The CHAIN rule is adequate for dealing with cases in which the number of intermediate stages is bounded by a constant (\( b \leq d \)) for the case of BAKERY-2). However, there are cases in which the number of intermediate stages cannot be bounded by a constant. Consider the trivial case of a sequential terminating loop.

How can we prove it?

Termination of this program can be specified by the response formula:

\[
0 < y < y : 0 \quad \text{while} \quad y > 0 \quad \text{do} \quad y := y
\]

Obviously, rule CHAIN cannot be used, because the number of intermediate stages depends on the initial value of variable \( y \).

Response Rules with Variable Number of Intermediate Stages

A. Pnueli

Response with Variable Ranking

Deductive Verification of Reactive Systems, NYU, Fall, 2007
There are however, some principles which are retained from rule \textsc{CHAIN}. We would like to have some measure of progress in the journey from \textit{b} to \textit{d}. Every intermediate stage should be associated with a helpful justice requirement such that a move that changes the status of \textit{f} from 0 to 1 decreases the measured distance to \textit{q}. In rule \textsc{CHAIN}, the distance was measured by the index of the assertion \textit{h} holding at the current state. In more general rules, we will introduce an explicit distance function, also called \textit{ranking} function.
Possible Domains for the Ranking Function

Possible Domains for the Ranking Function
Composite Well-Founded Domains

Given two well-founded domains \((A_1; \preceq_1)\) and \((A_2; \preceq_2)\), we introduce two ways to construct a composite well-founded domain.

**Claim 6.** If both \((A_1; \preceq_1)\) and \((A_2; \preceq_2)\) are well-founded, then so are \(A_1 \times \leq \times A_2\) and \(A_1 \operatorname{lex} A_2\).

**Proof.** It is sufficient to show that \(A_1 \times \leq \times A_2\) is well-founded.

Assume to the contrary, that there exists an infinitely descending sequence \((a_1, b_1) \leq (a_2, b_2) \leq \ldots\). Then so are \(A_1 \times \leq A_2\) and \(A_1 \times \operatorname{lex} A_2\).

The lexicographic product \(A_1 \times \leq A_2\), where \(\preceq\) is the well-founded domain \((A_1; \preceq_1)\), is the well-founded domain \((A_1 \times \leq A_2; \preceq)\), where

\[(a_1, b_1) \leq (a_2, b_2) \iff (a_1 \preceq_1 a_2) \land (b_1 \leq b_2) \land (a_2 \preceq_2 a_1) \land (b_1 \preceq_1 b_2) \text{ and } (a_2 \preceq_2 a_1) \land (b_1 \preceq_1 b_2)
\]

and the cross product \(A_1 \times A_2\), where \(\preceq\) is the well-founded domain \((A_1; \preceq_1)\) and \((A_2; \preceq_2)\), we introduce two ways to
must be infinitely descending, contradicting the well-foundedness of $\forall x$. Therefore, there exists some position $k$ such that $a_k = a_{k+1}$. Since $A_1$ is well-founded, it follows that $A_1$ satisfies $\forall x. \exists y. x < y$. From the definition of $\forall x$, it follows that the sequence of first pairs members
Rule WELL

\[ b \Diamond \iff d \]

\[ \neg \exists \psi \iff \neg \exists \psi \]

\[ (\exists \phi \wedge \exists \psi) \wedge \left( \exists \phi \right) \iff (\exists \phi \wedge \exists \psi) \iff d \wedge \exists \psi \]

For \( i = 1, \ldots, m \),

\[ \exists \psi \wedge (\exists \phi \wedge \exists \psi) \iff d \]

\[ \forall \Rightarrow \exists : \psi_0, \ldots, \psi_m, \phi_0, \ldots, \phi_m = b, d \]

Justice requirements

For a well-founded domain \( \langle A, \preceq \rangle \),

Rule WELL
Assume that the premises of rule WELL are valid. Let $d : 0 \vdash \square \phi$. We have to show that there exists a computation of and let hold at position $d$. We have to show that there exists a position $b$ such that holds at position $b$. Therefore, is false at all $j$.

By premise W3, and consequently (due to W2), there must exist an index such that that beyond $u$, since $A$ is a well-founded domain, there must exist an index $i$ is non-increasing. Since $A$ is a well-founded domain, the sequence of values of the corresponding ranking functions at the respective states. By premise W2, the sequence $d_j$ is non-increasing. Since $A$ is a well-founded domain, there must exist an index $i$.

In this way we proceed to establish an infinite sequence of indices $i_j; i_j+1; \ldots$ Denote this index by $i_j+1$. By argument similar to the above, for some $i$, denote by $i_j+1$. By premise W3, the successor state of $\phi$ at state $s_j$, and $\phi$ such that $\phi$ holds at state $i$. Since never holds beyond $b$, we must assume the contrary, that no position beyond $b$. By premise W1, $b$ must also satisfy for some $i$. Denote this index by $i_j+1$. By premise W2, the successor state of $\phi$ at state $s_j$, and $\phi$ such that $\phi$ holds at state $i$. Since never holds beyond $b$, we must assume the contrary, that no position beyond $b$. By premise W1, $b$ must also satisfy for some $i$. Denote this index by $i_j+1$. By premise W2, the successor state of $\phi$ at state $s_j$, and $\phi$ such that $\phi$ holds at state $i$.

\[ b \diamond \leq_2 \phi \text{ satisfying } f \leq_2 \phi \text{ implies that requirement } f \text{ is false at all } j = \ldots = 1 + u_j = u_j = \ldots = 1 + u p = u p \]

Thus, $\phi$ violates the justice requirement of rule WELL, and therefore is false at all $j$.

Claim 7. Rule WELL is sound for proving the response property.
### Lecture 7: Response with Variable Ranking

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0, 0, 0)</td>
<td>0 = (\bar{h} = x)</td>
<td>initially</td>
<td>0 = (\bar{h} = x)</td>
<td>naturally</td>
</tr>
<tr>
<td>(0, 0, 0, 0)</td>
<td>0 = (\bar{h} = x)</td>
<td>initially</td>
<td>0 = (\bar{h} = x)</td>
<td>naturally</td>
</tr>
<tr>
<td>(0, 0, 0, 0)</td>
<td>0 = (\bar{h} = x)</td>
<td>initially</td>
<td>0 = (\bar{h} = x)</td>
<td>naturally</td>
</tr>
<tr>
<td>(0, 0, 0, 0)</td>
<td>0 = (\bar{h} = x)</td>
<td>initially</td>
<td>0 = (\bar{h} = x)</td>
<td>naturally</td>
</tr>
</tbody>
</table>

As a well-founded domain, we choose \( \mathbb{N}^{\times 1} \times \mathbb{N}^{\times 1} \times \mathbb{N} = \forall x \cdot (\forall a, t \cdot \forall m \cdot \forall \bar{m} \cdot (a \geq 0 \land t \leq m) \land (x = 1 \lor t = 1 \land m = 1)) \). We wish to prove, using rule well, the response property are given by:

\[
\begin{align*}
\text{P} &::= [I - \bar{h} =: \bar{h} : I] \\
\text{P} &::= [I + \bar{h} =: \bar{h} : I] \\
\text{P} &::= \text{while } 0 < \bar{h} \text{ do } \text{P} \\
\text{P} &::= \text{while } 0 = x \text{ do } \text{P}
\end{align*}
\]

Application to Program up-down

A. Pnueli

Application to Program up-down

A. Pnueli
In many cases of parameterized systems $P_1 \times \cdots \times P_n$, it is possible to identify a global ranking which can be presented as the cross-product $Q_1 \times \cdots \times Q_n$. This leads to the following rule DIST-RANK:

**Rule DIST-RANK**

For a well-founded domain $(A; J_1, \ldots, J_m)$, assertions justice requirements $p, q = h_0, h_1, \ldots, h_m$, and ranking functions $0, 1, \ldots, m$

\[
W_1. \quad \forall \begin{array}{c} \exists \end{array} : \begin{array}{c} \forall \end{array} Q_0, \ldots, Q_n \quad q_0, q_1, \ldots, q_n = b, d
\]

\[
W_2. \quad d \iff q_1 \bigwedge_{\mu} = 0 \Rightarrow d \bigwedge_{\mu} q_1
\]

\[
W_3. \quad (f_\mu \geq f_\mu) \bigvee_{\mu} d \iff d \bigvee_{\mu} q_1
\]

\[
W_4. \quad ?f_0 \iff q_0 \Rightarrow d
\]

\[
L \quad b \iff d
\]
Example: Mutual Exclusion by Token Passing
The Processes

\[ \text{Processes} \]

\[ \begin{align*}
\forall n \in \mathbb{N} : & \text{wait at } \mathcal{F}_n \quad \text{Critical} \\
\forall n \in \mathbb{N} : & \text{wait at } \mathcal{F}_n \quad \text{Non-critical} \\
0 : & \text{loop forever do} \\
\end{align*} \]

\[ \begin{align*}
\forall i : & \text{request } \mathcal{F}_i \\
\forall i, j \in \mathbb{N} : & \text{if at } \mathcal{F}_i \text{ then \ if at } \mathcal{F}_j \\
0 : & \text{loop forever do} \\
\end{align*} \]

\[ \begin{align*}
\forall i, j \in \mathbb{N} : & \text{release at } \mathcal{F}_i \\
\forall i, j \in \mathbb{N} : & \text{request at } \mathcal{F}_i \\
\end{align*} \]
Together they imply mutual exclusion:

\[(\forall i \exists j \neg \text{at} \iff [i] \exists m \neg \text{at} \iff [j] \forall \neg \text{at}) \lor (\forall j \exists i \neg \text{at} \iff [j] \exists m \neg \text{at} \iff [i] \forall \neg \text{at})\]

\[I = ([i] \forall i \exists m \neg \text{at} \iff [i] \forall \neg \text{at}) \lor ([j] \forall i \exists m \neg \text{at} \iff [j] \forall \neg \text{at})\]

The following are invariants of TOKEN-RING:

\[
\begin{aligned}
\text{wait at} & \quad : m_4 \\
\text{critical} & \quad : m_3 \\
\text{wait at} & \quad : m_2 \\
\text{non-critical} & \quad : m_1 \\
\end{aligned}
\]

\[
\begin{aligned}
\text{loop forever} & \quad : I_0 \\
\end{aligned}
\]

\[
\begin{aligned}
\text{release} & \quad : J_4 \\
\text{wait at} & \quad : J_3 \\
\text{if at} & \quad : J_2 \\
\text{then} & \quad : J_1 \\
\text{request} & \quad : J_0 \\
\end{aligned}
\]

\[
0 = [N]_0 = \cdots = [2]_0 = I_1 = [1]_1 = [\text{local array}]_1 \text{ of boolean where } [N]_1 \forall \text{array}
\]

First Some Invariants

A. Pnueli
The choice of helpful justice requirements, assertions, and ranking functions for use in rule \texttt{WELL} is given by the following table:

<table>
<thead>
<tr>
<th>Process</th>
<th>Successors</th>
<th>Accessible</th>
<th>(\mathit{Liveness})</th>
</tr>
</thead>
<tbody>
<tr>
<td>([i + 1) (\vdash \downarrow)]</td>
<td>((i', i, i + 1))</td>
<td>(\downarrow)</td>
<td>(\mathit{Liveness})</td>
</tr>
<tr>
<td>([i + 1) (\vdash \downarrow)]</td>
<td>((i', i, i + 1))</td>
<td>(\downarrow)</td>
<td>(\mathit{Liveness})</td>
</tr>
<tr>
<td>([i + 1) (\vdash \downarrow)]</td>
<td>((i', i, i + 1))</td>
<td>(\downarrow)</td>
<td>(\mathit{Liveness})</td>
</tr>
</tbody>
</table>

We define the cyclic distance between \(i\) and \(z\) as

\[
N \mod (i - z) = (z', i) \downarrow
\]

for some process \(z\).

Accessibility can be specified by

Now to \texttt{Liveness}
Lecture 7: Response with Variable Ranking

Together, they imply mutual exclusion.

\[ (\forall i \in \mathbb{N}) \\left( i \neq \hat{i} \Rightarrow [i]\hat{y} \land 0 = [i]\hat{y} \right) \]

Some useful invariants:

Program: The Bakery Algorithm.

```
loop forever do
  Critical
  \[ [\forall j \neq i \Rightarrow [j]\hat{y} = 0 \land [i]\hat{y} = (\forall y \in \mathbb{N}) \max (y_{[i]}, \ldots, y_{[N]}) \] \]
  \[ 0 = \hat{y} \land \text{array of natural where} \quad \hat{y} \]
  Non-Critical
  \[ \forall y \in \mathbb{N}, \exists i \in \mathbb{N} : y = [i]\hat{y} \]
  \[ 0 < [i]\hat{y} \land \text{natural where} \quad N \]

Deductive Verication of Reactive Systems, NYU, Fall, 2007

A. Pnueli
Next, let us verify accessibility, specifiable by

\[ z = z \]

We intend to use rule \textsc{dist-rank}. For a transition \( t \), we will define

\[ \text{at} \quad \text{at-rank} \quad \Leftrightarrow \quad [z] \text{at} \]

Next, let us verify accessibility, specifiable by

Verifying Accessibility

A. Pnueli

Deductive Verification of Reactive Systems, NYU, Fall, 2007
\[ t = (t) = 0. \]

For all other transitions and all other cases, \( G(t) = 0. \)
**Alternately, Using Rule WELL**

Wecanalsouserule WELL for proving the accessibility property at `2\[z\]` at `4\[z\]` for the BAKERY algorithm.

A ranking function is defined as:
\[ R(i) = \text{count}(\text{processes with positive tickets whose values do not exceed the value of } y[i]) \]

The following table summarizes the helpful transitions and their rankings as required by rule WELL.

<table>
<thead>
<tr>
<th>{[\text{at-}z]}</th>
<th>(&lt;z,0,0))</th>
<th>([z]_2^{\geq} )</th>
<th>([z]_2^{=} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>([z]_4^{=} )</td>
<td>(&lt;z,0,0))</td>
<td>([z]_4^{=} )</td>
<td>([z]_4^{=} )</td>
</tr>
</tbody>
</table>

We can also use rule WELL for proving the accessibility property:

\[ [z]_4^{=} \text{at-} \diamond \text{ at-} [z]_2^{=} \]

**AlTERNATElY, USING RULE WELL**

Deductive Verification of Reactive Systems, NYU, Fall, 2007

\[ R(i) = \text{count}(\text{processes with positive tickets whose values do not exceed the value of } y[i]) \]
Example: A Simpler Version of Token Ring

A. Pnueli

Local $k = i$ where $[N]$ where $k = i$
It implies mutual exclusion!

$P_n \Leftarrow [i] \forall i \in \mathbb{N}$

The following is an invariant of TOKEN-RING:

First Some Invariants

\[
\begin{align*}
\text{local } k & = 1 \\
\text{loop forever do} & \\
\text{if } k & = i \text{ then} & \\
\text{end if} & \\
\text{await } k & = i & \\
\text{go to} & \\
\{ e, f, g, . . . \} & \\
\text{Critical} & \\
\oplus \Delta =: \lambda & \\
\{ \} & \\
\text{if } k & = 0 \text{ then} & \\
\text{end if} & \\
\text{local } k & = 1 \\
\end{align*}
\]
The choice of helpful transitions is given by the following table:

<table>
<thead>
<tr>
<th>Trans.</th>
<th>( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>( i )</td>
</tr>
<tr>
<td>( N )</td>
<td>( i )</td>
</tr>
<tr>
<td>( N )</td>
<td>( i )</td>
</tr>
<tr>
<td>( N )</td>
<td>( i )</td>
</tr>
</tbody>
</table>

where, due to symmetry, we can take process \([N]P\) as a representative process.

\[
[N] \sqcap \Diamond \leq [N]^3
\]

Accessibility can be specified by

**Now to Liveness**
is at process \( P \) but is just about to be released to process \( \}[1, \}(. \[N\])P. The special case \( \{\}\). \( P \). \( \) captures the situation that the token is at process \( \}[1, \}(. \( N \)) but is just about to be released to process \( \}[1, \}(. \( N \)). The special case \( \{\}\). \( P \). \( \) captures the situation that the token will eventually reach each of the processes. The \( k \)th \( \) disjunct captures the situation that the token will eventually reach process \( \}[1, \}(. \( i \)) but is just about to be released to process \( \}[1, \}(. \( i \)). The special case \( \{\}\). \( P \). \( \) captures the situation that the token is at process \( \}[1, \}(. \( N \)) but is just about to be released to process \( \}[1, \}(. \( 1 \)).

Thus, for transition \( \{\}\). \( P \). \( \) helpful, the distributed rank is given by

\[
> k \lor \{\}\]. \( P \). \( n \) \land \{\}\]. \( P \). \( 0 \}
\]

Choice of Distributed Ranking

For a given transition \( \{\}\). \( \tau \) we define \( \{\}\). \( \tau \) = \( \{\}\) \( \) if either \( \tau \) is helpful now, or there is a computation segment from the current state to the goal \( \) in which \( \tau \) may become helpful.
Proceeding in a similar way, we construct the following ranking table:

| ? = % \land [? 3] at \land ? > % \land [? 2] 3 \land ( [?] 3) % | [?] 2 % |
| ? = % \land [? 2] 3 \land ? > % \land [? 2] 3 \land ( [?] 2) % | [?] 2 % |
| ? = % \land [? 2] 1 \land ? > % \land [? 2] 0 \land ( [?] 1) % | [?] 2 % |
| ? > % \land [? 0] 0 \land at \land ( [?] 0) % | [?] 0 % |
| ? > % \land [? 0] 0 \land at \land ( [?] 1) % | [?] 0 % |
| ? > % \land [? 0] 0 \land at \land ( [?] 2) % | [?] 0 % |
| Trans. % | % |

\[ t(t) \]
Checking in TLV

In the token smv, we place:

Deductive Verification of Reactive Systems, NYU, Fall, 2007
In file `token Pf`, we place:

```plaintext
The Script File
```

A Pruelli
Lecture 7: Response with Variable Ranking

A. Pnueli

Call dist (p[N] . loc=3, p[N] . loc=4, \forall i \in \{N\})

set.args = \forall i \in \{N\} \ (d[i+5] = h[i+5] | P[i] . loc in \{0, 2, 3\} \wedge \neg k < i)\
\neg (P[N] . loc=3 \wedge P[i] . loc=4)\

End -- For (i in 1...N)

End -- To set.args

End -- To compute-inv

To compute-inv!

End -- To compute-inv

For (i in 1...N)

Let inv = \forall i \in \{1, 2, 3, 4\} \ (d[i+6] = h[i+6] | P[i] . loc in \{0, 2\}) \\
\neg (P[N] . loc=3 \wedge P[i] . loc=5) \\
Let d[p+6] = [6, p+6] . loc=3 \\
Let h[p+6] = [p+6, p] . loc=3 \\
Let p[4] = \forall i \in \{1, 2, 3\} \ (d[i+1] = h[i+1] | P[i] . loc in \{0, 2\}) \\
\neg (P[N] . loc=5 \wedge P[i] . loc=4)

Call binv (inv)

set.args; Call distr (P[N] . loc=3, P[N] . loc=4, inv, h, d)
Alternately, using rule \textsc{well} for proving the accessibility property at $\{ 3 \}^N$ for the \textsc{token-ring} algorithm.

Denote a ranking function $p = N - y$ defined as $p = N - y$.

Define a ranking function for the \textsc{token-ring} algorithm.

We can also use rule \textsc{well} for proving the accessibility property.

\begin{center}
\begin{tabular}{|c|c|c|c|}
  \hline
  $(t, 0, \triangledown)$ & $[N]^{\to t}$ & $\forall t \in [N]^{\to t}$ & $\forall t \in [N]^{\to t}$ \\
  \hline
  $(t, 1, \triangledown)$ & $[N]^{\to t}$ & $\forall t \in [N]^{\to t}$ & $\forall t \in [N]^{\to t}$ \\
  \hline
  $(t, 2, \triangledown)$ & $[N]^{\to t}$ & $\forall t \in [N]^{\to t}$ & $\forall t \in [N]^{\to t}$ \\
  \hline
  $(t, 3, \triangledown)$ & $[N]^{\to t}$ & $\forall t \in [N]^{\to t}$ & $\forall t \in [N]^{\to t}$ \\
  \hline
  $(t, 4, \triangledown)$ & $[N]^{\to t}$ & $\forall t \in [N]^{\to t}$ & $\forall t \in [N]^{\to t}$ \\
  \hline
  $(t, 5, \triangledown)$ & $[N]^{\to t}$ & $\forall t \in [N]^{\to t}$ & $\forall t \in [N]^{\to t}$ \\
  \hline
\end{tabular}
\end{center}
In TLV: \texttt{File: token-ring-lex.pt}

\begin{verbatim}
\begin{verbatim}
\textbf{\Lecture{7}}: Response with Variable Ranking
\end{verbatim}
\end{verbatim}
Lecture 7: Response with Variable Ranking

A. Pnueli

CALL set-args!

CALL compute-inv!

END -- To compute-inv!

CALL p-inv(\nu \in \{1,4,5\} in N.\[p]\[N. loc = 4\], inv, \[p]\[N. loc = 3])

CALL binv(inv);

CALL wellx-lex(p[N. loc = 3], p[N. loc = 4], inv, h, d, 2);

END -- For (i in 1...N)

END -- To set-args!

Deductive Verification of Reactive Systems, NYU, Fall, 2007
Together, they imply mutual exclusion.

\[
\frac{[\ell] \bar{h} > [i] \bar{h} \land 0 = [\ell] \bar{h} : \ell \neq \ell'}{[i] \bar{h} : \ell} \quad \text{all} \quad \exists \sigma \\
\frac{[i] \bar{h} : \ell \leftrightarrow 0 < [\ell] \bar{h} : \ell}{\text{all} \quad \exists \sigma}
\]

Some useful invariants:

Program Bakery: the Bakery Algorithm.

\[
\begin{array}{l}
0 = [i] \bar{h} : [j] \\
[\ell] \bar{h} > [i] \bar{h} \land 0 = [\ell] \bar{h} : \ell \neq \ell' \quad \text{wait} [j] \\
([N] \bar{h}, \ldots, [1] \bar{h}) \text{ max} < [i] \bar{h} : [j] \\
\text{all} \quad \exists \sigma \\
\text{loop forever do}
\end{array}
\]

0 = \bar{h} \text{ of natural where } [N] \bar{h} : \bar{h} \\
0 < N \text{ of natural where } N
For all other transitions and all other cases, \( h(t) = (t) \theta = 0 \).

The following table identifies for all transitions \( t \) when they are helpful and their ranking function:

| \( [I] h \geq [?] h \lor [?] \forall a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \Diamond a t \land [I] t \lor [?] \forall a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \Diamond a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \forall a t \land [I] t | \( [?] t \lor [?] \forall a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \Diamond a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \Diamond a t \land [I] t \lor [?] \forall a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \forall a t \land [I] t | \( [?] t \lor [?] \forall a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \Diamond a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \Diamond a t \land [I] t \lor [I] \Diamond a t \land [I] t \lor [?] \forall a t \land [I] t | \( [I] t | \( [I] t | \( [I] t | \( [I] t |

\( (t) \theta \) \hfill \( (t) \eta \) \hfill \( t \)

Deductive Verification of Reactive Systems, NYU, Fall 2007

A. Pnueli
Doing it in TLV: File bakery.smv

A. Pnueli, Lecture 7: Response with Variable Ranking
Declaring $\lambda$:

$$\mathrm{justice} \quad \text{loc} = 1, \quad \text{loc} = 3, \quad \text{loc} = 4$$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**: $\forall [\tau] \lambda : \quad 1$

**Case**
A. Pnueli

Script File: bakery-dist\_pf

To set\_args;
Let \_nh := nJ(1);
For (i in 1...\_nh)
Let h[i] := 0;
Let d[i] := 0;
End -- For (i in 1...\_nh)

For (i in 1...N)
Let b := (i - 1)*4;
Let h[b+1] := i=1 & C[i].loc=1;
Let d[b+1] := h[b+1];
Let h[b+2] := C[1].loc=2 & C[i].loc=2 & C[i].cond2;
Let d[b+2] := C[1].loc=1 | C[i].loc=2 & y[i] <= y[1];
Let h[b+3] := C[1].loc=2 & C[i].loc=3;
Let d[b+3] := C[1].loc=1 | C[i].loc=2..3 & y[i] <= y[1];
Let h[b+4] := C[1].loc=2 & C[i].loc=4;
Let d[b+4] := C[1].loc=1 | C[i].loc=2..4 & y[i] <= y[1];
End -- For (i in 1...N)

End -- To set\_args;
To compute inv:

```plaintext
let inv := 1
for i in 1...N
    let inv := inv & ((C[i].loc > 1) <-> (y[i] > 0))
               & (C[i].loc in 3..4 -> C[i].cond2)
               & (for j = i+1 to N { y[i] = 0 | y[i] != y[j] })
end
end
```

compute_inv;
call binv(inv);

set-args;
call distr(C[1].loc = 1, C[1].loc = 3, inv, h, d);

Deductive Verication of Reactive Systems, NYU, Fall, 2007
Alternately, using Rule WELL

The following table summarizes the helpful transitions and their rankings as required by rule WELL:

```
<table>
<thead>
<tr>
<th>State (i, J)</th>
<th>Constraints</th>
<th>Rankings</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, \Delta 0)</td>
<td>[i]_J \wedge \exists J' \Delta J' \wedge [i]_J</td>
<td>\Delta J'</td>
</tr>
<tr>
<td>(1, \Delta 0)</td>
<td>[i]_J \wedge \exists J' \Delta J' \wedge [i]_J</td>
<td>\Delta J'</td>
</tr>
<tr>
<td>(2, \Delta 0)</td>
<td>[i]_J \wedge \exists J' \Delta J' \wedge [i]_J</td>
<td>\Delta J'</td>
</tr>
<tr>
<td>(0, 0)</td>
<td>[i]_J \wedge \exists J' \Delta J' \wedge [i]_J</td>
<td>\Delta J'</td>
</tr>
</tbody>
</table>

Deductive Verification of Reactive Systems, NYU, Fall, 2007

We can also use Rule WELL for proving the accessibility property.

Define a ranking function for the BAKERY algorithm:

\[ [i]_J \wedge \exists J' \Delta J' \wedge [i]_J \Rightarrow [i]_J \wedge \bigvee \Delta J' \wedge [i]_J \]

\[ \Delta J' \]

\[ \bigvee \]

Alternately, using Rule WELL

Alternatively, Response with Variable Ranking
Using TLV: File bakery-lex.pf

End -- To set-args:

End -- For (t in 1..N:

For (i in 1..._nh:

Let nh := nJ(1):

For (i in 1..._nh:

Leth[i] := 0:

Let d[i] := 0:

Let c[i].loc := 2 & c[i].Toc := 4:

For (i in 1...N:

Let b := (i-1)*4:

Leth[b+1] := c[i].loc = 1 & c[i].cond2:

Let d[b+1][1] := 1;

Let d[b+1][2] := 0;

Let d[b+1][3] := 0:

Leth[b+2] := c[i].loc = 2 & c[i].cond2:

Let d[b+2][1] := 0:

Let d[b+2][2] := Del:

Let d[b+2][3] := 2:

Leth[b+3] := c[i].loc = 3:

Let d[b+3][1] := 0:

Let d[b+3][2] := Del:

Let d[b+3][3] := 1:

Leth[b+4] := c[i].loc = 4:

Let d[b+4][1] := 0:

Let d[b+4][2] := Del:

Let d[b+4][3] := 0:

End -- For (i in 1...N:

{(i) = (j) & (i) > 0} (1+1=N=t=t=t) For det + := det

End -- For (t in 1..nQ:

0 := (t) d:

0 := (t) nh:

For (t in 1..nQ:

nQ := (t) nh:

To set-args:
A. Pnueli

![LaTeX code]

Deductive Verification of Reactive Systems, NYU, Fall, 2007