} \quad \tau_{h_{n-1}} \quad \ldots \quad \varphi_1 \quad \tau_{h_1} \quad p
\]

3. Rule **WELL-J**
   (well-founded response under justice)
   The number of helpful transitions required to establish the property is unbounded

In all cases we will be able to use verification diagrams to represent the proof.

In practice, verification diagrams are often the preferred way to prove progress properties, because they represent the temporal structure of the program relative to the property.
Single-step rule (Motivation)

\[ p \Rightarrow \Diamond q \]

Justice requirement: it is not the case that a just transition is continuously enabled but never taken.

Single-step rule

For assertions \( p, q, \varphi \), and helpful transition \( \tau_h \in \mathcal{J} \),

\[
\begin{align*}
\text{J1.} & \quad p \rightarrow q \lor \varphi \\
\text{J2.} & \quad \{\varphi\} \mathcal{T} \{q \lor \varphi\} \\
\text{J3.} & \quad \{\varphi\} \tau_h \{q\} \\
\text{J4.} & \quad \varphi \Rightarrow En(\tau_h) \\
\end{align*}
\]

\[ p \Rightarrow \Diamond q \]

Premise J2 requires all transitions to preserve \( \varphi \) (or establish \( q \), in which case we are done).

Premise J4 ensures that the helpful transition \( \tau_h \) will be continuously enabled.

It ensures, by the justice requirement, that \( \tau_h \) will eventually be taken.

Premise J3 guarantees that it will establish \( q \).

Single-step rule (Cont’d)

In practice, this rule is not very useful:

Very few properties rely on just a single helpful transition.

This leads to the Chain rule, where we have several intermediate properties.

Useful rules

- Monotonicity:
  \[ p \Rightarrow q \quad q \Rightarrow \Diamond r \quad r \Rightarrow t \]
  \[ p \Rightarrow \Diamond t \]

- Reflexivity:
  \[ p \Rightarrow \Diamond p \]

- Transitivity:
  \[ p \Rightarrow \Diamond q \quad q \Rightarrow \Diamond r \]
  \[ p \Rightarrow \Diamond r \]

- Case analysis:
  \[ p \Rightarrow \Diamond r \quad q \Rightarrow \Diamond r \]
  \[ (p \lor q) \Rightarrow \Diamond r \]
Chain rule (Motivation)

\[ p \Rightarrow \Box q \]

Chain rule

For assertions \( p, q = \varphi_0 \) and \( \varphi_1, \ldots, \varphi_m \)
and helpful transitions \( \tau_{h_1}, \ldots, \tau_{h_m} \in J \)

\[ \begin{align*}
J_1 & : p \rightarrow \bigvee_{j=0}^{m} \varphi_j \\
J_2 & : \{ \varphi_i \} T \left\{ \bigvee_{j \leq i} \varphi_j \right\} \\
J_3 & : \{ \varphi_i \} \tau_{h_i} \left\{ \bigvee_{j < i} \varphi_j \right\} \text{ for } i = 1, \ldots, m \\
J_4 & : \varphi_i \rightarrow E_n(\tau_{h_i})
\end{align*} \]

\[ p \Rightarrow \Box q \]

J2: rank never increases
J3: rank decreases

Chain rule (Cont’d)

It is our task to find the intermediate assertions \( \varphi_m, \ldots, \varphi_1 \).

Premise J2 ensures that all transitions either preserve the current assertion or move down to a lower-ranked assertion.

Premise J4 ensures that the helpful transition \( \tau_{h_i} \) is enabled for \( \varphi_i \), which makes it impossible to stay in \( \varphi_i \) forever, by the justice requirement.

Premise J3 guarantees that the helpful transition moves down to a strictly lower-ranked assertion.

Since premises J2–J4 hold for every \( 1 \leq i \leq m \), this ensures that \( \varphi_0 = q \) will be reached eventually.

Verification Diagrams

Nodes: labeled by assertions \( \varphi_i \)
Terminal node \( \varphi_0 \)

Edges: labeled by transitions

single-lined (represents a regular transition)

double-lined (represents a helpful transition)
well-formedness conditions:

- weakly acyclic in $\rightarrow$:
  
  \[
  \text{if } \varphi_i \rightarrow \varphi_j \text{ then } i \geq j
  \]

- acyclic in $\Rightarrow$:
  
  \[
  \text{if } \varphi_i \Rightarrow \varphi_j \text{ then } i > j
  \]

- every nonterminal node has a double edge departing from it

- no transition can label both a single and a double edge departing from the same node.

Chain diagram verification conditions (Cont’d)

2. double $\tau$-edges

\[
\{\varphi\}_{\tau}\{\varphi \lor \varphi_1 \lor \ldots \lor \varphi_n\}
\]

3. enabling condition

\[
\varphi \rightarrow \text{En}(\tau)
\]
Example: Program mux-pet1 (Fig. 3.4)
(Peterson’s Algorithm for mutual exclusion)

\[
\begin{align*}
\text{local } y_1, y_2 & : \text{boolean where } y_1 = F, y_2 = F \\
 s & : \text{integer where } s = 1 \\

\ell_0 & : \text{loop forever do} \\
& \quad \begin{cases} 
\ell_1 & : \text{noncritical} \\
\ell_2 & : (y_1, s) := (T, 1) \\
\ell_3 & : \text{await } (-y_2) \lor (s \neq 1) \\
\ell_4 & : \text{critical} \\
\ell_5 & : y_1 := F 
\end{cases}
\end{align*}
\]

P_1 ::

\[
\begin{align*}
m_0 & : \text{loop forever do} \\
& \quad \begin{cases} 
m_1 & : \text{noncritical} \\
m_2 & : (y_2, s) := (T, 2) \\
m_3 & : \text{await } (-y_1) \lor (s \neq 2) \\
m_4 & : \text{critical} \\
m_5 & : y_2 := F 
\end{cases}
\end{align*}
\]


Example (Cont’d)

We now want to establish accessibility, expressed by

\[
\text{at}_-\ell_3 \Rightarrow \Box \text{at}_-\ell_4
\]

Since the two properties seem similar we would like to transform the \text{WAIT} diagram into a \text{CHAIN} diagram. This requires a double edge departing from every node. The edges labeled by \(m_3\) and \(m_4\) can be converted into double edges immediately since we have

\[
\varphi_3 \rightarrow En(m_3) \quad \text{and} \quad \varphi_2 \rightarrow En(m_4)
\]

However, \(\varphi_1 \not\rightarrow En(\ell_3)\), so we have to do some more work on \(\varphi_1\).

Example: Accessibility for MUX-PET1

In Chapter 3 of the SAFETY book we established 1-bounded overtaking, expressed by

\[
\text{at}_-\ell_3 \Rightarrow \neg \text{at}_-\ell_4 \quad \text{for MUX-PET1 with the following WAIT-diagram}
\]

The problem with \(\varphi_1: (at_-m_{0,2,5} \lor (at_-m_3 \land s = 2)) \land at_-\ell_3\) is the disjunct \(at_-m_5\), because

\[
\text{at}_-\ell_5 \rightarrow \neg En(\ell_3)
\]

Therefore we separate this disjunct and create two new assertions

\[
\varphi_1' : \quad \text{at}_-\ell_5 \land at_-\ell_3
\]

As helpful transition for \(\varphi_1'\) we identify \(m_5\). Clearly

\[
\varphi_1' \rightarrow En(m_5)
\]

and \(m_5\) leads from \(\varphi_1'\) to \(\varphi_1''\). Now we have

\[
\varphi_1'' \rightarrow En(\ell_3)
\]

and \(\ell_3\) leads from \(\varphi_1''\) to \(\varphi_0\), as required.

With some rearrangement of assertion numbers, and simplification of \(\varphi_1''\), this leads to the following chain diagram.
Example (Cont’d)

In practice one would not construct a deductive proof like this to prove accessibility (or any property) of mux-pet1:

MUX-PET1 is a finite-state program (due to the invariant $\chi_1 : s = 1 \lor s = 2$) and therefore fully automatic algorithmic methods are available.

However, the proof by verification diagram does give insight in why the property holds and the possible flows of the program to reach the goal.

Ranking functions: Motivation

In the CHAIN-J rule we used the index of the intermediate assertions as a measure of the distance from the goal. From an intermediate assertion $\varphi_n$ it takes at most $n$ helpful transitions to reach the goal.

We can generalize this idea of measuring the distance from the goal and define a distance function on the state space, and require that helpful transitions reduce the distance and all other transitions do not increase the distance. This ensures that the goal will eventually be reached.

We will measure the distance with ranking functions which map states into a well-founded domain.

Well-founded domains

Well-founded domain

$(A, \prec)$

where $A$ is a set and

$\prec$ is a well-founded order

i.e., there does not exist an infinitely descending sequence $a_0 \succ a_1 \succ a_2 \ldots$

Note: A well-founded order is transitive and irreflexive.

Examples:

$(N, <)$ is well-founded:

$n > n - 1 > n - 2 > \ldots > 0$

$(Z, <)$ is not well-founded:

$n > n - 1 > \ldots > 0 > -1 > -2 \ldots$

$(Z, |<|)$ with $x |>| y$ iff $|x| > |y|$ is well-founded:

$-7 |> -3 |> 2 |> -1 |> 0$

(Rationals in $[0, 1], <$) is not well-founded:

$1 > \frac{1}{2} > \frac{1}{4} > \frac{1}{8} > \frac{1}{16} > \ldots$
Lexicographic Product

Well-founded domains \((A_1, \prec_1)\) and \((A_2, \prec_2)\) can be combined into their **lexicographic product** \((A_1 \times A_2, \prec)\)

where
\[
(a_1, a_2) \prec (b_1, b_2) \quad \text{if} \quad a_i, b_i \in A_i
\]
iff
\[
a_1 \prec_1 b_1 \quad \text{or} \quad (a_1 = b_1 \text{ and } a_2 \prec_2 b_2).
\]

\((A_1 \times A_2, \prec)\) is also a well-founded domain.

In general, well-founded domains
\((A_1, \prec_1), \ldots, (A_n, \prec_n)\)
can be combined into their lexicographic product
\((A_1 \times \cdots \times A_n, \prec)\) where
\[
(a_1, \ldots, a_n) \prec (b_1, \ldots, b_n) \quad a_i, b_i \in A_i
\]
iff for some \(j, 1 \leq j \leq n,
\]
\[
a_1 = b_1, \ldots, a_{j-1} = b_{j-1}, a_j \prec_j b_j
\]

\((A_1 \times \cdots \times A_n, \prec)\) is also a well-founded domain.

Motivation (Cont’d)

Using \textsc{chain} diagrams to prove this, we would need a separate diagram for each value of \(N:\)

\[
\begin{array}{c}
\text{at}_-\ell_0 \\
\downarrow \ell_0 \\
\text{at}_-\ell_1 \land i = N \\
\downarrow \ell_1 \\
\text{at}_-\ell_2 \land i = N \\
\downarrow \ell_2 \\
\vdots \\
\text{at}_-\ell_j \land i = N-1 \\
\downarrow \ell_j \\
\end{array}
\]

which does not seem practical.

Well-founded rule (Motivation)

Consider program \(N:\)

\[
\begin{array}{c}
in N: \text{integer where } N > 0 \\
\text{local } i: \text{integer} \\
\ell_0: i := N \\
\ell_1: \text{while } i > 0 \text{ do} \\
\quad \ell_2: i = i - 1 \\
\ell_3: \\
\end{array}
\]

We want to prove that for program \(N:\)

\[
\text{at}_-\ell_0 \Rightarrow \Diamond \text{ at}_-\ell_3
\]

What we would like is something like the following diagram:

The problem with this diagram is that it is not acyclic in \(\Rightarrow\).

So how can we be sure that it will eventually exit the cycle to reach the goal?
Rule **WELL-J**

For assertions $p, q = \varphi_0$ and $\varphi_1, \ldots, \varphi_m$, helpful transitions $\tau_{h1}, \ldots, \tau_{hm} \in J$, a well-founded domain $(A, \prec)$, and ranking functions $\delta_0, \ldots, \delta_m : \Sigma \rightarrow A$:

- **Premise JW1:**
  \[ p \rightarrow \bigvee_{j=0}^{m} \varphi_j \]

- **Premise JW2:**
  \[ \rho_\tau \land \varphi_i \rightarrow \left[ \bigvee_{j=0}^{m} (\varphi'_j \land \delta_i \succ \delta'_j) \right] \text{ for every } \tau \in T \]

- **Premise JW3:**
  \[ \rho_{\tau_{hi}} \land \varphi_i \rightarrow \bigvee_{j=0}^{m} (\varphi'_j \land \delta_i \succ \delta'_j) \]

- **Premise JW4:**
  \[ \varphi_i \rightarrow \text{En}(\tau_{hi}) \]

\[ p \Rightarrow \Diamond q \]

(\#) for $i = 1, \ldots, m$

**Premise JW2:**
In the \textsc{chain} rule we required that all transitions resulted in a move down to a lower-ranked assertion or stay in the same assertion.

Progress towards the goal was measured by the assertion index.

Here, progress is measured by the value of the ranking function, so if a transition reduces the ranking function it may go to any assertion. If it cannot reduce the ranking function it should stay in the same assertion to keep the identity of the helpful transition.

**Premise JW3:**
The helpful transition is required to reduce the ranking function.

**Premise JW4:**
Same as in the \textsc{chain-j} rule. It ensures that the helpful transition will eventually be taken, by the justice requirement.

Since $(A, \prec)$ is well-founded there can only be a finite number of those steps, ensuring that eventually $\varphi_0$ is reached.

![Diagram](image-url)
RANK diagrams

Nodes: labeled by assertions and ranking functions

\[ \varphi_i, \delta_i \]

Terminal Nodes:

\[ \varphi_0, \delta_0 \]

Well-formedness constraint:
- Every nonterminal node \( \varphi_i, i > 0 \), has a double edge departing from it.
- No transition can label both a single and a double edge departing from the same node.

Verification conditions (Cont’d)

\[ \{ \varphi_1, \delta_1 \} \tau \{ \varphi_1 \land u \succ \delta_1 \} \lor \ldots \lor \{ \varphi_n, \delta_n \} \]

Claim: A P-valid rank diagram establishes that

\[ \bigvee_{j=0}^{m} \varphi_j \Rightarrow \Diamond \varphi_0 \]

is P-valid.

With \( p \rightarrow \bigvee_{j=0}^{m} \varphi_j \) and \( \varphi_0 \rightarrow q \), we can conclude the P-validity of

\[ p \Rightarrow \Diamond q \]
**Example: Program N**

Verification diagram for program N and property

\[ \text{at}_{-\ell_0} \Rightarrow \bigcirc \text{at}_{-\ell_3} \]

\[ \varphi_3 : \text{at}_{-\ell_0} \text{ } \delta_3 : (N, 3) \]

\[ \varphi_2 : \text{at}_{-\ell_1} \text{ } \delta_2 : (i, 2) \]

\[ \varphi_1 : \text{at}_{-\ell_2} \text{ } \delta_1 : (i, 1) \]

\[ \varphi_0 = q : \text{at}_{-\ell_3} \text{ } \delta_0 : (0, 0) \]

**Example (Cont’d): Verification conditions**

- \[ \text{at}_{-\ell_0} \rightarrow \text{at}_{-\ell_0} \lor \varphi_2 \lor \varphi_1 \lor \varphi_0 \]

Four double lines:

- \[ \varphi_1 \Rightarrow \varphi_2 : \text{at}_{-\ell_2} \land \text{at}'_{-\ell_1} \land i' = i - 1 \land \ldots \]

\[ \varphi_{\rho_{\ell_2}} \land \text{at}_{-\ell_2} \land u = (i, 1) \rightarrow \]

\[ \text{at}_{-\ell_1} \land ((i, 1) \times (i', 2)) \]

- \[ \text{at}_{-\ell_2} \rightarrow \text{at}_{-\ell_2} \text{ En}(\ell_2) \]

RANK diagram for program INC representing the proof of

\[ \text{at}_{-\ell_0} \Rightarrow \bigcirc \text{at}_{-\ell_2} \]

**Example: Program INC**

local \( y, inc : \text{integer} \) where \( y \geq 0 \land inc = 1 \)

\[
\begin{align*}
\ell_0 : & \text{ while } y > 0 \text{ do } \\
\ell_1 : & y := y + inc \\
\ell_2 : &
\end{align*}
\]

||
\[
\begin{align*}
m_0 : & \text{ inc := 0 } \\
m_1 : & \text{ inc := -1 } \\
m_2 : &
\end{align*}
\]

We want to prove for program INC

\[ \text{at}_{-\ell_0} \Rightarrow \bigcirc \text{at}_{-\ell_2} \]

Invariants:

\[ \text{at}_{-m_0} \rightarrow \text{inc} = 1 \]

\[ \text{at}_{-m_1} \rightarrow \text{inc} = 0 \]

\[ \text{at}_{-m_2} \rightarrow \text{inc} = -1 \]

While at \( m_0 \) and at \( m_1 \) no progress is made by traversing the loop \( \ell_0 - \ell_1 \). Progress is made only by moving to \( m_2 \).

While at \( m_2 \), progress is made by executing \( \ell_0 \) and \( \ell_1 \), so the loop is made explicit in the diagram.