Chapter 2

Response for Parameterized Programs

In this chapter we study methods for proving response properties of parameterized programs.

Parameterized programs allow the variable size (dynamic) statements

$$\overset{M}{\underset{j=1}{\|}} S[j] \qquad \text{and} \qquad \overset{M}{\underset{j=1}{\mathbf{OR}}} S[j].$$

The elaboration of such a statement is deferred until execution reaches it. At that point, the variable (or expression) M is evaluated and the program prepares to execute the statement

$$S[1] \,\|\, \ldots \,\|\, S[m] \qquad \text{or} \qquad S[1] \;\mathbf{or}\; \ldots \;\mathbf{or}\; S[m],$$

respectively, where m is the current value of M. Each parameterized statement $S[i]$, for $i = 1, \ldots, m$, is obtained by substituting the numeral i for the formal parameter j appearing in the text of the program and using it as a subscript in the labels and locally declared variables. Note that the variable M may be an input variable or may even be computed during the execution up to this point. In both cases, it is required that, on arrival to the dynamic statement, M has the value of a positive integer.

We have already considered such parameterized programs when studying methods for verifying their safety properties (Section 2.1 of the SAFETY book). To do so, we introduced several notations: If ℓ_i is a location within the parameterized statement $S[k]$, we write $at\_\ell_i[k]$ to denote that control is currently at location ℓ_i within statement $S[k]$. For a cooperation statement, we write L_i for the set of processes whose control is currently at ℓ_i, i.e.,

$$L_i \;=\; \big\{k \in [1..M] \mid at\_\ell_i[k]\big\},$$

and

$$N_i = |L_i|$$

for the size of this set. These notations are extended to sets and ranges of locations. Thus,

$$\begin{aligned}
L_{i_1,i_2,\ldots,i_k} &= L_{i_1} \cup L_{i_2} \cup \cdots \cup L_{i_k} \\
L_{i..j} &= L_i \cup L_{i+1} \cup \cdots \cup L_j \\
N_{i_1,i_2,\ldots,i_k} &= |L_{i_1,i_2,\ldots,i_k}| \\
N_{i..j} &= |L_{i..j}|.
\end{aligned}$$

The study of methods for verifying safety properties of parameterized programs in that section led to the conclusion that, with the appropriate notations, the basic rules are applicable and no special extensions are required to deal with parameterized progrmas. We intend to establish a similar conclusion for response properties, while illustrating the utility of the special notations for proving response properties of parameterized programs.

In most of the chapter, we consider the simpler case of response formulas $p \Rightarrow \diamondsuit q$, where p and q are assertions. The generalization necessary to handle the more general case that p and q are arbitrary past formulas is straightforward. Instead of assertions, we use past formulas, and instead of requiring that implications such as $p \rightarrow q$ (for assertions p and q) be P-state valid, we require that the entailment $p \Rightarrow q$, for past formulas p and q be P-valid.

2.1 Parameterized Well-Founded Rule

In most of the cases, response proofs of parameterized programs must be based on a version of rule WELL-J (Fig. 1.26), rather than on the weaker rule CHAIN-J (Fig. 1.7). This is due to the potentially unbounded number of helpful steps, which may depend on the value of the parameter M.

Rule WELL-J assumes that we can find a fixed number m of intermediate assertions $\varphi_0, \ldots, \varphi_m$, with corresponding helpful transitions $\tau_1, \ldots, \tau_m$, and ranking functions $\delta_0, \ldots, \delta_m$. To verify properties of parameterized programs, we can still use a fixed number of intermediate formulas and helpful transitions but they may refer to an additional variable k which is a process index.

To improve readability of formulas, we write $\rho_{\tau_i[k]}$ as $\rho_{\tau_i}[k]$.

Example (program TRIVIAL)

Consider the parameterized program TRIVIAL of Fig. 2.1. This program satisfies the response property

in M: **integer where** $M > 0$

$$\mathop{\Big\|}_{i=1}^{M} P[i] :: \quad [\ell_0\text{: } \textbf{skip}; \ \ell_1\text{: } \textbf{skip}; \ \ell_2\text{: }]$$

Fig. 2.1. Program TRIVIAL.

$$\underbrace{N_0 > 0}_{p} \quad \Rightarrow \quad \diamondsuit \underbrace{N_1 > 0}_{q = \varphi_0},$$

which states that, if some process is at ℓ_0, then eventually some process will be at ℓ_1.

A natural intermediate assertion to be used for proving this property is

$$\varphi[k]: \quad at\_\ell_0[k].$$

This assertion refers to the process index k such that $P[k]$ is currently at ℓ_0. The transition which is helpful for states satisfying $\varphi[k]$ is

$$\tau[k]: \quad \ell_0[k].$$

This shows that the identity of the helpful transition may also depend on the process index parameter k. ▟

In Fig. 2.2 we present rule WELL-JP, a version of rule WELL-J, extended to accommodate parameterized intermediate formulas and corresponding parameterized helpful transitions. For each $i = 1, \ldots, m$, the parameter k in $\varphi_i[k]$ and $\tau_i[k]$ is assumed to range over some nonempty set R_i whose size may depend on the inputs to the program. For $i \neq j$, the ranges R_i and R_j may be different. For each $j = 1, \ldots, m$, we use the notation $\widehat{\varphi}_j$ as an abbreviation for

$$\widehat{\varphi}_j: \quad \exists u \in R_j\text{: } \varphi_j[u]$$

indicating that $\widehat{\varphi}_j$ holds at a state if $\varphi_j[u]$ holds there, for some $u \in R_j$.

We refer to $\widehat{\varphi}_j$ as the *summary* of the parameterized formula $\varphi_j[k]$.

As usual, it is sufficient to check premise JP2 for $\tau \neq \tau_i[k]$.

Example (program TRIVIAL)

We show how to prove the response formula

$$\underbrace{N_0 > 0}_{p} \quad \Rightarrow \quad \diamondsuit \underbrace{N_1 > 0}_{q}$$

For assertions p, and $q = \varphi_0,\ \varphi_1[k], \ldots, \varphi_m[k]$,
transitions $\tau_1[k], \ldots, \tau_m[k] \in \mathcal{J}$,
a well-founded domain $(\mathcal{A}, \succ)$, and
ranking functions $\delta_0, \ldots, \delta_m\colon\ \Sigma \mapsto \mathcal{A}$

$$\text{JP1.}\quad p \;\rightarrow\; \bigvee_{j=0}^{m} \widehat{\varphi}_j$$

$$\left.\begin{array}{ll}
\text{JP2.}\quad \rho_\tau \;\land\; \varphi_i[k] \;\rightarrow & \left[\begin{array}{l} \displaystyle\bigvee_{j=0}^{m} (\widehat{\varphi}'_j \;\land\; \delta_i \succ \delta'_j) \\ \lor\ \big(\varphi'_i[k] \;\land\; \delta_i = \delta'_i\big) \end{array}\right] \\
 & \qquad \text{for every } \tau \in \mathcal{T} \\
\text{JP3.}\quad \rho_{\tau_i[k]} \;\land\; \varphi_i[k] \;\rightarrow & \displaystyle\bigvee_{j=0}^{m} (\widehat{\varphi}'_j[r] \;\land\; \delta_i \succ \delta'_j) \\
\text{JP4.}\quad \varphi_i[k] \;\rightarrow\; En\big(\tau_i[k]\big) &
\end{array}\right\} \begin{array}{c}\text{for } i = 1, \ldots, m, \\ \text{and } k \in R_i\end{array}$$

$$p \;\Rightarrow\; \diamondsuit\, q$$

Fig. 2.2. Rule WELL-JP (parameterized well-founded response under justice).

for program TRIVIAL (Fig. 2.1), using rule WELL-JP. Take $m = 1$ and the following parameterized intermediate assertions, helpful transition, and ranking functions:

$$\begin{array}{lll}
\varphi_1[k]\colon\ at_\ell_0[k], & \tau_1[k]\colon\ \ell_0[k], & \delta_1\colon\ 1 \\
q = \varphi_0[k]\colon\ N_1 > 0, & & \delta_0\colon\ 0,
\end{array}$$

where the parameter k ranges over R: $[1..M]$. Note that

$$\widehat{\varphi}_1 \quad = \quad \exists u \in [1..M]\colon\ at\_\ell_0[u] \quad = \quad N_0 > 0.$$

Let us check that all premises of rule WELL-JP are satisfied by this choice. Since $m = 1$ all multiple disjunctions over j collapse to a single disjunct.

- Premise JP1 requires

$$\underbrace{N_0 > 0}_{p} \;\rightarrow\; \cdots \lor \underbrace{N_0 > 0}_{\widehat{\varphi}_1},$$

which is valid.

- For premise JP2, it suffices to show

$$\rho_\tau \;\wedge\; \underbrace{at\_\ell_0[k]}_{\varphi[k]} \;\rightarrow\; \cdots \vee \cdots \vee \; \underbrace{at'\_\ell_0[k]}_{\varphi'[k] \,\wedge\, \delta=\delta'} \;,$$

  for every $\tau \neq \ell_0[k]$. Clearly, any such transition preserves the truth of $at\_\ell_0[k]$.

- Premise JP3 requires

$$\rho_{\ell_0}[k] \;\wedge\; \underbrace{at\_\ell_0[k]}_{\varphi[k]} \;\rightarrow\; \underbrace{N_1' > 0}_{q'} \;\vee\; \cdots,$$

  which is obviously valid.

- Premise JP4 requires the obviously valid implication

$$\underbrace{at\_\ell_0[k]}_{\varphi[k]} \;\rightarrow\; \underbrace{at\_\ell_0[k]}_{En(\tau[k])} \;.$$

This established the validity of $N_0 > 0 \Rightarrow \Diamond(N_1 > 0)$ over program TRIVIAL. ■

A Useful Well-Founded Domain

Progress in parameterized programs can often be measured by the number of processes that satisfy some requirements. For example, the response property

$$N_0 > 0 \;\Rightarrow\; \Diamond(N_0 = 0)$$

states that, starting from a state at which some processes are at location ℓ_0, eventually we will get to a state at which no process is executing at ℓ_0.

The most natural progress measure for such cases is the set of indices of the processes that currently satisfy the requirement of interest.

For example, we can respecify eventual evacuation of location ℓ_0 by the formula

$$L_0 \neq \emptyset \;\Rightarrow\; \Diamond(L_0 = \emptyset),$$

and use L_0 as the ranking function.

It is straightforward to show that the domain of subsets of $\{1, \ldots, M\}$ with the ordering relation $\succ$ taken as set inclusion, i.e., the domain $(2^{[1..M]}, \supset)$ is well founded.

While proving verification conditions involving a ranking function of the form L_i, we have to establish inclusions of the form $L_i \supseteq L_i'$ and $L_i \supset L_i'$.

To prove $L_i \supseteq L_i'$, we will establish

$$\forall j \in [1..M]\colon\; at'\_\ell_i[j] \;\rightarrow\; at\_\ell_i[j],$$

which guarantee that any index j belongs to L_i' only if it also belongs to L_i.

To prove $L_i \supset L'_i$, we will establish

$$\big(\forall j \in [1..M]\colon\ at'\_\ell_i[j] \ \rightarrow\ at\_\ell_i[j]\big) \land \big(\exists j \in [1..M]\colon\ at\_\ell_i[j] \land \neg at'\_\ell_i[j]\big).$$

The first clause of this conjunction guarantees that $L_i \supseteq L'_i$, while the second clause requires the identification of, at least, one process which has just left L_i.

Example (evacuation of ℓ_0 in program TRIVIAL)

We prove the response formula

$$\underbrace{L_0 \neq \emptyset}_{p} \ \Rightarrow\ \Diamond \underbrace{L_0 = \emptyset}_{q},$$

specifying eventual evacuation of location ℓ_0 for program TRIVIAL.

We take $m = 1$, and choose as follows:

$$\begin{array}{lll} \varphi_1[k]\colon\ at\_\ell_0[k], & \tau_1[k]\colon\ \ell_0[k], & \delta_1\colon\ L_0 \\ \varphi_0[k]\colon\ L_0 = \emptyset, & & \delta_0\colon\ \emptyset. \end{array}$$

Let us check that all premises of rule WELL-JP are satisfied by this choice.

- To establish JP1, it is sufficient to show

  $$\underbrace{L_0 \neq \emptyset}_{p} \ \rightarrow\ \cdots \lor \underbrace{\exists j \in [1..M]\colon\ at\_\ell_0[j]}_{\widehat{\varphi}_1}$$

  which is valid.

- To establish JP2, it is sufficient to show

  $$\rho_\tau \ \land\ \underbrace{at\_\ell_0[k]}_{\varphi_1[k]} \ \rightarrow\ \underbrace{at'\_\ell_0[k] \ \land\ L_0 \supseteq L'_0}_{\varphi'_1[k]\ \land\ \delta_1 \succeq \delta'_1},$$

  for every $\tau \neq \ell_0[k]$. There are two cases to be considered: $\tau = \ell_0[i]$ for $i \neq k$, and $\tau = \ell_1[i]$. It is clear that none of these transitions can invalidate $at\_\ell_0[k]$, or cause a new process to join L_0.

- For premise JP3, it is sufficient to show

  $$\rho_{\ell_0}[k] \ \land\ \underbrace{at\_\ell_0[k]}_{\varphi_1[k]} \ \rightarrow\ (L_0 \supset L'_0) \ \land\ \big(\underbrace{L'_0 = \emptyset}_{\varphi'_0} \ \land\ \underbrace{\exists j\colon\ at'\_\ell_0[j]}_{\widehat{\varphi}'_1}.$$

  The strict inclusion $L_0 \supset L'_0$ follows from the fact that transition $\ell_0[k]$ causes $P[k]$ to move away from ℓ_0 to ℓ_1, and does not change the location of any other process.

- Premise JP4,

  $$\underbrace{at\_\ell_0[k]}_{\varphi_1[k]} \ \rightarrow\ \underbrace{at\_\ell_0[k]}_{En(\tau_1[k])}$$

is obviously valid. ∎

Example (program MAX-ARRAY)

As a more advanced example, consider program MAX-ARRAY for parallel computation of the maximum of an integer array. This program was first presented in Fig. 2.5 of the SAFETY book and is reproduced here in Fig. 2.3.

$$
\begin{array}{ll}
\textbf{in} & M\colon \textbf{integer where } M \geq 1 \\
 & x\ \colon \textbf{array } [1..M] \textbf{ of integer} \\
\textbf{local} & y\ \colon \textbf{array } [1..M] \textbf{ of boolean where } y = \mathrm{T} \\
\textbf{out} & z\ \colon \textbf{integer}
\end{array}
$$

$$
\begin{array}{l}
\ell_0\colon \ \underset{i,j\in[1..M]}{\Big\|} P[i,j] :: \left[\begin{array}{l}\ell_1\colon \textbf{ if } x[i] < x[j] \textbf{ then } \ell_2\colon\ y[i] := \mathrm{F} \\ \ell_3\colon\end{array}\right] \\
\ell_4\colon \ \overset{M}{\underset{i=1}{\Big\|}} Q[i] :: \left[\begin{array}{l}\ell_5\colon \textbf{ if } y[i] \textbf{ then } \ell_6\colon\ z := x[i] \\ \ell_7\colon\end{array}\right] \\
\ell_8\colon
\end{array}
$$

Fig. 2.3. Program MAX-ARRAY (finding the maximum of an array).

In Section 2.2 of the SAFETY book, dealing with safety, we proved the partial correctness of this program, expressed by

$$at\_\ell_8 \quad \Rightarrow \quad maximal(z,\ x).$$

To complete the verification of this program, we have to prove its termination, expressed by response property

$$\underbrace{at\_\ell_0}_{p} \quad \Rightarrow \quad \Diamond\, \underbrace{at\_\ell_8}_{q}\ .$$

Intuitively, termination is obvious, since each process has a straight line program with no loops. To prove it formally, using rule WELL-JP, we identify a well-founded domain

$$(\mathcal{A}, \succ)\colon \quad \big([1..8],\ >\big) \ \times\ \big(2^{[1..M]},\ \supset\big).$$

Note that $(r_1, L_1) \succ (r_2, L_2)$ for $r_1, r_2 \in \{1, \ldots, 8\}$ and $L_1, L_2 \subseteq [1..M]$ if

$$r_1 > r_2 \qquad \text{or} \qquad (r_1 = r_2)\ \wedge\ (L_1 \supset L_2).$$

The parameterized intermediate assertions and helpful transitions, and the ranking functions are given in the table of Fig. 2.4. To compare a ranking function such as δ_6: $(6, L_2)$ with a shorter rank such as δ_4: 5, we pad the shorter tuple on the right by the empty set $\emptyset$. Thus, we are actually comparing $(6, L_2)$ with $(5, \emptyset)$.

φ_8 : at_ℓ_0	τ_8 : ℓ_0	δ_8: 8
$\varphi_7[i,j]$: $at\_\ell_1[i,j] \wedge N_{1..3} = M^2$	$\tau_7[i,j]$: $\ell_1[i,j]$	δ_7: $(7, L_1)$
$\varphi_6[i,j]$: $at\_\ell_2[i,j] \wedge N_{2,3} = M^2$	$\tau_6[i,j]$: $\ell_2[i,j]$	δ_6: $(6, L_2)$
φ_5 : $N_3 = M^2$	τ_5 : ℓ_3	δ_5: 5
φ_4 : at_ℓ_4	τ_4 : ℓ_4	δ_4: 4
$\varphi_3[i]$: $at\_\ell_5[i] \wedge N_{5..7} = M$	$\tau_3[i]$: $\ell_5[i]$	δ_3: $(3, L_5)$
$\varphi_2[i]$: $at\_\ell_6[i] \wedge N_{6,7} = M$	$\tau_2[i]$: $\ell_6[i]$	δ_2: $(2, L_6)$
φ_1 : $N_7 = M$	τ_1 : ℓ_7	δ_1: 1
$q = \varphi_0$: at_ℓ_8		δ_0: 0

Fig. 2.4. Assertions, transitions and rankings for program MAX-ARRAY.

Note that we refer to the exit transitions from the cooperation statements ℓ_0 and ℓ_4 by the names ℓ_3 and ℓ_7, respectively. These are *single* transitions that are enabled only when all processes within the cooperation statement have reached their terminal locations ℓ_3 and ℓ_7, respectively.

The summaries of the parameterized assertions are given by

$$\widehat{\varphi}_7:\quad N_1 > 0 \ \wedge\ N_{1..3} = M^2 \qquad\qquad \widehat{\varphi}_6:\quad N_2 > 0 \ \wedge\ N_{2,3} = M^2$$
$$\widehat{\varphi}_3:\quad N_5 > 0 \ \wedge\ N_{5..7} = M \qquad\qquad \widehat{\varphi}_2:\quad N_6 > 0 \ \wedge\ N_{6,7} = M.$$

As we see, the table refers to eight parameterized assertions and helpful transitions. Assertions φ_6, φ_7, and their corresponding transitions are parameterized by $i, j \in [1..M]$, while assertions φ_2, φ_3, and their corresponding transitions are parameterized by $i \in [1..M]$. Thus, assertions φ_6 and φ_7 can be viewed as parameterized by $k = \langle i, j\rangle$, ranging over $R_6 = R_7$: $[1..M] \times [1..M]$, assertions φ_2 and φ_3 are parameterized by $k = i$, ranging over $R_2 = R_3$: $[1..M]$, and assertions φ_1, φ_4, φ_5, and φ_6 can be viewed as trivially parameterized by k ranging over $R_1 = R_4 = R_5 = R_6$: $[1]$. Note that, for $i \in \{1, 4, 5, 6\}$, $\widehat{\varphi}_i = \varphi_i$.

The proof uses the following invariant

$$\psi: \begin{cases} N_{0,4,8} = 1 & \wedge \; N_{1..3,5..7} = 0 \;\; \vee \\ N_{1..3} \;\; = M^2 & \wedge \; N_{0,4..8} \;\;\; = 0 \;\; \vee \\ N_{5..7} \;\; = M & \wedge \; N_{0..4,8} \;\;\; = 0. \end{cases}$$

It is not difficult to check the premises of rule WELL-JP and verify that they are all satisfied.

For example, let us check premise JP3 for assertion $\varphi_2[i]$. We show that the following implication is state valid

$$\rho_{\ell_6}[i] \;\wedge\; \underbrace{at\_\ell_6[i] \;\wedge\; N_{6,7} = M}_{\varphi_2[i]} \;\rightarrow\; \cdots \;\vee\; \underbrace{N_7' = M \;\wedge\; (2, L_6) \succ (1, 0)}_{\widehat{\varphi}_1' \;\wedge\; \delta_2 \succ \delta_1'} \;\vee$$
$$\underbrace{N_6' > 0 \;\wedge\; N_{6,7}' = M \;\wedge\; (2, L_6) \succ (2, L_6')}_{\widehat{\varphi}_2' \;\wedge\; \delta_2 \succ \delta_2'}\;.$$

Assertion $\varphi_2[i]$: $at\_\ell_6[i] \wedge N_{6,7} = M$ implies $N_7 \leq M - 1$. To show the validity of the implication, we consider two cases. If $N_7 = M - 1$, then transition $\ell_6[i]$ leads to a successor state in which $N_7 = M$, establishing φ_1'. If $N_7 < M - 1$, there exists another index $k \neq i$ such that $at\_\ell_6[k]$ holds at the state before the transition and therefore also at the successor state. This establishes $\widehat{\varphi}_2'$ and a rank decrease at the successor state. ▟

2.2 Representation by Diagrams

To represent a proof of parameterized program by verification diagrams using rule WELL-JP, we follow the same conventions as for RANK diagrams (see Section 1.4), but introduce a special notation for parameterized nodes and edges.

A parameterized node is identified by a parameterized assertion of the form

$$\varphi_j: \quad \lambda k \in R_j\colon\; f[k],$$

where R_j is the range of the parameter k, which may be different for different j's, and f is an assertion that may refer to the parameter k.

When there is no danger of ambiguity (and the range R_j is understood from the context), we may also represent the parameterized assertion as

$$\varphi_j[k]: \quad f[k].$$

To make the treatment uniform, we view nodes labeled by an unparameterized assertion φ_u as though they were labeled by the trivially parameterized assertion

$$\varphi_u: \quad \lambda k \in \{1\}\colon\; f_u[k],$$

where $f_u[k] = \varphi_u$.

Edges departing from a node parameterized by k are of the following forms:

- A double-line edge labeled by $\tau[k]$.
- A single-line edge labeled by a *set-expression* T_r which stands for the set of transition instances $\{\tau_r[i] \mid i \in R_r\}$, where R_r is the range of the parameterized transition $\tau_r[i]$.

Example (program TRIVIAL)

In Fig. 2.5, we present a P-RANK diagram for program TRIVIAL of Fig. 2.1.

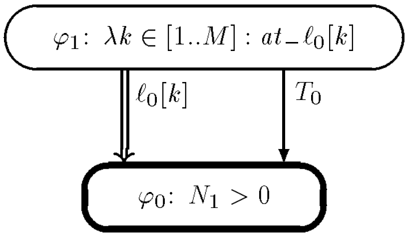

Fig. 2.5. P-RANK diagram for program TRIVIAL.

Node φ_1 in this diagram is parameterized by $k \in [1..M]$, while node φ_0 is unparameterized.

Assertion φ_0 can be trivially represented as parameterized by writing

$$\varphi_0: \quad \lambda k \in \{1\}: N_1 > 0.$$

Verification and Enabling Conditions for P-RANK Diagrams

Consider a non-terminal node labeled by the assertion $\varphi[k]$ and the ranking δ. Let L be a label of some edge departing from the node φ.

Let $\varphi_1, \ldots, \varphi_n$, $n > 0$, be the successors of φ by L-labeled edges and let $\delta_1, \ldots, \delta_n$ be the ranking functions of these successors.

- If $L = \tau[k]$ (labeling double edges), then the following verification conditions are implied:

$$\rho_\tau[k] \;\wedge\; \varphi[k] \quad\rightarrow\quad \big((\widehat{\varphi}'_1 \;\wedge\; \delta \succ \delta'_1) \;\vee\; \cdots \;\vee\; (\widehat{\varphi}'_n \;\wedge\; \delta \succ \delta'_n)\big)$$

$$\varphi[k] \quad\rightarrow\quad En\big(\tau[k]\big).$$

- If $L = T_r$ (labeling single edges), then the following verification condition is implied:

$$\rho_{\tau_r}[i] \wedge \varphi[k] \quad\rightarrow\quad \big((\varphi'[k] \wedge \delta \succeq \delta') \vee (\widehat{\varphi}'_1 \wedge \delta \succ \delta'_1) \vee \cdots \vee (\widehat{\varphi}'_n \wedge \delta \succ \delta'_n)\big).$$

In addition, Let τ_r be any transition not appearing in the label of any edge departing from φ. Then, the following verification condition is implied:

- $\varphi_{\tau_r}[i] \ \wedge\ \varphi[k] \quad\rightarrow\quad \varphi'[k] \ \wedge\ \delta \succeq \delta'.$

This corresponds to transitions labeling $n = 0$ single edges.

Example (program TRIVIAL)

The P-RANK diagram of Fig. 2.5 implies the following verification conditions:

$$\rho_{\ell_0}[k] \ \wedge\ at\_\ell_0[k] \quad\rightarrow\quad N_1' > 0 \ \wedge\ \underbrace{1}_{\delta_1} \ >\ \underbrace{0}_{\delta_0'}$$

$$at\_\ell_0[k] \quad\rightarrow\quad \underbrace{at\_\ell_0[k]}_{En\left(\ell_0[k]\right)}$$

$$\rho_{\ell_0}[i] \ \wedge\ at\_\ell_0[k] \quad\rightarrow\quad \left(at\_\ell_0'[k] \ \wedge\ 1 \ \geq\ 1\right) \ \vee\ \left(N_1' \ >\ 0 \ \wedge\ 1 \ >\ 0\right)\right)$$

$$\rho_{\ell_1}[i] \ \wedge\ at\_\ell_0[k] \quad\rightarrow\quad at\_\ell_0'[k] \ \wedge\ 1 \ \geq\ 1.$$

The last verification condition corresponds to transition $\ell_1[i]$ which labels no edge departing from φ_1.

In Fig. 2.6, we present a P-RANK diagram for the evacuation property

$$L_0 \neq \emptyset \quad\Rightarrow\quad \diamondsuit(L_0 = \emptyset)$$

for program TRIVIAL.

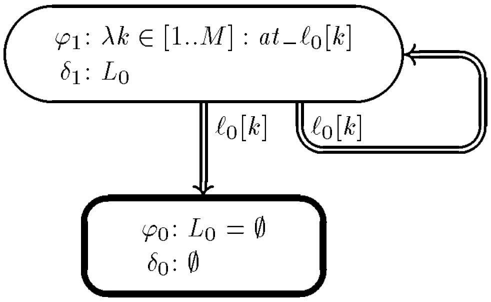

Fig. 2.6. P-RANK diagram for eventual evacuation of ℓ_0.

Note that while $P[k]$ is at ℓ_0, no other transition $\ell_0[i]$, for $i \neq k$, can cause L_0 to become empty. This is why we do not need a single edge labeled by T_0 connecting node φ_1 to node φ_0. ▪

Valid P-RANK Diagrams

A P-RANK diagram is said to be *valid over program P* (*P-valid* for short) if all the verification conditions associated with the diagram are P-state valid.

The consequences of having a valid P-RANK diagram are stated in the following claim:

Claim 2.1 (P-RANK diagrams)

A P-valid P-RANK diagram establishes that the response formula

$$\bigvee_{j=0}^{m} \quad \Rightarrow \quad \diamondsuit\, \widehat{\varphi}_0$$

is P-valid.

If, in addition, we can establish the P-state validity of the implications

$$p \quad \rightarrow \quad \bigvee_{j=0}^{m} \widehat{\varphi}_j \qquad \text{and} \qquad \widehat{\varphi}_0 \quad \rightarrow \quad q$$

then, we can conclude the P-validity of

$$p \quad \Rightarrow \quad \diamondsuit\, q.$$

Justification The first part of the claim follows from the soundness rule WELL-JP, and the observation that the verification conditions induced by a P-RANK diagram imply premises JP2–JP4 of rule WELL-JP. Given the two additional implications, we can also infer premise JP1 and the replacement of $\widehat{\varphi}_0$ by q. ∎

Example (program MAX-ARRAY)

We return to the example of program MAX-ARRAY (Fig. 2.3), for which we wish to prove termination, specified by

$$\underbrace{at\_\ell_0}_{p} \quad \Rightarrow \quad \diamondsuit\, \underbrace{at\_\ell_8}_{q}\ .$$

In Fig. 2.7 we present a P-RANK diagram for this property.

In this diagram, we use encapsulation conventions to factor out the prefix $\lambda_{i,j} \in [1..M]$ from φ_6 and φ_7 and write it as a label of the supernode containing these nodes. A similar factoring was applied to nodes φ_2 and φ_3.

When we compute the default verification conditions corresponding to transitions that do not label departing edges, it is important to consider these transitions only for the values of the parameter which do not appear in labels of departing edges.

Consider, for example, the verification conditions for the parameterized assertion φ_6: $\lambda i, j \in [1..M]$: $at\_\ell_2[i,j] \vee N_{2,3} = M^2$. The verification condition corresponding to the double edge departing from φ_6 is:

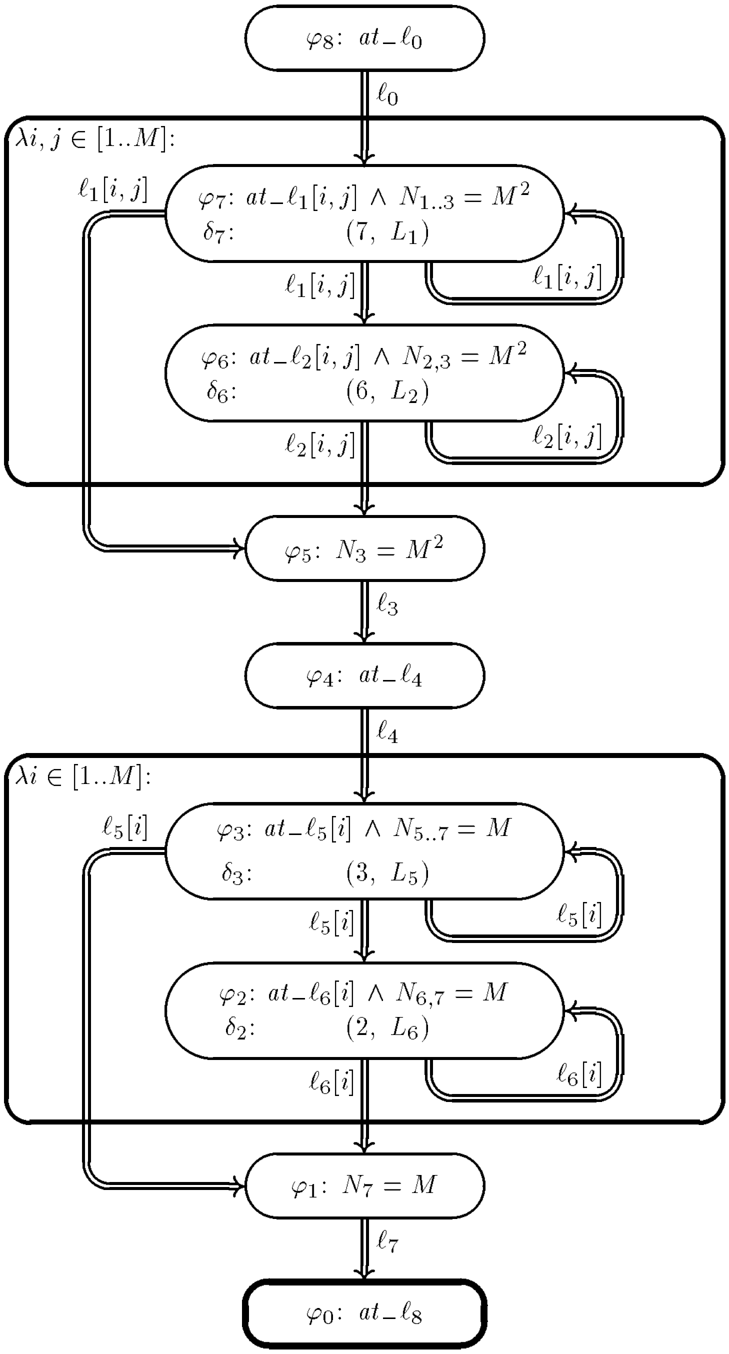

Fig. 2.7. P-RANK diagram by rule WELL-JH for program MAX-ARRAY.

$$\rho_{\ell_2}[i,j] \ \wedge \ \underbrace{at\_\ell_2[i,j] \ \wedge \ N_{2,3} = M^2}_{\varphi_6} \ \rightarrow$$

$$\big(at'\_\ell_2[i,j] \ \wedge \ N'_{2,3} = M^2 \ \wedge \ (6, L_2) \ \succ \ (6, L'_2)\big) \ \vee \ \big(N'_3 = M^2 \ \wedge \ (6, L_2) \ \succ \ 5\big).$$

Among the default verification conditions, we should include

$$(k,r) \neq (i,j) \ \wedge \ \rho_{\ell_2}[k,r] \ \wedge \ at\_\ell_2[i,j] \ \wedge \ N_{2,3} = M^2 \ \rightarrow$$
$$at'\_\ell_2[i,j] \ \wedge \ N'_{2,3} = M^2 \ \wedge \ (6, L_2) \ \succ \ (6, L'_2),$$

as asll we

$$\rho_{\ell_1}[k,r] \ \wedge \ at\_\ell_2[i,j] \ \wedge \ N_{2,3} = M^2 \ \rightarrow$$
$$at'\_\ell_2[i,j] \ \wedge \ N'_{2,3} = M^2 \ \wedge \ (6, L_2) \ \succ \ (6, L'_2).$$

Note the difference between the condition corresponding to $\ell_2[k,r]$ and the one corresponding to $\ell_1[k,r]$. In the case of ℓ_2, we have to exclude the parameter value $(k,r) = (i,j)$, since $\ell_2[i,j]$ already labels an existing edge departing from $\varphi[i,j]$. To do so, we add the conjunct $(k,r) \neq (i,j)$ to the antecedent of the implication. In the case of ℓ_1, which labels no edge departing from $\varphi_7[i,j]$, no such restriction is necessary.

Example (mutual exclusion by turn setting)

Consider parameterized program TURN of Fig. 2.8 which manages mutual exclusion between an arbitrary number of processes by turn setting. The program uses a shared variable t, which is initially 0. Each process $P[i]$, interested in entering its critical section, loops in $\ell_{2..5}$ trying to change t to the identity number of $P[i]$, namely i. The grouped statement at ℓ_3 is such that it sets t to i only if it finds $t = 0$. If $P[i]$ finds $t \neq 0$ it does not modify t but still moves to ℓ_4. At ℓ_4 the process checks whether its execution of ℓ_3 succeeded in setting t to i. Finding a value $t = i$ at ℓ_4, $P[i]$ enters its critical section ℓ_5. Otherwise it returns to ℓ_2 and then to ℓ_3 to try again.

Mutual exclusion is ensured by the invariant

$$\chi: \quad i \in L_{5,6} \ \rightarrow \ t = i.$$

We wish to show a weak-accessibility property, formulated by

$$\psi: \quad \underbrace{N_2 > 0 \ \wedge \ t = 0}_{p} \ \Rightarrow \ \diamondsuit \underbrace{N_5 > 0}_{q} \ .$$

This formula states that if some process is at ℓ_2 with $t = 0$, then eventually some process will get to ℓ_5. We refer to it as *weak accessibility* (sometimes also as *communal accessibility*), since it does not guarantee that the same process detected at ℓ_2 with $t = 0$ will be the one to enter ℓ_5. The stronger property

$$
\begin{array}{l}
\textbf{in} \quad M\text{: }\textbf{integer where } M > 1 \\
\textbf{local } t \ \text{: }\textbf{integer where } t = 0
\end{array}
$$

$$
\overset{M}{\underset{i=1}{\|}} P[i] :: \left[\begin{array}{l}
\ell_0\text{: }\textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\text{: }\textbf{noncritical} \\
\ell_2\text{: }\textbf{while } t \neq i \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_3\text{: } \langle \textbf{if } t = 0 \textbf{ then } t := i \rangle \\
\ell_4\text{: } \textbf{if } t = i \textbf{ then} \\
\quad\quad \ell_5\text{: }\textbf{critical}
\end{array}\right] \\
\ell_6\text{: } t := 0
\end{array}\right]
\end{array}\right]
$$

Fig. 2.8. Program TURN (mutual exclusion by turn setting).

of accessibility, claiming that any individual process visiting ℓ_2 with $t = 0$, will eventually reach ℓ_5, which can be expressed by the formula

$$at\_\ell_2[i] \ \land \ t = 0 \quad \Rightarrow \quad \diamondsuit \ at\_\ell_5[i],$$

is not valid for this program. In **Problem 2.1**, the reader is requested to show that accessibility is not guaranteed for program TURN.

To verify the weak-accessibility property expressed by ψ, we use the following in rule WELL-JP

$$
\begin{array}{llllll}
\varphi_3[k]\text{:} & at\_\ell_2[k] \ \land \ N_3 = 0 \ \land \ t = 0 & \delta_3\text{:} & 3 & \tau_3[k]\text{:} & \ell_2[k] \\
\varphi_2[k]\text{:} & at\_\ell_3[k] \ \land \ t = 0 & \delta_2\text{:} & 2 & \tau_2[k]\text{:} & \ell_3[k] \\
\varphi_1\text{:} & at\_\ell_4[t] \ \land \ t \neq 0 & \delta_1\text{:} & 1 & \tau_1\text{:} & \ell_4[t] \\
q = \varphi_0\text{:} & N_5 > 0 & \delta_0\text{:} & 0. & &
\end{array}
$$

We proceed to check the premises of rule WELL-JP.

- Premise JP1 requires showing $p \rightarrow \cdots \lor \widehat{\varphi}_3$. Since $N_2 > 0$ implies the assertion $\widehat{\varphi}_3$: $\exists k$: $at\_\ell_2[k]$, the premise follows.
- Premise JP2 will be checked for each $i = 1, 2, 3$.
  - $\varphi_3[k]$: $at\_\ell_2[k] \land \ N_3 = 0 \ \land \ t = 0$

  Most of the transitions satisfy JP2 for φ_3 by preserving φ_3 and δ_3. The only transitions that may falsify φ_3 are transitions of the form $\ell_2[i]$. Transition $\ell_2[i]$ leads to a state satisfying $\widehat{\varphi}_2$: $\exists r$: $at\_\ell_2[r] \ \land \ t = 0$.

  - $\varphi_2[k]$: $at\_\ell_3[k] \ \land \ t = 0$

The only transitions that can falsify φ_2 are of the form $\ell_3[i]$. However, such a transition leads to a state satisfying φ_1: $at\_\ell_4[t] \wedge t \neq 0$.

- φ_1: $at\_\ell_4[t] \;\wedge\; t \neq 0$

The only transition that can falsify φ_1 is $\ell_4[t]$ which leads to a state satisfying the goal assertion q: $N_5 > 0$.

- Premise JP3 will be checked for each $i = 1, 2, 3$.

- $\varphi_3[k]$: $at\_\ell_2[k] \;\wedge\; N_3 = 0 \;\wedge\; t = 0$

Here we show

$$\rho_{\ell_2}[k] \quad \wedge \quad \underbrace{\cdots \;\wedge\; t = 0}_{\varphi_3} \quad \rightarrow \quad \underbrace{\exists r\colon\; at'\_\ell_3[r] \;\wedge\; t' = 0}_{\widehat{\varphi}'_2} \quad \wedge \;\; 3 > 2.$$

Since $\rho_{\ell_2}[k]$ implies $at'\_\ell_3[k]$ and $t' = t$, the implication is valid.

- $\varphi_2[k]$: $at\_\ell_3[k] \;\wedge\; t = 0$

Here we show

$$\rho_{\ell_3}[k] \quad \wedge \quad \underbrace{N_3 > 0 \;\wedge\; t = 0}_{\varphi_2} \quad \rightarrow \quad \underbrace{at'\_\ell_4[t] \;\wedge\; t' \neq 0}_{\widehat{\varphi}'_1} \quad \wedge \;\; 2 > 1.$$

Since $\rho_{\ell_3}[k]$ under $t = 0$ implies $at'\_\ell_4[k]$ and $t' = k \neq 0$, the right-hand side follows.

- φ_1: $at\_\ell_4[t] \;\wedge\; t \neq 0$

We will show

$$\rho_{\ell_4}[t] \quad \wedge \quad \underbrace{at\_\ell_4[t] \;\wedge\; t \neq 0}_{\varphi_1} \quad \rightarrow \quad \underbrace{N'_5 > 0}_{q'} \;.$$

Since $\rho_{\ell_4}[t]$ and $t \neq 0$ imply $at'\_\ell_5[i]$, the right-hand side follows.

- Premise JP4 is obviously satisfied, since $\varphi_i[k]$ implies $En\big(\tau_i[k]\big)$ for $i = 1, 2, 3$.

We can summarize this WELL-JP proof in the P-RANK diagram presented in Fig. 2.9.

▟

Program RACE

Consider program RACE, presented in Fig. 2.10. This program consists of M competing processes. The first process $P[k_1]$ to perform statement ℓ_1 will set its local variable $t[k_1]$ to 1, while incrementing the shared variable y to 2. This process eventually gets back to location ℓ_0 and, on finding $t[k_1] = 1$ will proceed to ℓ_2. All processes performing statement ℓ_1 later than $P[k_1]$ will obtain values of $t[k]$ greater than 1, and cannot exit the while loop at ℓ_0.

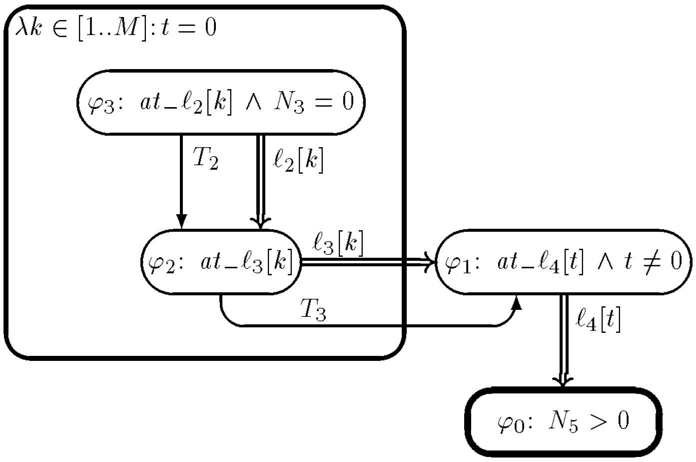

Fig. 2.9. P-RANK diagram for program TURN.

$$\begin{array}{ll}\textbf{in} & M\colon \textbf{integer where } M > 0\\ \textbf{local } y & \colon \textbf{integer where } y = 1\end{array}$$

$$\mathop{\Big\|}_{k=1}^{M} P[k] :: \left[\begin{array}{l} \textbf{local } t\colon \textbf{integer where } t = 0\\ \ell_0\colon \textbf{while } t \neq 1 \textbf{ do}\\ \qquad \ell_1\colon (t,\ y) := (y,\ y+1)\\ \ell_2\colon \end{array}\right]$$

Fig. 2.10. Program RACE.

The property we wish to establish for this program is that eventually some process reaches ℓ_2. This property is stated by the following response formula.

$$\underbrace{N_0 > 0 \ \land\ y = 1}_{p} \ \Rightarrow\ \Diamond\, \underbrace{N_2 > 0}_{q}\ .$$

It is easy to trace the progress of the computation towards the goal $N_2 > 0$ through the stages

$$\widehat{\varphi}_3\colon\quad N_0 > 0 \ \land\ y = 1$$

(initial) and

$$\widehat{\varphi}_2\colon\quad N_1 > 0 \ \land\ y = 1,$$

up to

$$\varphi_1[k]: \quad at\_\ell_0[k] \ \wedge \ t[k] = 1.$$

The next helpful step takes transition $\ell_0[k]$ and reaches

$$q = \varphi_0: \quad N_2 > 0.$$

Consequently, we choose assertions, helpful transitions, and ranking functions as follows:

$\varphi_3[k]$:	$at\_\ell_0[k] \ \wedge \ y = 1$	$\tau_3[k]$: $\ell_0[k]$	δ_2: 3
$\varphi_2[k]$:	$at\_\ell_1[k] \ \wedge \ y = 1$	$\tau_2[k]$: $\ell_1[k]$	δ_2: 2
$\varphi_1[k]$:	$at\_\ell_0[k] \ \wedge \ t[k] = 1$	$\tau_1[k]$: $\ell_0[k]$	δ_1: 1
φ_0:	$N_0 > 0.$		

In Fig. 2.11, we present a P-RANK diagram representing this proof.

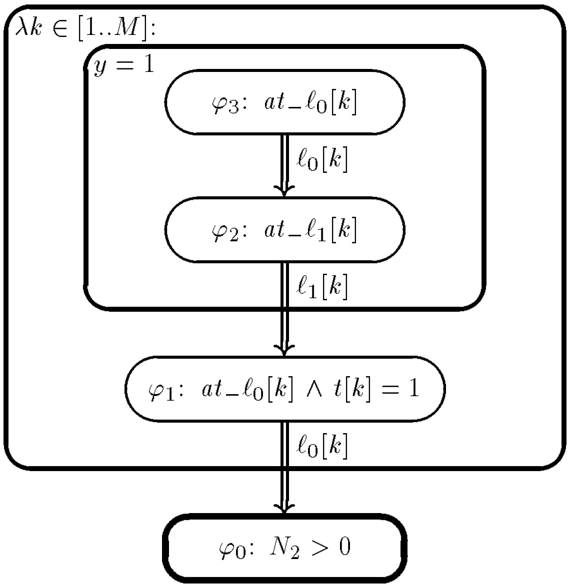

Fig. 2.11. P-RANK diagram for program RACE.

Advanced Example: Resource-Allocator Program

In some cases, response properties of a parameterized program can be proven without the need for explicit parameterization of the assertions and helpful transitions. Thus, rules CHAIN-J (Fig. 1.7) and WELL-J (Fig. 1.26) can be used directly.

We illustrate this on the example of a resource allocator program.

We reconsider program RES-SV for the allocation of a resource between several customers (the shared-variables version). This program was first presented in Section 2.2 of the SAFETY book (Fig. 2.9), and is reproduced here in Fig. 2.12. We denote $t \oplus_M 1 = (t \ mod \ M) + 1$.

$$
\begin{array}{l}
\textbf{in} \quad M\ : \textbf{integer where } M > 0 \\
\textbf{local } g,\ r\text{: } \textbf{array } [1..M] \textbf{ of boolean where } g = \text{F},\ r = \text{F} \\
\\
\qquad A :: \left[\begin{array}{l}
\textbf{local } t\text{: } \textbf{integer where } t = 1 \\
m_0\text{: } \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
m_1\text{: } \textbf{if } r[t] \textbf{ then} \\
\quad \left[\begin{array}{l}
m_2\text{: } g[t] := \text{T} \\
m_3\text{: } \textbf{await } \neg r[t] \\
m_4\text{: } g[t] := \text{F}
\end{array}\right] \\
m_5\text{: } t := t \oplus_M 1
\end{array}\right]
\end{array}\right] \\
\\
\qquad \| \\
\\
\qquad \overset{M}{\underset{i=1}{\|}} C[i] :: \left[\begin{array}{l}
\ell_0\text{: } \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\text{: } \textbf{noncritical} \\
\ell_2\text{: } r[i] := \text{T} \\
\ell_3\text{: } \textbf{await } g[i] \\
\ell_4\text{: } \textbf{critical} \\
\ell_5\text{: } r[i] := \text{F} \\
\ell_6\text{: } \textbf{await } \neg g[i]
\end{array}\right]
\end{array}\right]
\end{array}
$$

Fig. 2.12. Program RES-SV (resource allocator).

We are interested in proving the response properties of this program. The main response property is that of accessibility, stated by the formula

$$\psi\text{:} \quad at\_\ell_2[k] \ \Rightarrow \ \diamondsuit\, at\_\ell_4[k].$$

This formula states that any process that is interested in entering its critical section, will eventually succeed in doing so.

We are aided in this proof by the invariant

$$\varphi[i]:\quad idle[i] \ \lor\ requesting[i] \ \lor\ granted[i] \ \lor\ releasing[i]$$

holding for every $i = 1, \ldots, M$, where

$$\begin{array}{ll}
idle[i]: & \big(at_m_{0,1,5} \ \lor\ t \neq i\big) \ \land\ at_\ell_{0..2,6}[i] \ \land\ \neg r[i] \ \land\ \neg g[i] \\
requesting[i]: & \big(at_m_{0..2,5} \ \lor\ t \neq i\big) \ \land\ at_\ell_3[i] \ \land\ r[i] \ \land\ \neg g[i] \\
granted[i]: & at_m_3 \ \land\ t = i \ \land\ at_\ell_{3..5}[i] \ \land\ r[i] \ \land\ g[i] \\
releasing[i]: & at_m_{3,4} \ \land\ t = i \ \land\ at_\ell_6[i] \ \land\ \neg r[i] \ \land\ g[i].
\end{array}$$

In Fig. 2.13, we present a diagram (duplicated from Fig. 2.10 of the SAFETY book) which represents the possible movements of the system between the four cases comprising assertion $\varphi[i]$.

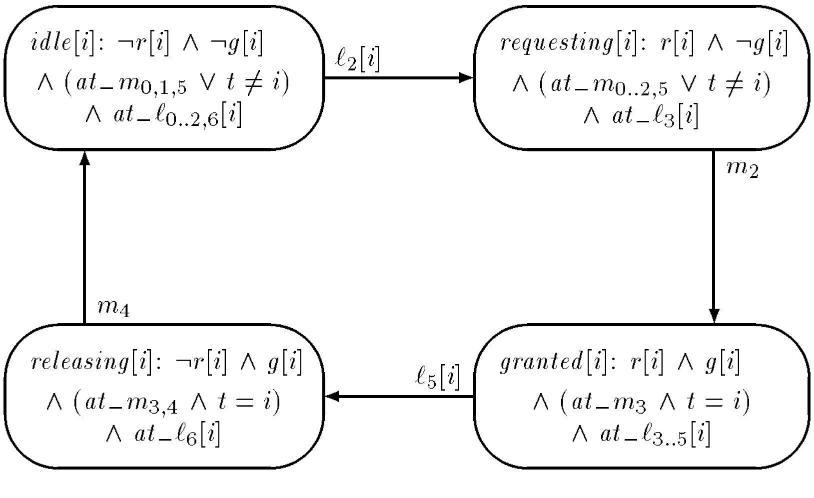

Fig. 2.13. Four phases in the life of a customer process.

The proof is established by the following lemmas:

Lemma A $\quad at\_\ell_2[k] \ \Rightarrow\ \diamondsuit\Big(at\_\ell_3[k] \ \land\ \big((at\_m_{0..3} \ \land\ t = k) \ \lor\ t \neq k\big)\Big)$

Lemma B $\quad at\_\ell_3[k] \ \land\ t \neq k \ \Rightarrow\ \diamondsuit\big(at\_\ell_3[k] \ \land\ at\_m_{0..3} \ \land\ t = k\big)$

Lemma C $\quad at\_\ell_3[k] \ \land\ at\_m_{0..3} \ \land\ t = k \ \Rightarrow\ \diamondsuit\, at\_\ell_4[k].$

Lemma A traces the progress of $C[k]$ from $\ell_2[k]$ to $\ell_3[k]$. Deriving at $\ell_3[k]$, we distinguish between two cases according to whether the allocator is currently paying attention to $C[k]$ $(t = k)$ or to some other customer $C[i]$, $t = i \neq k$. If the allocator is currently devoted to $C[i]$, $i \neq k$, then Lemma B ensures that $C[k]$'s turn will eventually come. Lemma C shows that, once the allocator pays attention to $C[k]$, process $C[k]$ eventually advances to $\ell_4[k]$.

Clearly, by the transitivity of response (rule TRNS-R), we obtain the required accessibility property.

- *Proof of Lemma A*

The proof of the response property

$$at\_\ell_2[k] \;\Rightarrow\; \Diamond\Big(at\_\ell_3[k] \;\wedge\; \big((at\_m_{0..3} \;\wedge\; t = k) \;\vee\; t \neq k\big)\Big)$$

using rule CHAIN-J, is presented in the diagram of Fig. 2.14.

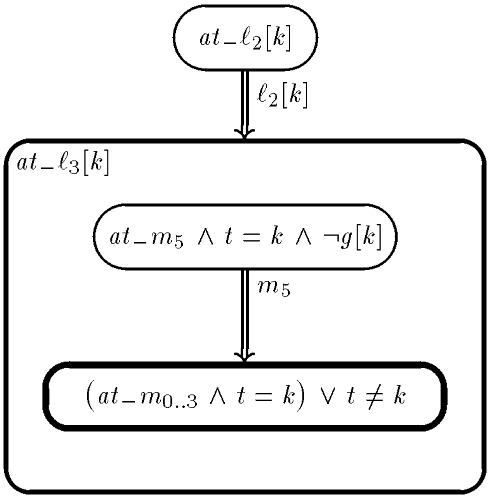

Fig. 2.14. Lemma A for program RES-SV.

The proof of this diagram relies on the invariant $\varphi[k]$, by which

$$at\_\ell_3[k] \;\rightarrow\; \begin{bmatrix} \big(at\_m_{0..3} \;\wedge\; t = k\big) \;\vee \\ \big(at\_m_5 \;\wedge\; t = k \;\wedge\; \neg g[k]\big) \;\vee \\ t \neq k \end{bmatrix}.$$

To see this observe that, by $\varphi[k]$, $at\_\ell_3[k]$ implies that either $requesting[k]$ or $granted[k]$ hold. The assertion $requesting[k]$ implies

$$(at\_m_{0..2,5} \ \lor \ t \neq k) \ \land \ \neg g[k],$$

which is equivalent to

$$\big((at\_m_{0..2,5} \ \land \ t = k) \ \lor \ t \neq k\big) \ \land \ \neg g[k].$$

This implies

$$(at\_m_{0..2} \ \land \ t = k) \ \lor \ \big(at\_m_5 \ \land \ t = k \ \land \ \neg g[k]\big) \ \lor \ t \neq k.$$

Taking the disjunction of this formula with $at\_m_3 \land t = k$, which is implied by $granted[k]$, we obtain the right-hand side of the implication above.

- *Proof of Lemma B*

The proof of the response property

$$at\_\ell_3[k] \ \land \ t \neq k \quad \Rightarrow \quad \Diamond\big(at\_\ell_3[k] \ \land \ at\_m_{0..3} \ \land \ t = k\big),$$

using rule WELL-J, is presented in Fig. 2.15.

Let us trace progress towards the goal φ_0 in this verification diagram, starting from a state in which $C[k]$ is at ℓ_3 and $t \neq k$. The diagram analyze the various locations in which A can currently be. If A is at m_0 (φ_9) it eventually moves to m_1 (φ_8). At m_1, A tests the value of $r[t]$. If $r[t] = \text{T}$, A moves to m_2 (φ_7). Otherwise, A moves to m_5 (φ_1). According to invariant $\varphi[t]$ (i.e., $\varphi[i]$ when $i = t$), when A is at m_2, $g[t] = \text{F}$ and since, on entering m_2, A has just tested $r[t]$ to be T, we obtain the conjunction $at\_m_2 \land r[t] \land \neg g[t]$ in φ_7. By $\varphi[t]$, $r[t] \land \neg g[t]$ implies $at\_\ell_3[t]$ which complete φ_7. From m_2, A eventually sets $g[t]$ to T and moves to m_3. At this point, we analyze the various possible locations of $C[t]$, i.e., the customer process $C[i]$, $i = 1, \ldots, M$, such that currently $t = i$.

Since at φ_7, $C[t]$ is at $at\_\ell_3[t]$ and $r[t] = \text{T}$, transition m_2 leads to the state described by φ_6. From ℓ_3, $C[t]$ moves to ℓ_4 (φ_5) and from there to ℓ_5 (φ_4).

Taking ℓ_5, $r[t]$ is reset to F and $C[t]$ moves to ℓ_6 (φ_3) waiting for $g[t]$ to become F. Up until this point, A cannot move because it waits for $r[t]$ to become F. Once $r[t] = \text{F}$, A moves to m_4 (φ_2) and from there to m_5 (φ_1).

At m_5, A sets t to $t \oplus_M 1$ advancing to the next process index in cyclic order. There are two cases. If $t = (k-1)\ mod\ M$ then $t \oplus_M 1 = k$ and transition m_5 leads to a state in which A is at m_0, $t = k$, and $C[k]$ is still waiting at ℓ_3.

If $t \neq (k-1)\ mod\ M$ then m_5 leads again to a state covered by φ_9. Here we must show that this transition causes a rank decrease. This means that

$$\big(\delta(t,k),\ 1\big) \ > \ \big(\delta(t \oplus_M 1,\ k),\ 9\big),$$

for $t \neq k$. To show this, we will prove $\delta(t,k) > \delta(t \oplus_M 1, k)$ for $t \neq k$.

Recall that the cyclic distance function was defined in Section 3.4 of the SAFETY book as

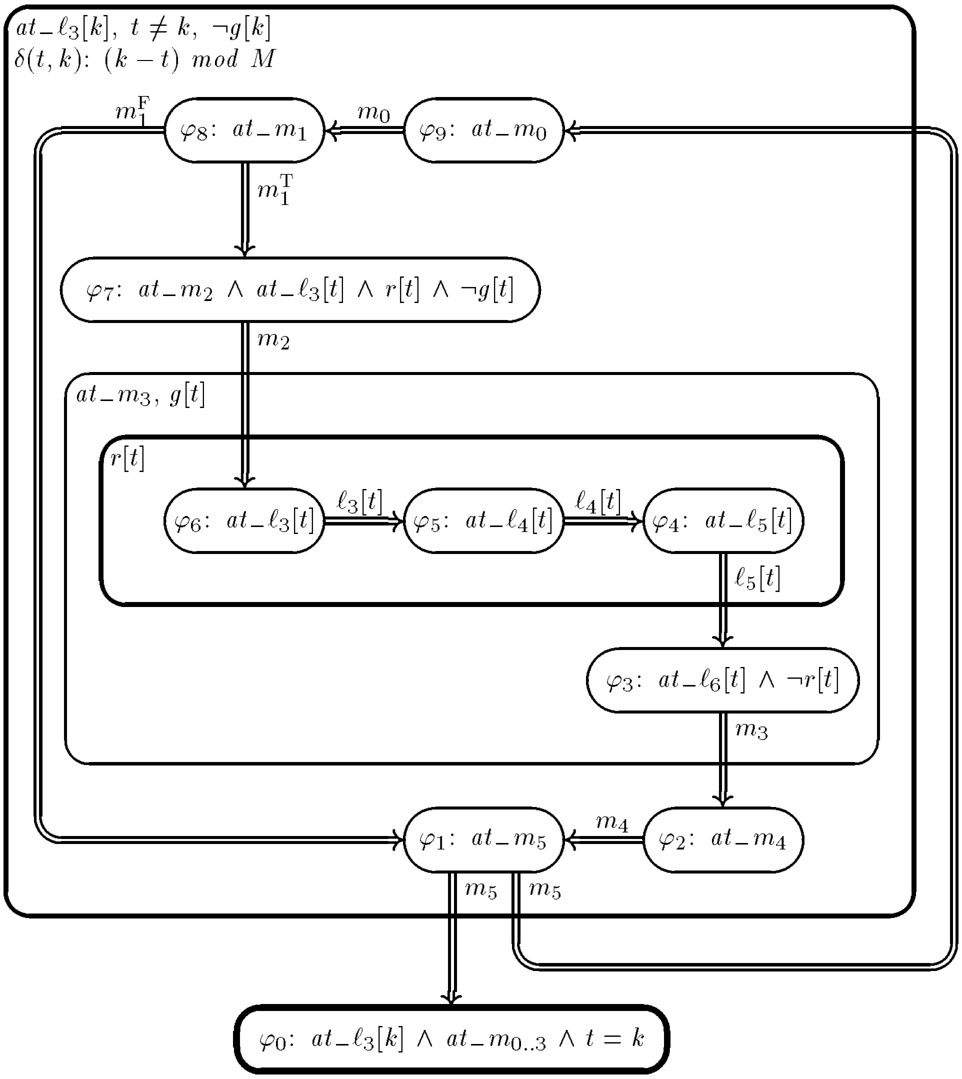

Fig. 2.15. Lemma B for program RES-SV.

$$\delta(t,k): \quad (k-t)\ mod\ M.$$

This function measures the number of times t has to be increased by 1 modulo M until it equals k.

It is obvious that if $t \neq k$ then $t \oplus_M 1$ is closer to k in a cyclic distance than t. For example, consider a situation where $M = 8$, $t = 8$ and $k = 3$.

Observe that

$$t \oplus_8 1 \;\;=\;\; 8 \oplus_8 1 \;\;=\;\; (8 \; mod \; 8) + 1 \;\;=\;\; 1.$$

The two cyclic distances are

$$\delta(t,k) \;\;=\;\; \delta(8,3) \;\;=\;\; (3-8) \; mod \; 8 \;\;=\;\; (-5) \; mod \; 8 \;\;=\;\; 3$$

$$\delta(t \oplus_8 1, k) \;\;=\;\; \delta(1,3) \;\;=\;\; (3-1) \; mod \; 8 \;\;=\;\; 2 \; mod \; 8 \;\;=\;\; 2.$$

Note that the restriction to $t \neq k$ is essential, because if $t = k$ then

$$\delta(t,k) \;\;=\;\; \delta(k,k) \;\;=\;\; 0 \;\not>$$
$$\delta(t \oplus_M 1,\; k) \;\;=\;\; \delta(k \oplus_M 1,\; k) \;\;=\;\; (-1) \; mod \; M \;\;=\;\; M-1.$$

Let us prove the rank decrease for arbitrary $M > 1$ and arbitrary $t \neq k$, using the mathematical definitions of $\oplus_M$ and $\delta(t,k)$.

Let n denote $\delta(t,k) = \big((k-t) \; mod \; M\big)$. Obviously $0 \leq n < M$. From the facts that $0 < t \leq M$, $0 < k \leq M$, and $t \neq k$, it follows that $n \neq 0$. Using $0 < n \leq M$, we compute

$$\begin{aligned}
\delta(t \oplus_M 1,\; k) \;&=\; \big(k - (t \oplus_M 1)\big) \; mod \; M \;\;=\;\; \Big(k - \big((t \; mod \; M) + 1\big)\Big) \; mod \; M \\
&=\; \big((k-1) - (t \; mod \; M)\big) \; mod \; M \\
&=\; (k-1-t) \; mod \; M \\
&=\; \big(-1 + (k-t)\big) \; mod \; M \\
&=\; \big(-1 + ((k-t) \; mod \; M)\big) \; mod \; M \\
&=\; (-1+n) \; mod \; M \;\;=\;\; (n-1) \; mod \; M \\
&=\; n-1 < \;\; n \;\;=\;\; \delta(t,k).
\end{aligned}$$

This computation uses the fact that, for C satisfying $0 \leq C < M$, $C \; mod \; M = C$, and the following property of the mod operator:

$$\big(a \pm (b \; mod \; M)\big) \;\;=\;\; (a \pm b) \; mod \; M,$$

which allows the introduction and elimination of $mod \; M$ within the scope of another $mod \; M$.

- *Proof of Lemma C*

  The response property

$$at\_\ell_3[k] \;\wedge\; at\_m_{0..3} \;\wedge\; t = k \;\;\Rightarrow\;\; \Diamond \, at\_\ell_4[k],$$

is proven by rule CHAIN-J. The proof is presented in the diagram of Fig. 2.16. ▪

The Bakery Algorithm

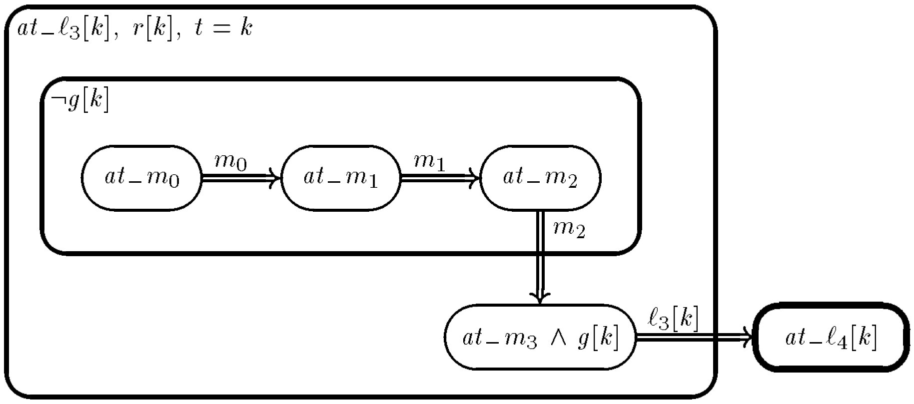

Fig. 2.16. Lemma C for program RES-SV.

In Fig. 2.17, we present an atomic version of the bakery algorithm for mutual exclusion among $M > 0$ processes.

The algorithm is called the *bakery* algorithm, since it is based on the idea that customers, as they enter, pick numbers which form an ascending sequence. Then, a customer with a lower number has higher priority in accessing its critical section. Statement ℓ_2 ensures that the number assigned to $y[i]$ is greater than the current value of $ticket[j]$, for all $j = 1, \ldots, M$. Then statement ℓ_3 admits process $P[i]$ into the critical section only if, for all $j \neq i$, either $y[j] = 0$, implying that process $P[j]$ is currently not interested in entering its cricial section, or $ticket[j] > ticket[i]$ which implise that $P[j]$ has a higher turn number and, therefore, a lower priority than $P[i]$.

We refer to this version of the program as *atomic* because it assumes that taking the maximum of $ticket[1], \ldots, ticket[M]$ can be done in one step and, similarly, testing that all $P[j]$ for $j \neq i$ have lower priority than $P[i]$ can be done in one step.

Proving Mutual Exclusion

To prove the property of mutual exclusion, we first establish some local invariants, relating the value of $ticket[i]$ to the location of process $P[i]$.

$$\chi_0:\quad ticket[i] \geq 0$$

$$\chi_1:\quad ticket[i] > 0 \quad\leftrightarrow\quad at\_\ell_{3..5}[i].$$

Consider two distinct processes, $P[i]$ and $P[k]$, for $i \neq k$. Their mutual exclusion can be specified by the invariance of the assertion

$$\psi:\quad \neg\big(at\_\ell_4[i] \ \wedge\ at\_\ell_4[k]\big).$$

$$
\begin{array}{l}
\textbf{in} \quad M \quad : \textbf{integer where } M > 0 \\
\textbf{local } \mathit{ticket}\text{: } \textbf{array}[1..M] \textbf{ of integer} \quad \textbf{where } \mathit{ticket} = 0 \\[1em]
\underset{i=1}{\overset{M}{\Big\|}} \; P[i] :: \left[\begin{array}{l}
\ell_0\text{: } \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\text{: } \textbf{noncritical} \\
\ell_2\text{: } \mathit{ticket}[i] := 1 + \mathit{max}(\mathit{ticket}[1], \ldots, \mathit{ticket}[M]) \\
\ell_3\text{: } \textbf{await } \forall j \neq i\text{: } \mathit{ticket}[j] = 0 \;\vee\; \mathit{ticket}[i] < \mathit{ticket}[j] \\
\ell_4\text{: } \textbf{critical} \\
\ell_5\text{: } \mathit{ticket}[i] := 0
\end{array}\right]
\end{array}\right]
\end{array}
$$

Fig. 2.17. Program BAKERY-AT (atomic bakery algorithm).

Applying the technique of assertion propagation (described in Chapter 1 of the SAFETY book), we obtain the following necessary condition for ψ being invariant (taking into account invariant χ_0):

$$\kappa\text{:} \quad at\_\ell_3[i] \;\wedge\; at\_\ell_4[k] \quad \rightarrow \quad \exists j \neq i\text{: } 0 < \mathit{ticket}[j] \leq \mathit{ticket}[i].$$

A good heuristic is to try to establish mutual exclusion of $P[i]$ and $P[k]$, using assertions that only refer to the variables of $P[i]$ and $P[k]$. Therefore we may look for the weakest (i, k)-assertion which implies κ. This is given by

$$at\_\ell_3[i] \;\wedge\; at\_\ell_4[k] \quad \rightarrow \quad 0 < \mathit{ticket}[k] \leq \mathit{ticket}[i].$$

Since $at\_\ell_4[k]$ implies $\mathit{ticket}[k] > 0$ by χ_1, we can simplify the candidate assertion to

$$\chi_3\text{:} \quad at\_\ell_3[i] \;\wedge\; at\_\ell_4[k] \quad \rightarrow \; \mathit{ticket}[k] \leq \mathit{ticket}[i].$$

It is not difficult to ascertain that χ_3 is an invariant (using χ_0 and χ_1).

Informally, we may consider the two relevant transitions.

- Taking $\ell_3[k]$ while $at\_\ell_3[i]$. Transitions $\ell_3[k]$ can be taken only if $\mathit{ticket}[i] = 0$ or $\mathit{ticket}[k] < \mathit{ticket}[i]$. By χ_1, $at\_\ell_3[i]$ implies $\mathit{ticket}[i] > 0$. Therefore, taking the transition implies $\mathit{ticket}[k] < \mathit{ticket}[i]$ which guarantees $\mathit{ticket}[k] \leq \mathit{ticket}[i]$.
- Taking $\ell_2[i]$. This transitions sets $\mathit{ticket}[i]$ to a value which is not smaller than $1 + \mathit{ticket}[k]$. Hence $\mathit{ticket}[k] \leq \mathit{ticket}[i]$.

In a symmetric way, we propose and verify the invariance of

$$\chi_4\text{:} \quad at\_\ell_3[k] \;\wedge\; at\_\ell_4[i] \quad \rightarrow \quad \mathit{ticket}[i] \leq \mathit{ticket}[k].$$

It is now straightforward to verify the invariance ψ, relying on χ_3 and χ_4 as previously established invariances.

Proving Accessibility

The property of accessibility for process $P[i]$ can be specified by the formula

$$\alpha: \quad at\_\ell_2[i] \quad \Rightarrow \quad \Diamond\, at\_\ell_4[i].$$

It is trivial to follow the progress of $P[i]$ from ℓ_2 to ℓ_3. Consequently, the main verification task should establish the property

$$\beta: \quad at\_\ell_3[i] \quad \Rightarrow \quad \Diamond\, at\_\ell_4[i].$$

The only reason process $P[i]$ may be held at ℓ_3, not being able to proceed immediately to ℓ_4 is that there are some processes *superior* to $P[i]$. A process $P[j]$ is said to be superior to $P[i]$ if $0 < ticket[j] < ticket[i]$. Consequently, we define

$$superior(i) \;=\; \{j \in [1..M] \mid 0 < ticket[j] < ticket[i]\}$$

to be the set of all (indices of) processes superior to $P[i]$.

Consider what may happen to the set $superior(i)$ while $P[i]$ is waiting at ℓ_3. Processes may leave the set $superior(i)$ by performing ℓ_5, setting their *ticket* variable to 0. Once a process has left $superior(i)$ it cannot reenter this set because the only way for a process to attain a positive ticket value is by performing ℓ_2. However, execution of ℓ_2 by $P[j]$ while $P[i]$ is at ℓ_3, necessary leads to a value $ticket[j] > ticket[i]$, which shows that $P[j]$ is not in $superior(i)$ even after execution of ℓ_2.

We conclude that while $P[i]$ is waiting at ℓ_3, the set $superior(i)$ can never increase. We will now argue that if the set is non-empty, it will eventually decrease.

Consider process $P[j]$ which has the minimal positive value of $ticket[j]$. If $superior(i) \neq \emptyset$ then, obviously $j \in superior(i)$. Having a positive ticket value implies that $P[j]$ is somewhere in the range $\{\ell_3, \ell_4, \ell_5\}$. We claim that wherever it is, its progress is guaranteed. In particular, even if $P[j]$ is at ℓ_3 it cannot be blocked by any other process, since $ticket[j]$ is (positively) minimal.

Thus, $P[j]$ is guaranteed to progress until it reaches ℓ_5, whose execution decreases the size of $superior(i)$.

Consequently, while $P[i]$ is waiting at ℓ_3, the set $superior(i)$ repeatedly decreases, until it becomes empty. Once $superior(i)$ is empty, $P[i]$ has the minimal positive ticket and its progress to ℓ_4 is guaranteed.

In Fig. 2.18, we present a P-RANK diagram for the proof of property β based on the above arguments. The proof uses the assertion

$$minimal(k): \quad ticket(k) > 0 \wedge \forall j\colon ticket(j) > 0 \;\rightarrow\; ticket(k) \leq ticket(j).$$

In **Problem 2.3**, the reader is requested to consider several non-atomic versions of the bakery algorithm and verify their properties.

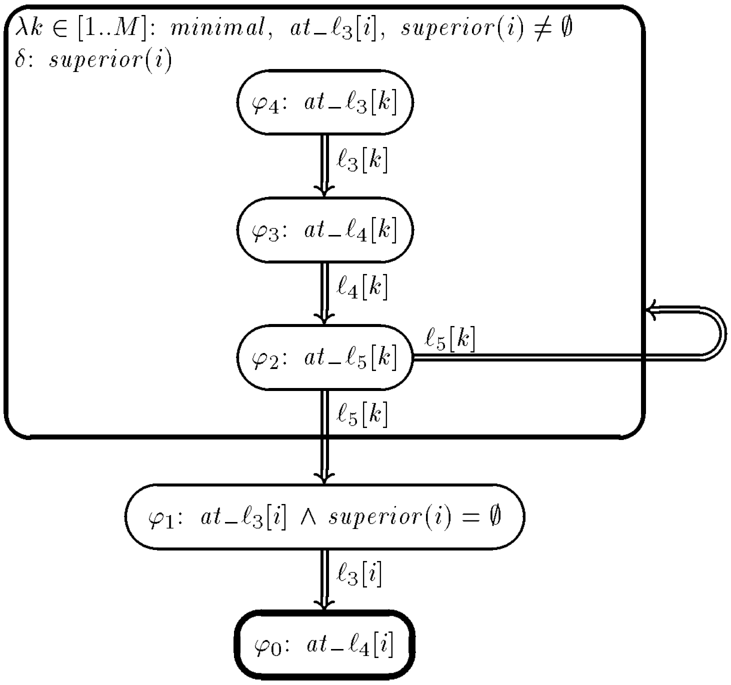

Fig. 2.18. A P-RANK diagram proving accessibility for program BAKERY-AT.

2.3 Coordination by Add-And-Store

As our next set of examples, illustrating the techniques for proving response properties of parameterized programs, we present a family of algorithms, based on a special synchronization instruction.

Some computers have special "add-and-store" instructions that fetch a value from a location in memory, add to it a number and store the result in the same location, all within one execution cycle. If several processors that have this instruction are connected to a common shared memory then, while one of them performs such an add-and-store instruction, no other processor can interfere with the value being fetched and stored at this location. This provides a natural implementation of the atomic instruction $y := y + e$ for a shared variable y and a *local expression* e. A local expression is an expression that depends only on variables local to the process. Usually, the value which is stored is also retained in an internal register. This enables a single transition implementation of the multiple assignment statement

$$(t,\, y) \;:=\; (y+e,\, y+e),$$

where y is a shared variable, e is a local expression, and t is a variable local to the process that is executing this instruction.

To emphasize the special role of this statement, it is written in the form

$$t \ := \ y \ := \ y + e,$$

in all the programs in this section.

Add-and-store statements of the described form have often been suggested as means for implementing semaphores. As we will see below, they are not as strong as semaphores, since they lack the compassion property associated with semaphores. That is, the transitions associated with add-and-store statements are not compassionate while semaphore transitions are.

We present two programs that coordinate entry to critical sections by means of add-and-store statements and investigate their response properties. Since these statements are not compassionate, they cannot directly ensure individual accessibility of each process to its critical section. Instead, they can only ensure the weaker property of cummunal accessibility by which, if some process wishes to enter the critical section, some process (not necessarily the same) will eventually enter its critical section. However, as will be shown, communal accessibility can be used to program more involved algorithms that ensure individual accessibility.

Notations and Inferences

To facilitate proofs of the programs in this section, we introduce some special notations that rely on the special structure of the considered programs. All programs considered here consist of a parallel composition of M processes $P[1], \ldots, P[M]$, where M is an input to the program. Each process has a local variable t. Let ℓ_a and ℓ_b, $a \leq b$ be two locations in the program. We define the following notations:

$$L^{t \sim c}_{a..b}: \quad \big\{ i \in [1..M] \ \mid \ at\_\ell_{a..b}[i] \ \wedge \ t[i] \sim c \big\},$$

$$N^{t \sim c}_{a..b}: \quad \big| L^{t \sim c}_{a..b} \big|$$

where $\sim$ is one of the six binary relations $\{<, \leq, =, \neq, \geq, >\}$.

Thus, $L^{t>c}_{3..4}$ denotes the set of all process indices $i \in [1..M]$, such that $P[i]$ is currently at ℓ_3 or at ℓ_4 and $t[i] > 0$, and $L^{t>c}_{3..4}$ denotes the total number of such processes.

For the case that $b = a$, we write $L^{t \sim c}_{a..b}$ and $N^{t \sim c}_{a..b}$ simply as $L^{t \sim c}_{a}$ and $N^{t \sim c}_{a}$. For the case that $b = a+1$, we write $L^{t \sim c}_{a..b}$ and $N^{t \sim c}_{a..b}$ simply as $L^{t \sim c}_{a,b}$ and $N^{t \sim c}_{a,b}$.

There is an obvious connection between assertions referring to expressions such as $N^{t \sim c}_{a..b}$ and assertions that use existential quantification over the index i. For example, the following equivalence is a direct consequence of the definitions.

$$N_{a..b}^{t>0} > 0 \quad \leftrightarrow \quad \exists i\colon\ \big(at\_\ell_{a..b}[i] \ \wedge\ t[i] > 0\big).$$

All quantified variables in this section are rigid variables denoting process indices which range over $[1..M]$. We therefore omit their range specifications from the formulas.

Some of the following proofs use rule EE-INTR, which can be derived in the proof system presented in Volume I.

Rule EE-INTR (introduction of existential quantifiers)

$$\frac{p \quad \Rightarrow \quad \diamondsuit\, q}{(\exists i\colon\ p) \quad \Rightarrow \quad \diamondsuit(\exists i\colon\ q)}$$

In the case that formula q has no free occurrences of i, we can use this rule to infer $(\exists i\colon\ p) \Rightarrow \diamondsuit\, q$ from $p \Rightarrow \diamondsuit\, q$.

Mutual Exclusion by Add-and-Store

We consider first program MUX-AST presented in Fig. 2.19, which implements mutual exclusion by add-and-store instructions.

$$
\begin{array}{l}
\textbf{in} \quad\ M\colon \textbf{integer where } M > 0 \\
\textbf{local } y\ :\textbf{integer where } y = 1
\end{array}
$$

$$
\mathop{\Big\|}_{i=1}^{M} P[i] :: \left[\begin{array}{l}
\textbf{local } t\colon \textbf{integer} \\
\ell_0\colon \textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
\ell_1\colon \textbf{noncritical} \\
\ell_2\colon t := -1 \\
\ell_3\colon \textbf{while } t < 0 \textbf{ do} \\
\qquad \left[\begin{array}{l}
\ell_4\colon \textbf{await } y > 0 \\
\ell_5\colon t := y := y - 1 \\
\ell_6\colon \textbf{if } t = 0 \textbf{ then} \\
\qquad \ell_7\colon \textbf{critical} \\
\ell_8\colon y := y + 1
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Fig. 2.19. Program MUX-AST (mutual exclusion by add-and-store).

The program consists of $M > 0$ processes which coordinate their entry to their critical sections via the shared "semaphore" variable y. In addition, each process $P[i]$ has a local variable t which is used in the program. The program for each process is a cycle, in which the process alternates between the execution of its noncritical section at ℓ_1 and the attempt to enter its critical section at ℓ_7.

The process may elect to stay forever in its noncritical section.

If the process exits its noncritical section, the protocol for trying to enter the critical section starts at ℓ_2, where the variable t is set to -1. Then, the process enters a loop at ℓ_3 which is terminated only when t becomes nonnegative. A nonnegative value of t (actually 0), signals that the process managed to enter its critical section, and may therefore return to ℓ_1.

Within the loop, the process waits first at ℓ_4 for y to become positive. When the process detects a positive y it performs an add-and-store statement at ℓ_5 with y and the constant -1, retaining the value stored in y also in t.

One may wonder why do we need further checks after having detected a positive y at ℓ_4. The answer is that two processes $P[i]$ and $P[j]$ may detect a positive y at ℓ_4 and move, one after the other, to ℓ_5. Assume $P[i]$ to be the first to perform ℓ_5. It will lower y to 0 and obtain a $t[i] = 0$. The second process $P[j]$, performing ℓ_5 one step later, will get to $y = t[j] = -1$. Therefore, it is necessary for a process to check its own t after the addition to see whether it was the first to perform ℓ_5.

Indeed, at ℓ_6 process $P[i]$ checks whether it was the lucky one. On finding $t[i] = 0$, $P[i]$ concludes that it was the first to perform ℓ_5 and proceeds to enter the critical section at ℓ_7. On exit it increments y, offsetting the subtraction at ℓ_5. Since $t[i] = 0$ in this case, the loop of ℓ_3 terminates, and lucky process $P[i]$ returns to ℓ_1.

If the process finds a negative t, it concludes that it is not its turn to enter, and proceeds to ℓ_8. Here it increments y to offset the subtraction at ℓ_5. Since $t < 0$, the loop of ℓ_3 does not terminate, and the process returns to ℓ_4 to try its luck once more.

Proving Safety Properties: Mutual Exclusion

As a first step in the verification of program MUX-AST, let us prove the safety property of mutual exclusion. This property states that at most one process may be at ℓ_7, and is expressible by:

$$N_7 \leq 1.$$

We prove several invariants that lead to the desired conclusion.

The first invariant is given by

$$I_1: \quad at\_\ell_7[i] \quad \rightarrow \quad t[i] = 0.$$

Observe that the assertions refer to the instance of variable t that belongs to process $P[i]$ as $t[i]$. This invariant can be verified by checking the relevant transitions. They are $\ell_6[i]$ which proceeds to ℓ_7 only if $t[i] = 0$, and $\ell_2[i]$, $\ell_5[i]$ which modify $t[i]$, but do not end up at $\ell_7[i]$.

A second invariant is given by

$$I_2: \quad N_{6..8} + y \;=\; 1.$$

This is easily verifiable by observing that initially $N_{6..8} = 0$ and $y = 1$. Then we check the relevant transitions and find that $\ell_5[i]$ decrements y by 1 but increments $N_{6..8}$ by 1. The latter is because process $P[i]$ has just joined the set $L_{6..8}$ by entering ℓ_6. Similarly, transition ℓ_8 increments y by 1 but decrements $N_{6..8}$ by 1. Thus, both preserve the sum $N_{6..8} + y$.

From this invariant it follows that $y \leq 1$ is also invariant.

The last invariant we consider is:

$$I_3: \quad i \neq j \;\;\rightarrow\;\; \neg\Big(\big(at\_\ell_{6..8}[i] \wedge t[i] = 0\big) \;\wedge\; \big(at\_\ell_{6..8}[j] \wedge t[j] = 0\big)\Big)$$

which states that at most one process $P[i]$ can be at $\ell_{6..8}$ with a zero t variable.

The only transition which may potentially falsify this assertion is taking $\ell_5[i]$ from a state satisfying $at\_\ell_{6..8}[j] \wedge t[j] = 0$ (symmetrically, taking $\ell_5[j]$ from a state satisfying $at\_\ell_{6..8}[i] \wedge t[i] = 0$). However, in this case $N_{6..8} > 0$ which, in view of I_2, implies $y \leq 0$. Since transition $\ell_5[i]$ leads to $t'[i] = y - 1$, it follows that $t'[i] < 0$. Therefore, i does not satisfy $at'\_\ell_{6..8}[i] \wedge t'[i] = 0$ and the assertion I_3 is not falsified.

This shows that all transitions preserve assertion I_3.

From I_1 and I_3 we conclude

$$N_7 \;\leq\; 1.$$

Proving Response Property: Communal Accessibility

As we have already commented, the add-and-store instructions are not strong enough to guarantee individual accessibility, i.e., that *each* process wishing to enter its critical section will eventually do so. Instead, we prove the weaker property of *communal accessibility* (also called *weak accessibility*). This property claims that, if some process wishes to enter its critical section, then *some* process (not necessarily the same) will eventually enter its critical section. This can be expressed by:

$$N_2 > 0 \;\;\Rightarrow\;\; \diamondsuit(N_7 > 0).$$

Let us prove this property.

Consider the progress of a process from ℓ_2 to ℓ_7. It can certainly advance unhindered until it reaches ℓ_4. At ℓ_4 it has to wait until y becomes positive. Due to I_2, y becomes positive only when the range $\ell_{6..8}$ is evacuated by all processes currently residing there. When y becomes positive, one or more processes will reach ℓ_5. The first process to execute ℓ_5, while y is positive, will get a zero value of t and will eventually get to ℓ_7.

Thus, we can partition the description of progress into phases, identified as follows:

1. Some process gets to ℓ_4.
2. While some processes are at ℓ_4 and $y \leq 0$, the range $\ell_{6..8}$ is evacuated until y becomes 1.
3. Some process executes ℓ_5 while y is still 1, gets a zero t, and proceeds to ℓ_7.

Consequently, we prove these stages in the progress towards entry to the critical section as separate lemmas.

Lemma A $\quad N_2 > 0 \quad \Rightarrow \quad \diamondsuit(N_4 > 0)$

In Fig. 2.20 we present a CHAIN-J proof of the response formula

$$at\_\ell_2[i] \quad \Rightarrow \quad \diamondsuit(N_4 > 0).$$

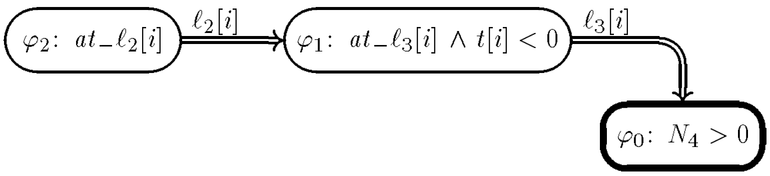

Fig. 2.20. CHAIN diagram for Lemma A.

Applying rule EE-INTR and observing that $N_4 > 0$ does not refer to the index i, we obtain

$$(\exists i\colon\ at\_\ell_2[i]) \quad \Rightarrow \quad \diamondsuit(N_4 > 0),$$

from which, by monotonicity, we conclude

$$N_2 > 0 \quad \Rightarrow \quad \diamondsuit(N_4 > 0).$$

Once we know that some process is at ℓ_4, there are two cases to consider, depending on whether y is positive. Consider first the more difficult one, in which $y \leq 0$. The following lemma claims that from the more difficult case, we are guaranteed to reach the easier case in which $N_4 > 0 \wedge y > 0$.

Lemma B $\quad N_4 > 0 \ \wedge\ y \leq 0 \quad \Rightarrow \quad \diamondsuit(N_4 > 0 \ \wedge\ y > 0)$

We prove this lemma by rule WELL-JP. The proof is presented by the P-RANK diagram of Fig. 2.21.

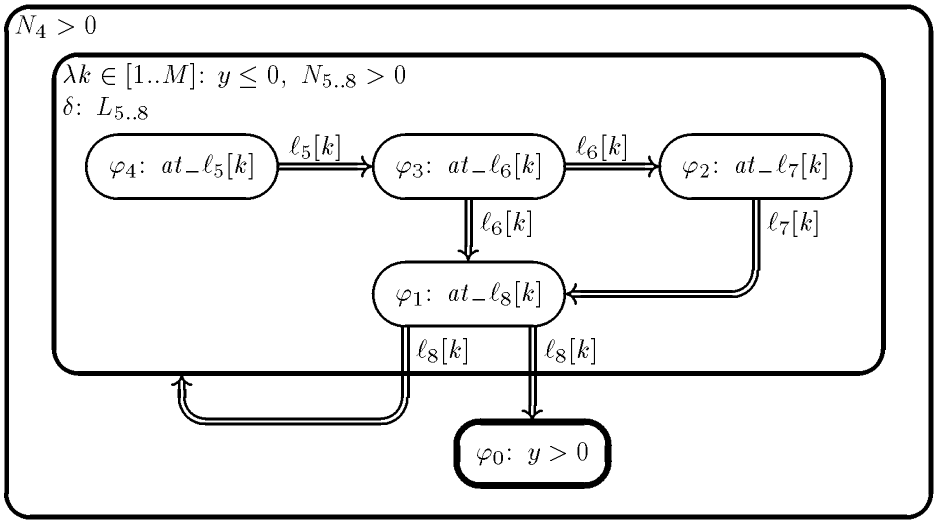

Fig. 2.21. P-RANK diagram for Lemma B.

Note that any transition that causes $N_{6..8}$ to become 0 leads, according to I_2, to a positive y.

The reason we use the ranking $N_{5..8}$ instead of $N_{6..8}$ which is mentioned in I_2, is that $N_{6..8}$ can increase by execution of ℓ_5. The rank $N_{5..8}$ cannot increase, while $y \leq 0$, because no process can pass ℓ_4 if $y \leq 0$.

Lemma C $\quad N_4 > 0 \ \wedge \ y > 0 \quad \Rightarrow \quad \diamondsuit(N_7 > 0)$

This lemma is proven again by rule WELL-JP, tracing the progress from ℓ_4 to ℓ_7. The proof is presented in the P-RANK diagram of Fig. 2.22.

We use the invariant I_2 to infer $y > 0 \rightarrow y = 1$.

The three lemmas can be combined into a single P-RANK diagram, presented in Fig. 2.23.

It is possible to omit most of the internal details of this diagram, and retain just the top-level structure, identifying the lemmas which lead from one phase to the next. This leads to the diagram of Fig. 2.24.

This schematic diagram identifies the partition of the proof into three lemmas. Each lemma is represented as a box, with exits that lead to subsequent

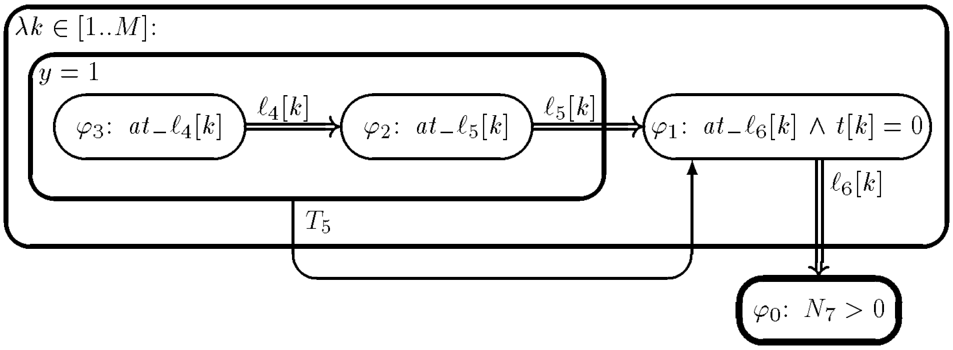

Fig. 2.22. P-RANK diagram for Lemma C.

boxes. The box represents the response property, guaranteeing that eventually the computation will exit to one of the successors of this box.

From Communal to Individual Accessibility

Program MUX-AST for mutual exclusion by add-and-store instructions does not fully satisfy our expectations from a satisfactory solution to the mutual exclusion problem. It falls short in guaranteeing communal accessibility rather than individual accessibility. However, once some process is admitted to the critical section we can appoint it as an arbitrator for the next round. This is because when being in the critical section it can perform activities such as determining which process will be the next to be admitted, without having to worry about possible interference from other processes.

In Fig. 2.25 we present program IMUX-AST, which implements individual accessibility. This program extends program MUX-AST in several ways. It contains an additional variable *next* ranging over $1..M$, and whose value represents the index of a process, that has been given higher priority in the last arbitration round. The local variable t of MUX-AST is transformed in IMUX-AST into an array $t[1..M]$ that can be inspected by each of the processes. Initially $next = 0$.

The protocol proceeds very much as before, except that at ℓ_4 we introduced a new gate. The function of the gate at ℓ_4 is to hold there all processes except the one that has been given higher priority. Note that, if no process has been given priority, $next = 0$ and all processes can pass the gate at ℓ_4. From there on, the protocol proceeds as before, except that there is an additional code performing an arbitration round at $\ell_{9..15}$. Since this code follows immediately after the critical section it is guaranteed to be performed in exclusion, by a single process $P[i]$.

The first question asked at ℓ_9 is whether a new round of arbitration is needed.

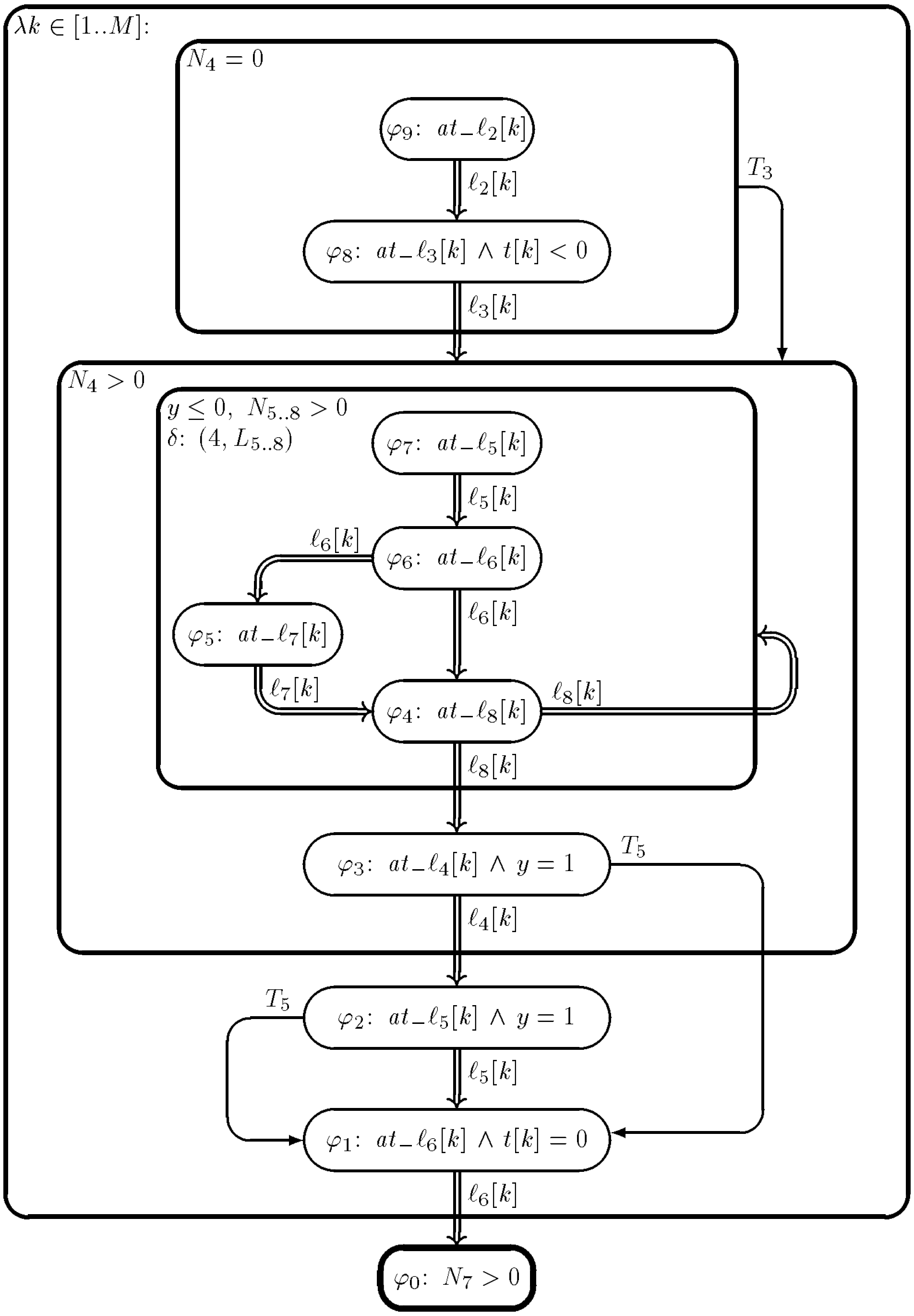

Fig. 2.23. P-RANK diagram for $N_2 > 0 \Rightarrow \diamondsuit(N_7 > 0)$.

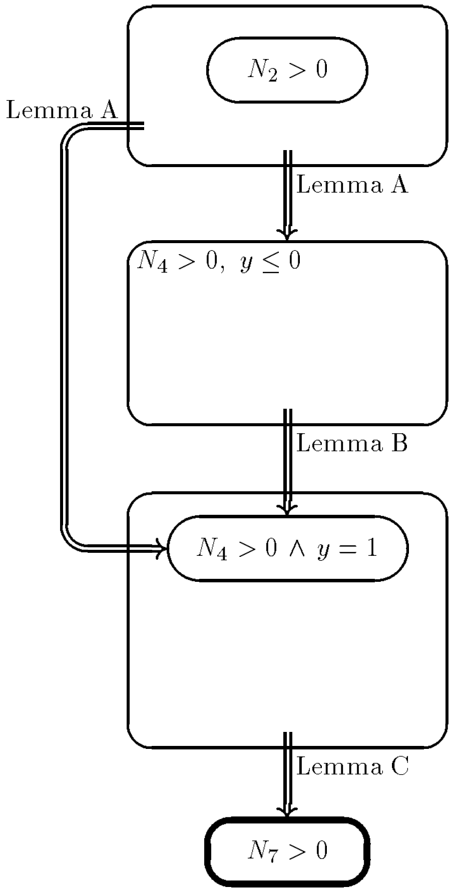

Fig. 2.24. Top-level structure of the proof.

A new round is needed if either no process has been assigned a high priority, i.e., $next = 0$, or if the process with the higher priority is $P[i]$ itself.

The other case is that some other process, $P[k]$, for $k \neq i$ has been assigned a high priority, but $P[i]$ entered the critical section ahead of $P[k]$. The algorithm is such that some overtaking is possible. In this case, $P[k]$ is still attempting entry to its critical section and we should not modify its priority. Hence in such a case, $P[i]$ proceeds directly to ℓ_{16} and no arbitration takes place.

On entering the arbitration section, variable $next$ is reset to 0 for the case that its previous value was i. Process $P[i]$ then sets variable j to the next index following i in cyclic order, and enters the while loop at ℓ_{12}. The loop searches

$$
\begin{array}{ll}
\textbf{in} \quad M & : \textbf{integer where } M = 0 \\
\textbf{local } j,\ y,\ next & : \textbf{integer where } y = 1,\ next = 0 \\
\qquad t & : \textbf{array } [1..M] \textbf{ of integer where } t = 0
\end{array}
$$

$$
\mathop{\Big|\Big|}_{i=1}^{M} P[i] :: \left[\begin{array}{l}
\ell_0\text{: }\textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\text{: }\textbf{noncritical} \\
\ell_2\text{: } t[i] := -1 \\
\ell_3\text{: }\textbf{while } t[i] < 0 \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_4\text{: }\textbf{await } next = 0 \lor next = i \\
\ell_5\text{: }\textbf{await } y > 0 \\
\ell_6\text{: } t[i] := y := y - 1 \\
\ell_7\text{: }\textbf{if } t[i] = 0 \textbf{ then} \\
\quad \left[\begin{array}{l}
\ell_8\text{: }\textbf{critical} \\
\ell_9\text{: }\textbf{if } next = 0 \ \lor\ next = i \textbf{ then} \\
\quad \left[\begin{array}{l}
\ell_{10}\text{: } next := 0 \\
\ell_{11}\text{: } j := (i \ mod\ M) + 1 \\
\ell_{12}\text{: }\textbf{while } next = 0 \ \land\ j \neq i \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_{13}\text{: }\textbf{if } t[j] < 0 \textbf{ then} \\
\qquad \ell_{14}\text{: } next := j \\
\ell_{15}\text{: } j := (j \ mod\ M) + 1
\end{array}\right]
\end{array}\right]
\end{array}\right] \\
\ell_{16}\text{: } y := y + 1
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Fig. 2.25. Program IMUX-AST — guaranteeing individual accessibility.

for the first j in cyclic order such that $t[j] < 0$. Note that $t[j] < 0$ is a reliable indicator that $P[j]$ is interested in entering the critical section but has not done so yet. If such a j is found, $P[i]$ sets $next$ to j, thus declaring j to have a high priority, and then exits the loop. Another possibility is that no such j has been found, and then j will close a full cycle and return to the value i. The loop terminates in both of these cases.

The reader is invited to verify that this program guarantees individual accessibility, i.e., that it satisfies the requirement

$$at\_\ell_2[k] \quad \Rightarrow \quad \diamondsuit\, at\_\ell_7[k].$$

Readers-Writers with Add-And-Store Instructions

An interesting extension of the mutual exclusion problem is the readers-writers problem. In this problem we distinguish two types of critical sections, called a *reading section* and a *writing section*. The required exclusion property is that

> while some process resides in a writing section, no other process may reside in either a reading or a writing section.

Note that this allows several processes to cohabit a reading section.

The Program

Program READ-WRITE for solving the readers-writers problem, using add-and-store instructions, is presented in Fig. 2.26. (Compare with program READ-WRITE of Fig. 2.11 of the SAFETY book, which uses generalized semaphores.) After the statement *noncritical* at ℓ_1, which represents the noncritical activity of the processes, each process branches nondeterministically to either the *read protocol* R, at location $\ell_{2..7}$, or to the *write protocol* W, at $\ell_{8..14}$.

In the *read protocol*, each reader tries to decrement y and achieve a nonnegative value in t (and in y). Since the initial value of y is M, up to M readers may decrement y by 1, and still obtain a nonnegative value. If a reader succeeds to obtain a nonnegative value for t, it proceeds to the *read section* at ℓ_6. If it does not, it corrects y at ℓ_7 and continues to loop.

The *writer's protocol* is similar, except that a writer subtracts M From y, attempting to obtain a nonnegative value. This is possible only if that particular writer is the only one currently subtracting from y. Even if one reader has subtracted and lowered y to $M-1$, the writer, subtracting a further M will already obtain -1, and be barred from entering the writing section.

Another special feature of the write protocol is the wait at ℓ_{10} for y to become M. This is similar to the **await** $y > 0$ we had in the mutual exclusion program MUX-AST (Fig. 2.25). This is necessary, because otherwise we can construct a computation in which two writers chase one another around the ℓ_9 loop, keeping y always negative, without any of them or a reader being able to enter any of the critical sections. With the *await* gate at ℓ_{10}, this cannot happen as we will prove below.

Proving Mutual Exclusion

The first property we prove is that of exclusion, as required by the problem. This is established by several invariants.

- *Invariant* I_1

$$N_{5..7} + M \cdot N_{12..14} + y = M.$$

This equality holds initially, since $y = M$ and $N_{5..7} = N_{12..14} = 0$. It is

```
in     M: integer where M > 0
local y : integer where y = M

         M
        ||  P[i] ::
        i=1
  [ local t: integer where t = 0
    ℓ0: loop forever do
        [ ℓ1: noncritical
          [        [ ℓ2: t := −1
                   [ ℓ3: while t < 0 do
            R ::   [       [ ℓ4: t := y := y − 1
                   [       [ ℓ5: if t ≥ 0 then
                   [       [       ℓ6: Read
                   [       [ ℓ7: y := y + 1
               or
                   [ ℓ8: t := −1
                   [ ℓ9: while t < 0 do
                   [       [ ℓ10: await y = M
            W ::   [       [ ℓ11: t := y := y − M
                   [       [ ℓ12: if t ≥ 0 then
                   [       [       ℓ13: Write
                   [       [ ℓ14: y := y + M           ] ] ] ]
```

Fig. 2.26. Program READ-WRITE (readers-writers with add-and-store).

preserved by transitions, such as ℓ_4 which increments $N_{5..7}$ by 1 and decrements y by 1, or ℓ_{11} which increments $N_{12..14}$ by 1 and decrements y by M. Similarly, it is preserved by ℓ_7 and ℓ_{14}.

- *Invariant* I_2

$$N^{t\geq 0}_{5..7} \ + \ M \cdot N^{t\geq 0}_{12..14} \ \leq \ M.$$

There are two transitions that may endanger the invariance of this assertion.

- $\ell_4[i]$ while $N^{t\geq 0}_{5..7} + M \cdot N^{t\geq 0}_{12..14} = M$

However, due to I_1,

$$y \ = \ M \ - \ N_{5..7} \ - \ M \cdot N_{12..14}$$

$$\leq \; M \; - \; N^{t\geq 0}_{5..7} \; - \; M \cdot N^{t\geq 0}_{12..14} \; = \; 0.$$

So the execution of $\ell_4[i]$ will produce $t'[i] < 0$, and hence $N^{t\geq 0}_{5..7}$ does not increase.

- $\ell_{11}[i]$ while $N^{t\geq 0}_{5..7} + M \cdot N^{t\geq 0}_{12..14} > 0$

This is a dangerous situation since by adding one more element to $N^{t\geq 0}_{12..14}$ the sum will increase beyond M.

However due to I_1, and a calculation identical to the one before we obtain

$$y \; \leq \; M \; - \; N^{t\geq 0}_{5..7} \; - \; M \cdot N^{t\geq 0}_{12..14} \; < \; M.$$

Consequently, the execution of $\ell_{11}[i]$ produces $t'[i] < 0$, and does not increase $N^{t\geq 0}_{12..14}$.

- *Invariant* I_3

$$L_6 \; \subseteq \; L^{t\geq 0}_{5..7}.$$

This is equivalent to the implication

$$at\_\ell_6[i] \quad \rightarrow \quad t[i] \geq 0,$$

claimed for every $i = 1, \ldots, M$. The invariant can be verified by considering the transitions that enter ℓ_6. Since a process $P[i]$ can enter ℓ_6 only if $t[i] \geq 0$, it follows that $i \in L^{t\geq 0}_{5..7}$, after the transition.

- *Invariant* I_4

Similarly to the above invariant, we can also prove

$$L_{13} \; \subseteq \; L^{t\geq 0}_{12..14},$$

which is equivalent to

$$at\_\ell_{13}[i] \quad \rightarrow \quad t[i] \geq 0.$$

From I_3 and I_4 we can conclude

$$N_6 \; + \; M \cdot N_{13} \; \leq \; N^{t\geq 0}_{5..7} \; + \; M \cdot N^{t\geq 0}_{12..14}.$$

Using I_2 we conclude

$$N_6 \; + \; M \cdot N_{13} \; \leq \; M.$$

From this it is easy to infer

$$N_{13} > 0 \quad \rightarrow \quad N_6 = 0 \; \wedge \; N_{13} = 1,$$

which is precisely the exclusion property required. It states that if some process is at ℓ_{13}, then it is the only one there, and no process is at ℓ_6.

Proving Accessibility

The response property we prove for this program is again that of *communal accessibility*. Since this program has two types of critical sections, a *read* and a *write* section, communal accessibility has an even broader interpretation. It states that if some process is interested in entering *one* of the critical sections, then some process will eventually enter *one* of the critical sections. We cannot even guarantee, for example, that if some process wants to read (write) then some process will eventually read (write). This property is expressible by

$$N_2 + N_8 > 0 \quad \Rightarrow \quad \diamondsuit(N_6 + N_{13} > 0).$$

We prove this property by two lemmas, concentrating first on the *writers*.

Lemma A $\quad N_8 > 0 \quad \Rightarrow \quad \diamondsuit(N_6 + N_{13} > 0)$

This lemma states that if a process is interested in writing, then eventually some process will either read (visit ℓ_6) or write (visit ℓ_{13}).

The proof of the lemma is established by a sequence of simpler lemmas, identifying important intermediate stages in getting from $N_8 > 0$ to $N_6 + N_{13} > 0$. We refer the reader to Fig. 2.27 which presents a diagram showing the structure of the proof.

Lemma A1 $\quad N_8 > 0 \quad \Rightarrow \quad \diamondsuit(N_{10} > 0)$

This lemma ensures that if some process is currently at ℓ_8 then eventually some process will arrive at ℓ_{10}.

This simple property is a direct consequence of the property

$$at\_\ell_8[i] \quad \Rightarrow \quad \diamondsuit\ at\_\ell_{10}[i],$$

which is established by the RANK diagram of Fig. 2.28.

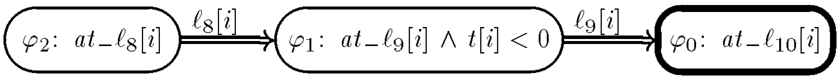

Fig. 2.28. P-RANK diagram for Lemma A1.

Arriving at a state satisfying $N_{10} > 0$, there are two cases to be considered. The easier one is that $y = M$, and Lemma A4 below shows how to get from this situation to the goal $N_6 + N_{13} > 0$. The more complicated situation is when $y < M$ (in view of the invariant I1, always $y \leq M$). The two lemmas, A2 and A3, show that being at $N_{10} > 0 \wedge y < M$ we eventually get to the easier case of $N_{10} > 0 \wedge y = M$. They split the case $N_{10} > 0 \wedge y < M$ into two subcases, according to whether $N_{11..14}$ is positive or zero, and show that from both we eventually arrive to $N_{10} > 0 \wedge y = M$.

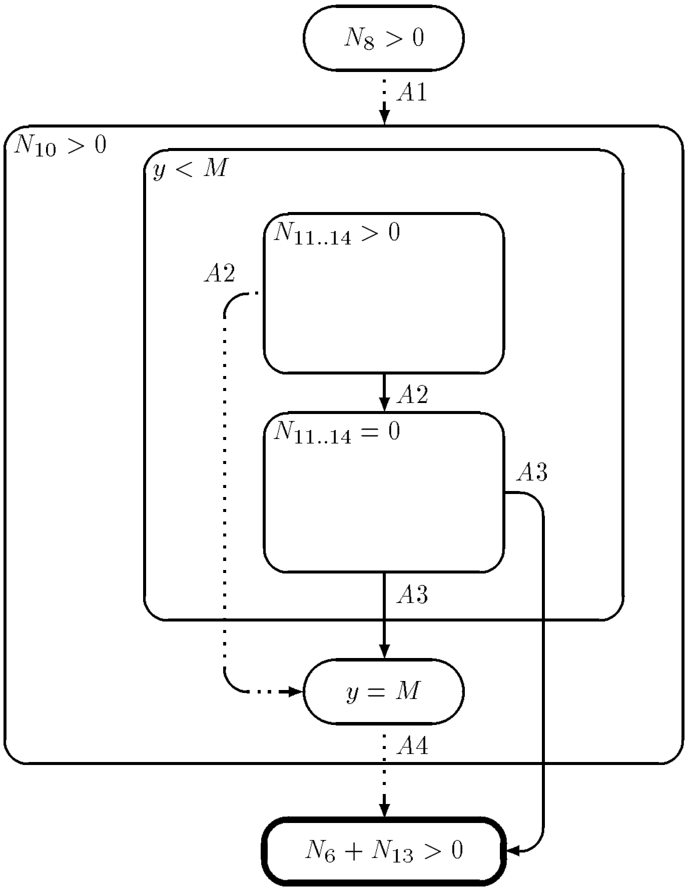

Fig. 2.27. Structure of the proof of Lemma A.

Lemma A2 $\quad N_{10} > 0 \ \wedge \ y < M \ \wedge \ N_{11..14} > 0 \quad \Rightarrow$

$$\diamondsuit\big(N_{10} > 0 \ \wedge \ (N_{11..14} = 0 \ \vee \ y = M)\big)$$

The proof of this lemma can be based on rule WELL-JP, and is presented in the P-RANK diagram of Fig. 2.29.

Note that transitions from ℓ_{14} may either retain $N_{11..14} > 0$ and $y < M$, or evacuate $L_{11..14}$ completely, or increase y to become M. It is possible for y to equal M, while some processes are still at $\ell_{10,11}$.

Note that we have to consider also the possibility that a transition ℓ_7 increases y to become M.

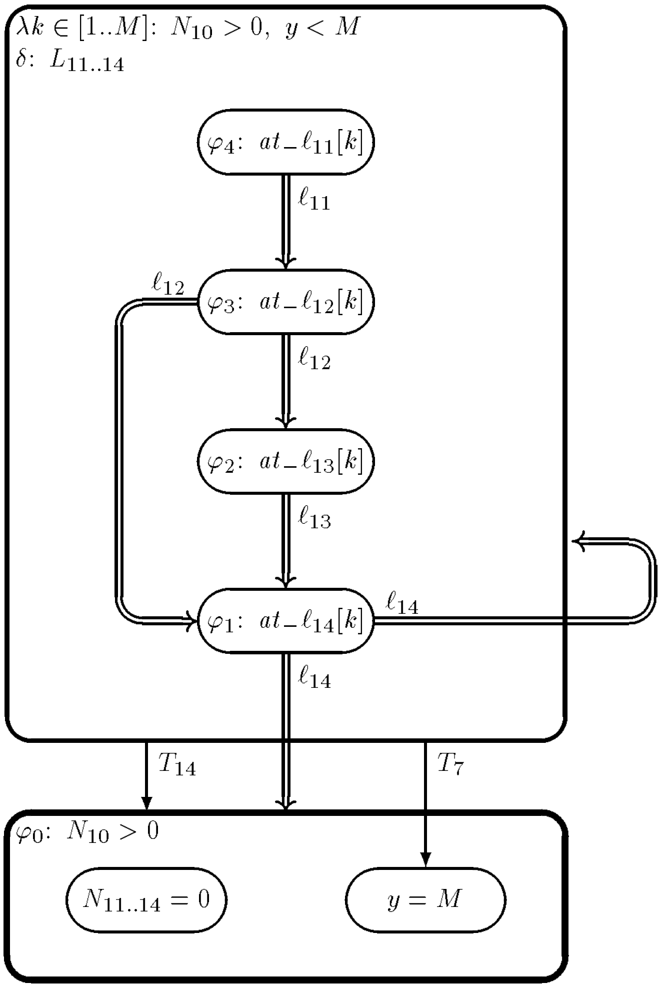

Fig. 2.29. P-RANK diagram for Lemma A2.

In the proof we rely on the fact that while $y < M$, all transitions of the form $\ell_{10}[i]$ are disabled.

Lemma A3 $N_{10} > 0 \;\wedge\; y < M \;\wedge\; N_{11..14} = 0 \quad\Rightarrow$

$$\Diamond\big((N_{10} > 0 \;\wedge\; y = M) \;\vee\; N_6 > 0\big)$$

The proof of the lemma is presented in the P-RANK diagram of Fig. 2.30.

We observe that, due to I_1, the antecedent implies that $0 < N_{4..7} < M$ and

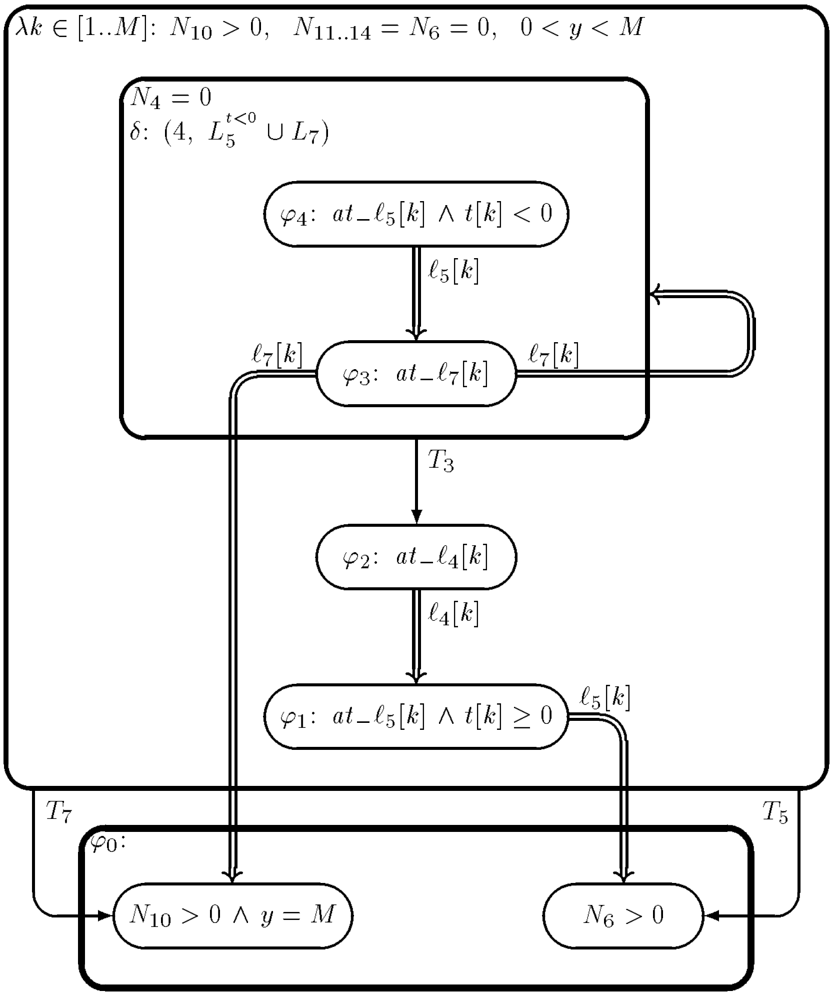

Fig. 2.30. P-RANK diagram for Lemma A3.

$0 < y < M$. This situation is split in the diagram into several subcases:

- $N_4 = N_6 = 0 \ \wedge \ N_5^{t<0} + N_7 > 0$ is covered by φ_3 and φ_4
- $N_4 > 0$ is covered by φ_2
- $N_5^{t\geq 0} > 0$ is covered by φ_1
- $N_6 > 0$ is the goal φ_0.

The sum $N_5^{t<0} + N_7$ decreases on each ℓ_7 transition. It must eventually drop to zero, ensuring $y = M$, unless N_4 becomes positive before.

Combining Lemmas A2 and A3 together, we obtain

$$N_{10} > 0 \ \wedge\ y < M \quad \Rightarrow \quad \Diamond\big((N_{10} > 0 \ \wedge\ y = M) \ \vee\ N_6 > 0\big).$$

It remains to show that from $(N_{10} > 0) \wedge (y = M)$ we are also guaranteed to reach $N_{13} + N_6 > 0$. This is claimed by the next lemma.

Lemma A4 $\quad N_{10} > 0 \ \wedge\ y = M \quad \Rightarrow \quad \Diamond(N_6 + N_{13} > 0)$

The proof of this lemma is presented in the P-RANK diagram of Fig. 2.31.

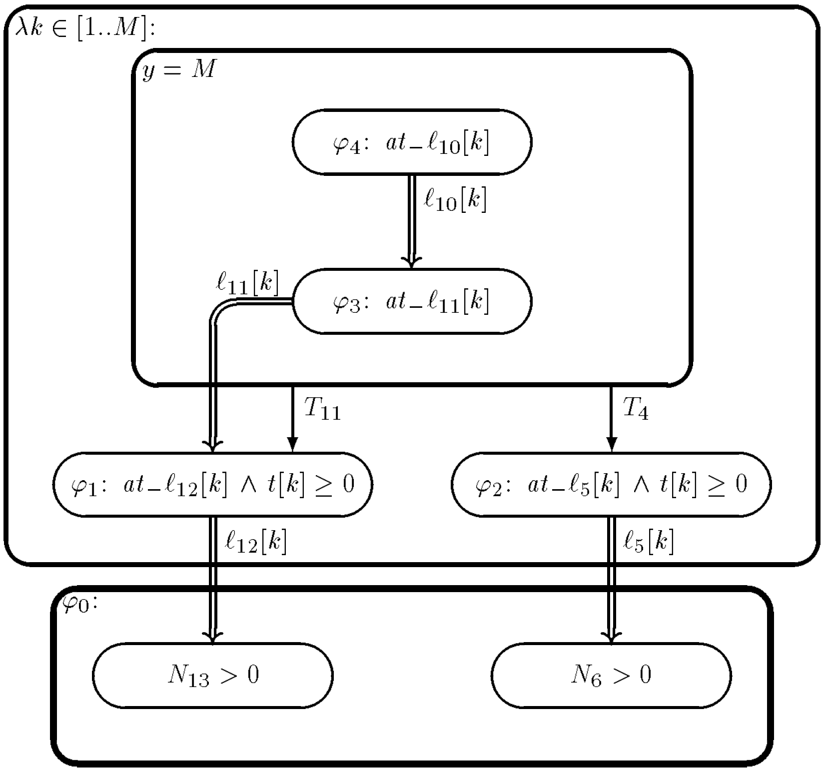

Fig. 2.31. P-RANK diagram for Lemma A4.

This concludes the proof of Lemma A, which states that if a process wishes to write then eventually some process will read or some process will write.

By combining Lemmas A1, A2, and A3, we can also obtain Corollary A

Corollary A $N_{10} > 0 \;\Rightarrow\; \diamondsuit(N_6 + N_{13} > 0)$

The second case we have to handle is the possibility that a process wishes to read. This is covered by

Lemma B $N_2 > 0 \;\Rightarrow\; \diamondsuit(N_6 + N_{13} > 0)$

The proof of Lemma B is also split into several intermediate stages. In Fig. 2.32 we present these stages and the lemmas that lead from one stage to the next.

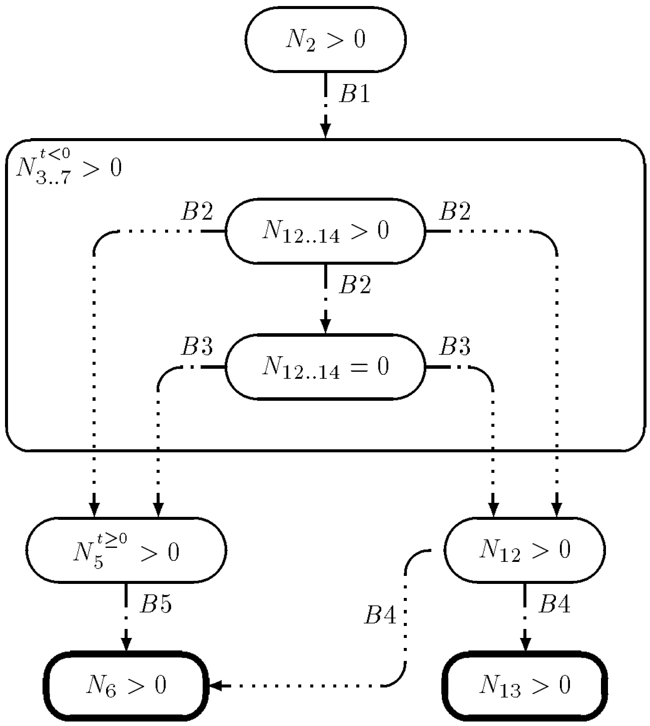

Fig. 2.32. Structure of the proof of Lemma B.

Lemma B1 $N_2 > 0 \;\Rightarrow\; \diamondsuit\left(N^{t<0}_{3..7} > 0\right)$

This simple lemma follows a process that wishes to read, from location ℓ_2 to location ℓ_3, where it arrives with $t[i] < 0$. The lemma is so simple that we skip its proof.

Arriving at ℓ_3, it is important to distinguish between the case that $y > 0$ and the case $y \le 0$. Due to the invariant I_1, if $y \le 0$ then either $N_{12..14} > 0$ or $N_{5..7} = M$. Since $N_3 > 0$, it is impossible for all processes to be at $\ell_{5..7}$, and therefore we conclude that $N_{12..14} > 0$. The next lemma shows that this case must develop to the simpler case in which $N_{12..14} = 0$. However, we cannot hope that while this happens, the process which is currently at ℓ_3 will stand still. What can it do? It can loop around $\ell_{3..7}$ and perhaps even get to ℓ_5 with a nonnegative $t[i]$. If this happens then we are close to our goal $N_6 > 0$. If it does not happen, we should at least make sure that this process does not escape the loop $\ell_{3..7}$ without visiting ℓ_6 first. To contain some processes in the loop we use the set $L_{3..7}^{t<0}$, observing that the only way a process can leave this set, is by moving to $L_5^{t\ge0}$.

Consequently, the next lemma establishes

Lemma B2

$$N_{3..7}^{t<0} > 0 \ \wedge\ N_{12..14} > 0 \quad\Rightarrow\quad \diamondsuit \begin{bmatrix} \left(N_{3..7}^{t<0} > 0 \ \wedge\ N_{12..14} = 0\right) \vee \\ N_5^{t\ge0} > 0 \ \vee\ N_{12} > 0 \end{bmatrix}$$

The proof of this lemma is presented in the P-RANK diagram of Fig. 2.33. It uses the ranking function $N_{12..14}$ which decreases on each ℓ_{14} transition. By I1, as long as $N_{12..14} > 0$, y is smaller than M.

In a more general situation we would have to worry about the possibility of $N_{12..14}$ increasing due to an ℓ_{11} transition. Here, however, any ℓ_{11} transition makes L_{12} nonempty, which is one of the goals of this lemma.

Similarly, the only transition that can violate $N_{3..7}^{t<0} > 0$ is L_4 from a positive y. This, however, makes $L_5^{t\ge0}$ nonempty, which is another goal.

The next lemma considers a situation in which there is still some unsatisfied process (i.e., $t[i] < 0$) in the $\ell_{3..7}$ loop, while $N_{12..14} = 0$. It shows that eventually one of the sets $L_5^{t\ge0}$ or L_{12} must become nonempty.

Lemma B3 $\quad N_{3..7}^{t<0} > 0 \ \wedge\ N_{12..14} = 0 \quad\Rightarrow\quad \diamondsuit\left(N_5^{t\ge0} > 0 \ \vee\ N_{12} > 0\right)$

The lemma is proved in P-RANK diagram of Fig. 2.34. Note that while being at ℓ_6, $t[i] \ge 0$, so that $L_6^{t<0}$ is always empty. Therefore, having $N_{3..7}^{t<0} > 0$ implies that we must have a nonempty $L_{3..5,7}$. If L_4 is nonempty, then due to $N_{12..14} = 0$ and to I1, we are guaranteed to have a positive y. Therefore, any ℓ_4 transition from such a state, leads to a nonempty $L_5^{t\ge0}$.

Lemma B4 $\quad N_{12} > 0 \quad\Rightarrow\quad \diamondsuit(N_6 + N_{13} > 0)$

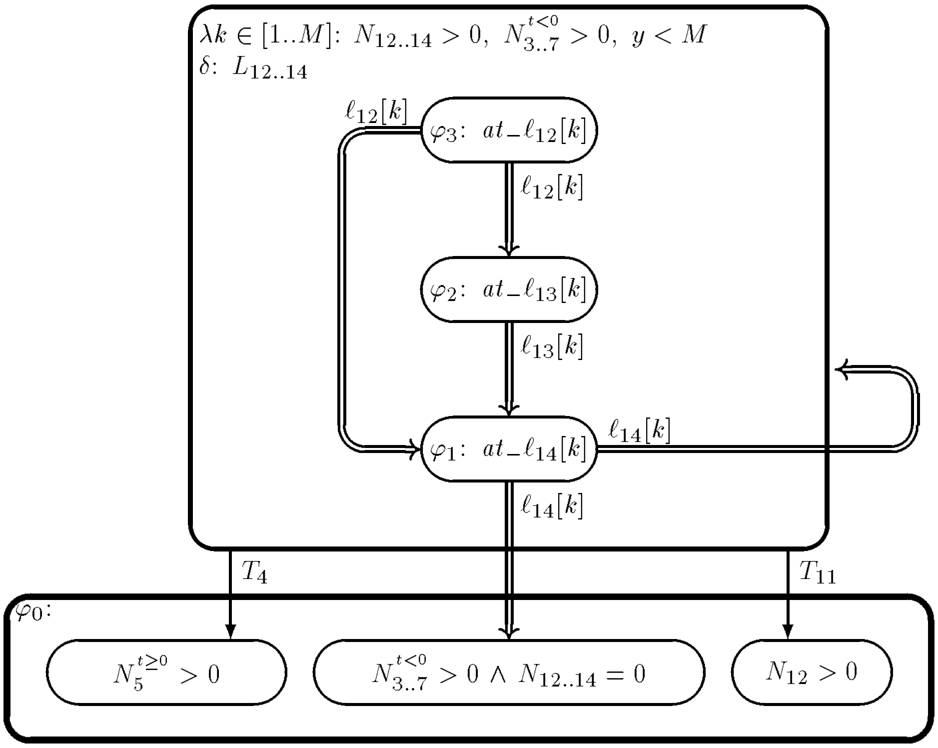

Fig. 2.33. P-RANK diagram for Lemma B2.

This lemma starts from a situation in which we identify a writer at ℓ_{12}. If $t[i] \geq 0$ for this process, then we can easily prove that eventually $N_{13} > 0$. Otherwise, $t[i] < 0$, and we can easily trace the progress of this process from ℓ_{12}, through ℓ_{14}, ℓ_9 until it reaches ℓ_{10}. We now use Lemma A to show that if L_{10} is nonempty, then eventually some process will reach ℓ_6 or ℓ_{13}.

Lemma B5 $\quad N^{t\geq 0}_5 > 0 \quad \Rightarrow \quad \diamondsuit(N_6 > 0)$

This lemma requires one helpful step, performed by any of the processes in $L^{t\geq 0}_5$, to achieve a nonempty L_6.

Problems

Problem 2.1 (mutual exclusion by turn setting) page 105

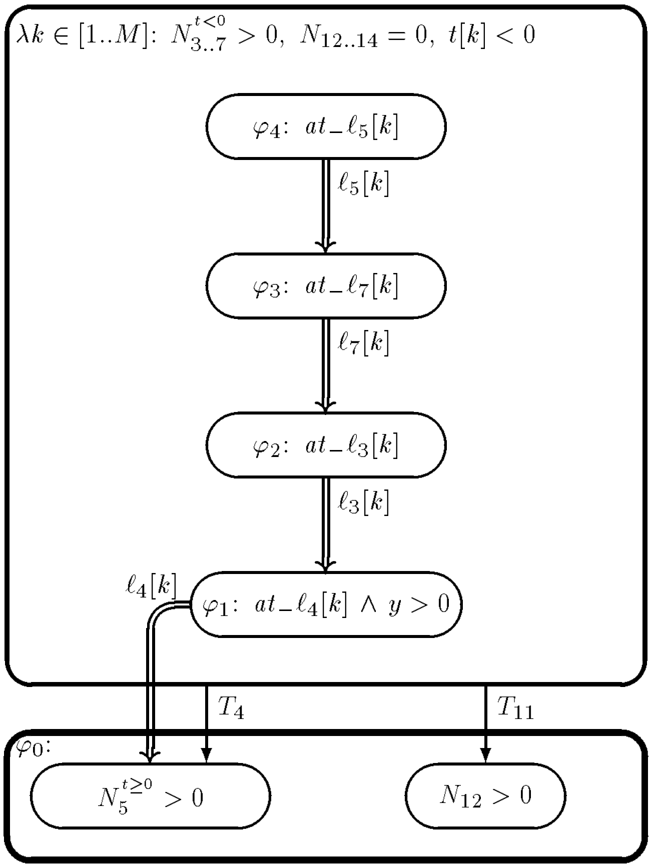

Fig. 2.34. P-RANK diagram for Lemma B3.

Show that the property of (individual) accessibility, specified by

$$at\_\ell_2[i] \land t = 0 \Rightarrow \Diamond\, at\_\ell_5[i]$$

is not guaranteed for program TURN (Fig. 2.8).

Problem 2.2 (mutual exclusion with central manager) page 105

Program PMUX-MAN of Fig. 2.35 implements mutual exclusion, using a central allocator process A which receives requests in variable y and responds in variable x. Using the parameterized rules presented in this chapter, prove accessibility for this program, which can be specified by the response formula

$$N_2 > 0 \Rightarrow \Diamond(N_4 > 0),$$

where N_2 counts the number of processes $P[j]$ currently executing at location m_2, while N_4 counts the number of processes $P[j]$ currently executing at location m_4

local x, y: **integer where** $x = 0$, $y = 0$

$$A :: \left[\begin{array}{l} \ell_0\text{: }\textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: }\textbf{while } y = 0 \textbf{ do} \\ \qquad \ell_2\text{: }\textbf{skip} \\ \ell_3\text{: } x := y \\ \ell_4\text{: }\textbf{while } x \neq 0 \textbf{ do} \\ \qquad \ell_5\text{: }\textbf{skip} \\ \ell_6\text{: } y := 0 \end{array}\right] \end{array}\right]$$

$$\|$$

$$\mathop{\Big\|}_{j=1}^{M} P[j] :: \left[\begin{array}{l} m_0\text{: }\textbf{loop forever do} \\ \qquad \left[\begin{array}{l} m_1\text{: }\textbf{noncritical} \\ m_2\text{: }\textbf{while } x \neq j \textbf{ do} \\ \qquad m_3\text{: } y := j \\ m_4\text{: }\textbf{critical} \\ m_5\text{: } x := 0 \end{array}\right] \end{array}\right]$$

Fig. 2.35. Program PMUX-MAN.

Note that this program cannot guarantee individual accessibility but only communal accessibility (see page 104).

Problem 2.3 (molecular bakery algorithms) page 117

In Fig. 2.17, we presented the atomic version of the bakery algorithm and verified its properties.

This atomic version is obviously not an LCR-program and represents a very simplified version which cannot be directly implemented on existing hardware.

A first attempt to transform program BAKERY-AT into an LCR program breaks each of the non-LCR statements ℓ_2 and ℓ_3 into a *for*-loop. This leads to program BAKERY-1, presented in Fig. 2.36.

While program BAKERY-1 obeys the LCR restriction, it is no longer a correct solution to the mutual exclusion problem.

```
in M     : integer where M > 1
   ticket: array [1..M] of integer where ticket = 0

              ⎡ j, k, t: integer where j = k = t = 0
              ⎢ ℓ0: loop forever do
              ⎢    ⎡ ℓ1: noncritical
              ⎢    ⎢ ℓ2: t := 0
              ⎢    ⎢ ℓ3: for k = 1 to M do
 M            ⎢    ⎢        ℓ4: t := max(t, ticket[k])
 ||  P[i] ::  ⎢    ⎢ ℓ5: ticket[i] := t + 1
i=1           ⎢    ⎢ ℓ6: for j = 1 to M do
              ⎢    ⎢      ⎡              (   j = i                   ) ⎤
              ⎢    ⎢      ⎢ ℓ7: await   ( ∨ ticket[j] = 0           ) ⎥
              ⎢    ⎢      ⎣              ( ∨ ticket[i] < ticket[j]   ) ⎦
              ⎢    ⎢ ℓ8: critical
              ⎣    ⎣ ℓ9: ticket[i] := 0
```

Fig. 2.36. Program BAKERY-1: an LCR version.

(a) Present a segment of a computation of program BAKERY-1 which violates mutual exclusion.

As can be seen from your counter-example, the fault can be traced to the fact that while process $P[i]$ is computing its next *ticket* value at locations ℓ_3–ℓ_5, the value of $ticket[i]$ is unreliable and should not be checked by other processes at statement ℓ_7.

To correct this problem, we introduce a protection mechanism, by which $ticket[i]$ cannot be accessed while $P[i]$ is at the location range ℓ_3–ℓ_5.

This leads to program BAKERY-2 of Fig. 2.37.

The protection mechanism implemented in BAKERY-2, indeed solves the problem of mutual-exclusion violation.

(b) Prove that program BAKERY-2 ensures mutual exclusion.

While program BAKERY-2 guarantees mutual exclusion, it fails to maintain the complementary required property of accessibility.

(c) Show a computation of program BAKERY-2 which fails to satisfy the property

$$
\begin{array}{l}
\textbf{in } M \quad\quad\;\; : \textbf{integer where } M > 1 \\
\quad \mathit{choosing}\text{: } \textbf{array } [1..M] \textbf{ of boolean where } \mathit{choosing} = \mathrm{F} \\
\quad \mathit{ticket} \quad\; : \textbf{array } [1..M] \textbf{ of integer where } \mathit{ticket} = 0
\end{array}
$$

$$
\overset{M}{\underset{i=1}{\Big\|}} P[i] :: \left[
\begin{array}{l}
j,\ k,\ t\text{: } \textbf{integer where } j = k = t = 0 \\
\ell_0\text{: } \textbf{loop forever do} \\
\quad \left[
\begin{array}{l}
\ell_1\text{: } \textbf{noncritical} \\
\ell_2\text{: } \mathit{choosing}[i] := \mathrm{T} \\
\ell_3\text{: } t := 0 \\
\ell_4\text{: } \textbf{for } k = 1 \textbf{ to } M \textbf{ do} \\
\quad \ell_5\text{: } t := \max\big(t,\ \mathit{ticket}[k]\big) \\
\ell_6\text{: } \mathit{ticket}[i] := t + 1 \\
\ell_7\text{: } \mathit{choosing}[i] := \mathrm{F} \\
\ell_8\text{: } \textbf{for } j = 1 \textbf{ to } M \textbf{ do} \\
\quad \left[
\begin{array}{l}
\ell_9\text{: } \textbf{await } \neg \mathit{choosing}[j] \\
\ell_{10}\text{: } \textbf{await } \left(
\begin{array}{l}
\quad j = i \\
\vee\ \mathit{ticket}[j] = 0 \\
\vee\ \mathit{ticket}[i] < \mathit{ticket}[j]
\end{array}
\right)
\end{array}
\right] \\
\ell_{11}\text{: } \textbf{critical} \\
\ell_{12}\text{: } \mathit{ticket}[i] := 0
\end{array}
\right]
\end{array}
\right]
$$

Fig. 2.37. Program BAKERY-2: ticket computation is protected.

of accessibility for one of the processes.

To overcome the difficulty illustrated by your counter-example, we augment the priority comparison mechanism of statement ℓ_7 by a secondary component, which is the process index. Thus, if both $\mathit{ticket}[i]$ and $\mathit{ticket}[j]$ are different from zero, then $P[i]$ has a higher priority if

$$\big(\mathit{ticket}[i],\ i\big) \prec \big(\mathit{ticket}[j],\ j\big),$$

where the comparison is lexicographic.

This leads to program BAKERY of Fig. 2.38.

This version is fianlly satisfactory in that it satisfies the properties of mutual exclusion and accessibility.

(d) Prove that program BAKERY satisfies the property of mutual exclusion.

$$
\begin{array}{l}
\textbf{in } M \quad\; : \textbf{integer where } M > 1 \\
\quad \mathit{choosing}: \textbf{array } [1..M] \textbf{ of boolean where } \mathit{choosing} = \mathrm{F} \\
\quad \mathit{ticket} \quad : \textbf{array } [1..M] \textbf{ of integer where } \mathit{ticket} = 0 \\[1em]
\overset{M}{\underset{i=1}{\|}} P[i] :: \left[\begin{array}{l}
j,\ k,\ t\text{: } \textbf{integer where } j = k = t = 0 \\
\ell_0\text{: } \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\text{: } \textbf{noncritical} \\
\ell_2\text{: } \mathit{choosing}[i] := \mathrm{T} \\
\ell_3\text{: } t := 0 \\
\ell_4\text{: } \textbf{for } k = 1 \textbf{ to } M \textbf{ do} \\
\quad \ell_5\text{: } t := \max\big(t,\ \mathit{ticket}[k]\big) \\
\ell_6\text{: } \mathit{ticket}[i] := t + 1 \\
\ell_7\text{: } \mathit{choosing}[i] := \mathrm{F} \\
\ell_8\text{: } \textbf{for } j = 1 \textbf{ to } M \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_9\text{: } \textbf{await } \neg \mathit{choosing}[j] \\
\ell_{10}\text{: } \textbf{await } \left(\begin{array}{l} \quad j = i \\ \vee\ \mathit{ticket}[j] = 0 \\ \vee\ \mathit{ticket}[i] \prec \mathit{ticket}[j] \end{array}\right)
\end{array}\right] \\
\ell_{11}\text{: } \textbf{critical} \\
\ell_{12}\text{: } \mathit{ticket}[i] := 0
\end{array}\right]
\end{array}\right]
\end{array}
$$

Fig. 2.38. Program BAKERY: Final version.

(e) Prove that program BAKERY satisfies the property of accessibility.

$*$ **Problem 2.4** (Peterson's algorithm for n processes) page 117

Program MUX-PET-N of Fig. 2.39 presents a solution to the mutual-exclusion problem for an arbitrary number of processes.

The *while* statement at ℓ_3 gradually increases the priority of process $P[i]$ from 1 to n. When the priority, expressed by j ($j[i]$ for process $P[i]$), grows beyond n, $P[i]$ is admitted to its critical section. On entering priority level j, process $P[i]$ records (at ℓ_4) its current priority in $y[i]$ and writes its identity number (i) in $s[j]$ the signature logbook for this level. It then remains at level j until it detects one of two occurrences. Either the number of processes of priority not smaller than j becomes 1, which means that $P[i]$ is the only one with such priority, or the logbook $s[j]$ assumes a value different than i, implying that some of the processes

```
in     n    : integer where n ≥ 2
local y, s: array[1..n] of integer  where y = s = 0

          n
          ||  P[i] ::
         i=1
  [local j, k, above-me: integer
   ℓ0: loop forever do
     [ℓ1: noncritical
      ℓ2: j := 1
      ℓ3: while j < n do
        [ℓ4: y[i] := j
         ℓ5: s[j] := i
         ℓ6: above-me := n
         ℓ7: while above-me > 1 ∧ s[j] = i do
           [ℓ8: above-me := 0
            ℓ9: k := 1
            ℓ10: while k ≤ n do
              [ℓ11: if y[k] ≥ j then
                      [ℓ12: above-me := above-me + 1]
               ℓ13: k := k + 1                        ]]
         ℓ14: j := j + 1                               ]
      ℓ15: critical
      ℓ16: y[i] := 0                                    ]]
```

Fig. 2.39. Program MUX-PET-N (mutual exclusion for n processes).

entered level j *after* $P[i]$.

To check for the first occurrence, statements ℓ_8 to ℓ_{13} repeatedly count the number of processes with priority not smaller than j. Note that one cannot assume that on termination of the *while* statement ℓ_{10} the value of *above-me* necessarily equals the number of processes with priority not smaller than j. While $P[i]$ counts, some processes with high priority may go through the critical section and lower their priority, and some processes with low priority may raise their priority to the tested level after being considered.

Prove that program MUX-PET-N guarantees accessibility to each of its processes, by showing that the response formula

$$at\_\ell_2[i] \;\Rightarrow\; \Diamond\, at\_\ell_{15}[i]$$

is P-valid over MUX-PET-N.