Chapter 1

Response

In the preceding chapters we discussed methods for proving safety properties. While the class of safety properties is very important, and normally occupies a large portion of a specification, it must be complemented by properties in the other classes.

It is interesting to note that the expression of safety properties and their verification use relatively little temporal logic. The main emphasis there is directed towards finding assertions that are invariant over the computation of a program. Most of the verification efforts are concentrated on showing that the assertions are inductive. This requires proving a set of verification conditions, which are expressed by nontemporal state formulas. Temporal logic is used mainly for stating the final result of invariance of the assertion. It is true that, when considering precedence properties, we extensively use the past part of temporal logic. But as we commented there, an equivalent, though sometimes less elegant, state formulation of these properties can be managed through auxiliary or history variables.

It is only when we enter the realm of more general properties, that temporal logic becomes an essential and irreplaceable tool. Thus if, for some reason, one is willing to restrict himself to the study of safety properties of reactive programs, he does not need the full power of temporal logic.

A related observation is that, only when we go beyond safety properties does fairness become meaningful. Recall the definition of a *run* as a state sequence that satisfies the requirements of initiation and consecution but not necessarily any of the fairness requirements. It can be shown that a safety formula holds over all runs of a program if and only if it holds over all computations, i.e., fair runs. Thus, safety properties cannot distinguish between fair and unfair runs.

This is no longer the case with progress (nonsafety) properties. Note, for example, that the infinite sequence $s_0, s_0, \ldots$ is a legal run of any program P, provided $s_0 \vDash \Theta$. This run is generated by continuously taking the idling transition τ_I. There are very few progress properties that hold over this run.

Consequently, while safety properties do not depend on the fairness requirements for their validity, progress properties do. In Chapters 1–3 we concentrate on the important class of *response* properties, which is one of the progress classes. This class contains properties that can be expressed by a formula of the form

$$\Box \Diamond p,$$

for a past formula p. In these chapters, we introduce a family of rules for response properties that rely on the different fairness requirements. Chapters 1–2 present rules that rely on *justice*, while Chapter 3 presents rules that rely on (justice and) *compassion*.

Chapter 4 completes the picture by presenting rules for the highest progress class, that of reactivity.

Chapter 1 deals with response properties that rely on the just transitions of the system for their validity. Chapter 3 generalizes the treatment to properties that rely on both justice and compassion for their validity.

In Section 1.1 we consider a single-step rule that relies on the activation of a single just transition.

Section 1.2 shows how to combine several applications of the single-step rule into a rule that relies on a fixed number of activations of just transitions.

Section 1.3 generalizes the rule to the case that the number of just activations necessary to achieve the goal is not fixed and may depend, for example, on an input parameter.

In Section 1.4, we extend all the above methods to prove properties expressed by response formulas that contain past subformulas.

Section 1.5 deals with the class of guarantee formulas, treating them as a special case of response properties.

In a similar way, Section 1.6 considers the class of obligation properties. Again, their verification is based on their consideration as a special case of the response class.

1.1 Response Rule

Even though there are several different classes of progress properties, their verification is almost always based on the establishment of a single construct — the *response formula*

$$p \Rightarrow \Diamond q,$$

for past formulas p and q. This formula states that any position in the computation which satisfies p must be followed by a later position which satisfies q. Since the canonical response formula $\square \diamond q$ is equivalent to $\mathrm{T} \Rightarrow \diamond q$, it follows that every response property can be expressed by a formula of the form $p \Rightarrow \diamond q$.

A general response property allows p and q to be general past formulas. In Sections 1.1–1.3 we consider the simpler case that p and q are assertions. In Section 1.4 we will generalize the treatment to the case that p and q are past formulas.

A Single-Step Rule

A single-step rule, relying on justice, is provided by rule RESP-J presented in Fig. 1.1.

For assertions p, q, φ, and transition $\tau_h \in \mathcal{J}$,

$$\begin{array}{ll} \text{J1.} & p \rightarrow q \vee \varphi \\ \text{J2.} & \{\varphi\}\ \mathcal{T}\ \{q \vee \varphi\} \\ \text{J3.} & \{\varphi\}\ \tau_h\ \{q\} \\ \text{J4.} & \varphi \rightarrow En(\tau_h) \\ \hline & p \Rightarrow \diamond q \end{array}$$

Fig. 1.1. Rule RESP-J (single-step response under justice).

The rule calls for the identification of an intermediate assertion φ and a just transition $\tau_h \in \mathcal{J}$, to which we refer as the *helpful transition*.

Premise J1 of the rule states that, in any position satisfying p, either the goal formula q already holds, or the intermediate formula φ, bridging the passage from p to q, holds. The q-disjunct of this premise covers the case that the distance between the p-position and the q-position is 0. The φ-disjunct and the other premises cover the case that the distance between these two positions is positive.

Premise J2 requires that every transition leads from a φ-position to a position that satisfies $q \vee \varphi$. That is, either a position satisfying the goal formula q is attained or, if not, then at least the intermediate φ is maintained.

Premise J3 requires that the helpful transition τ_h always leads from a φ-position to a q-position.

Premise J4 requires that the helpful transition τ_h is enabled at every φ-position.

Justification To justify the rule, consider a computation σ which satisfies the four premises of the rule. Assume that p holds at position k, $k \geq 0$. We wish to show that q holds at some position i, $i \geq k$. Assume, to the contrary, that it does not. That means that for all i, $i \geq k$, q does not hold at i. By J1, φ holds at position k. By J2, every successor of a φ-state is a $(q \vee \varphi)$-state.

Since we assumed that q never occurs beyond k, it follows that φ holds continuously beyond k. By J4, the just transition τ_h must be continuously enabled. However, τ_h is never taken beyond k. This is because if τ_h were taken, it would have been taken from a φ-position, and by J3 the next position would have satisfied q. Thus we have that τ_h is continuously enabled, but never taken beyond k. It follows that the sequence σ is not just with respect to τ_h, and is, therefore, not a computation.

This shows that, for all computations, there must exist an i, $i \geq k$ such that q holds at i. ▟

In applications of the rule, it is sufficient to establish premise J2 for all $\tau \neq \tau_h$, since J2 for $\tau = \tau_h$ is implied by J3. It is also unnecessary to check premise J2 for $\tau = \tau_I$, the idling transition, since $\{\varphi\}\ \tau_I\ \{\varphi\}$ is trivially state valid.

Example (program ANY-Y)

Program ANY-Y of Fig. 1.2 illustrates a simple program consisting of two processes communicating by the shared variable x, initially set to 0. Process P_1 keeps incrementing variable y as long as $x = 0$. Process P_2 has only one statement, which sets x to 1. Obviously, once x is set to 1, process P_2 terminates, and some time later so does P_1, as soon as it observes that $x \neq 0$.

$$\textbf{local}\ \ x,\ y\text{: }\textbf{integer where } x = y = 0$$

$$P_1 :: \begin{bmatrix} \ell_0\text{: } \textbf{while } x = 0 \textbf{ do} \\ \quad \ell_1\text{: } y := y + 1 \\ \ell_2\text{: } \end{bmatrix} \quad \| \quad P_2 :: \begin{bmatrix} m_0\text{: } x := 1 \\ m_1\text{: } \end{bmatrix}$$

Fig. 1.2. Program ANY-Y.

We illustrate the use of rule RESP-J for proving the response property

$$at\_m_0 \ \Rightarrow\ \Diamond(x = 1)$$

for program ANY-Y.

As the helpful transition τ_h we take m_0. As the intermediate assertion φ we take p: at_m_0. Premise J1 assumes the form

$$\underbrace{at\_m_0}_{p} \rightarrow \underbrace{\cdots}_{q} \vee \underbrace{at\_m_0}_{\varphi},$$

which is obviously valid. Premise J2 is established by showing that all transitions, excluding m_0, preserve φ: at_m_0, which is clearly the case.

Premise J3 requires showing that m_0 leads from any φ-state to a q-state, expressed by

$$\underbrace{\cdots \wedge x' = 1}_{\rho_{m_0}} \wedge \underbrace{\cdots}_{\varphi} \rightarrow \underbrace{x' = 1}_{q'},$$

which is obviously valid. Finally, J4 requires

$$\underbrace{at\_m_0}_{\varphi} \rightarrow \underbrace{at\_m_0}_{En(m_0)},$$

which is also valid. This establishes that the property specified by the response formula $at\_m_0 \Rightarrow \diamondsuit(x = 1)$ is valid over program ANY-Y. ▪

Combining Response Properties

Rule RESP-J by itself is not a very strong rule, and is sufficient only for proving *one-step* response properties, i.e., properties that can be achieved by a single activation of a helpful transition. For example, while program ANY-Y always terminates, its termination cannot be proven by a single aplication of rule RESP-J.

In general, most response properties of the form $p \Rightarrow \diamondsuit q$ require several helpful steps in order to get from a p-position to a q-position.

To establish such properties we may use several rules that enable us to combine response properties, each established by a single application of rule RESP-J. These rules are based on general properties of response formulas that allow us to form these combinations. We list some of these properties as proof rules. All of these rules can be established as derived rules, using the standard deductive system for temporal logic[1].

Monotonicity

An important property of response formulas is the monotonicity of both the antecedent and the consequent. This can be summarized in the form of the (monotonicity) rule MON-R, presented in Fig. 1.3.

[1] For example, the one presented in Chapter 3 of Volume I.

$$\frac{p \Rightarrow q \qquad q \Rightarrow \Diamond r \qquad r \Rightarrow t}{p \Rightarrow \Diamond t}$$

Fig. 1.3. Rule MON-R (monotonicity of response)

Rule MON-R enables us to strengthen the antecedent and weaken the consequent. Thus, if we managed to prove the response formula

$$at\_\ell_0 \;\Rightarrow\; \Diamond(x = 1),$$

we can infer from it, using rule MON-R, the formula

$$at\_\ell_0 \;\Rightarrow\; \Diamond(x > 0).$$

Reflexivity

Property RFLX-R of Fig. 1.4 states that the $\Diamond$ operator is reflexive.

$$p \;\Rightarrow\; \Diamond p$$

Fig. 1.4. Property RFLX-R (reflexivity of response)

We may use this property to prove simple response formulas such as

$$x = 0 \;\Rightarrow\; \Diamond(x = 0).$$

Transitivity

The transitivity property of response formulas is expressed by the (transitivity) rule TRNS-R, presented in Fig. 1.5.

$$\frac{p \Rightarrow \Diamond q \qquad q \Rightarrow \Diamond r}{p \Rightarrow \Diamond r}$$

Fig. 1.5. Rule TRNS-R (transitivity of response)

Thus, if we managed to prove for program ANY-Y the two response formulas

$$at\_\ell_1 \ \wedge\ at\_m_1 \ \wedge\ x = 1 \quad \Rightarrow \quad \diamond(at\_\ell_0 \ \wedge\ at\_m_1 \ \wedge\ x = 1)$$

$$at\_\ell_0 \ \wedge\ at\_m_1 \ \wedge\ x = 1 \quad \Rightarrow \quad \diamond(at\_\ell_2 \ \wedge\ at\_m_1),$$

we may use rule TRNS-R to conclude

$$at\_\ell_1 \ \wedge\ at\_m_1 \ \wedge\ x = 1 \quad \Rightarrow \quad \diamond(at\_\ell_2 \ \wedge\ at\_m_1).$$

The soundness of rule TRNS-R is obvious. Consider a computation σ such that the first two premises are valid over σ. Let i be a position satisfying p. By the first premise, there exists a position j, $j \geq i$, satisfying q. By the second premise, there exists a position k, $k \geq j$, satisfying r. Thus, we are ensured of a position k, $k \geq i$, satisfying r, which establishes $p \Rightarrow \diamond r$.

Proof by Cases

Another useful property of response formulas is that it is amenable to proof by cases. This possibility is presented by rule CASES-R of Fig. 1.6.

$$\frac{p \ \Rightarrow\ \diamond r \qquad q \ \Rightarrow\ \diamond r}{(p \ \vee\ q) \ \Rightarrow\ \diamond r}$$

Fig. 1.6. Rule CASES-R (case analysis for response)

Assume, for example, that we have proved for program ANY-Y the two following reponse formulas.

$$at\_\ell_0 \ \wedge\ at\_m_1 \ \wedge\ x = 1 \quad \Rightarrow \quad \diamond(at\_\ell_2 \ \wedge\ at\_m_1)$$

$$at\_\ell_1 \ \wedge\ at\_m_1 \ \wedge\ x = 1 \quad \Rightarrow \quad \diamond(at\_\ell_2 \ \wedge\ at\_m_1).$$

Then, we may use rule CASES-R to conclude

$$(at\_\ell_0 \ \wedge\ at\_m_1 \ \wedge\ x = 1) \ \vee\ (at\_\ell_1 \ \wedge\ at\_m_1 \ \wedge\ x = 1) \quad \Rightarrow \quad \diamond(at\_\ell_2 \ \wedge\ at\_m_1),$$

from which, by rule MON-R, we can infer

$$at\_\ell_{0,1} \ \wedge\ at\_m_1 \ \wedge\ x = 1 \quad \Rightarrow \quad \diamond(at\_\ell_2 \ \wedge\ at\_m_1).$$

Example (program ANY-Y)

We will illustrate the use of these rules by proving termination of program ANY-Y. This property can be expressed by the response formula

$$\Theta \;\Rightarrow\; \Diamond(at\_\ell_2 \;\wedge\; at\_m_1),$$

where

$$\Theta\!: \quad \pi = \{\ell_0, m_0\} \;\wedge\; x = 0 \;\wedge\; y = 0$$

is the initial condition of program ANY-Y.

The proof consists of the following steps:

1. $at\_\ell_0 \;\wedge\; at\_m_0 \;\wedge\; x = 0 \;\Rightarrow\; \Diamond(at\_\ell_{0,1} \;\wedge\; at\_m_1 \;\wedge\; x = 1)$
   by rule RESP-J, taking τ_h: m_0 and φ: $at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x = 0$

2. $at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\Rightarrow\; \Diamond(at\_\ell_2 \;\wedge\; at\_m_1)$
   by rule RESP-J, taking τ_h: ℓ_0 and φ: $at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x = 1$

3. $at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\Rightarrow\; \Diamond(at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x = 1)$
   by rule RESP-J, taking τ_h: ℓ_1 and φ: $at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x = 1$

4. $at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\Rightarrow\; \Diamond(at\_\ell_2 \;\wedge\; at\_m_1)$
   by rule TRNS-R, applied to 3 and 2

5. $(at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x = 1) \;\vee\; (at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x = 1) \;\Rightarrow\; \Diamond(at\_\ell_2 \;\wedge\; at\_m_1)$
   by rule CASES-R, applied to 2 and 4

6. $at\_\ell_{0,1} \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\rightarrow\; (at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x = 1) \;\vee\; (at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x = 1)$
   an assertional validity

7. $at\_\ell_{0,1} \;\wedge\; at\_m_1 \;\wedge\; x = 1 \;\Rightarrow\; \Diamond(at\_\ell_2 \;\wedge\; at\_m_1)$
   by rule MON-R, using 5 and 6

8. $at\_\ell_0 \;\wedge\; at\_m_0 \;\wedge\; x = 0 \;\Rightarrow\; \Diamond(at\_\ell_2 \;\wedge\; at\_m_1)$
   by rule TRNS-R, applied to 1 and 7

9. $\Theta \;\rightarrow\; at\_\ell_0 \;\wedge\; at\_m_0 \;\wedge\; x = 0$ an assertional validity

10. $\Theta \;\Rightarrow\; \Diamond(at\_\ell_2 \;\wedge\; at\_m_1)$ by rule MON-R, applied to 8 and 9. ▟

1.2 Chain Rule

The proof of the last example follows a very specific pattern that occurs often in proofs of response properties. According to this pattern, to establish $p \Rightarrow \Diamond q$, we identify a sequence of intermediate situations described by assertions $\varphi_m, \varphi_{m-1}, \ldots, \varphi_0$ such that p implies one of $\varphi_m, \ldots, \varphi_0$, and q is identical with

φ_0 (or is implied by φ_0). We then show that for every $i > 0$, being at φ_i implies that eventually we must reach φ_j for some $j < i$.

We can interpret the index i of the intermediate formula φ_i as a measure of the distance of the current state from a state that satisfies the goal q. Thus, the lower the index, the closer we are to achieving the goal q. For a position j, let φ_i be the intermediate formula with the smallest i s.t. φ_i holds at j. We refer to the index i as the *rank* of position j.

This proof pattern is summarized in rule CHAIN-J (Fig. 1.7).

For assertions p and $q = \varphi_0, \varphi_1, \ldots, \varphi_m$ and transitions $\tau_1, \ldots, \tau_m \in \mathcal{J}$

$$\text{J1.}\quad p \rightarrow \bigvee_{j=0}^{m} \varphi_j$$

$$\left.\begin{array}{ll} \text{J2.} & \{\varphi_i\}\ \mathcal{T}\ \left\{\bigvee_{j \leq i} \varphi_j\right\} \\ \text{J3.} & \{\varphi_i\}\ \tau_i\ \left\{\bigvee_{j < i} \varphi_j\right\} \\ \text{J4.} & \varphi_i \rightarrow En(\tau_i) \end{array}\right\} \text{ for } i = 1, \ldots, m$$

$$\overline{\qquad p \Rightarrow \Diamond q \qquad}$$

Fig. 1.7. Rule CHAIN-J (chain rule under justice).

According to premise J1, p implies that one of the intermediate formulas φ_i (possibly φ_0 implying q) holds. Premise J2 requires that taking any transition from a φ_i-position results in a next position which satisfies φ_j, for some $j \leq i$. Premise J3 requires that taking the helpful transition τ_i from a φ_i-position results in a next position which satisfies φ_j for $j < i$. We can view premise J2 as stating that the rank never increases, while premise J3 states that the helpful transition guarantees that the rank decreases. Premise J4 claims that the helpful transition τ_i is enabled at every φ_i-position.

Justification Assume that all four premises are P-state valid. Consider a P-computation σ and a position t that satisfies p. We wish to prove that some later position satisfies q. Assume to the contrary that all positions later than t (including t itself) do not satisfy q. By J1, position t satisfies φ_j for some $j \geq 0$. Index j cannot be 0 because $\varphi_0 = q$ and we assumed that no position beyond t satisfies q. Thus, position t satisfies φ_j, for some $j > 0$. By J2, position $t+1$ satisfies some φ_k, $k \leq j$. Again, $k > 0$ due to $\varphi_0 = q$. Continuing in this manner, it follows that every position beyond t satisfies some φ_j for $j > 0$, to which we refer as the rank of the position.

By J2, the rank of the position can either decrease or remain the same. It follows that there must exist some position $k \geq t$, beyond which the rank never decreases.

Assume that i is the rank of the state at position k. Since q is never satisfied and the rank never decreases beyond position k, it follows (by J2) that φ_i holds continually beyond k. By J3, τ_i cannot be taken beyond k, because that would have led to a rank decrease. By J4, τ_i is continually enabled beyond k yet, by the argument above, it is never taken. This violates the requirement of justice for τ_i.

It follows that if all the premises of the rule are P-state valid then the conclusion $p \Rightarrow \diamondsuit\, q$ is P-valid. ▟

Note that since premise J3 implies premise J2 for $\tau = \tau_i$, it is sufficient to check premise J2 for a given $i = 1, \ldots, m$, only for $\tau \neq \tau_i$. Also, it is unnecessary to check premise J2 for $\tau = \tau_I$, since $\{\varphi_i\}\ \tau_I\ \{\varphi_i\}$ trivially holds.

Example (Reproving termination of program ANY-Y)

Let us show how termination of program ANY-Y can be proved (again) by a single application of rule CHAIN-J.

The property we wish to prove is

$$\underbrace{at\_\ell_0 \ \wedge\ at\_m_0 \ \wedge\ x = 0 \ \wedge\ y = 0}_{p = \Theta} \quad \Rightarrow \quad \diamondsuit\, \underbrace{at\_\ell_2 \ \wedge\ at\_m_1}_{q}\ .$$

Inspired by our previous proof of this property, we choose four assertions and corresponding helpful transitions as follows:

$$\begin{array}{lll}
\varphi_3: & at_\ell_{0,1} \ \wedge\ at_m_0 \ \wedge\ x = 0 & \tau_3:\ m_0 \\
\varphi_2: & at_\ell_1 \ \wedge\ at_m_1 \ \wedge\ x = 1 & \tau_2:\ \ell_1 \\
\varphi_1: & at_\ell_0 \ \wedge\ at_m_1 \ \wedge\ x = 1 & \tau_1:\ \ell_0 \\
\varphi_0 = q: & at_\ell_2 \ \wedge\ at_m_1. &
\end{array}$$

Let us consider each of the premises of rule CHAIN-J.

- Premise J1

  This premise calls for proving

$$p \;\rightarrow\; \bigvee_{j=0}^{3} \varphi_j.$$

We will prove $p \rightarrow \varphi_3$, which amounts to

$$\underbrace{at\_\ell_0 \;\wedge\; at\_m_0 \;\wedge\; x=0 \;\wedge\; y=0}_{p} \;\;\rightarrow\;\; \underbrace{at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x=0}_{\varphi_3}$$

which obviously holds.

- Premises J2–J4 for $i = 3, 2, 1$

  We list below the premises that are proven for each i, $i = 3, 2, 1$.

  - Assertion φ_3: $at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x=0$

$$\Big\{\underbrace{at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x=0}_{\varphi_3}\Big\} \;\; \tau \;\; \Big\{\underbrace{at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x=0}_{\varphi_3}\Big\}$$

for each $\tau \neq m_0$

$$\Big\{\underbrace{at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x=0}_{\varphi_3}\Big\} \;\; m_0 \;\; \left\{\begin{array}{l} \underbrace{at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_2} \;\;\vee \\ \underbrace{at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_1} \end{array}\right\}$$

$$\underbrace{at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x=0}_{\varphi_3} \;\;\rightarrow\;\; \underbrace{at\_m_0}_{En(m_0)}\;.$$

  - Assertion φ_2: $at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x=1$

$$\Big\{\underbrace{at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_2}\Big\} \;\; \tau \;\; \Big\{\underbrace{at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_2}\Big\}$$

for every $\tau \neq \ell_1$

$$\Big\{\underbrace{at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_2}\Big\} \;\; \ell_1 \;\; \Big\{\underbrace{at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_1}\Big\}$$

$$\Big[\underbrace{at\_\ell_1 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_2}\Big] \;\;\rightarrow\;\; \underbrace{at\_\ell_1}_{En(\ell_1)}\;.$$

  - Assertion φ_1: $at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x=1$

$$\Big\{\underbrace{at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_1}\Big\} \;\; \tau \;\; \Big\{\underbrace{at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_1}\Big\}$$

for every $\tau \neq \ell_0$

$$\{\underbrace{at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_1}\}\;\; \ell_0 \;\; \{\underbrace{at\_\ell_2 \;\wedge\; at\_m_1}_{\varphi_0}\}$$

$$\underbrace{at\_\ell_0 \;\wedge\; at\_m_1 \;\wedge\; x=1}_{\varphi_1} \;\;\rightarrow\;\; \underbrace{at\_\ell_0}_{En(\ell_0)} .$$

All of these implications and verification conditions are obviously state valid, which establishes the conclusion

$$\Theta \;\Rightarrow\; \Diamond(at\_\ell_2 \;\wedge\; at\_m_1). \qquad ▟$$

Relation to rule NWAIT

There is a strong resemblance between the premises of rule CHAIN-J and those of rule NWAIT (Fig. 3.6 on page 267 of the SAFETY book). This is not surprising, since in both cases we wish to establish the evolution from p to q (q_0 in the case of NWAIT) by successively passing through $\varphi_m, \varphi_{m-1}, \ldots, \varphi_1, \varphi_0$.

The main difference is that, in rule NWAIT, we are quite satisfied if, from a certain point on, we stay forever within φ_j for some $j > 0$. This is unacceptable in a response rule, where we are anxious to establish eventual arrival at φ_0. This difference is expressed in premise J3 which requires that activation of the helpful transition τ_i takes us out of a φ_i-state, and premise J4 which requires that τ_i is enabled on all φ_i-states. These premises have no counterparts in rule NWAIT. This excludes computations that consist of states whose rank, from a certain point on, never decreases below some $j > 0$.

Another difference is that, in rule NWAIT, we allow a transition to lead from φ_i, $i > 0$, back to φ_i. This is expressed by the disjunction, $\bigvee\limits_{j \leq i} \varphi_j$, appearing in the postcondition of premise N3. Rule CHAIN-J allows this in premise J2 for all but the helpful transition, which is required in J3 to lead to a position with a strictly lower rank.

In spite of these differences, many of the approaches used in the study of precedence properties are also applicable to the analysis of response properties. One of these useful approaches is that of *verification diagrams* introduced in Section 3.3 of the SAFETY book.

1.3 Chain Diagrams

As observed in the previous example (program ANY-Y), in many cases it suffices to specify the intermediate assertions $\varphi_0, \varphi_1, \ldots, \varphi_m$ and to identify the helpful transitions $\tau_1, \ldots, \tau_m$. The proofs of the actual verification conditions is a detail that can be left to skeptical readers, and eventually to an automated system. Only some of the verification conditions raise interesting questions, and those are usually elaborated in a presentation of a proof. However, the main structure of the proof is adequately represented by the list of assertions and their corresponding helpful transitions.

A concise visual summary of this information, and some additional details, is provided by verification diagrams. Verification diagrams were already introduced in Section 3.3 of the SAFETY book to represent proofs using rule NWAIT. However, we give here an independent description of the diagrams that support proofs of response properties by rule CHAIN-J.

Verification Diagrams

A verification diagram is a directed labeled graph constructed as follows:

- *Nodes* in the graph are labeled by assertions. We will often refer to a node by the assertion labeling it.
- *Edges* in the graph represent transitions between assertions. The diagrams presenting proofs by rule CHAIN-J allow edges of two types, represented graphically by *single* (*-lined*) and *double* (*-lined*) arrows. Each edge of either type departs from one assertion, connects to another, and is labeled by the name of a transition. We refer to an edge labeled by τ as a τ*-edge*.
- One of the nodes may be designated as a *terminal node* ("goal" node). In the graphical representation, this node is distinguished by having a boldface boundary. No edges depart from a terminal node. Terminal nodes correspond to "goal" assertions such as φ_0 in rule CHAIN-J.

Chain Diagrams

A verification diagram is said to be a *chain diagram* if its nodes are labeled by assertions $\varphi_0, \ldots, \varphi_m$, with φ_0 being the terminal node, and if it satisfies the following requirements:

- If a single (-line) edge connects node φ_i to node φ_j, then $i \geq j$.
- If a double (-line) edge connects node φ_i to node φ_j, then $i > j$.
- Every node φ_i, $i > 0$, has a double edge departing from it. This identifies the transition labeling such an edge as *helpful* for assertion φ_i. All helpful transitions must be just.

The first two requirements ensure that the diagram is weakly acyclic in the sense defined in Section 3.3 of the SAFETY book for WAIT diagrams. That is, the terminal node is labeled by φ_0 and whenever node φ_i is connected by an edge (single or double) to node φ_j, then $j \leq i$. The stronger second requirement ensures that the subgraph based on the double edges is acyclic, forbidding self-connections by double edges. The third requirement demands that every nonterminal assertion (i.e., φ_i for $i > 0$) has at least one helpful transition associated with it.

Verification Conditions for CHAIN Diagrams

The assertions labeling nodes in a diagram are intended to represent the intermediate assertions appearing in a CHAIN-J proof. A τ-labeled edge connecting node φ_i to node φ_j implies that it is possible for a φ_i-state to have a τ-successor satisfying φ_j. A double edge departing from node φ and labeled by transition τ identifies τ as helpful for assertion φ. Consequently, we associate verification conditions with nodes and the edges departing from them. These conditions, expressed by implications, represent premises J2–J4 of rule CHAIN-J.

For a node φ_i and transition τ, connecting φ_i to φ_j, we say that φ_j is a τ-successor of φ_i. Let φ be a nonterminal node and $\varphi_1, \ldots, \varphi_k$, $k \geq 0$, be the τ-successors of φ.

V1. If all the edges connecting φ to its τ-successors are single (-lined), then we associate with φ and τ the verification condition

$$\{\varphi\}\ \tau\ \{\varphi\ \vee\ \varphi_1\ \vee\ \cdots\ \vee\ \varphi_k\}.$$

Transition τ, labeling only single edges, is identified as unhelpful for φ. This condition, similar to premise J2, allows τ to lead from a φ-state back to a φ-state, recording no progress.

The case of a transition τ that does not label any edges departing from φ is interpreted as though τ labels $k = 0$ single-lined edges departing from φ. That is, with such a transition we associate the verification condition

$$\{\varphi\}\ \tau\ \{\varphi\}.$$

V2. If some edge departing from φ is double (hence $k > 0$), we associate with φ and τ the verification condition

$$\{\varphi\}\ \tau\ \{\varphi_1\ \vee\ \cdots\ \vee\ \varphi_k\}.$$

This condition corresponds to premise J3, requiring a transition τ, identified as helpful, to lead away from φ.

V3. If τ labels some double edge departing from φ, we require

$$\varphi\ \rightarrow\ En(\tau).$$

This condition corresponds to premise J4, requiring that a transition helpful for φ is enabled on all φ-states. We refer to this requirement as the *enabling requirement*.

Valid CHAIN Diagrams

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

The consequences of having a P-valid CHAIN diagram are stated by the following claim:

Claim 1.1 (CHAIN diagrams)

A P-valid CHAIN diagram establishes that the response formula

$$\bigvee_{j=0}^{m} \varphi_j \;\Rightarrow\; \diamondsuit\, \varphi_0$$

is P-valid.

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

$$\text{(J1)} \quad p \;\to\; \bigvee_{j=0}^{m} \varphi_j \qquad \text{and} \qquad \text{(J0)} \quad \varphi_0 \;\to\; q$$

then, we can conclude the P-validity of

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

Justification First, we show the first part of the claim, stating the P-validity of

$$\bigvee_{j=0}^{m} \varphi_j \;\Rightarrow\; \diamondsuit\, \varphi_0.$$

We use rule CHAIN-J with p: $\bigvee_{j=0}^{m} \varphi_j$, $q = \varphi_0$ and, for each $i = 1, \ldots, m$, we take τ_i (the helpful transition for φ_i) to be the transition labeling the double edge departing from φ_i.

For our choice of p and q, premise J1 of rule CHAIN-J assumes the form

$$\text{(J1)} \quad \bigvee_{j=0}^{m} \varphi_j \;\to\; \bigvee_{j=0}^{m} \varphi_j,$$

which is trivially state-valid. We proceed to show that the P-state validity of premises J2–J4 follows from the P-validity of the diagram, for each $i = 1, \ldots, m$.

Premise J2 requires showing

$$\text{(J2)} \quad \rho_\tau \;\wedge\; \varphi_i \;\to\; \varphi_0' \;\vee\; \varphi_1' \;\vee\; \cdots \;\vee\; \varphi_i',$$

for each $\tau \in \mathcal{T}$. Let $\varphi_{i_1}, \ldots, \varphi_{i_k}$ be the τ-successors of φ_i in the P-valid diagram, for $\tau \in \mathcal{T} - \{\tau_i, \tau_I\}$. By the requirement of weak acyclicity $i_1 \leq i, \ldots, i_k \leq i$. Since $\tau \neq \tau_i$, all the τ-edges departing from node φ are single-line edges and the following verification condition holds:

$$\text{V1.} \quad \rho_\tau \;\wedge\; \varphi_i \;\to\; \varphi_i' \;\vee\; \varphi_{i_1}' \;\vee\; \cdots \;\vee\; \varphi_{i_k}'.$$

Since $i_j \leq i$ for each $j = 1, \ldots, k$, the disjunction on the right-hand side of V1 is taken over a subset of the assertions appearing on the right-hand side of premise J2. It follow that J2 is state-valid for assertion φ_i and transition τ. For $\tau = \tau_I$, premise J2 holds trivially since ρ_{τ_I} implies $\varphi_i' = \varphi_i$. For $\tau = \tau_i$, premise J2 is implied by J3.

Premise J3 requires

$$\text{(J3)} \quad \rho_{\tau_i} \ \wedge \ \varphi_i \ \rightarrow \ \varphi_0' \ \vee \ \varphi_1' \ \vee \ \cdots \ \vee \ \varphi_{i-1}'.$$

Let $\varphi_{i_1}, \ldots, \varphi_{i_k}$ be the τ_i-successors of φ_i in the P-valid diagram. Since all τ_i-edges departing from φ_i are double, $i_j < i$ for $j = 1, \ldots, k$ and the following verification condition holds:

$$\text{V2.} \quad \rho_\tau \ \wedge \ \varphi_i \ \rightarrow \ \varphi_{i_1}' \ \vee \ \cdots \ \vee \ \varphi_{i_k}'.$$

Repeating the subset argument, this implies the state validity of premise J3.

Premise J4 is identical to condition V3 for every φ_i and τ_i, $i = 1, \ldots, m$.

Next, we consider the more general case of p and q which are not identical to $\bigvee_{j=0}^{m} \varphi_j$ and φ_0, respectively, but satisfy the implications J1 and J0. Applying rule MON-R to p, $\bigvee_{j=0}^{m} \varphi_j$, φ_0, and q (standing for p, q, r, and t in MON-R), we obtain the conclusion $p \Rightarrow \Diamond q$. ▟

Note that chain diagrams and their notion of validity are a conservative extension of the WAIT diagrams, introduced in Section 3.3 of the SAFETY book. The additional requirements that disallow a self-connecting edge all refer to double edges which are not present in WAIT diagrams. It follows that a P-valid CHAIN diagram is also P-valid for proving the nested waiting-for formula

$$\bigvee_{j=0}^{m} \varphi_j \quad \Rightarrow \quad \varphi_m \ \mathcal{W} \ \varphi_{m-1} \cdots \varphi_1 \ \mathcal{W} \ \varphi_0.$$

Example (program ANY-Y)

Consider again program ANY-Y (Fig. 1.2). The CHAIN diagram of Fig. 1.8 provides a graphical representation for the proof of the response property

$$\underbrace{at\_\ell_0 \ \wedge \ at\_m_0 \ \wedge \ x = 0}_{p=\Theta} \quad \Rightarrow \quad \Diamond \Big(\underbrace{at\_\ell_2 \ \wedge \ at\_m_1}_{q=\varphi_0} \Big) \ ,$$

for program ANY-Y.

The diagram identifies $\varphi_3, \ldots, \varphi_0$ as the intermediate assertions and m_0, ℓ_1, ℓ_0 as their corresponding helpful transitions. This CHAIN diagram is valid over program ANY-Y, which establishes the P-validity of

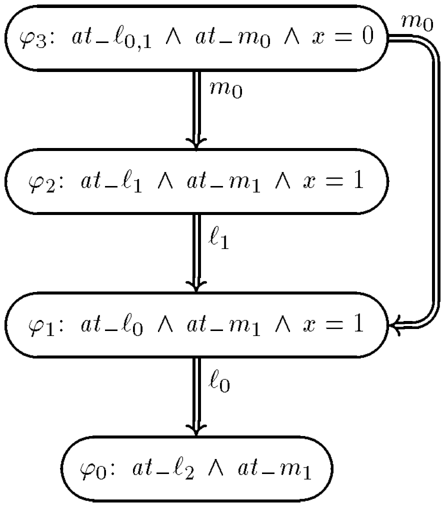


Fig. 1.8. CHAIN diagram for termination.

$$\bigvee_{j=0}^{3} \varphi_j \quad \Rightarrow \quad \Diamond(at\_\ell_2 \ \wedge \ at\_m_1)$$

over this program. Since (as shown above) $\Theta \rightarrow \varphi_3$, the second part of Claim 1.1 establishes the P-validity of the termination property

$$\Theta \quad \Rightarrow \quad \Diamond(at\_\ell_2 \ \wedge \ at\_m_1). \quad ▟$$

The Advantages of Diagrams

One of the advantages of the presentation of a CHAIN-J proof sketch by a CHAIN diagram, over its presentation by a list of assertions and their corresponding helpful transitions is that the diagram provides a stronger (and more detailed) version of premises J2 and J3 than is standardly provided by rule CHAIN-J and a list of the assertions and helpful transitions.

Consider, for example, the proof presented in Fig. 1.8. Both the diagram and the textual proof identify φ_2 as $at\_\ell_1 \wedge at\_m_1 \wedge x = 1$ and ℓ_1 as its helpful transition.

However, while rule CHAIN-J suggests that we prove for premise J3 the verification condition

$$\{\varphi_2\} \ \ell_1 \ \{\varphi_0 \vee \varphi_1\},$$

the diagram claims that the even stronger condition

$$\{\varphi_2\} \ \ell_1 \ \{\varphi_1\}$$

is P-state valid. This results from the fact that there is no ℓ_1-edge connecting φ_2 to φ_0.

Encapsulation Conventions

There are several encapsulation conventions that lead to more structured hierarchical diagrams and improve the readability and manageability of large complex diagrams. These conventions were introduced in Section 3.3 of the SAFETY book and we reproduce them here briefly, to make the presentation self contained. The basic construct of encapsulation is that of a *compound node* that may contain several internal nodes. The encapsulation conventions attribute to a compound node aspects and relations that are common to all of their contained nodes. We refer to the contained nodes as *descendants* of the compound node. Nodes that are not compound are called *basic nodes*. We use three encapsulation conventions.

- *Departing edges*

  An edge departing from a compound node is interpreted as though it departed from each of its descendants. This is represented by the graphical equivalence of Fig. 1.9.

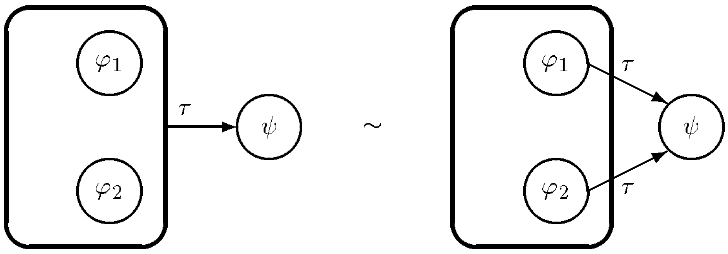


Fig. 1.9. Departing edges.

- *Arriving edges*

  In a similar way, an edge arriving at a compound node is interpreted as though it arrived at each of its descendants. This is represented in the graphical equivalence of Fig. 1.10.

- *Common factors*

  An assertion φ labeling a compound node is interpreted as though it were a conjunct added to each of the labels of its descendants. This is represented by the graphical equivalence of Fig. 1.11. We refer to φ as a *common factor* of the two nodes.

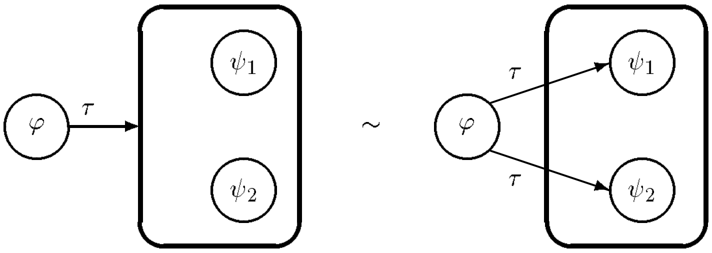


Fig. 1.10. Arriving edges.

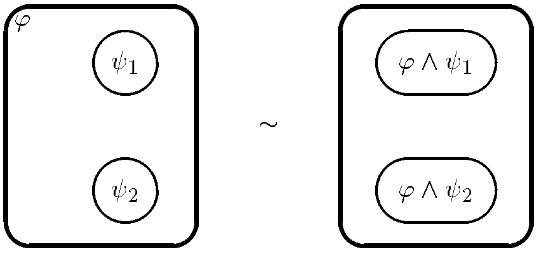


Fig. 1.11. Common factors.

Example In Fig. 1.12 we present a verification diagram which is the encapsulated version of the verification diagram of Fig. 1.8.

This encapsulation uses the arriving edge convention to denote by a single arrow the two edges connecting φ_3 to φ_2 and to φ_1. It uses the common factor convention to simplify the presentation of φ_1 and φ_2. ■

Additional Examples

Let us consider a few more examples for the application of rule CHAIN-J and CHAIN diagrams, illustrating the encapsulation conventions.

Example (Peterson's Algorithm — version 1)

For the next example, we return to program MUX-PET1, Peterson's algorithm for mutual exclusion (Fig. 1.13). This program was previously studied in Section 1.4 of the SAFETY book, where we established for it the following invariants:

$$\psi_0:\quad s = 1 \;\vee\; s = 2$$

$$\psi_1:\quad y_1 \;\leftrightarrow\; at\_\ell_{3..5}$$

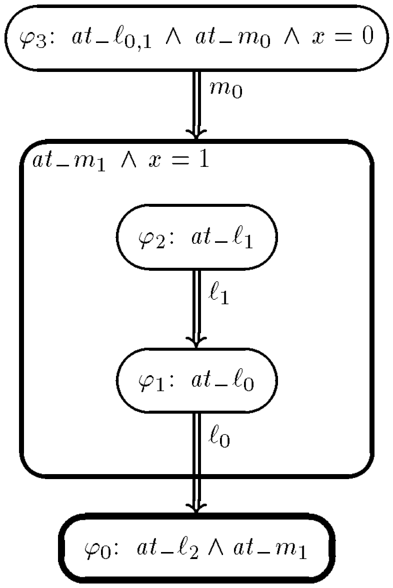


Fig. 1.12. Encapsulated version of CHAIN diagram for termination.

$$\psi_2: \quad y_2 \;\leftrightarrow\; at\_m_{3..5}.$$

We wish to prove for this program the response property of accessibility, given by

$$\underbrace{at\_\ell_2}_{p} \;\Rightarrow\; \Diamond\, \underbrace{at\_\ell_4}_{q}\ .$$

To use rule CHAIN-J or its diagram representation, we have to identify intermediate assertions that characterize the intermediate situations between the starting assertion p: at_ℓ_2 and the goal assertion q: at_ℓ_4. It is obvious that the first helpful step in the progress from ℓ_2 to ℓ_4 is process P_1 moving from ℓ_2 to ℓ_3. Consequently, we can safely take φ_m to be at_ℓ_2 and the helpful transition τ_m to be ℓ_2. We cannot yet determine the value of m because it depends on the number of helpful steps necessary to get from ℓ_3 to ℓ_4. We can now concentrate on showing how to get from ℓ_3 to ℓ_4.

Similar to the heuristics employed in the application of rule NWAIT (Section 3.3 of the SAFETY book), it is often useful to work backwards from the goal assertion at_ℓ_4. Consequently, we take φ_0 to be at_ℓ_4.

For the previous intermediate assertion φ_1, we look for situations that are only one helpful step away from φ_0. Clearly, the only transition that can accomplish φ_0 in one step is ℓ_3. If we choose the helpful transition τ_1 to be ℓ_3, then

local y_1, y_2: **boolean where** $y_1 = y_2 = \text{F}$
s : **integer where** $s = 1$

$$P_1 :: \left[\begin{array}{l} \ell_0: \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1: \textbf{noncritical} \\ \ell_2: (y_1,\ s) := (\text{T},\ 1) \\ \ell_3: \textbf{await } \neg y_2 \lor s \neq 1 \\ \ell_4: \textbf{critical} \\ \ell_5: y_1 := \text{F} \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} m_0: \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1: \textbf{noncritical} \\ m_2: (y_2,\ s) := (\text{T},\ 2) \\ m_3: \textbf{await } \neg y_1 \lor s \neq 2 \\ m_4: \textbf{critical} \\ m_5: y_2 := \text{F} \end{array}\right] \end{array}\right]$$

Fig. 1.13. Program MUX-PET1 (Peterson's algorithm) — version 1.

the appropriate φ_1 is the assertion characterizing all the states on which ℓ_3 is enabled. Therefore, as φ_1 we take the enabling condition of ℓ_3

$$\varphi_1: \quad at\_\ell_3 \ \land \ (\neg y_2 \ \lor \ s \neq 1).$$

Assertion φ_1 does not yet cover all the accessible states satsifying at_ℓ_3. Consequently, we cannot take m to be 2, and must search for additional assertions, characterizing states that satisfy at_ℓ_3 and that are one helpful step away from φ_1. Therefore, we look for transitions of P_2 that may change the disjunction $\neg y_2 \lor s = 2$ from F to T. The only candidate transition is m_5, which sets y_2 to F. Consequently, we take

$$\varphi_2: \quad at\_\ell_3 \ \land \ at\_m_5 \ \land \ y_2 \ \land \ s = 1.$$

The conjunct $y_2 \land s = 1$ can be safely added to φ_2 since all the states satisfying $at\_\ell_3 \land (\neg y_2 \lor s \neq 1)$ are already covered by φ_1.

Looking for $(at\_\ell_3 \land y_2 \land s = 1)$-states that are one helpful step away from φ_2, we easily identify

$$\varphi_3: \quad at\_\ell_3 \ \land \ at\_m_4 \ \land \ y_2 \ \land \ s = 1.$$

In a similar way, we can identify the preceding assertion as

$$\varphi_4: \quad at\_\ell_3 \ \wedge \ at\_m_3 \ \wedge \ y_2 \ \wedge \ s=1.$$

Note that m_3, which is the helpful transition for φ_4, is enabled on all states satisfying φ_4.

At this point, we find out that the disjunction $\varphi_1 \vee \cdots \vee \varphi_4$ covers the range of all accessible (at_ℓ_3)-states. This is because P_2 must be in one of the locations $m_0, \ldots, m_4$. Due to ψ_2, the range $m_{0..2}$ is covered by φ_1 under the $\neg y_2$ disjunct. Locations $m_3 \ldots, m_5$ for $s \neq 1$ are covered by φ_1 under the disjunct $s \neq 1$, while the same locations for the case that $s = 1$ are covered by assertions $\varphi_4, \varphi_3, \varphi_2$, respectively.

In Fig. 1.14 we present a CHAIN diagram using the intermediate assertions constructed through the preceding analysis.

Note that we have grouped under φ_1 many possible states of P_2, and have not represented the movement of P_2 through them. This is justified by the fact that ℓ_3 is enabled on all of these states and is the transition declared as helpful for φ_1. In contrast, we separated m_3, m_4, and m_5, because the helpful transitions there changed from one of these states to the next. ▟

In **Problem 1.1**, the reader is requested to establish accessibility for another algorithm for mutual exclusion.

Example (Peterson's Algorithm — Version 2)

Consider the refined program MUX-PET2, version 2 of Peterson's algorithm, presented in Fig. 1.15.

In Section 1.4 of the SAFETY book, we established the following invariants for this program:

$$\chi_0: \quad s=1 \ \vee \ s=2$$

$$\chi_1: \quad y_1 \ \leftrightarrow \ at\_\ell_{3..6}$$

$$\chi_2: \quad y_2 \ \leftrightarrow \ at\_m_{3..6}.$$

We intend to verify the property of accessibility for program MUX-PET2, which can be expressed by the response formula

$$\underbrace{at\_\ell_2}_{p} \quad \Rightarrow \quad \Diamond\, \underbrace{at\_\ell_5}_{q} \ .$$

The construction of the appropriate verification diagram starts in a similar way to the diagram for program MUX-PET1 of the previous example. We take φ_m to be at_ℓ_2. From ℓ_2, process P_1 can proceed at its own pace to ℓ_3, which we take as φ_{m-1}. The next step taken by P_1 leads into ℓ_4 where a more detailed analysis is necessary.

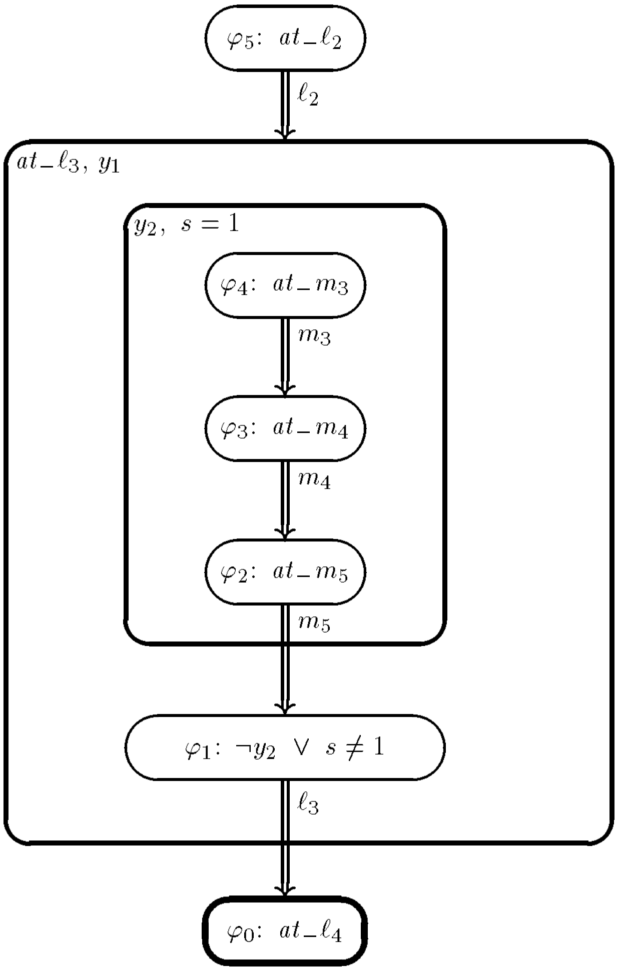


Fig. 1.14. CHAIN diagram for program MUX-PET1.

To perform this detailed analysis we take φ_0 to be the goal assertion at_ℓ_5. What should we take as φ_1? In the preceding case, we characterized φ_1 as being one helpful step away from φ_0. This characterization is not sufficient here. Another requirement is that if s' is a successor of a φ_1-state, then s' should satisfy either φ_1 or φ_0: at_ℓ_5. This shows that we cannot take φ_1 to be, as before, the assertion $at\_\ell_4 \wedge (\neg y_2 \vee s \neq 1)$. This is because the accessible state

$$s: \quad \big\langle \pi\colon \{\ell_4, m_2\},\ y_1\colon \mathrm{T},\ y_2\colon \mathrm{F},\ s\colon 1 \big\rangle$$

satisfies the candidate assertion $at\_\ell_4 \wedge (\neg y_2 \vee s \neq 1)$ but has an m_2-successor

$$\begin{array}{l} \textbf{local}\ y_1,\ y_2\colon \textbf{boolean where}\ y_1 = y_2 = \text{F} \\ \qquad\quad s \qquad\ \colon \textbf{integer where}\ s = 1 \end{array}$$

$$P_1 :: \left[\begin{array}{l} \ell_0\colon\ \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} \ell_1\colon\ \textbf{noncritical} \\ \ell_2\colon\ y_1 := \text{T} \\ \ell_3\colon\ s := 1 \\ \ell_4\colon \left[\begin{array}{l} \ell_4^a\colon\ \textbf{await}\ \neg y_2 \\ \quad \textbf{or} \\ \ell_4^b\colon\ \textbf{await}\ s \neq 1 \end{array}\right] \\ \ell_5\colon\ \textbf{critical} \\ \ell_6\colon\ y_1 := \text{F} \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} m_0\colon\ \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} m_1\colon\ \textbf{noncritical} \\ m_2\colon\ y_2 := \text{T} \\ m_3\colon\ s := 2 \\ m_4\colon \left[\begin{array}{l} m_4^a\colon\ \textbf{await}\ \neg y_1 \\ \quad \textbf{or} \\ m_4^b\colon\ \textbf{await}\ s \neq 2 \end{array}\right] \\ m_5\colon\ \textbf{critical} \\ m_6\colon\ y_2 := \text{F} \end{array}\right] \end{array}\right]$$

Fig. 1.15. Program MUX-PET2 (Peterson's algorithm) — version 2.

given by

$$s'\colon \quad \big\langle \pi\colon \{\ell_4, m_3\},\ y_1\colon \text{T},\ y_2\colon \text{T},\ s\colon 1 \big\rangle$$

which satisfies neither the candidate assertion nor φ_0.

We observe that the cause for this problem is the disjunct $\neg y_2$ which can be falsified (changed from T to F) by transition m_2 of P_2. There is no such problem with the disjunct $s \neq 1$ which cannot be falsified by P_2. Consequently, we take

$$\varphi_1\colon \quad at\_\ell_4\ \wedge\ s \neq 1.$$

The only transition that can lead from a $\neg\varphi_1$-state to a φ_1-state is m_3. Therefore, we take φ_2 to be

$$\varphi_2: \quad at\_\ell_4 \ \land \ at\_m_3 \ \land \ s = 1.$$

We also realize that the transition leading into φ_2 is m_2 which changes y_2 from F to T and preserves the value of s. Consequently, we take φ_3 to be

$$\varphi_3: \quad at\_m_{0..2} \ \land \ \neg y_2 \ \land \ s = 1.$$

By now, we have covered all the states satisfying $at\_\ell_4 \land (\neg y_2 \lor s \neq 1)$. From now on, the analysis proceeds as it did for program MUX-PET1. The final CHAIN diagram is presented in Fig. 1.16.

This diagram partitions the range $m_{0..4}$ into three regions. The region $m_{0..2}$, represented by φ_3, guarantees the enableness of ℓ_4^a (but not the enableness of m_2 which is therefore drawn as a single edge). However it may evolve into φ_2, where no transition of P_1 is guaranteed to be enabled. Being at φ_2, m_3 is the helpful transition which eventually leads into φ_1. In φ_1, ℓ_4^b is enabled, and since P_2 cannot falsify $s \neq 1$, eventually ℓ_4^b is taken and leads to φ_0.

The edge labeled ℓ_4^a connecting node φ_1 to node φ_0 represents the possibility that ℓ_4^a may be enabled on a state satisfying $s \neq 1$. A more careful analysis shows that φ_1 in this diagram can be strenghened to the assertion

$$\widehat{\varphi}_1: \quad at\_m_4 \ \land \ s \neq 1 \ \land \ y_2,$$

and then this edge is unnecessary. ▪

In **Problem 1.2**, the reader is requested to verify accessibility for a family of mutual exclusion algorithms, known as the *bakery algorithms*.

Example (Dekker's algorithm)

Dekker's algorithm for solving the mutual exclusion problem is presented in program MUX-DEK of Fig. 1.17.

In comparison to Peterson's algorithm, Dekker's algorithm has a relatively simple safety proof but rather elaborate proof of accessibility.

- *Invariants*

In Section 1.4 of the SAFETY book we derived the following invariants for program MUX-DEK:

$$\chi_1: \quad t = 1 \ \lor \ t = 2$$

$$\chi_2: \quad y_1 \ \leftrightarrow \ (at\_\ell_{3..5,8..10})$$

$$\chi_3: \quad y_2 \ \leftrightarrow \ (at\_m_{3..5,8..10})$$

These are the invariants we needed to prove the mutual exclusion property, i.e., the invariance of

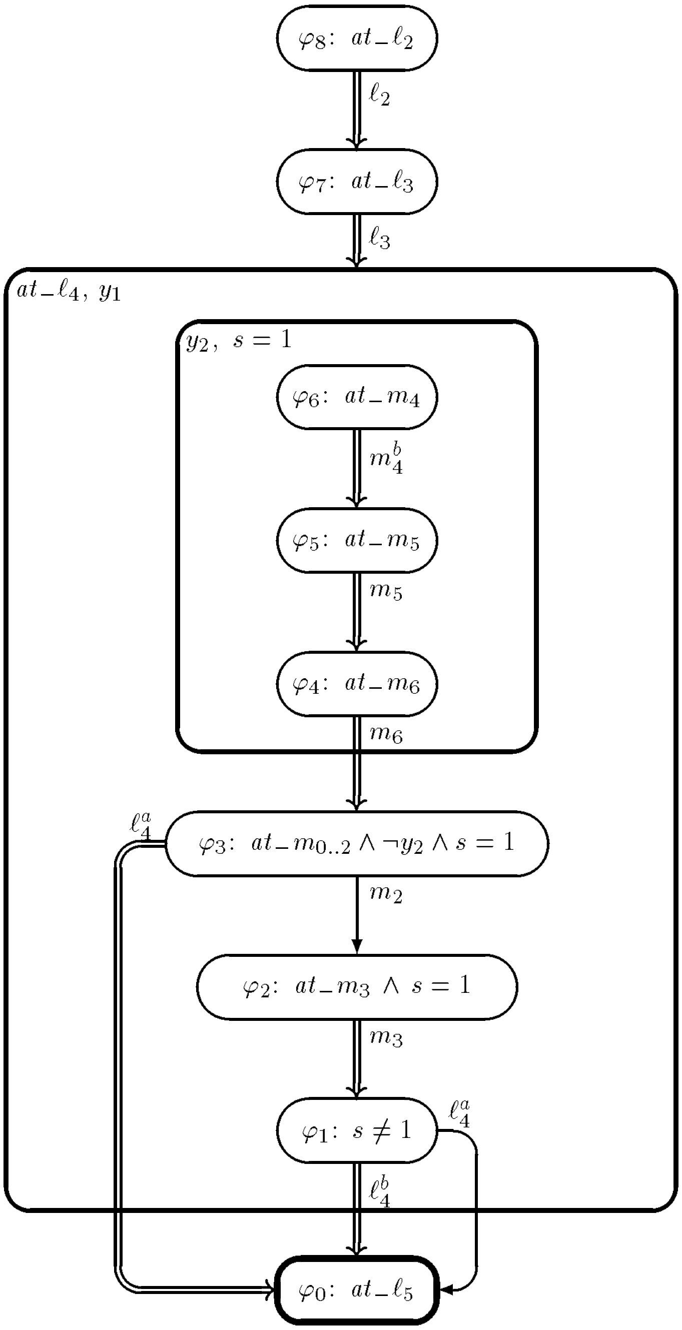


Fig. 1.16. CHAIN diagram for program MUX-PET2.

local $y_1,\ y_2$: **boolean where** $y_1 = \mathrm{F},\ y_2 = \mathrm{F}$
t : **integer where** $t = 1$

$$
P_1 :: \left[\begin{array}{l}
\ell_0\text{: }\textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
\ell_1\text{: } \textbf{noncritical} \\
\ell_2\text{: } y_1 := \mathrm{T} \\
\ell_3\text{: } \textbf{while } y_2 \textbf{ do} \\
\qquad \ell_4\text{: } \textbf{if } t = 2 \\
\qquad\qquad \textbf{then } \left[\begin{array}{l}
\ell_5\text{: } y_1 := \mathrm{F} \\
\ell_6\text{: } \textbf{await } t = 1 \\
\ell_7\text{: } y_1 := \mathrm{T}
\end{array}\right] \\
\ell_8\text{: } \textbf{critical} \\
\ell_9\text{: } t := 2 \\
\ell_{10}\text{: } y_1 := \mathrm{F}
\end{array}\right]
\end{array}\right]
$$

$$\|$$

$$
P_2 :: \left[\begin{array}{l}
m_0\text{: }\textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
m_1\text{: } \textbf{noncritical} \\
m_2\text{: } y_2 := \mathrm{T} \\
m_3\text{: } \textbf{while } y_1 \textbf{ do} \\
\qquad m_4\text{: } \textbf{if } t = 1 \\
\qquad\qquad \textbf{then } \left[\begin{array}{l}
m_5\text{: } y_2 := \mathrm{F} \\
m_6\text{: } \textbf{await } t = 2 \\
m_7\text{: } y_2 := \mathrm{T}
\end{array}\right] \\
m_8\text{: } \textbf{critical} \\
m_9\text{: } t := 1 \\
m_{10}\text{: } y_2 := \mathrm{F}
\end{array}\right]
\end{array}\right]
$$

Fig. 1.17. Program MUX-DEK (Dekker's algorithm for mutual exclusion).

$$\chi_4\text{:}\quad \neg(at\_\ell_{8..10}\ \land\ at\_m_{8..10}).$$

As we will see, additional invariants are needed for the support of the response property. We will develop them as they are needed.

- *Response*

The main response property of this algorithm is, of course, that of accessibility. It is stated by

$$\psi: \quad at\_\ell_2 \;\Rightarrow\; \diamondsuit\, at\_\ell_8.$$

We partition the proof of the accessibility property into three lemmas, proving respectively.

Lemma A $\quad at\_\ell_2 \;\Rightarrow\; \diamondsuit\Big((at\_\ell_4 \,\wedge\, t=2) \,\vee\, (at\_\ell_{3..7} \,\wedge\, t=1) \,\vee\, at\_\ell_8\Big)$

Lemma B $\quad at\_\ell_4 \,\wedge\, t=2 \;\Rightarrow\; \diamondsuit(at\_\ell_{3..7} \,\wedge\, t=1)$

Lemma C $\quad at\_\ell_{3..7} \,\wedge\, t=1 \;\Rightarrow\; \diamondsuit\, at\_\ell_8.$

Obviously, the difficult part of the protocol is the loop at $\ell_{3..7}$. Being within this loop, P_1 is considered to have a higher priority when $t=1$. Lemma A claims that if P_1 is just starting its journey towards the critical section, then it will either reach ℓ_4 with a lower priority, or get to $\ell_{3..7}$ with a higher priority, or reach ℓ_8. Lemma B claims that if P_1 is at ℓ_4 with a low priority it will stay within the loop and eventually gain a high priority. Lemma C shows that if P_1 is within this loop and has a higher priority, then it will eventually get to ℓ_8.

Clearly, by combining these three response properties, using the transitivity of response rule TRNS-R we obtain the required accessibility property.

- *Proof of Lemma A*

The proof of the response property

$$at\_\ell_2 \;\Rightarrow\; \diamondsuit\Big((at\_\ell_4 \,\wedge\, t=2) \,\vee\, (at\_\ell_{3..7} \,\wedge\, t=1) \,\vee\, at\_\ell_8\Big)$$

is presented in the CHAIN diagram of Fig. 1.18.

It is easy to follow P_1 from ℓ_2 to ℓ_3. If $t=1$ on entry to ℓ_3, then we are already at the goal $at\_\ell_{3..7} \wedge t=1$. Otherwise, we enter ℓ_3 with $t=2$, setting y_1 to T. Here we examine the possible locations of P_2. Assertions φ_1, φ_2, and φ_3 cover all the possibilities. The possible motions of P_2 within these three assertions consist of taking m_9, setting t to 1, which raises the priority of P_1 and attains the goal $at\_\ell_{3..7} \wedge t=1$. The other possible movements are from φ_3 to φ_2, and then to φ_1. Being at $m_{3,4}$ with $y_1 = $ T and $t=2$, P_2 cannot move elsewhere. Transition mode ℓ_3^{T} is enabled on φ_1 and φ_3 states, while mode ℓ_3^{F} is enabled on φ_2. Both are helpful since they lead to $at\_\ell_4 \wedge t=2$ and at_ℓ_8, respectively.

- *Proof of Lemma B*

The proof of the response property

$$at\_\ell_4 \,\wedge\, t=2 \;\Rightarrow\; \diamondsuit(at\_\ell_{3..7} \,\wedge\, t=1)$$

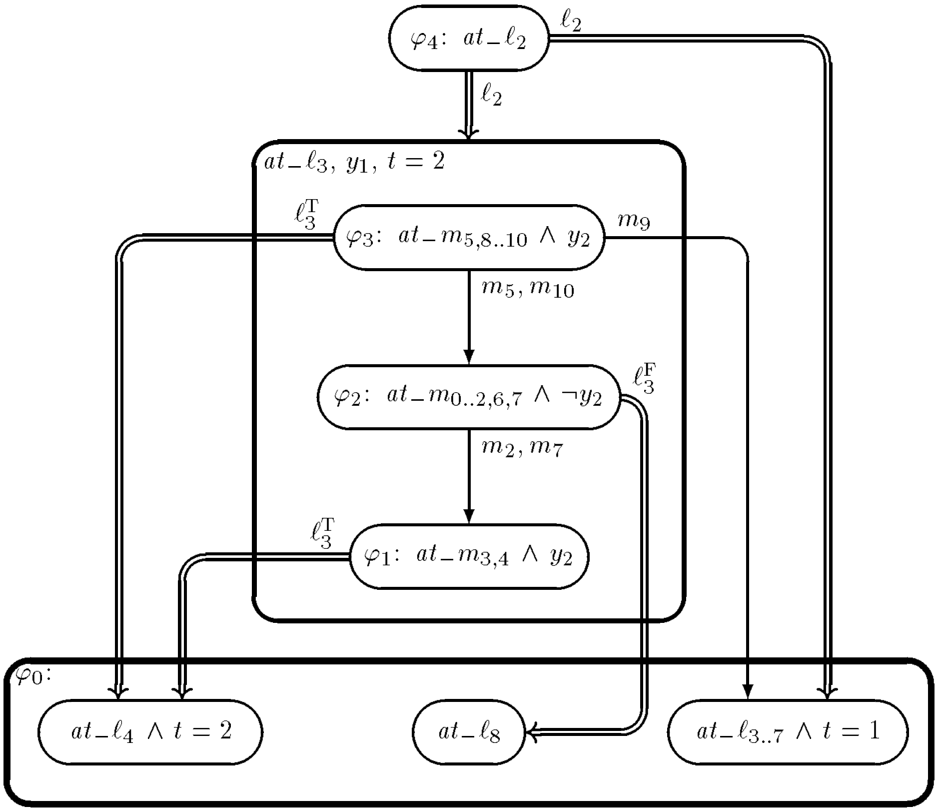


Fig. 1.18. CHAIN diagram for Lemma A.

is presented in the CHAIN diagram of Fig. 1.19.

From ℓ_4, P_1 proceeds to ℓ_5 since $t = 2$, and then to ℓ_6 while resetting y_1 to F. While being at $\ell_{4,5}$, P_2 may still set t to 1 by performing m_9, which leads to the goal $at\_\ell_{3..7} \wedge t = 1$.

However, once P_1 enters ℓ_6, it stays at ℓ_6 waiting for t to change to 1. At that point we have to inspect where P_2 may currently be. We consider as possible locations of P_2 all of m_2–m_9, tracing their possible flow under the relatively stable situation of $t = 2$, $y_1 =$ F. We see that all transitions are enabled and lead to m_9 which eventually sets t to 1 as required.

A tacit assumption made in this diagram is the exclusion of m_{10}, m_0, and m_1, as possible locations while P_1 is at ℓ_6 with $y_1 =$ F and $t = 2$. This assumption must hold for the program, if we believe Lemma B to be valid. Indeed, consider the situation that P_1 is waiting at ℓ_6 with $y_1 =$ F and $t = 2$, while P_2 is at m_1.

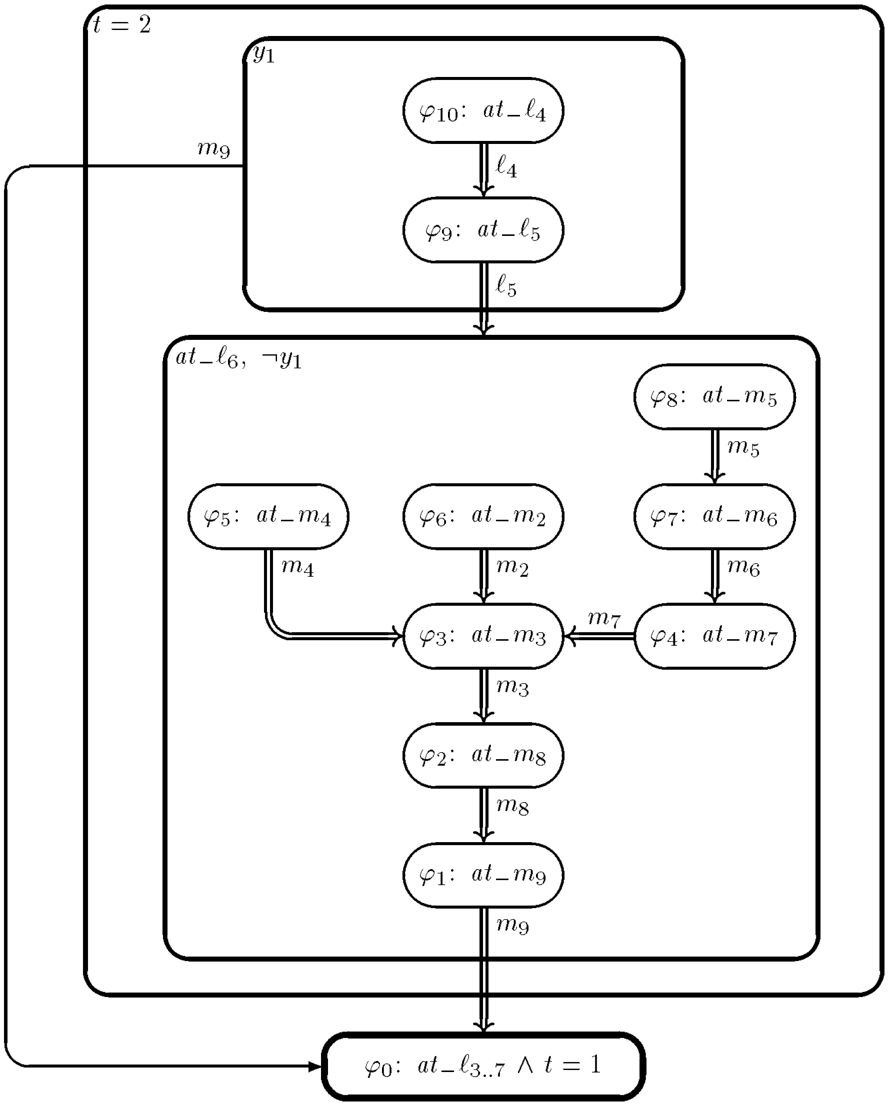


Fig. 1.19. CHAIN diagram for Lemma B.

Since P_2 is allowed to stay at the noncritical section forever, this would lead to a deadlock, denying accessibility from P_1.

We must therefore conclude that if the algorithm is correct, and guarantees accessibility to both processes, then the following assertion must be invariant

$$at\_\ell_6 \ \wedge\ t = 2 \ \rightarrow\ at\_m_{2..9}.$$

This invariance follows from the stronger invariant

$$\chi_5: \quad at\_\ell_{4..6} \ \wedge\ t = 2 \ \rightarrow\ at\_m_{2..9}$$

which we will prove.

By symmetry one can also require the invariance of

$$\chi_6: \quad at\_m_{4..6} \ \wedge\ t = 1 \ \rightarrow\ at\_\ell_{2..9}.$$

■ *Proof of invariant* χ_5 ⟪ Exercise? ⟫

Clearly,

$$\underbrace{\cdots \wedge\ at\_\ell_0 \ \wedge\ \cdots}_{\Theta} \ \rightarrow\ \underbrace{at\_\ell_{4..6} \ \wedge\ t = 2 \ \rightarrow\ at\_m_{2..9}}_{\chi_5}$$

holds.

Let us check the verification conditions for assertion χ_5, which are of the form

$$\underbrace{at\_\ell_0 \ \wedge\ at\_m_0 \ \wedge\ \neg y_1 \ \wedge\ \neg y_2 \ \wedge\ t = 1}_{\Theta} \ \wedge\ \underbrace{at\_\ell_{4..6} \ \wedge\ t = 2 \ \rightarrow\ at\_m_{2..9}}_{\chi_5} \ \rightarrow$$
$$\underbrace{at'\_\ell_{4..6} \ \wedge\ t' = 2 \ \rightarrow\ at'\_m_{2..9}}_{\chi_5'} \ .$$

There are three transitions that may potentially falsify assertion χ_5.

- Transition m_9

  Sets t to 1 which makes $t' = 2$ false and hence preserves the assertion.

- Transition ℓ_9

  Leads to at_ℓ_{10} which makes $at'\_\ell_{4..6}$ false.

- ℓ_3^{T} while $t = 2$

  This is possible only if $y_2 = \mathrm{T}$ which, by χ_3, implies $at\_m_{3..5} \vee at\_m_{8..10}$, and therefore $at\_m_{2..10}$. This almost gives us $at\_m_{2..9}$, with the exception of m_{10}. We thus need additional information that will exclude the possibility of P_2 being at m_{10} while $t = 2$.

Clearly, while entering m_{10} from m_9, P_2 sets t to 1. Can P_1 change it back to 2, while P_2 is still at m_{10}? The answer is no, because m_{10}, as we see in χ_4, is still a part of the critical section and is therefore exclusive of ℓ_9, the only statement capable of changing t to 2.

This suggests the invariant

$$\chi_7: \quad at\_m_{10} \ \rightarrow\ t = 1$$

and its symmetric counterpart

$$\chi_8: \quad at\_\ell_{10} \rightarrow t = 2.$$

To prove χ_7, we should inspect two transitions:

- Transition m_9

  Sets t to 1.

- ℓ_9 while at_m_{10}

  Impossible due to χ_4.

This establishes χ_7 and similarly χ_8. Having χ_7 we can use it to show that the last transition considered in the proof of χ_5, namely ℓ_3^{T} while $t = 2$, implies $at\_m_{2..9}$, which establishes χ_5.

- *Proof of Lemma C*

Lemma C states that if P_1 is within the waiting loop $\ell_{3..7}$ with higher priority, i.e., $t = 1$, then eventually it will reach ℓ_8. It is stated by

$$at\_\ell_{3..7} \wedge t = 1 \Rightarrow \Diamond\, at\_\ell_8.$$

The proof is presented in the CHAIN diagram of Fig. 1.20.

Note our efforts to minimize the number of assertions by grouping together situations with different control configurations, wherever possible. Thus for all the states where $y_1 = \mathrm{T}$ and P_1 is either at ℓ_3 or at ℓ_4, we do not distinguish between these two possibilities, but partition the diagram according to the location of P_2. This is because, in this general situation, it is P_2 which is the helpful process and we have to trace its progress. On the other hand, when $y_2 = \mathrm{F}$, P_1 becomes the helpful process and we start distinguishing between the cases of at_ℓ_3 and at_ℓ_4, while lumping together the locations of P_2 into two groups: $m_{0..2,7}$ and m_6. These two groups must be distinguished because it is possible (though not guaranteed) to exit the first group into a situation where $y_2 = \mathrm{T}$, but it is impossible to exit m_6 into such a situation. This is because when P_2 is at m_6 with $t = 1$, it cannot progress until t is changed to 2. ▟

In **Problem 1.3**, the reader is requested to prove accessibility for two variants of Dekker's algorithm.

Case Splitting According to the Helpful Transitions

In the preceding examples, the main reason for using rule CHAIN-J with $m > 1$ intermediate assertions has been that the program requires m helpful steps to reach the goal. In most of these applications there always was a worst case computation that actually visited each of the assertions, starting with φ_m and proceeding through $\varphi_{m-1}, \varphi_{m-2}, \ldots$ up to $\varphi_0 = q$.

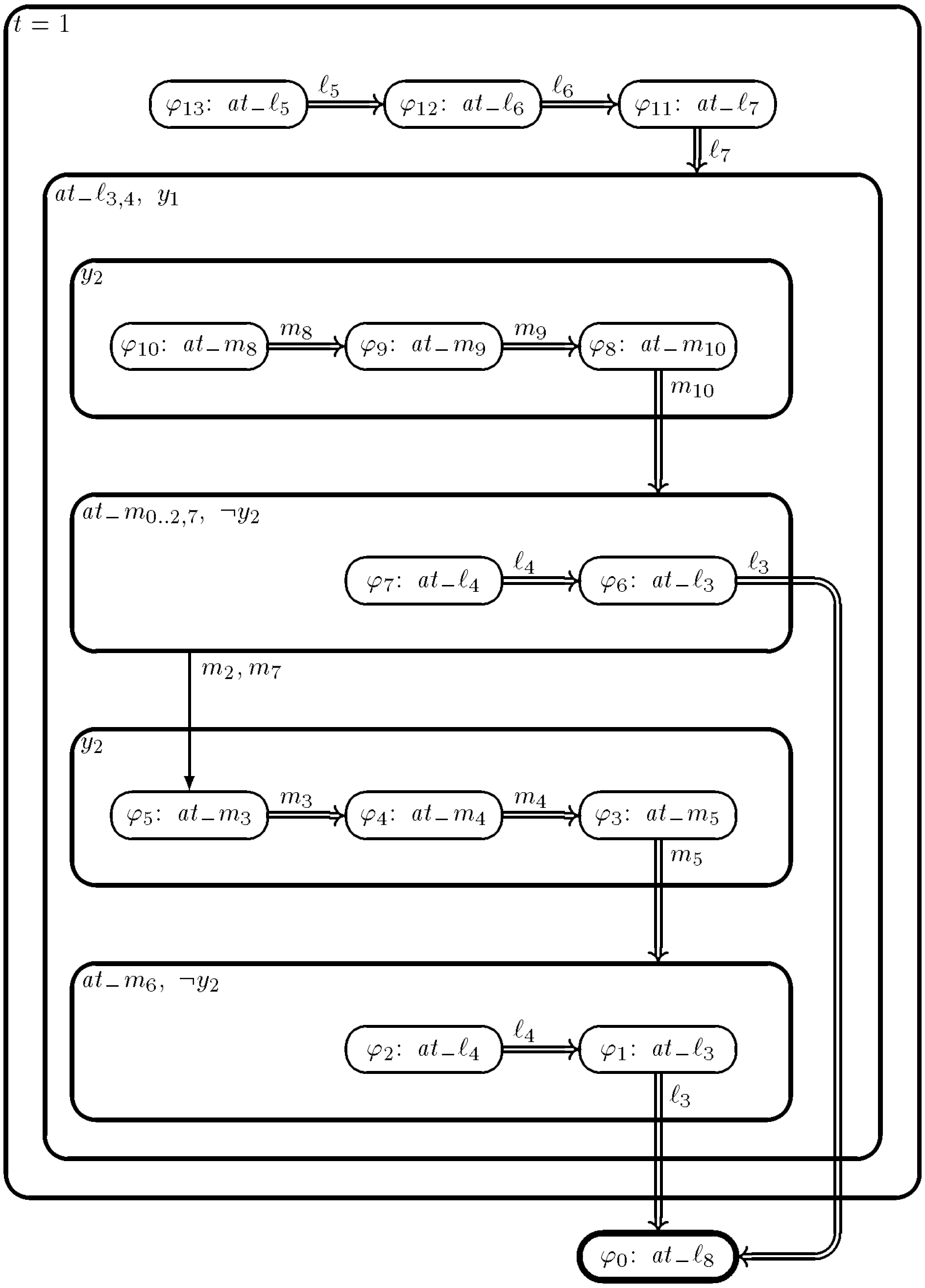


Fig. 1.20. CHAIN diagram for Lemma C.

This is not the only motivation for using several intermediate assertions. Another good reason for wishing to partition the state space lying between p and q into several assertions is that different states in that space may require different helpful transitions for getting them closer to the goal.

Example (maximum)

Consider, for example, program MAX presented in Fig. 1.21. This program places in output variable z the maximum of inputs x and y. The program consists of two parallel statements that compare the values of x and y.

$$\begin{array}{l} \textbf{in} \quad x,\ y\text{: } \textbf{integer} \\ \textbf{out}\ z \quad\ \text{: } \textbf{integer} \end{array}$$

$$\begin{bmatrix} \ell_0\text{: } \textbf{if } x \geq y \textbf{ then} \\ \qquad \ell_1\text{: } z := x \\ \ell_2\text{:} \end{bmatrix} \quad || \quad \begin{bmatrix} m_0\text{: } \textbf{if } x \leq y \textbf{ then} \\ \qquad m_1\text{: } z := y \\ m_2\text{:} \end{bmatrix}$$

Fig. 1.21. Program MAX (maximum).

The response statement we would like to prove for this program is

$$\underbrace{at\_\ell_0 \ \wedge \ at\_m_0}_{p} \quad \Rightarrow \quad \Diamond\, \underbrace{maximal(z,\ x,\ y)}_{q}\ ,$$

where $maximal(z, x, y)$ stands for the formula

$$maximal(z,\ x,\ y)\text{:} \quad (z = x \ \vee \ z = y) \ \wedge \ z \geq x \ \wedge \ z \geq y,$$

claiming that z is the maximum of x and y.

Clearly, the goal of this response property is achieved in the helpful steps which are either ℓ_0 and ℓ_1 or m_0 and m_1. Perhaps one would expect a proof of this property by rule RESP-J that only uses one intermediate assertion φ.

However, no such proof exists. The reason for this is that we cannot identify a *single* transition that is helpful for all the states satisfying p: $at\_\ell_0 \wedge at\_m_0$. Clearly, for all states satisfying $p \wedge x \geq y$, ℓ_0 is the helpful transition, while for states satisfying $p \wedge x \leq y$, m_0 is the helpful transition. Consequently, we need at least *four* intermediate assertions in a proof of this property by rule CHAIN-J.

We choose the following assertions and helpful transitions

$$\varphi_4\text{:} \quad at\_m_0 \ \wedge \ x \leq y \qquad\qquad \tau_4\text{:} \quad m_0$$

$$\varphi_3\text{:} \quad at\_\ell_0 \ \wedge \ x \geq y \qquad\qquad \tau_3\text{:} \quad \ell_0$$

$$\begin{array}{llll} \varphi_2: & at\_m_1 \ \wedge \ x \leq y & \tau_2: & m_1 \\ \varphi_1: & at\_\ell_1 \ \wedge \ x \geq y & \tau_1: & \ell_1 \\ \varphi_0: & q. & & \end{array}$$

It is straightforward to verify that all the premises of rule CHAIN-J are satisfied by this choice.

In Fig. 1.22, we present a CHAIN diagram for the proof of the considered response property.

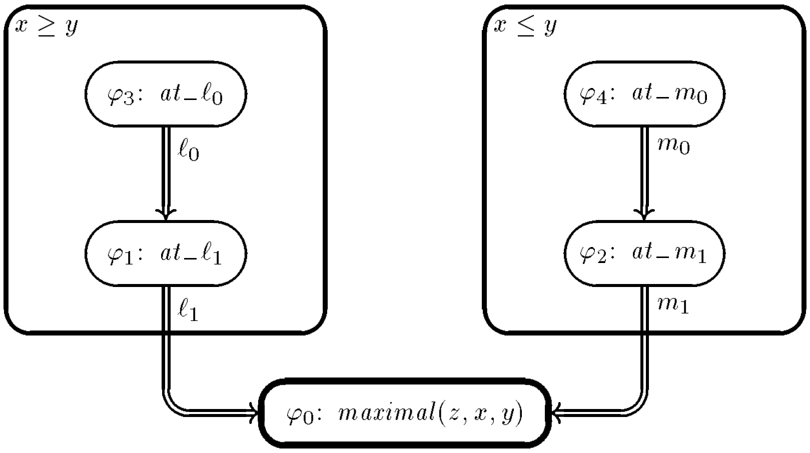


Fig. 1.22. CHAIN diagram for program MAX.

Assertions φ_3 and φ_4 partition (non-exclusively) the situation $at\_\ell_0 \wedge at\_m_0$ into states for which ℓ_0 is helpful and has not been taken yet, and states for which m_0 is helpful and has not been taken yet.

It is not difficult to verify that taking ℓ_0 from a φ_3-state, as well as taking m_0 from a φ_4-state, leads to φ_1 and φ_2, respectively. Choosing φ_4 to rank above φ_3 is quite arbitrary. In particular, we do not have a computation that goes from φ_4-states to φ_3-states. Every computation follows either the φ_3, φ_1 route or the φ_4, φ_2 route. ▪

1.4 Well-Founded Rule

In rule CHAIN-J we treated each of the participating assertions $\varphi_0, \ldots, \varphi_m$ as separate entities, and made no attempt to find a uniform representation for φ_j as a single formula involving j. This approach is adequate for response properties which require a *bounded* number of steps for their achievement, e.g., at most five in the case of program MUX-PET1 (Fig. 1.14). The bound must be uniform and independent of the initial state.

There are many cases, however, in which no such bound can be given a priori. To deal with these cases, we must generalize the induction over a fixed finite subrange of the integers, such as $0, 1, \ldots, m$ in rule CHAIN-J, into an explicit induction over an arbitrary well-founded relation.

Well-Founded Domains

We define a *well-founded domain* $(\mathcal{A}, \succ)$ to consist of a set $\mathcal{A}$ and a *well-founded order* relation $\succ$ on $\mathcal{A}$. A binary relation $\succ$ is called an *order* if it is

- transitive: $a \succ b$ and $b \succ c$ imply $a \succ c$, and
- irreflexive: $a \succ a$ for no $a \in \mathcal{A}$.

The relation $\succ$ is called *well-founded* if there does not exist an infinitely descending sequence $a_0, a_1, \ldots$ of elements of $\mathcal{A}$ such that

$$a_0 \succ a_1 \succ \cdots .$$

A typical example of a well-founded domain is $(\mathbb{N}, >)$, where $\mathbb{N}$ are the natural numbers (including 0) and $>$ is the greater-than relation. Clearly, $>$ is well-founded over the natural numbers, because there cannot exist an infinitely descending sequence of natural numbers

$$n_0 > n_1 > n_2 > \ldots .$$

For an arbitrary order relation $\succ$ on $\mathcal{A}$, we define its *reflexive extension* $\succeq$ to hold between a, $a' \in \mathcal{A}$, written $a \succeq a'$, if either $a \succ a'$ or $a = a'$.

The Lexicographic Product

Given two well-founded domains, $(\mathcal{A}_1, \succ_1)$ and $(\mathcal{A}_2, \succ_2)$, we can form their *lexicographical product* $(\mathcal{A}, \succ)$, where

$\mathcal{A}$ is defined as $\mathcal{A}_1 \times \mathcal{A}_2$, i.e., the set of all pairs (a_1, a_2), such that $a_1 \in \mathcal{A}_1$ and $a_2 \in \mathcal{A}_2$.

$\succ$ is an order defined for $(a_1, a_2), (b_1, b_2) \in \mathcal{A}$ by

$$(a_1, a_2) \succ (b_1, b_2) \quad \text{iff} \quad a_1 \succ_1 b_1 \quad \text{or} \quad a_1 = b_1 \ \land \ a_2 \succ_2 b_2.$$

Thus, in comparing the two pairs (a_1, a_2) and (b_1, b_2), we first compare a_1 against b_1. If $a_1 \succ_1 b_1$, then this determines the relation between the pairs to be $(a_1, a_2) \succ (b_1, b_2)$. If $a_1 = b_1$, we compare a_2 with b_2, and the result of this comparison determines the relation between the pairs.

The order $\succ$ is called *lexicographic*, which implies that, as when searching in a dictionary, we locate the position of a word by checking the first letter first and only after locating the place where the first letter matches, do we continue matching the subsequent letters.

The importance of the lexicographic product follows from the following claim:

Claim (lexicographic product)

If the domains $(\mathcal{A}_1, \succ_1)$ and $(\mathcal{A}_2, \succ_2)$ are well-founded, then so is their lexicographic product $(\mathcal{A}, \succ)$.

Clearly, by the above, the domain $(\mathbb{N}^2, \succ)$, where $\succ$ is the lexicographic order between pairs of natural numbers, is well-founded. This order is defined by

$$(n_1, n_2) \ \succ \ (m_1, m_2) \qquad \text{iff} \qquad n_1 > m_1 \quad \text{or} \quad n_1 = m_1 \ \land \ n_2 > m_2.$$

According to this definition

$$(10, 20) \ \succ \ (5, 15) \qquad (1, 0) \ \succ \ (0, 100) \qquad (1, 5) \ \succ \ (1, 3).$$

New well-founded domains can be constructed by taking lexicographic products of more than two well-founded domains. Applying this construction to the domain $(\mathbb{N}, >)$ of natural numbers, we obtain the domain $(\mathbb{N}^k, \succ)$, for $k \geq 2$, where $\succ$ is the lexicographic order between k-tuples of natural numbers. The order $\succ$ is defined by

$$(n_1, \ldots, n_k) \ \succ \ (m_1, \ldots, m_k) \qquad \text{iff} \qquad n_1 = m_1, \ \ldots, \ n_{i-1} = m_{i-1}, \ n_i > m_i$$
$$\text{for some } i, \ 1 \leq i \leq k.$$

For example, for $k = 3$

$$(7, \ 2, \ 1) \ \succ \ (7, \ 0, \ 45).$$

It is easy to show that the domain $(\mathbb{N}^k, \succ)$ is well-founded.

The Rule

Let $(\mathcal{A}, \succ)$ be a well-founded domain. As in rule CHAIN-J, we use several intermediate assertions $\varphi_1, \ldots, \varphi_m$ to describe the evolution from p to $q = \varphi_0$. Rule CHAIN-J uses the index of the assertion as a measure of the distance from the goal q. The rule presented here associates an explicit *ranking function* δ_i with each assertion φ_i, $i = 0, \ldots, k$. The function δ_i maps states into the set $\mathcal{A}$ and is intended to measure the distance of the current state to a state satisfying the goal q.

We refer to the value of δ_i in a φ_i-state as a *rank* of the state. The well-founded rule WELL for response properties is given in Fig. 1.23. Premise W1 states that every p-position satisfies one of $\varphi_0, \ldots, \varphi_m$. Premise W2 states that every φ_i-position with positive i and rank u is eventually followed by a position which satisfies some φ_j, with a rank lower than u.

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

$$\text{W1.} \quad p \;\rightarrow\; \bigvee_{i=0}^{m} \varphi_i$$

$$\text{W2.} \quad \varphi_i \;\wedge\; \delta_i = u \;\;\Rightarrow\;\; \diamondsuit\left(\bigvee_{j=0}^{m} \left(\varphi_j \;\wedge\; u \succ \delta_j\right)\right) \quad \text{for } i = 1, \ldots, m$$

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

Fig. 1.23. Rule WELL (well-founded response).

Justification It is straightforward to justify rule WELL. Consider a computation σ that satisfies premises W1, W2, and let t_1 be a position in σ which satisfies p. By W1, some φ_i is satisfied at t_1. If it is $\varphi_0 = q$, we are done. Otherwise, let φ_{i_1}, $i_1 > 0$, be the assertion holding at t_1 and let u_1 denote the rank of the state at position t_1. By W2, there exists a position t_2, $t_2 \geq t_1$, such that some φ_j holds at t_2 with a rank $u_2 \in \mathcal{A}$, such that $u_1 \succ u_2$. If $j = 0$, we are done. Otherwise, we proceed to locate a position $t_3 \geq t_2$.

In this way we construct a sequence of positions

$$t_1 \;\leq\; t_2 \;\leq\; t_3 \;\leq\; \cdots,$$

and a corresponding sequence of elements from $\mathcal{A}$ (ranks)

$$u_1 \;\succ\; u_2 \;\succ\; u_3 \;\succ\; \cdots,$$

such that either the sequence is of length k and $q = \varphi_0$ holds at the position t_k, or the sequence is infinite and some φ_j, $j > 0$, holds at each t_i with rank $\delta_j = u_i$ there. The later case is impossible since that would lead to an infinitely descending sequence of elements of $\mathcal{A}$, in contrast to the well-foundedness of $\succ$ over $\mathcal{A}$. It follows that for some $t_k \geq t_1$, $q = \varphi_0$ holds at t_k, which shows that $\diamondsuit\, q$ holds at t_1. ▟

Example (factorial)

Consider program FACT of Fig. 1.24. This program computes in z the factorial of a nonnegative integer x. We wish to prove for this program the response property

$$\underbrace{at\_\ell_0 \ \land \ x \geq 0 \ \land \ y = x \ \land \ z = 1}_{p} \quad \Rightarrow \quad \diamondsuit \, \underbrace{at\_\ell_2 \ \land \ z = x!}_{q = \varphi_0} \ .$$

$$\begin{array}{l}
\textbf{in} \quad\ \ x\colon \ \textbf{integer where } x \geq 0 \\
\textbf{local } y\colon \ \textbf{integer where } y = x \\
\textbf{out} \quad z\colon \ \textbf{integer where } z = 1 \\
\\
\left[\begin{array}{l}
\ell_0\colon \ \textbf{while } y > 0 \textbf{ do} \\
\qquad \ell_1\colon \ (y, \ z) := (y - 1, \ z \cdot y) \\
\ell_2\colon
\end{array}\right]
\end{array}$$

Fig. 1.24. Program FACT (factorial).

Intending to use rule WELL with $m = 1$, it only remains to choose the assertion φ_1, and the ranking functions δ_0 and δ_1. This necessitates the identification of a well-founded domain $(\mathcal{A}, \succ)$, where $\mathcal{A}$ serves as the range of δ_i. Obviously, φ_1 should describe the intermediate stage in the process of getting from p to q, and δ_1 should measure the distance of this intermediate stage from the goal $q = \varphi_0$. Premise W2 ensures that steps in the computation always bring us closer to the goal.

For program FACT, a good measure of the distance from termination is the value of y. This is because when we are at ℓ_0, there are y more iterations of the *while* loop before the program terminates. We therefore choose $(\mathbb{N}, >)$ as our well-founded domain and $|y|$ as the ranking function. Thus,

$$(\mathcal{A}, \ \succ) \ = \ (\mathbb{N}, \ >), \qquad \delta_0\colon 0, \qquad \text{and} \qquad \delta_1\colon \ |y| + 1.$$

The choice of $\delta_0 = 0$ is natural because, being at a φ_0-state, we are already at the goal, and the distance to the goal can therefore be taken as 0.

The intermediate assertion φ_1 should represent the progress the computation has made, so that when $y = 0$, we can infer that $z = x!$. Clearly, the way the program operates is that it accumulates in z the product of the terms $x \cdot (x-1) \cdots$. In an intermediate stage, z contains the product $x \cdot (x - 1) \cdots (y + 1)$, which can also be expressed as $x!/y!$, provided $0 \leq y \leq x$.

We thus arrive at the intermediate assertion

$$\varphi\colon \quad at\_\ell_0 \ \land \ 0 \leq y \leq x \ \land \ z = x!/y! \ .$$

It only remains to show that premises W1,W2 are satisfied by these choices.

- Premise W1

This premise requires

$$\underbrace{at\_\ell_0 \ \wedge\ x \geq 0 \ \wedge\ y = x \ \wedge\ z = 1}_{p} \quad \rightarrow$$

$$\underbrace{\cdots}_{\varphi_0} \ \vee\ \underbrace{at\_\ell_0 \ \wedge\ 0 \leq y \leq x \ \wedge\ z = x!/y!}_{\varphi_1}.$$

This implication is obviously valid.

- Premise W2

This premise requires showing

$$\underbrace{at\_\ell_0 \ \wedge\ 0 \leq y \leq x \ \wedge\ z = x!/y!}_{\varphi_1} \ \wedge\ |y| + 1 = n \quad \Rightarrow$$

$$\diamondsuit \left(\begin{array}{l} \underbrace{at\_\ell_2 \ \wedge\ z = x!}_{\varphi_0} \ \wedge\ n > \underbrace{0}_{\delta_0} \\ \qquad\quad \vee \\ \underbrace{at\_\ell_0 \ \wedge\ 0 \leq y \leq x \ \wedge\ z = x!/y!}_{\varphi_1} \ \wedge\ n > \underbrace{|y| + 1}_{\delta_1} \end{array} \right).$$

Since $\varphi_1 \wedge |y| + 1 = n$ implies that $n > 0$, it is sufficient to prove this implication for every $n > 0$. As φ_1 implies $y \geq 0$, we may replace $|y|$ by y.

Case $n = 1$:

For this value of n, we prove

$$\underbrace{at\_\ell_0 \ \wedge\ 0 \leq y \leq x \ \wedge\ z = x!/y!}_{\varphi_1} \ \wedge\ y + 1 = 1 \quad \Rightarrow$$

$$\diamondsuit \Big(\underbrace{at\_\ell_2 \ \wedge\ z = x!}_{\varphi_0} \ \wedge\ 1 > 0 \Big),$$

which simplifies to

$$at\_\ell_0 \ \wedge\ z = x! \ \wedge\ y = 0 \quad \Rightarrow \quad \diamondsuit(at\_\ell_2 \ \wedge\ z = x!).$$

This, of course, can be proven by a single application of rule RESP-J, observing that, under the situation described by the antecedent, only transition ℓ_0 is enabled, and taking it leads to $at\_\ell_2 \wedge z = x!$.

Case $n > 1$:

In this case, we will prove

$$\underbrace{at\_\ell_0 \ \land\ 0 \le y \le x \ \land\ z = x!/y!}_{\varphi_1} \ \land\ y+1 = n > 1 \quad \Rightarrow$$

$$\Diamond\Big(\underbrace{at\_\ell_0 \ \land\ 0 \le y \le x \ \land\ z = x!/y!}_{\varphi_1} \ \land\ n > y+1\Big) .$$

This can be proven by rule CHAIN-J using three assertions, ψ_0, ψ_1, and ψ_2.

The top assertion ψ_2 corresponds to $\varphi_1 \land y+1 = n$. Assertion ψ_1 describes the intermediate state, after passing the test of the *while* statement, and being at ℓ_1. The final assertion ψ_0 implies $\varphi_1 \land y+1 = n-1 < n$, and describes the situation after performing the assignment ℓ_1 and arriving back at ℓ_0. The verification diagram in Fig. 1.25 describes this proof.

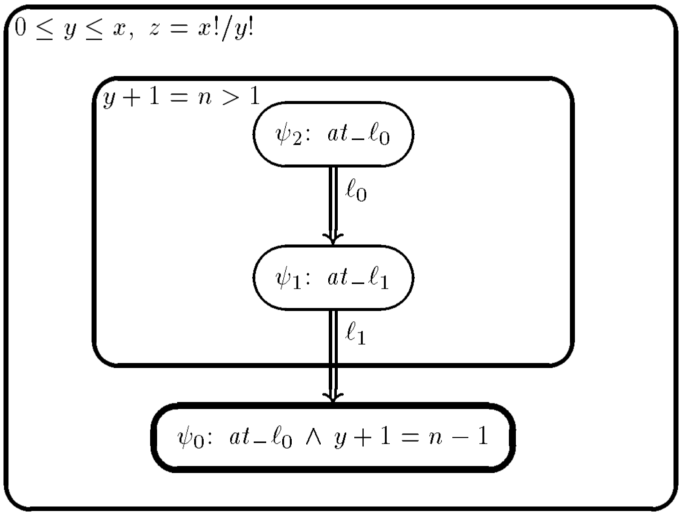


Fig. 1.25. Verification diagram for case $n > 1$.

Thus, by treating separately the cases $n = 1$ and $n > 1$, we conclude that premise W2 holds for every $n \ge 1$. This establishes that the conclusion

$$\underbrace{at\_\ell_0 \ \land\ x \ge 0 \ \land\ y = x \ \land\ z = 1}_{p} \quad \Rightarrow \quad \Diamond\, \underbrace{at\_\ell_2 \ \land\ z = x!}_{q}$$

of rule WELL is valid. ▪

A Rule with Nontemporal Premises

Rule WELL, used for proving response formulas, has as its premise W2, another response formula. This allows a recursive use of the rule, by which the temporal premise W2 is proved either by the simpler rule RESP-J, or by rule WELL again, only applied to simpler assertions. As a matter of fact, if we closely examine the proof of the previous example, we can identify there the use of those two options. For proving W2 for $n = 1$, we used rule RESP-J, since the response property for this case is accomplished in one step. On the other hand, the case of $n > 1$ accomplishes W2 in two steps, and we therefore had to use rule CHAIN-J.

However, in many cases, we do not need the recursive application of the rule, which means that premise W2 is proved directly by rule RESP-J. In these cases it is advantageous to replace the temporal premise W2 by the nontemporal premises of rule RESP-J, which are necessary for its derivation. This leads to a (combined) form of the rule in which all premises are nontemporal. Such a form is often more satisfactory because it explicitly manifests the power of the rule to derive temporal statements from nontemporal ones.

This leads to rule WELL-J (Fig. 1.26).

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

$$\text{JW1.} \quad p \;\rightarrow\; \bigvee_{j=0}^{m} \varphi_j$$

$$\left.\begin{array}{ll}
\text{JW2.} & \rho_\tau \;\land\; \varphi_i \;\rightarrow\; \left[\begin{array}{l} \displaystyle\bigvee_{j=0}^{m} (\varphi'_j \;\land\; \delta_i \succ \delta'_j) \\ \lor\;\; (\varphi'_i \;\land\; \delta_i = \delta'_i) \end{array}\right] \\
 & \qquad\qquad\qquad\qquad \text{for every } \tau \in \mathcal{T} \\
\text{JW3.} & \rho_{\tau_i} \;\land\; \varphi_i \;\rightarrow\; \displaystyle\bigvee_{j=0}^{m} (\varphi'_j \;\land\; \delta_i \succ \delta'_j) \\
\text{JW4.} & \varphi_i \;\rightarrow\; En(\tau_i)
\end{array}\right\} \text{ for } i = 1, \ldots, m$$

$$p \;\Rightarrow\; \Diamond\, q$$

Fig. 1.26. Rule WELL-J (well-founded response under justice).

The rule requires finding auxiliary assertions φ_i and transitions τ_i, $i = 1, \ldots, m$, a well-founded domain $(\mathcal{A}, \succ)$, and ranking functions $\delta_i\colon \Sigma \mapsto \mathcal{A}$. Each assertion φ_i, $i > 1$, is associated with the transition τ_i that is helpful at positions satisfying φ_i, and with its own ranking function δ_i.

Premise JW1 requires that every p-position satisfies one of $\varphi_0, \ldots, \varphi_m$.

Premises JW2–JW4 impose three requirements for each $i = 1, \ldots, m$.

Premise JW2 requires that, taking any transition from a φ_i-position k, always leads to a successor position $k' = k+1$, such that

- either some φ_j, $j = 0, \ldots, m$, holds at k' with a rank δ'_j lower than δ_i at k, or
- φ_i holds at k' with a rank δ'_i equal to the rank δ_i at k.

The main implication of premise JW2 is that if the situation has not improved in any noticeable way in going from k to k', i.e., the new rank still equals the old rank, at least we have not lost the identity of the helpful transition and the transition that was helpful in k is also helpful at k'.

Premise JW3 requires that transition τ_i, which is helpful for φ_i, always leads from a φ_i-position k to a next position which satisfies some φ_j and has a rank lower than that of k, i.e., $\delta_i \succ \delta'_j$.

Premise JW4 requires that the helpful transition τ_i is enabled at every φ_i-position.

Justification To justify the rule, assume a computation such that p holds at position k, and no later position $i \geq k$, satisfies $q = \varphi_0$. By this assumption and JW1, some φ_j, $j > 0$, must hold at position k. Let φ_{i_1} be the formula holding at k, and denote the rank δ_{i_1} at k by u_1. By JW4, transition τ_{i_1} is enabled at position k.

Consider the transition τ taken at position k, leading into position $k+1$. By JW2 and JW3, either position $k+1$ has a lower rank u_2, $u_2 \prec u_1$, or it has the same rank, but then τ_{i_1} is still the helpful transition at $k+1$ and is enabled there. In the case that the rank is still u_1, we can continue the argument from $k+1$ to $k+2$, $k+3$, etc. However, we cannot have all positions $i > k$ with the same rank. To see this, assume that all positions beyond k do have the same rank. By JW2 and JW4, this implies that τ_{i_1} is continuously helpful and enabled. By JW3, τ_{i_1} is not taken beyond k because taking it would have led to a state with a rank lower than u_1. Thus, our assumption that all positions beyond k have the same rank leads to the situation that τ_{i_1} is continuously enabled and not taken, violating the justice requirement for τ_{i_1}.

Thus, eventually, we must reach a position k_2, $k_2 > k$, with lower rank u_2, where $u_2 \prec u_1$. In a similar way we can establish the existence of a position $k_3 > k_2$, with rank u_3 where $u_3 \prec u_2$. Continuing in this manner, we construct an infinitely descending sequence $u_1 \succ u_2 \succ u_3 \succ \cdots$ of elements of $\mathcal{A}$. This is impossible, due to the well-foundedness of $\succ$ on $\mathcal{A}$.

We conclude that every p-position must be followed by a q-position, establishing the consequence of the rule. ▟

Note that since premise JW3 implies premise JW2 for $\tau = \tau_i$, it is sufficient to check premise JW2 only for $\tau \neq \tau_i$.

Rule CHAIN-J can be viewed as a special case of rule WELL-J which uses $\delta_i = i$, for $i = 0, \ldots, m$, as ranking functions. It is not difficult to see that the premises J2, J3, and J4 of rule CHAIN-J correspond precisely to premises JW2, JW3, and JW4 of rule WELL-J. The well-founded domain used in this special case is the finite segment $[0..m]$ of the natural numbers ordered by $>$.

Example (factorial)

We use rule WELL-J to prove that program FACT of Fig. 1.24 satisfies the response property of total correctness

$$\underbrace{at\_\ell_0 \;\wedge\; x \geq 0 \;\wedge\; y = x \;\wedge\; z = 1}_{p} \quad\Rightarrow\quad \diamondsuit\, \underbrace{at\_\ell_2 \;\wedge\; z = x!}_{q = \varphi_0} \;.$$

Obviously, except for the terminating state, execution of program FACT alternates between states satisfying at_ℓ_0 in which ℓ_0 is the helpful transition, and states satisfying at_ℓ_1 in which ℓ_1 is helpful.

Consequently, we take $m = 2$ and use the following intermediate assertions.

$$\varphi_2\colon \quad at\_\ell_0 \;\wedge\; 0 \leq y \leq x \;\wedge\; z = x!/y!$$

$$\varphi_1\colon \quad at\_\ell_1 \;\wedge\; 1 \leq y \leq x \;\wedge\; z = x!/y!$$

$$\varphi_0\colon \quad at\_\ell_2 \;\wedge\; z = x! \;.$$

Note that when control is at ℓ_1, y is required to be greater than or equal to 1.

It remains to determine the ranking functions δ_i, $i = 0, 1, 2$. Our previous analysis of the considered response property for program FACT (using rule WELL) identified $|y|$ as a good measure of progress over $(\mathbb{N}, >)$. Variable y keeps decreasing as the program gets closer to termination. Unfortunately, premise JW3 of rule WELL-J requires that δ decreases on *each* activation of a helpful transition. As we see, not every helpful transition causes $|y|$ to decrease. In particular, ℓ_0 does not change $|y|$. Consequently, we have to supplement $|y|$ by an additional component that will decrease when $|y|$ stays the same. This leads to the following choice:

$$\delta_2\colon \quad (|y|,\ 2)$$

$$\delta_1\colon \quad (|y|,\ 1)$$

$$\delta_0\colon \quad (0,\ 0)\,.$$

The corresponding well-founded domain is $(\mathbb{N} \times \{1, 2\},\ \succ)$, where $\succ$ is the lexicographical order between pairs of integers.

In the previous proof of this property, we used the measure $|y| + 1$ to ensure that the rank decreases also on the transition from ℓ_0 to ℓ_2. Since the use of pairs guarantees such a decrease by a decreasing second component, we can omit the $+1$ increment and take the first component to be simply $|y|$.

We may view the ranking function δ_i: $(|y|, i)$ as consisting of a *major* and a *minor* measures of progress. Function $|y|$ measures large steps of progress, such as one full iteration of the loop at ℓ_0. The minor component i measures smaller steps of progress. Observe that transition ℓ_1 actually causes the minor measure i to increase from 1 to 2, but at the same time it decreases the major measure $|y|$.

Let us consider the premises of rule WELL-J. Since both φ_i's imply $y \geq 0$, we may replace $|y|$ by y.

- Premise JW1

We prove JW1 by showing

$$\underbrace{at\_\ell_0 \ \wedge\ x \geq 0 \ \wedge\ y = x \ \wedge\ z = 1}_{p} \quad\rightarrow$$

$$\cdots \ \vee\ \underbrace{at\_\ell_0 \ \wedge\ 0 \leq y \leq x \ \wedge\ z = x!/y!}_{\varphi_2},$$

which is obviously valid.

- Premises JW2, JW3 for $i = 2$

For $i = 2$ we will show

$$\rho_\tau \ \wedge\ \underbrace{at\_\ell_0 \ \wedge\ 0 \leq y \leq x \ \wedge\ z = x!/y!}_{\varphi_2} \quad\rightarrow$$

$$\left(\begin{array}{l} \underbrace{at'\_\ell_2 \ \wedge\ z = x!}_{\varphi_0'} \ \wedge\ \underbrace{(y, 2)}_{\delta_2} \succ \underbrace{(y', 0)}_{\delta_0'} \\ \qquad\quad \vee \\ \underbrace{at'\_\ell_1 \ \wedge\ 1 \leq y' \leq x \ \wedge\ z' = x!/y'!}_{\varphi_1'} \ \wedge\ \underbrace{(y, 2)}_{\delta_2} \succ \underbrace{(y', 1)}_{\delta_1'} \end{array}\right),$$

for each transition $\tau \in \{\ell_0, \ell_1\}$, not necessarily the helpful one. Obviously, this will satisfy both JW2 and JW3. Since at_ℓ_0 implies that transition ℓ_1 is disabled, the left-hand side of the implication for $\tau = \ell_1$ is false and the implication is trivially true.

For $\tau = \ell_0$, we prove the implication by separately considering the cases $y = 0$ and $y \neq 0$.

Case $y = 0$:

In this case ρ_{ℓ_0} implies at'_ℓ_2, $y' = y = 0$, and $z' = z$. Since $z = x!/y!$ and $y = 0$ imply $z = x!$, it follows that the left-hand side implies $\varphi'_0 \ \wedge\ (0,2) \succ (0,0)$.

Case $y \neq 0$:

In this case φ_2 implies that $y > 0$, which together with ρ_{ℓ_0}, implies at'_ℓ_1, $y' = y$, and $z' = z$. Assertion φ_2 implies $y \leq x \wedge z = x!/y!$ which together with $y > 0$ establishes φ'_1. The rank decrease $(y,2) \succ (y,1)$ is obvious.

- Premises JW2, JW3 for $i = 1$

For $i = 1$ we will show

$$\rho_\tau \ \wedge\ \underbrace{at\_\ell_1 \ \wedge\ 1 \leq y \leq x \ \wedge\ z = x!/y!}_{\varphi_1} \ \rightarrow$$

$$\cdots \ \vee\ \Big(\underbrace{at'\_\ell_0 \ \wedge\ 0 \leq y' \leq x \ \wedge\ z' = x!/y'!}_{\varphi'_2} \ \wedge\ \underbrace{(y,1)}_{\delta_1} \ \succ\ \underbrace{(y',2)}_{\delta'_2}\Big),$$

for each transition $\tau \in \{\ell_0, \ell_1\}$, not necessarily the helpful one. Obviously, this will satisfy both JW2 and JW3. Since at_ℓ_1 implies that transition ℓ_0 is disabled, the left-hand side of the implication for $\tau = \ell_0$ is false and the implication is trivially true.

For $\tau = \ell_1$, we observe that ρ_{ℓ_1} implies at'_ℓ_0, $y' = y - 1$, and $z' = z \cdot y$. Substituting these expressions in the right-hand side of the implication reduces the conjunction to

$$0 \leq y-1 \leq x \quad \wedge \quad z \cdot y = x!/(y-1)! \quad \wedge \quad (y,1) \succ (y-1,2),$$

all of which are either obviously valid or are implied by φ_1.

- Premises JW4

The helpful transitions for φ_1 and φ_2 are ℓ_1 and ℓ_0, with enabling conditions at_ℓ_1 and at_ℓ_0, respectively. Obviously, they satisfy

$$\underbrace{at\_\ell_1 \ \wedge\ \cdots}_{\varphi_1} \ \rightarrow\ \underbrace{at\_\ell_1}_{En(\tau_1)}$$

$$\underbrace{at\_\ell_0 \ \wedge\ \cdots}_{\varphi_2} \ \rightarrow\ \underbrace{at\_\ell_0}_{En(\tau_2)}$$

as required by premise JW4.

This establishes the four premises of rule WELL-J, proving the response property of total correctness for program FACT

$$at\_\ell_0 \ \wedge\ x \geq 0 \ \wedge\ y = x \ \wedge\ z = 1 \quad \Rightarrow \quad \diamondsuit(at\_\ell_2 \ \wedge\ z = x!). \qquad \blacksquare$$

A Condensed Representation of Ranking Functions

Many of our proofs consider ranking functions that consist of lexicographic pairs of natural numbers, i.e.,

$$\delta \;=\; (d_1, d_2).$$

Such a function decreases over a transition even if d_2 increases, provided d_1 decreases at the same time. Lexicographic order implies that even a small decrease in d_1 outweights an arbitrarily large increase in d_2. In some cases there exists a bound M, $M > 0$, which is larger than any possible increase in d_2. In these cases we may use the ranking function

$$\widehat{\delta} \;=\; M \cdot d_1 + d_2,$$

which ranges over $\mathbb{N}$, instead of the original $\delta = (d_1, d_2)$ which ranges over $\mathbb{N} \times \mathbb{N}$. We refer to $\widehat{\delta}$ as a *condensed representation* ranking function.

In **Problem 1.4** the reader is requested to prove that in such cases,

$$\widehat{\delta} \;=\; M \cdot d_1 + d_2 \;>\; \widehat{\delta}\,' \;=\; M \cdot d_1' + d_2' \qquad \text{iff} \qquad \delta\colon\ (d_1, d_2) \;\succ\; \delta'\colon\ (d_1', d_2').$$

For example, in the above proof of program FACT, we used the ranking functions

$$\delta_i\colon \quad (|y|,\ i).$$

Since the maximal increase in the value of i is 1 which is smaller than 2, we could have used instead the condensed ranking function

$$\widehat{\delta}_i\colon \quad 2 \cdot |y| \;+\; i.$$

Example (up down)

Consider program UP-DOWN presented in Fig. 1.27. This program can be viewed as an extension of program ANY-Y of Fig. 1.2. Process P_1 increments y, counting up in ℓ_0, ℓ_1, as long as $x = 0$. Once P_1 finds that x is different from 0, it proceeds to ℓ_2, ℓ_3, where y is decremented until it becomes 0. Process P_2's single action is to set x to 1. Obviously, due to justice, x will eventually be set to 1. However, one cannot predict the number of helpful steps required for P_1 to terminate. The longer P_2 waits before performing m_0, the higher the value y will attain on the move to ℓ_2. It is this value of y which determines the number of remaining steps to termination.

In fact, for every $n > 0$, we can construct a computation requiring more than $4n$ helpful steps to achieve $y = 0$. This computation allows P_1 to increase y up to n, and only then activates m_0. At least $2n$ more steps of P_1 are needed to decrement y back to 0.

Consequently, we need rule WELL-J to prove the response property

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

$$P_1 :: \begin{bmatrix} \ell_0\text{: } \textbf{while } x = 0 \textbf{ do} \\ \quad \ell_1\text{: } y := y + 1 \\ \ell_2\text{: } \textbf{while } y > 0 \textbf{ do} \\ \quad \ell_3\text{: } y := y - 1 \\ \ell_4\text{:} \end{bmatrix} \quad || \quad P_2 :: \begin{bmatrix} m_0\text{: } x := 1 \\ m_1\text{:} \end{bmatrix}$$

Fig. 1.27. Program UP-DOWN.

$$\underbrace{at\_\ell_0 \ \wedge \ at\_m_0 \ \wedge \ x = y = 0}_{p=\Theta} \ \Rightarrow \ \diamondsuit \underbrace{at\_\ell_4 \ \wedge \ at\_m_1}_{q}$$

for program UP-DOWN.

In order to construct the intermediate assertions and the ranking functions δ_i, we observe that there are three distinct phases in the achievement of $at\_\ell_4 \wedge at\_m_1$. The first phase waits for P_2 to perform m_0. This phase terminates when m_0 is executed. In the second phase, P_1 senses that x has been set to 1 and moves to ℓ_2. In the third phase, P_1 is within $\ell_{2,3}$ and decrements y until y reaches 0 and P_1 moves to ℓ_4.

Consequently, it seems advisable to use the well-founded domain $(\mathbb{N}^3, \succ)$ of lexicographic triples (n_1, n_2, n_3), whose first element n_1 identifies the phase, and whose remaining elements, n_2 and n_3, identify progress within the phase. Recall that lexicographic ordering on triples of natural numbers is defined by

$$(n_1,\ n_2,\ n_3) \ \succ \ (m_1,\ m_2,\ m_3) \qquad \text{iff} \qquad \begin{cases} n_1 > m_1 \quad \text{or} \\ n_1 = m_1,\ n_2 > m_2 \quad \text{or} \\ n_1 = m_1,\ n_2 = m_2,\ n_3 > m_3. \end{cases}$$

Obviously, this ordering is a well-founded relation on $\mathbb{N}^3$.

Consider the remaining elements needed to measure progress within a phase. The first phase terminates after one helpful step, m_0. The second phase terminates in two helpful steps, ℓ_1 followed by ℓ_0. The last phase has y measuring coarse progress, and it takes two steps to decrement y, ℓ_2 followed by ℓ_3.

Consequently, we define the following assertions, helpful transitions and ranking functions,

$$\begin{array}{llll} \varphi_5\text{:} & at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0 \wedge y \geq 0 & \tau_5\text{: } m_0 & \delta_5\text{: } (2,\ 0,\ 0) \\ \varphi_4\text{:} & at\_\ell_1 \quad \wedge at\_m_1 \wedge x = 1 \wedge y \geq 0 & \tau_4\text{: } \ell_1 & \delta_4\text{: } (1,\ 0,\ 1) \end{array}$$

$$\begin{array}{llll}
\varphi_3: & at_\ell_0 \;\wedge\; at_m_1 \wedge x = 1 \wedge y \geq 0 & \tau_3: \;\; \ell_0 & \delta_3: \;\; (1,\ 0,\ 0) \\
\varphi_2: & at_\ell_2 \;\wedge\; at_m_1 \wedge x = 1 \wedge y \geq 0 & \tau_2: \;\; \ell_2 & \delta_2: \;\; (0,\ |y|,\ 2) \\
\varphi_1: & at_\ell_3 \;\wedge\; at_m_1 \wedge x = 1 \wedge y > 0 & \tau_1: \;\; \ell_3 & \delta_1: \;\; (0,\ |y|,\ 1) \\
\varphi_0: & at_\ell_4 \;\wedge\; at_m_1 & & \delta_0: \;\; (0,\ 0,\ 0).
\end{array}$$

Note that progress within the second phase is measured by the third component, which moves from 1 to 0 on execution of ℓ_1. Progress within the third phase is measured by the pair $(y, 1 + at\_\ell_2)$ which decreases on execution of both ℓ_3 and ℓ_2.

⟪ Exercise?⟫

Let us show that all premises of rule WELL-J are satisfied by these choices.

- Premise JW1

For this premise we have to show the implication

$$\underbrace{at\_\ell_0 \;\wedge\; at\_m_0 \;\wedge\; x = y = 0}_{p} \;\;\rightarrow$$

$$\cdots \;\vee\; \underbrace{at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x = 0 \;\wedge\; y \geq 0}_{\varphi_5},$$

which is obvious.

- Premise JW2 for φ_5

It is sufficient to show the following for each $\tau \neq m_0$

$$\rho_\tau \;\wedge\; \underbrace{at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x = 0 \;\wedge\; y \geq 0}_{\varphi_5} \;\;\rightarrow$$

$$\cdots \;\vee\; \Big(\underbrace{at'\_\ell_{0,1} \;\wedge\; at'\_m_0 \;\wedge\; x' = 0 \;\wedge\; y' \geq 0}_{\varphi'_5} \;\wedge\; \underbrace{(2,\ 0,\ 0)}_{\delta_5} \;=\; \underbrace{(2,\ 0,\ 0)}_{\delta'_5}\Big)\ .$$

The only transitions $\tau \neq m_0$ enabled on φ_5-states are ℓ_0 and ℓ_1. For each of them, ρ_τ implies $at'\_\ell_{0,1}$, $x' = x = 0$, and $y' \geq y \geq 0$. Consequently, the implication is valid.

- Premise JW3 for φ_5

We show

$$\rho_{m_0} \;\wedge\; \underbrace{at\_\ell_{0,1} \;\wedge\; at\_m_0 \;\wedge\; x = 0 \;\wedge\; y \geq 0}_{\varphi_5} \;\;\rightarrow$$

$$\left[\begin{array}{l} \underbrace{at'\_\ell_1 \ \wedge\ at'\_m_1 \ \wedge\ x'=1 \ \wedge\ y' \geq 0}_{\varphi'_4} \ \wedge\ \underbrace{(2,\ 0,\ 0)}_{\delta_5} \succ \underbrace{(1,\ 0,\ 1)}_{\delta'_4} \\ \qquad \vee \\ \underbrace{at'\_\ell_0 \ \wedge\ at'\_m_1 \ \wedge\ x'=1 \ \wedge\ y' \geq 0}_{\varphi'_3} \ \wedge\ \underbrace{(2,\ 0,\ 0)}_{\delta_5} \succ \underbrace{(1,\ 0,\ 0)}_{\delta'_3} \end{array}\right].$$

Clearly, ρ_{m_0} implies $x' = 1$, $y' = y$, at'_m_1 and $at'\_\ell_i = at\_\ell_i$ for $i = 0, 1$. By φ_5, either at_ℓ_0 or at_ℓ_1 holds. In the case that $at\_\ell_0 = \mathrm{T}$, the second disjunct is implied. In the case that $at\_\ell_1 = \mathrm{T}$, the first disjunct is implied. The decrease in rank is obvious in both cases.

- Premises JW2, JW3 for φ_4

Since ℓ_1 is the only transition enabled on φ_4-states, the following implication establishes both JW2 and JW3 for φ_4:

$$\rho_{\ell_1} \ \wedge\ \underbrace{at\_\ell_1 \ \wedge\ at\_m_1 \ \wedge\ x=1 \ \wedge\ y \geq 0}_{\varphi_4} \ \rightarrow$$

$$\cdots \ \vee\ \Big(\underbrace{at'\_\ell_0 \ \wedge\ at'\_m_1 \ \wedge\ x'=1 \ \wedge\ y' \geq 0}_{\varphi'_3} \ \wedge\ \underbrace{(1,\ 0,\ 1)}_{\delta_4} \succ \underbrace{(1,\ 0,\ 0)}_{\delta'_3}\Big) .$$

Transition relation ρ_{ℓ_1} implies at'_ℓ_0, $at'\_m_1 = at\_m_1$, $x' = x$, and $y' \geq y$. By φ_4, it follows that φ'_3 holds, and the decrease in rank is obvious.

- Premises JW2, JW3 for φ_3

Since ℓ_0 is the only transition enabled on φ_3-states, the following implication establishes both JW2 and JW3 for φ_3

$$\rho_{\ell_0} \ \wedge\ \underbrace{at\_\ell_0 \ \wedge\ at\_m_1 \ \wedge\ x=1 \ \wedge\ y \geq 0}_{\varphi_3} \ \rightarrow$$

$$\cdots \ \vee\ \Big(\underbrace{at'\_\ell_2 \ \wedge\ at'\_m_1 \ \wedge\ x'=1 \ \wedge\ y' \geq 0}_{\varphi'_2} \ \wedge\ \underbrace{(1,\ 0,\ 0)}_{\delta_3} \succ \underbrace{(0,\ |y'|,\ 2)}_{\delta'_2}\Big)$$

Transition relation ρ_{ℓ_0} under $x = 1$ implies at'_ℓ_2 and $at'\_m_1 = at\_m_1$. It also implies $x' = x$ and $y' = y$. The rank decrease $(1, 0, 0) \succ (0, |y'|, 2)$ is obvious, since 1, the first component of the left-hand side, is larger than 0, the first component of the right-hand side.

- Premises JW2, JW3 for φ_2

Since ℓ_2 is the only transition enabled on φ_2-states, the following implication establishes both JW2 and JW3 for φ_2

$$\rho_{\ell_2} \;\wedge\; \underbrace{at\_\ell_2 \;\wedge\; at\_m_1 \;\wedge\; x=1 \;\wedge\; y \geq 0}_{\varphi_2} \;\rightarrow$$

$$\left(\begin{array}{l} \underbrace{at'\_\ell_4 \;\wedge\; at'\_m_1}_{\varphi'_0} \;\wedge\; \underbrace{(0,\; |y|,\; 2)}_{\delta_2} \succ \underbrace{(0,\; 0,\; 0)}_{\delta'_0} \\ \qquad \vee \\ \underbrace{at'\_\ell_3 \;\wedge\; at'\_m_1 \;\wedge\; x'=1 \;\wedge\; y'>0}_{\varphi'_1} \;\wedge\; \underbrace{(0,\; |y|,\; 2)}_{\delta_2} \succ \underbrace{(0,\; |y'|,\; 1)}_{\delta'_1} \end{array}\right).$$

We distinguish between two cases.

Case $y = 0$:

In this case, ρ_{ℓ_2} implies at'_ℓ_4, $y' = y = 0$, and $at'\_m_1 = at\_m_1$. Since φ_2 implies $at\_m_1 = \mathrm{T}$, the left-hand side of this verification condition implies the right-hand side disjunct $\varphi'_0 \;\wedge\; (0, |y|, 2) \succ (0, 0, 0)$.

Case $y \neq 0$:

By φ_2, it follows that $y > 0$. In this case, ρ_{ℓ_2} implies at'_ℓ_3, $at'\_m_1 = at\_m_1$, $x' = x$, and $y' = y > 0$. Together with φ_2, these imply φ'_1. To show the rank decrease, we observe that $|y| = |y'|$ and $2 > 1$.

- Premises JW2, JW3 for φ_1

Since ℓ_3 is the only transition enabled on φ_1-states, the following implication establishes both JW2 and JW3 for φ_1

$$\rho_{\ell_3} \;\wedge\; \underbrace{at\_\ell_3 \;\wedge\; at\_m_1 \;\wedge\; x=1 \;\wedge\; y>1}_{\varphi_1} \;\rightarrow$$

$$\cdots \;\vee\; \Big(\underbrace{at'\_\ell_2 \;\wedge\; at'\_m_1 \;\wedge\; x'=1 \;\wedge\; y' \geq 0}_{\varphi'_2} \;\wedge\; \underbrace{(0,\; |y|,\; 1)}_{\delta_1} \succ \underbrace{(0,\; |y'|,\; 2)}_{\delta'_2}\Big).$$

Transition relation ρ_{ℓ_3} implies at'_ℓ_2, $at'\_m_1 = at\_m_1$, $x' = x$, and $y' = y - 1$. By the clause $y > 0$ in φ_1 we have $y' = y - 1 \geq 0$. The decrease in rank follows from $y > 0$ and $(0, |y|, 1) = (0, y, 1) \succ (0, y-1, 2) = (0, |y'|, 2)$.

- Premise JW4

This premise requires showing the following implication for each $i = 1, \ldots, 5$.

$$\varphi_i \;\rightarrow\; En(\tau_i).$$

By inspecting φ_i for each $i = 1, \ldots, 5$, we see that this is indeed the case.

This concludes the proof. ▪

Persistence of the Helpful Transitions

Premise JW2 of rule WELL-J requires that, in the case that a transition does not attain a lower rank in the next state, it must maintain the rank and lead to a state that still satisfies φ_i, and therefore maintain τ_i as the helpful transition. We refer to this clause as a requirement for the persistence of helpful transitions. One may wonder how essential this requirement is, and whether it would be possible to relax this requirement. In **Problem 1.5**, we request the reader to consider a version of rule WELL-J in which premise JW2 has been relaxed to allow the helpful transition to change without rank decrease. The problem shows that the resulting rule is unsound.

1.5 Rank Diagrams

To represent by diagrams proofs of response properties that require the use of well-founded ranking, we have to add some more components to the labels of nodes.

A verification diagram is said to be a RANK *diagram* if its nodes are labeled by assertions $\varphi_0, \ldots, \varphi_m$, with φ_0 being the terminal node, and ranking functions $\delta_0, \ldots, \delta_m$, where each δ_i maps states into $\mathcal{A}$, and it satisfies the following requirement:

- Every node φ_i, $i > 0$, has a double edge departing from it. This identifies the transition labeling such an edge as *helpful* for assertion φ_i. All helpful transitions must be just.

Note that, unlike CHAIN diagrams, we allow node φ_i to be connected to φ_j for $j > i$.

Verification and Enabling Conditions for RANK Diagrams

Consider a nonterminal node labeled by assertion φ and ranking function δ, and let $\varphi_1, \ldots, \varphi_k$, $k \geq 0$, be the τ-successors of φ and $\delta_1, \ldots, \delta_k$ be their respective ranking functions.

- If transition τ is unhelpful for φ, i.e., labels only single edges departing from the node, then we associate with φ and τ the following verification condition

$$\Big\{\varphi \wedge \delta = u\Big\} \;\; \tau \;\; \Big\{(\varphi \wedge u \succeq \delta) \;\vee\; (\varphi_1 \wedge u \succ \delta_1) \;\vee\; \cdots \;\vee\; (\varphi_k \wedge u \succ \delta_k)\Big\}.$$

- If τ is helpful for φ (labels double edges), we associate with φ and τ the following verification condition

$$\Big\{\varphi \wedge \delta = u\Big\} \;\; \tau \;\; \Big\{(\varphi_1 \wedge u \succ \delta_1) \;\vee\; \cdots \;\vee\; (\varphi_k \wedge u \succ \delta_k)\Big\}.$$

- For every nonterminal node φ and a transition τ labeling a double edge departing from φ, we require

$$\varphi \quad \rightarrow \quad En(\tau).$$

Note that in the case of an unhelpful transition, we allow a τ-successor with a rank equal to that of φ, provided it satisfies the same assertion φ.

Valid RANK Diagrams

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

The consequences of having a valid RANK diagram are stated in the following claim.

Claim 1.2 (RANK diagrams)

A P-valid RANK diagram establishes that the response formula

$$\bigvee_{j=0}^{m} \varphi_j \quad \Rightarrow \quad \diamondsuit\, \varphi_0$$

is P-valid.

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

$$p \ \rightarrow \ \bigvee_{j=0}^{m} \varphi_j \qquad \text{and} \qquad \varphi_0 \ \rightarrow \ q$$

then, we can conclude the validity of

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

Justification It is not difficult to see that a valid RANK diagram establishes the premises of rule WELL-J with p: $\bigvee_{j=0}^{m} \varphi_j$, q: φ_0, and τ_i the transition helpful for φ_i being the transition labeling the double edge departing from φ_i in the diagram. This establishes the P-validity of

$$\bigvee_{j=0}^{m} \varphi_j \ \Rightarrow \ \diamondsuit\, \varphi_0.$$

Given assertions p and q, satisfying the implications

$$p \ \rightarrow \ \bigvee_{j=0}^{m} \varphi_j \qquad \text{and} \qquad \varphi_0 \ \rightarrow \ q,$$

we can use rule MON-R of Fig. 1.3 to infer

$$p \;\Rightarrow\; \diamondsuit q. \quad \blacksquare$$

Example (factorial)

The diagram of Fig. 1.28 presents a valid RANK diagram that establishes total correctness for program FACT (Fig. 1.24).

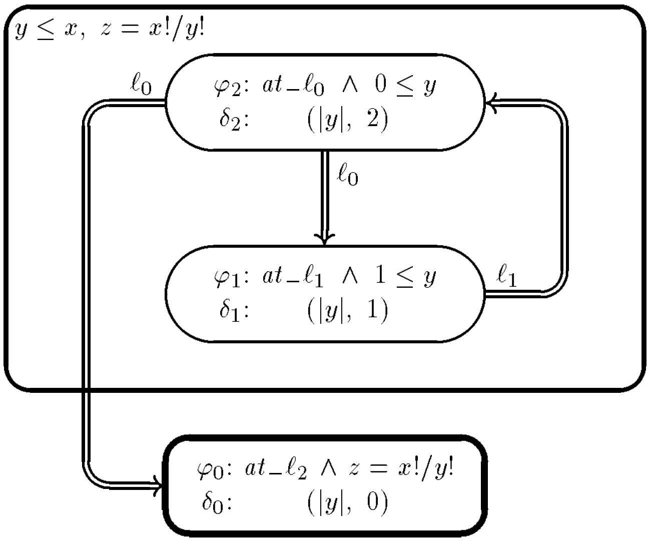


Fig. 1.28. RANK diagram for total correctness of program FACT.

This diagram contains a connection from φ_1 to φ_2 which is disallowed in chain diagrams. However, the validity of the implied verification conditions ensures that, whenever a transition is taken from a φ_2-state s_2 to a φ_1-state s_1, the rank decreases, i.e., $\delta_2(s_2) \succ \delta_1(s_1)$. $\blacksquare$

Distributing the Ranking Functions

To make RANK diagrams more readable, we introduce additional encapsulation conventions.

One of the useful conventions is that compound nodes may be labeled by a list of assertions. Such labeling indicates that the full assertion associated with a basic (noncompound) node n_i is a conjunction of the assertion labeling the node itself and all the assertions labeling compound nodes that contain n_i.

Thus, while the label of node φ_2 in the diagram of Fig. 1.28 is $at\_\ell_0 \wedge 0 \leq y$, the full assertion associated with this node is

$$at\_\ell_0 \;\wedge\; 0 \leq y \;\wedge\; y \leq x \;\wedge\; z = x!/y!\ .$$

We can view this representation as distribution of the full assertion into the part $at\_\ell_0 \wedge 0 \leq y$ labeling the node itself and the part $y \leq x \wedge z = x!/y!$ labeling the enclosing node, which is common to both φ_1 and φ_2.

In a similar way, we introduce a convention for distribution of ranking functions. The convention allows us to label a compound node by

$$\delta\colon\ f,$$

where f is some ranking function mapping states into a well-founded domain $\mathcal{A}$. In most of our examples, the domains are either $(\mathbb{N}, >)$ or lexicographic products of this domain.

Consider a basic node n_i labeled by assertion φ_i and local ranking function f_b. Assume that node n_i is contained in a nested sequence of compound nodes that are labeled by ranking labels $\delta\colon f_1, \ldots, \delta\colon f_m$, as we go from the outermost compound node towards n_i. This situation is depicted in Fig. 1.29. Then the full ranking function associated with the node φ_i is given by the tuple

$$\delta_i \;=\; (f_1,\ \ldots,\ f_m,\ f_b).$$

That is, we consider the outermost ranking f_1 to be the most significant component in δ_i, and the local ranking f_b to be the least significant component.

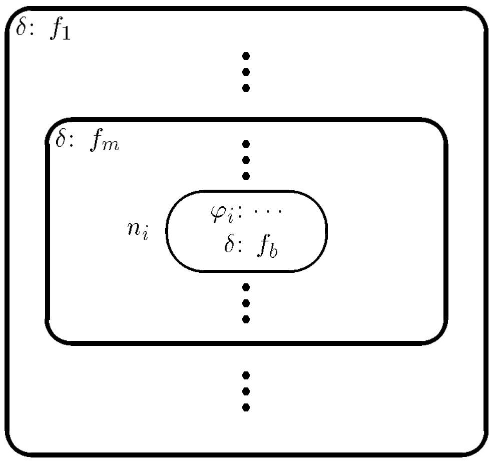


Fig. 1.29. Encapsulated sequence of nodes.

Example (factorial)

In Fig. 1.30, we present a version of the RANK diagram of Fig. 1.28, in which a common component of the ranking function appears as a ranking label of the enclosing compound state.

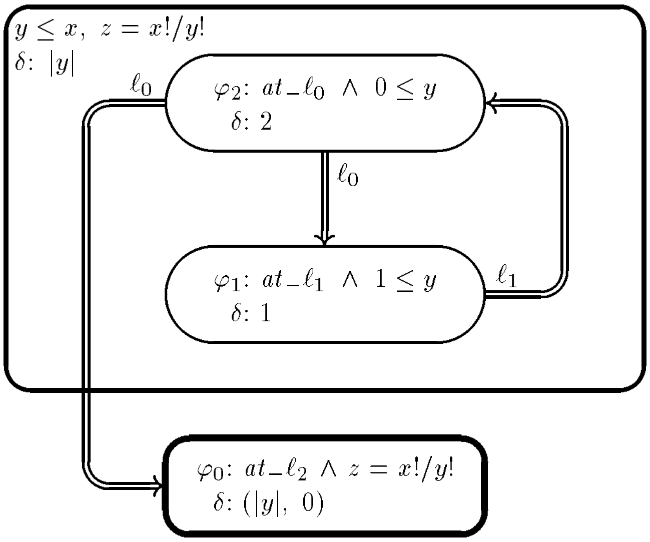


Fig. 1.30. RANK diagram with distributed ranking functions.

The full ranking functions associated with the nodes in the RANK diagram of Fig. 1.30 are identical to those appearing in Fig. 1.28. ▪

Another rank distribution convention allows one to omit the local rank labeling a node φ_i altogether. This is interpreted as if the node were labeled with the ranking function δ: i, where i is the index of the node (and the assertion labeling it). In Fig. 1.31, we present another version of the RANK diagram for program FACT, using this convention.

The full ranking functions associated with the nodes in this diagram are:

$$\delta_2\colon\ (|y|,\ 2), \qquad \delta_1\colon\ (|y|,\ 1), \qquad \text{and} \qquad \delta_0\colon\ 0.$$

This raises the question of how to compare lexicographic tuples of unequal lengths such as δ_2: $(|y|,\ 2)$ and δ_0: 0.

Since all our examples will be based on tuples of non-negative integers, we agree that the relation holding between $(a_1, \ldots, a_i)$ and $(b_1, \ldots, b_k)$ for $i < k$ is determined by lexicographically comparing $(a_1, \ldots, a_i, 0, \ldots, 0)$ to $(b_1, \ldots, b_i, b_{i+1},$

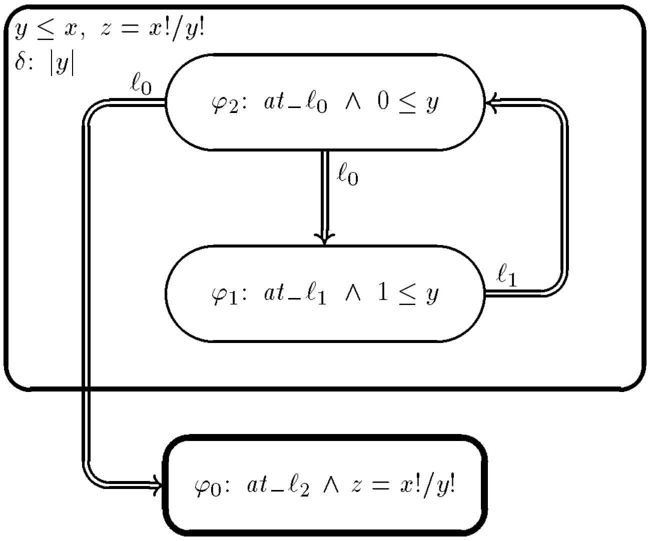


Fig. 1.31. RANK diagram with default local ranking.

$\ldots, b_k)$. That is, we pad the shorter tuple by zeros on the right until it assumes the length of the longer tuple.

According to this definition, $(|y|,\ 2) \succ 0$, since $(|y|,\ 2) \succ (0,\ 0)$.

Example (up-down)

In Fig. 1.32 we present a valid RANK diagram which implies, by monotonicity, the property of termination for program UP-DOWN (Fig. 1.27).

$$\underbrace{at\_\ell_0 \ \wedge\ at\_m_0 \ \wedge\ x = y = 0}_{p=\Theta} \quad\Rightarrow\quad \diamondsuit\, \underbrace{at\_\ell_4 \ \wedge\ at\_m_1}_{\varphi_0}\ .$$

The diagram provides a detailed description of the progress of the computation from φ_5 to φ_0. It shows that progress from φ_5 to φ_2 is due to a CHAIN-like reasoning. Then, the progress from φ_2 and φ_1 to φ_0 requires a well-founded argument with the measure $|y|$ for coarse progress, and the index $j = 1, 2$ of φ_j for measuring fine progress.

The ranking functions appearing in this diagram are somewhat different from the ones used originally. When padded to the maximum length of 3, they are given by

δ_5: $(5,\ 0,\ 0)$

δ_4: $(4,\ 0,\ 0)$

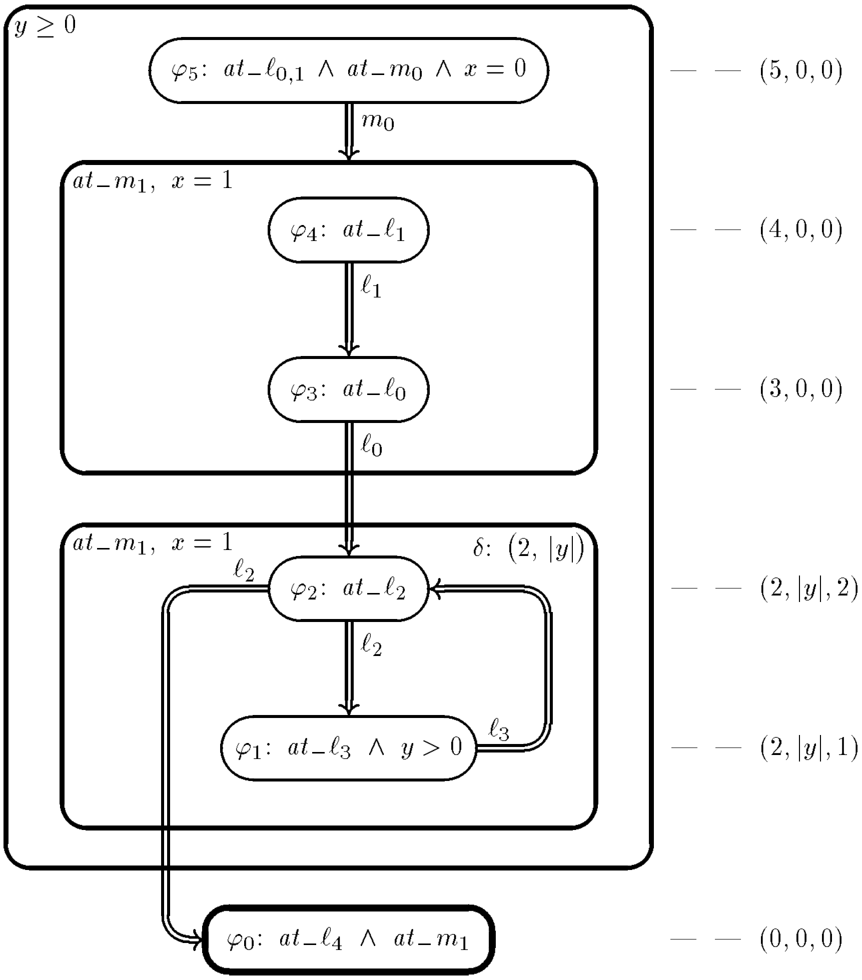


Fig. 1.32. RANK diagram for termination of UP-DOWN.

δ_3: $(3, \; 0, \; 0)$

δ_2: $(2, \; |y|, \; 2)$

δ_1: $(2, \; |y|, \; 1)$

δ_0: $(0, \; 0, \; 0)$.

Note that the diagram contains a connection from φ_1 to φ_2. This is allowed because ℓ_3 decrements y and leads to a decrease in rank, stated by

$$(2,\ |y|,\ 1)\ \succ\ (2,\ |y-1|,\ 2).$$ ▪

In **Problems 1.6–1.9**, the reader is requested to prove total correctness of several programs.

1.6 Response with Past Subformulas

In this section we generalize the methods and proof rules presented in the preceding sections to handle response formulas $p \Rightarrow \diamondsuit q$, where p and q are past formulas.

The generalization is straightforward. It involves the following systematic modifications and replacements.

- Wherever a rule calls for one or more intermediate assertions, the past-version of the rule requires finding past formulas.
- Each premise of the form $\varphi \rightarrow \psi$, for assertions φ and ψ, is replaced by an entailment $\widehat{\varphi} \Rightarrow \widehat{\psi}$ for corresponding past formulas $\widehat{\varphi}$ and $\widehat{\psi}$.
- A verification condition $\{p\}\,\tau\,\{q\}$, for past formulas p and q and transition τ, is interpreted as the entailment $\rho_\tau \wedge p \Rightarrow q'$, where the primed version of a past formula is calculated as in Section 4.1 of the SAFETY book.

For example, in Fig. 1.33, we present the past version of rule WELL-J.

Similar past versions can be derived for rules RESP-J and CHAIN-J.

Example Let us illustrate the use of the past version of rule WELL-J for proving the response property

$$0 \le n \le y \quad \Rightarrow \quad \diamondsuit\big(y = n\ \wedge\ \diamondminus(at\_\ell_0\ \wedge\ y = n)\big)$$

for program UP-DOWN of Fig. 1.27.

This property states that any position i at which y is greater or equal to some $n \ge 0$, is followed by a position j at which $y = n$ and such that, at a preceding position $k \le j$, y equaled n while control was at ℓ_0. This property characterizes a feature of program UP-DOWN by which a computation that achieves $y \ge n$ at some state, has at least two occurrences of states in which $y = n$. One occurrence has control at ℓ_0 while the other occurrence has control at ℓ_2.

In our proof, we use the following invariants for program UP-DOWN

$$\chi_0:\quad y \ge 0$$

$$\chi_1:\quad (at\_\ell_{0,1} \wedge at\_m_0 \wedge x = 0) \quad \vee \quad (at\_\ell_{0..4} \wedge at\_m_1 \wedge x = 1)$$

$$\chi_2:\quad at\_\ell_4 \ \rightarrow\ y = 0$$

$$\chi_3:\quad y < n\ \vee\ \diamondminus(at\_\ell_0\ \wedge\ y = n).$$

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

$$\text{JW1.}\quad p \;\Rightarrow\; \bigvee_{j=0}^{m} \varphi_j$$

$$\left.\begin{array}{ll}
\text{JW2.}\quad \rho_\tau \;\wedge\; \varphi_i \;\Rightarrow & \begin{bmatrix} \displaystyle\bigvee_{j=0}^{m} (\varphi'_j \;\wedge\; \delta_i \succ \delta'_j) \\ \vee\;\; (\varphi'_i \;\wedge\; \delta_i = \delta'_i) \end{bmatrix} \\
 & \qquad \text{for every } \tau \in \mathcal{T} \\
\text{JW3.}\quad \rho_{\tau_i} \;\wedge\; \varphi_i \;\Rightarrow & \displaystyle\bigvee_{j=0}^{m} (\varphi'_j \;\wedge\; \delta_j \succ \delta'_j) \\
\text{JW4.}\quad \varphi_i \;\Rightarrow\; En(\tau_i) &
\end{array}\right\} \text{ for } i = 1, \ldots, m$$

$$\overline{\qquad p \;\Rightarrow\; \diamondsuit\, q \qquad}$$

Fig. 1.33. Past version of rule WELL-J.

State invariants χ_0, χ_1, and χ_2 are derived and proven in the usual way.

Proving the Invariance of χ_3

Formula χ_3 is a past invariant, and can be proven by rule P-INV, taking $\varphi = \psi\colon\ y < n \vee \diamondminus(at\_\ell_0 \wedge y = n)$. Premise P1 of rule P-INV is trivial since $\varphi = \psi$. Premise P2 requires

$$\underbrace{\cdots at\_\ell_0 \;\wedge\; y = 0 \;\wedge\; \cdots}_{\Theta} \;\;\rightarrow\;\; \underbrace{y < n \;\vee\; (at\_\ell_0 \;\wedge\; y = n)}_{\varphi_0}\;.$$

As $n \geq 0$, we consider two cases. If $n > 0$ then $y = 0$ implies $y < n$. If $n = 0$ then $at\_\ell_0 \wedge y = 0$ implies $at\_\ell_0 \wedge y = n$.

Finally, premise P2 requires showing

$$\rho_\tau \;\;\wedge\;\; \underbrace{y < n \;\vee\; \diamondminus(at\_\ell_0 \;\wedge\; y = n)}_{\varphi} \;\;\Rightarrow$$

$$\underbrace{y' < n \ \lor \ (at'\_\ell_0 \ \land \ y' = n) \ \lor \ \diamondminus(at\_\ell_0 \ \land \ y = n)}_{\varphi'} \ .$$

This can be shown by temporal instantiation of the implication

$$\rho_\tau \ \land \ (y < n \ \lor \ p) \quad \rightarrow \quad \big(y' < n \ \lor \ (at'\_\ell_0 \ \land \ y' = n) \ \lor \ p\big).$$

Obviously if $p = \text{T}$ the implication is trivially valid. It therefore remains to show that the following holds:

$$\rho_\tau \ \land \ y < n \quad \rightarrow \quad y' < n \ \lor \ (at'\_\ell_0 \ \land \ y' = n).$$

This implication can be potentially falsified only by a transition that can transform a state satisfying $y < n$ into a next state satisfying $\neg(y' < n)$, i.e., $y' \geq n$. The only candidate transition is ℓ_1. Therefore, we consider

$$\underbrace{\cdots \ \land \ at'\_\ell_0 \ \land \ y' = y+1}_{\rho_\tau} \ \land \ y < n \quad \rightarrow \quad y' < n \ \lor \ (at'\_\ell_0 \ \land \ y' = n).$$

As $y < n$, we consider two cases. If $y < n-1$ then $y' = y+1 < n$. If $y = n-1$ then $at'\_\ell_0 \land y' = y+1$ implies $at'\_\ell_0 \land y' = n$.

This concludes the proof of past invariant χ_3.

Proving the Response Formula

To prove the response formula

$$\underbrace{0 \leq n \leq y}_{p} \quad \Rightarrow \quad \diamond \underbrace{y = n \ \land \ \diamondminus(at\_\ell_0 \ \land \ y = n)}_{q} \ ,$$

we use the past version of rule WELL-J as presented in Fig. 1.33.

The choice of intermediate past formulas φ_1–φ_5, helpful transitions and ranking functions is presented in the verification diagram of Fig. 1.34.

Note that each of $\varphi_0, \ldots, \varphi_5$ is a past formula. For example, the full formula φ_4 is given by

$$\varphi_4\!: \quad at\_\ell_1 \quad \land \quad at\_m_1 \quad \land \quad x = 1 \quad \land \quad y > n \geq 0 \quad \land \quad \diamondminus(at\_\ell_0 \ \land \ y = n).$$

Let us consider some of the premises required by rule WELL-J. Premise JW1 requires the following implication

$$0 \leq n \leq y \quad \Rightarrow$$

$$\underbrace{\diamondminus(at\_\ell_0 \ \land \ y = n) \ \land \ \left(y = n \quad \lor \quad y > n \land \begin{pmatrix} at\_\ell_{0,1} \ \land \ at\_m_0 \ \land \ x = 0 \\ \lor \\ at\_\ell_{0..3} \ \land \ at\_m_1 \ \land \ x = 1 \end{pmatrix}\right)}_{\varphi_0 \ \lor \ \cdots \ \lor \ \varphi_5} \ .$$

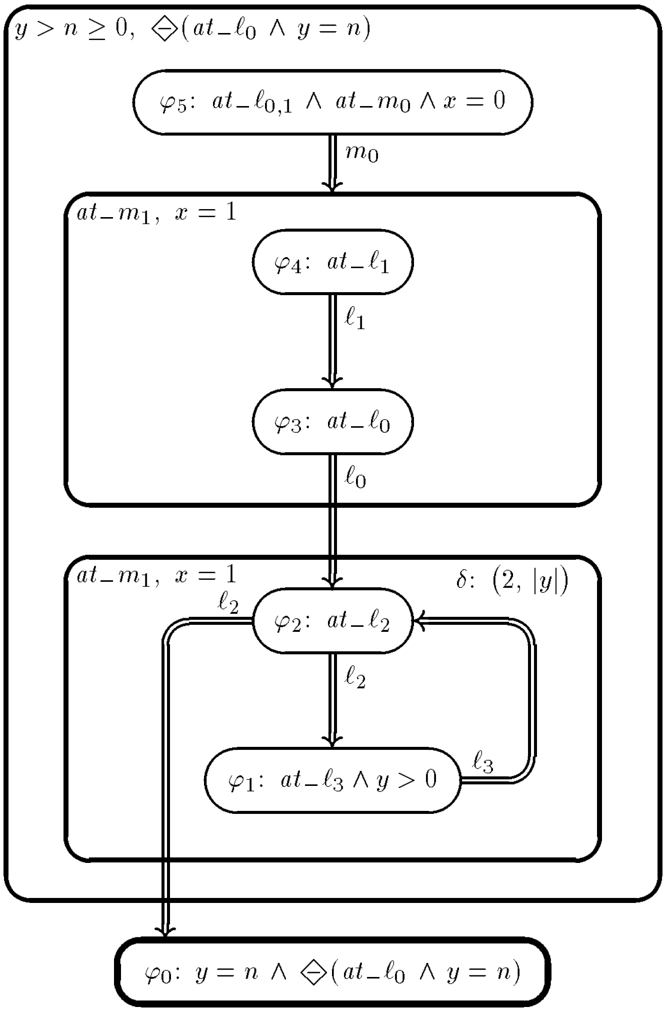


Fig. 1.34. RANK diagram for $(0 \leq n \leq y) \Rightarrow \diamond\big(y = n \land \diamondminus(at\_\ell_0 \land y = n)\big)$.

It is not difficult to see that this entailment follows from invariants χ_0–χ_3.

Observe that each φ_i, $i = 0, \ldots, 5$ can be written in the form

$$\varphi_i \quad = \quad \diamondminus(at\_\ell_0 \land y = n) \land \widehat{\varphi}_i,$$

where $\widehat{\varphi}_i$ is a state formula.

Consequently, premise JW2 can be written as follows

$\rho_\tau \land \widehat{\varphi}_i \land \diamondminus(at\_\ell_0 \land y = n) \Rightarrow$

$$\big(\diamondminus(at\_\ell_0 \ \wedge \ y = n)\big)' \ \wedge \ \left(\widehat{\varphi}'_0 \ \vee \ \bigvee_{j=1}^{5} (\widehat{\varphi}'_j \ \wedge \ \delta_i \succ \delta'_j) \ \vee \ (\widehat{\varphi}_i \ \wedge \ \delta_i = \delta'_i)\right).$$

Since $\diamondminus(at\_\ell_0 \wedge y = n)$ entails $\big(\diamondminus(at\_\ell_0 \wedge y = n)\big)'$, which expands to the disjunction

$$\big(\diamondminus(at\_\ell_0 \ \wedge \ y = n)\big)': \quad (at'\_\ell_0 \ \wedge \ y' = n) \ \vee \ \diamondminus(at\_\ell_0 \ \wedge \ y = n),$$

it only remains to establish the following state entailment

$$\rho_\mathcal{T} \ \wedge \ \widehat{\varphi}_i \quad \Rightarrow \quad \widehat{\varphi}'_0 \ \vee \ \bigvee_{j=1}^{5} (\widehat{\varphi}'_j \ \wedge \ \delta_i \succ \delta'_j) \ \vee \ (\widehat{\varphi}_i \ \wedge \ \delta_i = \delta'_i).$$

This entailment can be proven in a way similar to the proof of the verification conditions in the RANK diagram of Fig. 1.32, whose ranking functions are identical to those of Fig. 1.34.

The past conjunct $\diamondminus(at\_\ell_0 \ \wedge \ y = n)$ can be similarly factored out also for premise JW3.

This concludes the proof that property

$$0 \leq n \leq y \quad \Rightarrow \quad \diamond\big(n = y \ \wedge \ \diamondminus(at\_\ell_0 \ \wedge \ y = n)\big)$$

for program UP-DOWN. ∎

1.7 Compositional Verification of Response Properties

Compositional verification is a method intended to reduce the complexity of verifying properties of large programs. The method infers properties of the whole system from properties of its components, which are proven separately for each component.

We apply compositional verification to programs that can be decomposed into several top-level processes, called *components*, which communicate by shared variables and such that every variable of the program can be modified by at most one of these components. A variable which is modified by component P_i is said to be *owned* by P_i.

Modular Computations

Let $P :: \big[\text{declarations}; \ \big[P_1 :: \ [\ell_0^1\colon \ S_1]\| \cdots \|P_k :: \ [\ell_0^k\colon \ S_k]\big]\big]$ be a program, and P_i be a component of P. Denote by $V = \{\pi\} \cup Y$ the set of system variables of P, and let $Y_i \subseteq Y$ be the set of variables owned by P_i. Let L_i denote the set of locations of process P_i.

Assume that we have constructed the fair transition system (FTS) S_P: $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C}\rangle$ corresponding to P. We assume that the initial condition has the form

$$\Theta: \quad \pi = \{\ell_0^1, \ldots, \ell_0^k\} \ \wedge \ \bigwedge_{y \in Y} p_y(y),$$

where, for each $y \in Y$, $p_y(y)$ is an assertion constraining the initial values of variable y. Obviously, $p_y(y)$ is derived from one of the *where* clauses in the declarations of variables in P. If there is no *where* clause constaining y, then $p_y(y) = \text{T}$. Let T_i denote the transitions of S_P associated with the statements of P_i.

Based on the FTS S_P and process P_i, we construct a new FTS $S^M_{P_i}$: $\langle V_i, \Theta_i, \mathcal{T}_i, \mathcal{J}_i, \mathcal{C}_i\rangle$ called the *modular* FTS *corresponding to* P_i. The FTS $S^M_{P_i}$ is intended to capture the possible behavior of process P_i in any context (not necessarily that of P) which respects the ownership of Y_i by P_i. That is, we are ready to consider any context whose only restriction is that it cannot modify any variable owned by P_i. The constituents of $S^M_{P_i}$ ar given by:

- V_i: V

  The system variables of $S^M_{P_i}$ are identical to the system variables of the complete FTS S_P.

- Θ_i: $\left(\pi \cap L_i = \{\ell_0^i\}\right) \quad \wedge \quad \bigwedge_{y \in Y_i} p_y(y)$

  The initial condition of $S^M_{P_i}$ requires that, initially, the only L_i-location contained in π is ℓ_0^i and all the variables owned by P_i satisfy their initial constraints as specified in the *where* clauses of the program declarations. Except for π, nothing is required by Θ_i concerning the system variables not owned by P_i.

- $\mathcal{T}_i = T_i \cup \{\tau_E\}$

  The transitions of $S^M_{P_i}$ include all transitions associated with statements of P_i (T_i) and a special *environment transition* τ_E. Transition τ_E is intended to represent the actions of an arbitrary context which respects the ownership of Y_i by P_i. For each $\tau \in T_i$, the transition relation $S^M_{P_i}$ associates with τ is ρ_τ, the transition relation S_P associates with τ. The transition relation for τ_E is given by

  $$\rho_E: \quad (\pi' \cap L_i = \pi \cap L_i) \quad \wedge \quad press(Y_i).$$

  This transition relation guarantees the preservation of the L_i-part of π and preservation of the values of all variables owned by P_i. The special treatment of π can be described by saying that, in addition to owning the variables in Y_i, P_i also owns the L_i-part of π (projection of π on L_i).

- $\mathcal{J}_i \;=\; \mathcal{J} \,\cap\, T_i$

  The just transitions of $S^M_{P_i}$ are the just transitions among T_i.

- $\mathcal{C}_i \;=\; \mathcal{C} \,\cap\, T_i$

  The compassionate transitions of $S^M_{P_i}$ are the compassionate transitions among T_i.

There is no need to include the idling transition τ_I in $\mathcal{T}_i$ because the effect of τ_I, a transition that changes no system variable, can be obtained as a special case of τ_E.

We refer to each computation of FTS $S^M_{P_i}$ as a *modular computation* of process P_i. As previously explained, any such computation represents a possible behavior of process P_i when put in an arbitrary context which is only required to respect the ownership rights of P_i.

Example (program KEEPING-UP)

Consider program KEEPING-UP presented in Fig. 1.35. Top-level process P_1 owns variable x and the $\ell_{0..2}$-part of π. We use $\ell_{0..2}$ as abbreviation for $\{\ell_0, \ell_1, \ell_2\}$. We can construct $S^M_{P_1}$, the modular FTS corresponding to process P_1 as follows:

$$\textbf{local } x,\ y\text{: } \textbf{integer where } x = y = 0$$

$$P_1 :: \left[\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: } \textbf{await } x < y+1 \\ \ell_2\text{: } x := x+1 \end{array}\right] \end{array}\right] \;\|\; P_2 :: \left[\begin{array}{l} m_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1\text{: } \textbf{await } y < x+1 \\ m_2\text{: } y := y+1 \end{array}\right] \end{array}\right]$$

Fig. 1.35. Program KEEPING-UP.

- V_1: $\{\pi, x, y\}$
- Θ_1: $\big(\pi \,\cap\, \ell_{0..2} \;=\; \{\ell_0\}\big) \;\wedge\; x = 0$
- $\mathcal{T}_1$: $\{\tau_{\ell_0}, \tau_{\ell_1}, \tau_{\ell_2}, \tau_E\}$

  with the following transition relations (after some simplifications):

  $$\rho_{\ell_0}\text{: } \mathit{move}(\ell_0, \ell_1) \;\wedge\; \mathit{pres}(x, y)$$
  $$\rho_{\ell_1}\text{: } \mathit{move}(\ell_1, \ell_2) \;\wedge\; x < y+1 \;\wedge\; \mathit{pres}(x, y)$$
  $$\rho_{\ell_2}\text{: } \mathit{move}(\ell_2, \ell_0) \;\wedge\; x' = x+1 \;\wedge\; \mathit{pres}(y)$$

$$\rho_E\colon\ (\pi' \cap \ell_{0..2} \ = \ \pi \cap \ell_{0..2}) \ \wedge \ pres(x)$$

In this relations, we used the following abbreviation:

$$move(\ell_i, \ell_j)\colon\ at\_\ell_i \ \wedge \ \pi' = \big(\pi - \{\ell_i\}\big) \ \cup \ \{\ell_j\}$$

- $\mathcal{J}_1\colon\ \{\tau_{\ell_0}, \tau_{\ell_1}, \tau_{\ell_2}\}$
- $\mathcal{C}_1\colon\ \emptyset$

The following is a modular computation of process P_1:

$$\begin{aligned}\widehat{\sigma}\colon\ & \big\langle \pi\colon\{\ell_0, m_0\},\ x\colon 0,\ y\colon 0\big\rangle \xrightarrow{\ell_0} \big\langle \pi\colon\{\ell_1, m_0\},\ x\colon 0,\ y\colon 0\big\rangle \xrightarrow{\ell_1} \\ & \big\langle \pi\colon\{\ell_2, m_0\},\ x\colon 0,\ y\colon 0\big\rangle \xrightarrow{\tau_E} \big\langle \pi\colon\{\ell_2, m_0\},\ x\colon 0,\ y\colon -1\big\rangle \xrightarrow{\ell_2} \\ & \big\langle \pi\colon\{\ell_0, m_0\},\ x\colon 1,\ y\colon -1\big\rangle \ \cdots\ .\end{aligned}$$

In a similar way, we can construct $S^M_{P_2}$, the modular FTS corresponding to process P_2. ▟

The following claim establishes a connection between computations of the entire program and modular computations of its processes.

Claim 1.3 (computations of programs and modular computations)

Every computation of a program is a modular computation of each of its top-level processes.

Thus, the set of computations of the entire program is a subset of the set of modular computations of each of its top-level processes.

Example Consider, for example, the following computation of program KEEPING-UP

$$\begin{aligned}\sigma\colon\ & \big\langle \pi\colon\{\ell_0, m_0\},\ x\colon 0,\ y\colon 0\big\rangle \xrightarrow{\ell_0} \big\langle \pi\colon\{\ell_1, m_0\},\ x\colon 0,\ y\colon 0\big\rangle \xrightarrow{\ell_1} \\ & \big\langle \pi\colon\{\ell_2, m_0\},\ x\colon 0,\ y\colon 0\big\rangle \xrightarrow{m_0} \big\langle \pi\colon\{\ell_2, m_1\},\ x\colon 0,\ y\colon 0\big\rangle \xrightarrow{\ell_2} \\ & \big\langle \pi\colon\{\ell_0, m_1\},\ x\colon 1,\ y\colon 0\big\rangle \xrightarrow{m_1} \big\langle \pi\colon\{\ell_0, m_2\},\ x\colon 1,\ y\colon 0\big\rangle \xrightarrow{m_2} \\ & \big\langle \pi\colon\{\ell_0, m_0\},\ x\colon 1,\ y\colon 1\big\rangle \xrightarrow{\ell_0} \big\langle \pi\colon\{\ell_1, m_0\},\ x\colon 1,\ y\colon 1\big\rangle \ \cdots\ .\end{aligned}$$

Viewed as a modular computation of process P_1, this computation can be presented as:

$$\begin{aligned}
\sigma_1\colon\ & \langle \pi\colon\{\ell_0, m_0\},\ x\colon 0,\ y\colon 0\rangle \xrightarrow{\ell_0} \langle \pi\colon\{\ell_1, m_0\},\ x\colon 0,\ y\colon 0\rangle \xrightarrow{\ell_1}\\
& \langle \pi\colon\{\ell_2, m_0\},\ x\colon 0,\ y\colon 0\rangle \xrightarrow{\tau_E} \langle \pi\colon\{\ell_2, m_1\},\ x\colon 0,\ y\colon 0\rangle \xrightarrow{\ell_2}\\
& \langle \pi\colon\{\ell_0, m_1\},\ x\colon 1,\ y\colon 0\rangle \xrightarrow{\tau_E} \langle \pi\colon\{\ell_0, m_2\},\ x\colon 1,\ y\colon 0\rangle \xrightarrow{\tau_E}\\
& \langle \pi\colon\{\ell_0, m_0\},\ x\colon 1,\ y\colon 1\rangle \xrightarrow{\ell_0} \langle \pi\colon\{\ell_1, m_0\},\ x\colon 1,\ y\colon 1\rangle\ \cdots .
\end{aligned}$$

Viewed as a modular computation of process P_2, this computation can be presented as:

$$\begin{aligned}
\sigma_2\colon\ & \langle \pi\colon\{\ell_0, m_0\},\ x\colon 0,\ y\colon 0\rangle \xrightarrow{\tau_E} \langle \pi\colon\{\ell_1, m_0\},\ x\colon 0,\ y\colon 0\rangle \xrightarrow{\tau_E}\\
& \langle \pi\colon\{\ell_2, m_0\},\ x\colon 0,\ y\colon 0\rangle \xrightarrow{m_0} \langle \pi\colon\{\ell_2, m_1\},\ x\colon 0,\ y\colon 0\rangle \xrightarrow{\tau_E}\\
& \langle \pi\colon\{\ell_0, m_1\},\ x\colon 1,\ y\colon 0\rangle \xrightarrow{m_1} \langle \pi\colon\{\ell_0, m_2\},\ x\colon 1,\ y\colon 0\rangle \xrightarrow{m_2}\\
& \langle \pi\colon\{\ell_0, m_0\},\ x\colon 1,\ y\colon 1\rangle \xrightarrow{\tau_E} \langle \pi\colon\{\ell_1, m_0\},\ x\colon 1,\ y\colon 1\rangle\ \cdots .
\end{aligned}$$

This illustrates that a computation of a program is a modular computation of each of its top-level processes. ■

The weak converse of Claim 1.3 is not true. There are modular computations of process P_i which do not correspond to computations of the entire program. This is illustrated by $\widehat{\sigma}$ the previously presented modular computation of process P_1 in program KEEPING-UP. This computation contains a state with $y = -1$ as a τ_E-successor of a state with $y = 0$. No such state can occur in a computation of KEEPING-UP. This shows that the definition of modular computations of process P_i allows more general contexts than the actual context provided by the program containing P_i. The actual context of P_1 within program KEEPING-UP is process P_2 which can never change y from a value of 0 to a value of -1.

On the other hand, the strong converse of Claim 1.3 is true. Let σ be a model (infinite state sequence) such that the interpretation of π is a subset of the locations of P. The valid converse of Claim 1.3 states that if σ is simultaneously a modular computation of every top-level process of P then σ is a computation of P. In **Problem 1.10**, we request the reader to prove this fact.

Modular Validity and a Basic Compositionality Rule

For a top-level process P_i within program P, we say that formula φ is *modularly valid over* P_i, denoted

$$P_i \underset{m}{\models} \varphi,$$

if φ holds over all modular computations of P_i. For example, the formula $\square(x \geq x^-)$, stating that x never decreases, is modularly valid over process P_1 of program KEEPING-UP (Fig. 1.42), while $\square(y \geq y^-)$ is modularly valid over process P_2 of the same program[2].

[2] x^- and y^- denote the values of x and y in the preceding state.

Rule COMP-B, presented in Fig. 1.36, infers the P-validity of a formula φ from the premise that φ is modularly valid over some top-level process of P.

For a program p, P_i a top-level process of P, and φ a temporal formula

$$\frac{P_i \underset{m}{\models} \varphi}{P \models \varphi}$$

Fig. 1.36. Rule COMP-B (basic compositionality).

Soundness

Let P be a program and P_i be a top-level process of P. Assume that formula φ is modularly valid over P_i, i.e. $P_i \underset{m}{\models} \varphi$. This means that φ holds over all modular computations of P_i. By Claim 1.3, every computation of P is also a modular computation of P_i. It follows that all computations of P satisfy φ and, hence, φ is P-valid. ■

Rule COMP-B can be used to reduce the goal of establishing $P \models \varphi$ into the subgoals of establishing several modular validities (not necessarily of the same formula φ. In **Problem 1.11** the reader is requested to establish this fact.

A Compositional Rule for Safety Properties

In theory, rule COMP-B is adequate for compositional verification of any temporal formula. In practice, however, its application often proves inconvenient and calls for additional temporal reasoning. Therefore, it is advantageous to derive more specific rules, each of which is tailored to deal with temporal formulas of particular classes.

In Fig. 1.37 we present rule COMP-S which can be used for compositional verification of safety formulas.

Premise CS1 states that χ is an invariant of the entire program P. Premise CS2 states that the entailment $\boxminus \chi \Rightarrow p$ is modularly valid over some top-level process P_i. From these two assumptions, the rule infers that p is an invariant of P.

Justification Assume that premises CS1 and CS2 hold and let σ be a computation of program P. By premise CS1, formula χ holds at all positions of σ. Since

For P_i, a top-level process of program P, and past formulas χ, p,

$$\begin{array}{ll} \text{CS1.} & P \models \Box\chi \\ \text{CS2.} & P_i \underset{m}{\models} \boxminus\chi \Rightarrow p \\ \hline & P \models \Box p \end{array}$$

Fig. 1.37. Rule COMP-S (compositional verification of safety properties).

every computation of P is also a modular computation of P_i, premise CS2 implies that the formula $\boxminus\chi \rightarrow p$ holds at all positions of σ.

Consider an arbitrary position $j \geq 0$ of σ. By D1, χ holds at all positions $k \leq j$ and, therefore $\boxminus\chi$ holds at j. By CS2, $\boxminus\chi \rightarrow p$ holds at j and, therefore, so does p.

We conclude that p holds at all positions of σ. ▟

Rule COMP-S is often used in an incremental style. As a first step we take $\chi = \mathrm{T}$ and prove $P_i \underset{m}{\models} \Box p_1$. From this the rule infers

$$P \models \Box p_1.$$

Next, we take $\chi = p_1$ and prove $P_i \underset{m}{\models} \boxminus p_1 \Rightarrow p_2$. This leads to

$$P \models \Box p_2,$$

which may be followed by additional steps.

The advantage of this proof pattern is that in each step we concentrate on proving a modular validity over a single process P_i. If P_i is only a small part of the program, each compositional verification step has to consider only a small fraction of the transitions in the complete program.

We illustrate the use of rule COMP-S on a simple example.

Example (program KEEPING-UP)

Consider program KEEPING-UP presented in Fig. 1.35. Process P_1 in this program repeatedly increments x, provided x does not exceed $y + 1$. In a symmetric way, process P_2 repeatedly increments y, provided y does not exceed $x + 1$.

We wish to prove for this program the invariance of the assertion $|x - y| \leq 1$, i.e.,

$$\Box\, \underbrace{|x - y| \leq 1}_{p}\,,$$

claiming that the difference between x and y never exceeds 1 in absolute value.

We prove this property by compositional verification, using rules INV-P and COMP-S. We first show the P-validities

$$P \vDash \Box(x \geq x^-) \qquad \text{and} \qquad P \vDash \Box(y \geq y^-),$$

and then the P-validities

$$P \vDash \Box(x \leq y+1) \qquad \text{and} \qquad P \vDash \Box(y \leq x+1).$$

The invariants $x \leq y+1$ and $y \leq x+1$ imply the desired P-validity

$$P \vDash \Box(|x-y| \leq 1).$$

For more details of this proof, we refer the reader to Section 4.3 of the SAFETY book. ▟

A Compositional Rule for Response Properties

Next, we present a rule that can support compositional verification of response properties and illustrate its use. This is rule COMP-R, presented in Fig. 1.38.

For P_i, a top-level process of program P, and past formulas χ, p, and q,

$$\begin{array}{ll} \text{CR1.} & P \vDash \Box\,\chi \\ \text{CR2.} & P_i \underset{m}{\vDash} p \Rightarrow \Diamond(q \vee \neg\chi) \\ \hline & P \vDash p \Rightarrow \Diamond\, q \end{array}$$

Fig. 1.38. Rule COMP-R (compositional verification of response properties).

Justification Assume that premises CR1 and CR2 hold and let σ be a computation of program P. By premise CR1, formula χ holds at all positions of σ. Since every computation of P is also a modular computation of P_i, premise MR2 implies that the formula $p \to \Diamond(q \vee \neg\chi)$ holds at all positions of σ. Let j be a p-position of σ. By MR2, there exists a position $k \geq j$ such that either q holds at k or χ is false at k. The second alternative is impossible, due to CR1. We conclude that every p-position is followed by a q-position and, therefore, $p \Rightarrow \Diamond\, q$ is valid over P. ▟

Example (program PING-PONG)

$$\textbf{local } x,\ y,\ z\text{: } \textbf{integer where } x = y = z = 0$$

$$P_1 :: \left[\begin{array}{l} \textbf{own out } x,\ z \\ \ell_0\text{: } x := 1 \\ \ell_1\text{: } \textbf{await } y > 0 \\ \ell_2\text{: } z := 1 \\ \ell_3\text{:} \end{array}\right] \ \Big\| \ P_2 :: \left[\begin{array}{l} \textbf{own out } y \\ m_0\text{: } \textbf{await } x > 0 \\ m_1\text{: } y := 1 \\ m_2\text{:} \end{array}\right]$$

Fig. 1.39. Program PING-PONG.

We illustrate the use of rule COMP-R for compositional verification of response properties on an example. In Fig. 1.39, we present program PING-PONG.

The two processes of this program maintain a coordination protocol. The protocol starts by P_1 setting x to 1 at statement ℓ_0. This is sensed by P_2 at m_0, and is responded to by setting y to 1 at statement m_1. This is sensed by P_1 at ℓ_1, and is responded to by setting z to 1 at ℓ_2.

We wish to establish for this program the response property

$$\Theta \ \Rightarrow \ \diamondsuit(z = 1).$$

We start by proving, using rule P-INV, the modular invariance

$$P_1 \ \underset{m}{\models} \ \Box(x \geq x^-).$$

This is easy to prove since the local formula $x \geq x^-$ is inductive over the modular FTS corresponding to process P_1.

By rule COMP-B we can infer

$$P \ \models \ \Box(x \geq x^-).$$

In a similar way, we establish $P_2 \underset{m}{\models} \Box(y \geq y^-)$, leading to

$$P \ \models \ \Box(y \geq y^-).$$

Now, we use rule RESP-J to prove

$$P_1 \ \underset{m}{\models} \ \underbrace{\Theta}_{p} \ \Rightarrow \ \diamondsuit \underbrace{x > 0}_{q} \ .$$

As the intermediate assertion and helpful transition, we take φ: at_ℓ_0 and τ_h: ℓ_0.

Using rule COMP-R with χ: T, we conclude

$$P \ \models \ \Theta \ \Rightarrow \ \diamondsuit(x > 0).$$

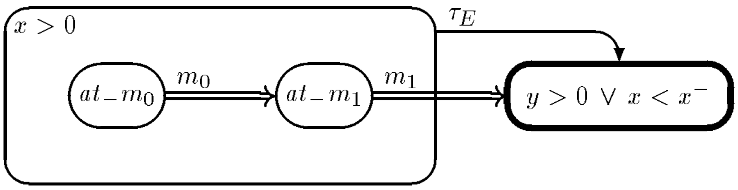
Next, we intend to establish

$$P_2 \underset{m}{\models} \quad x > 0 \ \Rightarrow \ \Diamond(y > 0 \ \vee \ x < x^-).$$

As a first step, we use rule INV to prove

$$P_2 \underset{m}{\models} \quad \Box \underbrace{at\_m_{0,1} \ \vee \ y > 0}_{\varphi_2} \ .$$

Having established the modular invariance of φ_2 over P_2, the following verification diagram proves that $x > 0 \Rightarrow \Diamond(y > 0 \vee x < x^-)$ is modularly valid over P_2.



Note that the diagram allows the possibility that the environment changes x from a positive value to a nonpositive one. However, such a change leads to a position satisfying $x < x^-$.

Now, use rule COMP-R with χ: $x \geq x^-$, p: $x > 0$, and q: $y > 0$, to conclude

$$P \ \models \ x > 0 \ \Rightarrow \ \Diamond(y > 0).$$

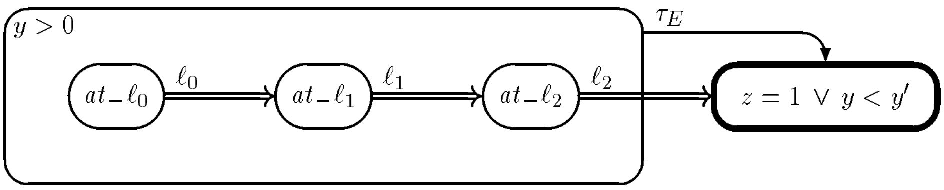
Next, we plan to establish

$$P_1 \underset{m}{\models} \quad y > 0 \ \Rightarrow \ \Diamond(z = 1 \ \vee \ y < y^-).$$

As a first step, we use rule INV to prove

$$P_1 \underset{m}{\models} \ \Box(at\_\ell_{0..2} \ \vee \ z = 1).$$

Having established this modular invariant, the following verification diagram proves that $y > 0 \Rightarrow \Diamond(z = 1 \vee y < y^-)$ is modularly valid over P_1.



Now, use rule COMP-R with χ: $y \geq y^-$, p: $y > 0$, and q: $z = 1$, to conclude

$$P \ \models \ y > 0 \ \Rightarrow \ \Diamond(z = 1).$$

Thus, we have shown that the three response properties $\Theta \Rightarrow \Diamond(x > 0)$, $x > 0 \Rightarrow \Diamond(y > 0)$, and $y > 0 \Rightarrow \Diamond(z = 1)$ are all valid over program PING-PONG. Using rule TRNS-R, we conclude

$$\Theta \;\Rightarrow\; \diamond(z = 1). \qquad \blacksquare$$

1.8 Guarantee Properties

In this and the following section we consider methods for proving properties belonging to the *guarantee* and *obligation* classes. Our approach to these classes is to consider them as special cases of the response class, and to use response rules with some simplifications for their verification.

As defined in Section 0.5 of the SAFETY book, guarantee properties are properties that can be specified by a formula of the form

$$\diamond\, r$$

for some past formula r.

Clearly, guarantee properties are a special case of response properties, $p \Rightarrow \diamond\, r$, where the antecedent p refers to the beginning of the computation. Consequently, an obvious rule for proving guarantee properties, is rule GUAR (Fig. 1.40).

For past formula r

$$\frac{\mathit{first} \;\wedge\; \Theta \;\;\Rightarrow\;\; \diamond\, r}{\diamond\, r}$$

Fig. 1.40. Rule GUAR (proving guarantee properties).

The rule requires as a premise a response property by which the initial condition of the program guarantees the eventual realization of r.

Example Consider system INC2, presented in Fig. 1.41.

We wish to establish the guarantee property

$$\diamond\big(x \geq 20 \;\wedge\; \diamondminus(x = 10)\big)$$

for system INC2, using rule GUAR. The premise of rule GUAR requires the response property

$$\overbrace{\mathit{first} \;\;\wedge\;\; \underbrace{x = 0}_{\Theta}}^{p} \;\;\Rightarrow\;\; \diamond\Big(\underbrace{x \geq 20 \;\wedge\; \diamondminus(x = 10)}_{r}\Big)\;.$$

This response property can be proven by rule WELL-J, using the intermediate past formula

V: $\{x\colon \textbf{integer}\}$

Θ: $x = 0$

$\mathcal{T}$: $\{\tau_I, \tau\}$ where $\rho_\tau\colon\ x' = x + 2$

$\mathcal{J}$: $\{\tau\}$

$\mathcal{C}$: $\emptyset$

Fig. 1.41. System INC2.

$$\varphi:\quad even(x)\ \land\ 0 \le x \le 18\ \land\ \big(x \ge 10\ \rightarrow\ \diamondsuit\!\!\!-(x = 10)\big),$$

the helpful transition τ, and the ranking function δ: $|20 - x|$.

Let us establish premise JW1 of rule WELL-J. It requires showing

$$\underbrace{\cdots\ \land\ x = 0}_{p}\quad \Rightarrow\quad \cdots\ \lor\ \underbrace{even(x)\ \land\ 0 \le x \le 18\ \land\ \big(x \ge 10\ \rightarrow\ \diamondsuit\!\!\!-(x = 10)\big)}_{\varphi}\ .$$

Clearly, $x = 0$ entails all three conjuncts comprising φ. Note that, in this case, we do not use the conjunct *first* which is part of p.

By rule GUAR, we conclude that property

$$\diamondsuit\big(x \ge 20\ \land\ \diamondsuit\!\!\!-(x = 10)\big)$$

is valid over system INC2. ▟

The premise of rule GUAR contains *first* as part of the antecedent. Its purpose is to ensure that we only consider Θ at the beginning of the computation. As illustrated in the last example, in many cases we do not use this conjunct and simply prove $\Theta \Rightarrow \diamondsuit\, p$. There are, however, some cases in which this conjunct is necessary, as illustrated below.

Example Consider the simple program

local x: **integer where** $x = 1$

ℓ_0: **loop forever do** $[\ell_1\colon$ **skip**; $\ell_2\colon\ x := -x]$.

We consider the guarantee property

$$\psi:\quad \diamondsuit\big(at\_\ell_2\ \land\ \boxminus(x = 1)\big).$$

This property states that every computation of the program contains a position j satisfying at_ℓ_2, and such that $x = 1$ at all positions $i \le j$.

This property is certainly valid for the program. To prove it, we have to establish the response property

$$\mathit{first} \;\wedge\; \mathit{at\_}\ell_0 \;\wedge\; x = 1 \quad\Rightarrow\quad \diamondsuit\big(\mathit{at\_}\ell_2 \;\wedge\; \boxminus(x = 1)\big).$$

Note, however, that the *first* conjunct is essential in this case, since the formula

$$\mathit{at\_}\ell_0 \;\wedge\; x = 1 \quad\Rightarrow\quad \diamondsuit\big(\mathit{at\_}\ell_2 \;\wedge\; \boxminus(x = 1)\big)$$

is not valid over the program.

Consider position i in the computation, corresponding to the third visit to ℓ_2. At this position $x = 1$, but there exist earlier positions in which $x = -1$. Therefore, there exists no position later than i, at which $\mathit{at\_}\ell_2 \wedge \boxminus(x = 1)$ holds. It follows that the implication $\mathit{at\_}\ell_0 \wedge x = 1 \rightarrow \diamondsuit\big(\mathit{at\_}\ell_2 \wedge \boxminus(x = 1)\big)$ does not hold at i.

To prove that the full premise

$$\underbrace{\mathit{first} \;\wedge\; \mathit{at\_}\ell_0 \;\wedge\; x = 1}_{p} \quad\Rightarrow\quad \diamondsuit\Big(\underbrace{\mathit{at\_}\ell_2 \;\wedge\; \boxminus(x = 1)}_{r}\Big)\,,$$

is valid, we may use rule CHAIN-J (Fig. 1.7) with the following intermediate past formulas and helpful transitions:

$$\begin{array}{rll} \varphi_2: & \mathit{at\_}\ell_0 \;\wedge\; \boxminus(x = 1) & \qquad \tau_2:\ \ell_0 \\ \varphi_1: & \mathit{at\_}\ell_1 \;\wedge\; \boxminus(x = 1) & \qquad \tau_1:\ \ell_1 \\ \varphi_0 = r: & \mathit{at\_}\ell_2 \;\wedge\; \boxminus(x = 1) & \qquad \tau_0:\ \ell_2. \end{array}$$

Premise J1 requires showing

$$\underbrace{\mathit{first} \;\wedge\; \mathit{at\_}\ell_0 \;\wedge\; x = 1}_{p} \quad\Rightarrow\quad \cdots \;\vee\; \underbrace{\mathit{at\_}\ell_0 \;\wedge\; \boxminus(x = 1)}_{\varphi_2}\,.$$

Clearly, the antecedent implies $\mathit{at\_}\ell_0$. To see that it also implies $\boxminus(x = 1)$, we observe that under *first*, any past formula p is congruent to $(p)_0$, the initial version of p. Since the initial version of $\boxminus(x = 1)$ is

$$\big(\boxminus(x = 1)\big)_0 \;=\; (x = 1),$$

the right-hand side simplifies to $\mathit{at\_}\ell_0 \wedge x = 1$ which is entailed by the left-hand side.

Another premise that has to be checked is J3 for transition ℓ_0,

$$\rho_{\ell_0} \;\wedge\; \underbrace{\mathit{at\_}\ell_0 \;\wedge\; \boxminus(x = 1)}_{\varphi_2} \quad\Rightarrow\quad \cdots \;\vee\; \underbrace{\mathit{at}'\_\ell_1 \;\wedge\; x' = 1 \;\wedge\; \boxminus(x = 1)}_{\varphi_1'}\,.$$

Since ρ_{ℓ_0} implies at'_ℓ_1 and $x' = x$, and $\boxminus(x = 1)$ implies $x = 1$, the entailment is valid.

The rest of the premises are proven in a similar way. This establishes the validity of $\diamondsuit\big(at\_\ell_2 \wedge \boxminus(x=1)\big)$. ▪

Completeness of Rule GUAR

Rule GUAR is obviously sound, which means that the P-validity of the premise $\mathit{first} \wedge \Theta \Rightarrow \diamondsuit\, r$ implies the P-validity of the conclusion $\diamondsuit\, r$.

The rule is also *complete*, which means that the P-validity of the conclusion implies the P-validity of the premise. Consider σ, an arbitrary computation of program P. By the assumption that $\diamondsuit\, r$ is P-valid, there exists a position k at which r holds. For σ to satisfy the premise we have to show that every position $i \geq 0$, satisfying $\mathit{first} \wedge \Theta$, is followed by a position j, $j \geq i$, satisfying r. Since 0 is the only position satisfying $\mathit{first} \wedge \Theta$, we can take j to be k.

Completeness of rule GUAR is important because it tells us that the rule is adequate for proving all P-valid guarantee formulas.

1.9 Obligation Properties

Before studying the class of obligation properties, we introduce a special class of response formulas.

Escape Formulas

Some response properties are naturally expressed by formulas of the form

$$p \quad \Rightarrow \quad \square\, q \;\vee\; \diamondsuit\, r,$$

for past formulas p, q, and r.

This formula claims that, following a p-state, either q will hold forever or r eventually occurs. We may view such a formula as stating that, following p, q should hold continually unless we escape to a state that eventually leads to r. Consequently, we refer to formulas of this form as *escape formulas*.

To see that this formula specifies a response property, observe that it is equivalent to

$$\neg q \;\wedge\; (\neg r)\,\Sigma\,(p \wedge \neg r) \quad \Rightarrow \quad \diamondsuit\, r.$$

In this form, the formula states that every $\neg q$-position preceded by a p-position such that no r has occured since, must be followed by an r-position.

While, in principle, it is possible to use the general response rules to establish escape formulas, it is more convenient to use a special rule, presented in Fig. 1.42.

For past formulas p, q, r, and φ

$$\begin{array}{ll} \text{E1.} & p \;\Rightarrow\; q\,\mathcal{W}\,\varphi \\ \text{E2.} & \varphi \;\Rightarrow\; \Diamond\, r \\ \hline & p \;\Rightarrow\; \Box\, q \;\vee\; \Diamond\, r \end{array}$$

Fig. 1.42. Rule ESC (escape).

Rule ESC uses an auxiliary past formula φ. Premise E1 requires that, following a p-position, either q will hold forever or q will hold until an occurrence of φ. Premise E2 requires that every φ-position is followed by an r-position. Typically, we prove E1 by rule P-WAIT, a past version of rule WAIT (Fig. 3.3 of the SAFETY book), and E2 by appropriate response rules.

Example Consider program MAY-HALT of Fig. 1.43. This trivial program has a nondeterministic choice at ℓ_1 between getting deadlocked at ℓ_2 or, taking the ℓ_1^a branch, proceeding to ℓ_3. Consequently, the program has some computations that reach ℓ_2 and stay there forever, and some computations that never halt.

$$\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: } \left[\begin{array}{l} \ell_1^a\text{: } \textbf{skip} \\ \quad \textbf{or} \\ \ell_1^b\text{: } \textbf{skip};\ \ell_2\text{: } \textbf{halt} \end{array}\right] \\ \ell_3\text{: } \textbf{skip} \\ \ell_4\text{: } \textbf{skip} \end{array}\right] \end{array}$$

Fig. 1.43. Program MAY-HALT (possible deadlock).

We use rule ESC to prove the property

$$\psi_1\text{:}\quad \underbrace{at\_\ell_{0,1}}_{p} \;\Rightarrow\; \Box\, \underbrace{at\_\ell_{0..2}}_{q} \;\vee\; \Diamond\, \underbrace{at\_\ell_4}_{r}$$

for this program.

Take

$$\varphi: \quad at\_\ell_3.$$

Consider the two premises of rule ESC.

- Premise E1

  This premise requires

$$\underbrace{at\_\ell_{0,1}}_{p} \quad \Rightarrow \quad \underbrace{at\_\ell_{0..2}}_{q} \;\; \mathcal{W} \;\; \underbrace{at\_\ell_3}_{\varphi}\;.$$

  It is straightforward to derive this property by rule WAIT.

- Premise E2

$$\underbrace{at\_\ell_3}_{\varphi} \quad \Rightarrow \quad \Diamond\, \underbrace{at\_\ell_4}_{r}\;.$$

  A single application of rule RESP-J establishes this property.

This establishes the considered escape property. ▟

From Escape to Obligation

The (simple) obligation class includes all the properties that can be specified by a formula of the form

$$\Box\, q \;\vee\; \Diamond\, r,$$

for past formulas q and r.

We observe that such a formula can be rewritten as

$$\underbrace{\mathit{first} \;\wedge\; \Theta}_{p} \quad \Rightarrow \quad \Box\, q \;\vee\; \Diamond\, r,$$

which represents it as a special case of an escape formula.

This observation inspires rule OBL (Fig. 1.44) for proving obligation proper-

For past formulas q, r, and φ,

$$\begin{array}{ll} \text{O1.} & \mathit{first} \land \Theta \;\; \Rightarrow \;\; q\,\mathcal{W}\,\varphi \\ \text{O2.} & \varphi \;\Rightarrow\; \Diamond\, r \\ \hline & \Box\, q \;\lor\; \Diamond\, r \end{array}$$

Fig. 1.44. Rule OBL (proving obligation properties).

ties.

Premise O1 requires that, from the beginning of the computation, q holds continuously and can be interrupted only at a position satisfying φ. Premise O2 states that φ guarantees an eventual r. Consequently, q can be interrupted only when r is guaranteed. It follows that $q\,\mathcal{W}\,(\Diamond\, r)$ is valid. By properties of the waiting-for operator, we may deduce $\Box\, q \lor \Diamond\, r$.

Example (incrementor-decrementor)

Consider Program INC-DEC presented in Fig. 1.45, which nondeterministically increments or decrements an integer variable y.

$$\begin{array}{l} \textbf{local}\ x\text{: }\textbf{boolean where}\ x = \mathrm{T} \\ \qquad\quad y\text{: }\textbf{integer where}\ y = 10 \end{array}$$

$$\begin{array}{l} \ell_0\text{: }\textbf{loop forever do} \\ \qquad \ell_1\text{: } \left[\begin{array}{l} \ell_1^a\text{: } \langle \textbf{when}\ x\ \textbf{do}\ y := y+1 \rangle \\ \qquad\quad \textbf{or} \\ \ell_1^b\text{: } \langle \textbf{when}\ x\ \textbf{do}\ x := \mathrm{F} \rangle \\ \qquad\quad \textbf{or} \\ \ell_1^c\text{: } \langle \textbf{when}\ \neg x\ \textbf{do}\ y := y-1 \rangle \end{array}\right] \end{array}$$

Fig. 1.45. Program INC-DEC (nondeterministic incrementor-decrementor).

We wish to prove for this program the obligation property

$$\Box\, \underbrace{y \geq 10}_{q} \;\lor\; \Diamond\, \underbrace{y = 0}_{r} \,.$$

Intending to apply rule OBL, it only remains to identify the intermediate formula φ. The main characterization of φ is that it describes the event whose occurrence guarantees the eventual realization of r.

Examining the program, we see that the first moment we realize that $y = 0$ is going to happen is when x becomes false. Consequently we take

$$\varphi: \quad \neg x \ \land \ y > 0.$$

The premises that have to be verified are as follows:

- Premise O1

To establish this premise it suffices to prove

$$\underbrace{\cdots \land \ \Theta}_{\widehat{p}} \quad \Rightarrow \quad \underbrace{y \geq 10}_{\widehat{q}=q} \ \ \mathcal{W} \ \ \underbrace{\neg x \ \land \ y > 0}_{\widehat{r}=\varphi} \ .$$

To prove this we use rule WAIT (Fig. 3.3 of the SAFETY book) with the intermediate assertion

$$\widehat{\varphi}: \quad x \ \land \ y \geq 10.$$

The three premises of rule WAIT require

$$\text{W1.} \qquad \underbrace{at\_\ell_0 \ \land \ x \ \land \ y = 10}_{\widehat{p}} \quad \rightarrow \quad \underbrace{x \ \land \ y \geq 10}_{\widehat{\varphi}} \ \lor \ \cdots ,$$

which is obviously state valid.

$$\text{W2.} \qquad \underbrace{x \ \land \ y \geq 10}_{\widehat{\varphi}} \quad \rightarrow \quad \underbrace{y \geq 10}_{\widehat{q}} ,$$

which is trivially state valid.

$$\text{W3.} \qquad \rho_\tau \ \land \ \underbrace{x \ \land \ y \geq 10}_{\widehat{\varphi}} \quad \rightarrow \quad \underbrace{x' \ \land \ y' \geq 10}_{\widehat{\varphi}'} \ \lor \ \underbrace{\neg x' \ \land \ y' > 0}_{\widehat{r}'} ,$$

for all transitions τ in the program. It is not difficult to see that all three requirements are valid.

- Premise O2

To establish this premise we have to prove

$$\underbrace{\neg x \ \land \ y > 0}_{\varphi} \quad \Rightarrow \quad \diamondsuit \underbrace{y = 0}_{r} \ .$$

This is proven by the RANK verification diagram presented in Fig. 1.46.

This concludes the proof of

$$\square(y \geq 10) \ \lor \ \diamondsuit(y = 0). \qquad ▟$$

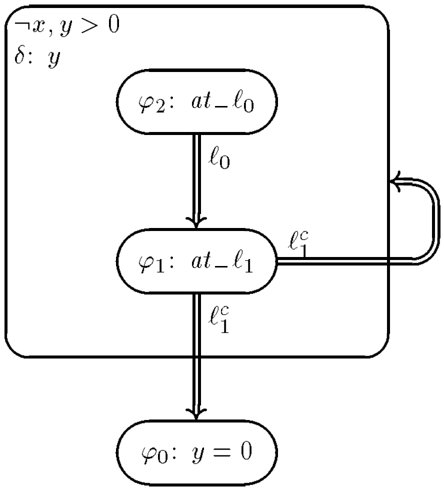


Fig. 1.46. Verification diagram for $\neg x \wedge y > 0 \Rightarrow \diamondsuit(y = 0)$.

A general obligation property is a conjunction of the form

$$\bigwedge_{i=1}^{n} (\square\, q_i \;\vee\; \diamondsuit\, r_i).$$

Consequently, to prove the validity of such a formula, it is sufficient (and necessary) to prove the validity of each conjunct, which is a simple obligation property.

Completeness of rule OBL

Rule OBL is *complete* for proving (simple) obligation properties. This means that, whenever $\square\, q \vee \diamondsuit\, r$ is P-valid, we can find a past formula φ, such that premises O1 and O2 are also P-valid. The choice we can always make is taking

$$\varphi: \qquad r \;\vee\; (\neg q \;\wedge\; \boxminus\, \neg r).$$

We will show that, if the property $\square\, q \vee \diamondsuit\, r$ is P-valid, then the two premises of rule OBL are also P-valid for this choice of φ.

- Premise O1

For premise O1 it suffices to show that

$$q \;\mathcal{W}\; \underbrace{r \;\vee\; (\neg q \;\wedge\; \boxminus\, \neg r)}_{\varphi}$$

is P-valid.

This formula states that either q holds forever, or it is interrupted by an r, or it is interrupted by a $\neg q$-position which is not preceded by any r-position. This formula is valid in general so, in particular, it holds over all computations.

- Premise O2

This premise requires

$$\underbrace{r \ \vee\ (\neg q \ \wedge\ \boxminus \neg r)}_{\varphi} \ \Rightarrow\ \Diamond r,$$

for which it is sufficient to show

$$\neg q \ \wedge\ \boxminus \neg r \ \Rightarrow\ \Diamond r.$$

Assume to the contrary, that there exists a computation σ and a position i such that $\neg q \wedge \boxminus \neg r$ holds at i, but r does not hold at any position $j \geq i$. Since $\neg q \wedge \boxminus r$ holds at i, r does not occur at any position $j < i$. Therefore σ does not satisfy $\Diamond r$. On the other hand, since $\neg q$ at i, σ also does not satisfy $\Box q$. This contradicts our assumption that $\Box q \vee \Diamond r$ is P-valid.

Consequently, premise O2 is also P-valid.

As we continuously remind the reader, the auxiliary formula φ constructed during a proof of completeness is not necessarily the one we recommend for actual use. In practice, we can almost always find better assertions.

Problems

Problem 1.1 (three values) page 22

Prove accessibility for process P_1 of program MUX-VAL-3 of Fig. 1.47. This program uses the shared integer variables y_1 and y_2. Obviously, these variables can only assume one of the values $\{-1, 0, 1\}$.

Accessibility for P_1 is stated by the response formula

$$at\_\ell_2 \ \Rightarrow\ \Diamond\, at\_\ell_4.$$

Problem 1.2 (bakery algorithms) page 25

(a) Prove accessibility for process P_1 of program MUX-BAK-A of Fig. 1.48. Note that the two processes are not exactly symmetric due to the difference between statements ℓ_3^b and m_3^b.

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. Statements ℓ_2 and m_2 ensure that the number assigned to y_i, $i = 1, 2$, is greater than the current value of y_j, $j \neq i$.

$$\textbf{local } y_1,\ y_2\text{: }\textbf{integer where } y_1 = y_2 = 0$$

$$\left[\begin{array}{l} \ell_0\text{: }\textbf{loop forever do} \\ \left[\begin{array}{l} \ell_1\text{: }\textbf{noncritical} \\ \ell_2\text{: }\left\langle\begin{array}{l}\textbf{if } y_2 = -1 \\ \textbf{then } y_1 := -1 \\ \textbf{else } y_1 := 1\end{array}\right\rangle \\ \ell_3\text{: }\textbf{await } y_2 \neq y_1 \\ \ell_4\text{: }\textbf{critical} \\ \ell_5\text{: } y_1 := 0 \end{array}\right] \end{array}\right] \quad \| \quad \left[\begin{array}{l} m_0\text{: }\textbf{loop forever do} \\ \left[\begin{array}{l} m_1\text{: }\textbf{noncritical} \\ m_2\text{: }\left\langle\begin{array}{l}\textbf{if } y_1 = -1 \\ \textbf{then } y_2 := 1 \\ \textbf{else } y_2 := -1\end{array}\right\rangle \\ m_3\text{: }\textbf{await } y_1 \neq -y_2 \\ m_4\text{: }\textbf{critical} \\ m_5\text{: } y_2 := 0 \end{array}\right] \end{array}\right]$$

Fig. 1.47. Program MUX-VAL-3.

$$\textbf{local } y_1,\ y_2\text{: }\textbf{integer where } y_1 = y_2 = 0$$

$$\left[\begin{array}{l} \ell_0\text{: }\textbf{loop forever do} \\ \left[\begin{array}{l} \ell_1\text{: }\textbf{noncritical} \\ \ell_2\text{: } y_1 := y_2 + 1 \\ \ell_3\text{: }\left[\begin{array}{l}\ell_3^a\text{: }\textbf{await } y_2 = 0 \\ \qquad \textbf{or} \\ \ell_3^b\text{: }\textbf{await } y_1 \leq y_2\end{array}\right] \\ \ell_4\text{: }\textbf{critical} \\ \ell_5\text{: } y_1 := 0 \end{array}\right] \end{array}\right] \quad \| \quad \left[\begin{array}{l} m_0\text{: }\textbf{loop forever do} \\ \left[\begin{array}{l} m_1\text{: }\textbf{noncritical} \\ m_2\text{: } y_2 := y_1 + 1 \\ m_3\text{: }\left[\begin{array}{l}m_3^a\text{: }\textbf{await } y_1 = 0 \\ \qquad \textbf{or} \\ m_3^b\text{: }\textbf{await } y_2 < y_1\end{array}\right] \\ m_4\text{: }\textbf{critical} \\ m_5\text{: } y_2 := 0 \end{array}\right] \end{array}\right]$$

Fig. 1.48. Program MUX-BAK-A.

(b) Program MUX-BAK-A does not obey the LCR restriction. In particular, statements ℓ_2 and m_2 each contain two critical references: to y_1 and to y_2. To correct this situation, we propose program MUX-BAK-C of Fig. 1.49. This LCR-program contains two additional *await* statements that ensure that processes do not wait too long at locations ℓ_3 or m_3. Show that program MUX-BAK-C guarantees accessibility for process P_1.

Problem 1.3 (variants of Dekker's algorithm)) page 32

$$
\begin{array}{l}
\textbf{local } y_1,\ y_2,\ t_1,\ t_2\text{: } \textbf{integer where } y_1 = y_2 = 0 \\
\qquad\quad x_1,\ x_2 \qquad\ \ \text{: } \textbf{integer where } x_1 = x_2 = 0
\end{array}
$$

$$
\left[\begin{array}{l}
\ell_0\text{: } \textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
\ell_1\text{: } \textbf{noncritical} \\
\ell_2\text{: } x_1 := 1 \\
\ell_3\text{: } t_1 := y_2 + 1 \\
\ell_4\text{: } y_1 := t_1 \\
\ell_5\text{: } x_1 := 0 \\
\ell_6\text{: } \textbf{await } x_2 = 0 \\
\ell_7\text{: } \left[\begin{array}{c}
\textbf{await } y_2 = 0 \\
\textbf{or} \\
\textbf{await } y_1 \leq y_2
\end{array}\right] \\
\ell_8\text{: } \textbf{critical} \\
\ell_9\text{: } y_1 := 0
\end{array}\right]
\end{array}\right]
\;\Big\|\;
\left[\begin{array}{l}
m_0\text{: } \textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
m_1\text{: } \textbf{noncritical} \\
m_2\text{: } x_2 := 1 \\
m_3\text{: } t_2 := y_1 + 1 \\
m_4\text{: } y_2 := t_2 \\
m_5\text{: } x_2 := 0 \\
m_6\text{: } \textbf{await } x_1 = 0 \\
m_7\text{: } \left[\begin{array}{c}
\textbf{await } y_1 = 0 \\
\textbf{or} \\
\textbf{await } y_2 < y_1
\end{array}\right] \\
m_8\text{: } \textbf{critical} \\
m_9\text{: } y_2 := 0
\end{array}\right]
\end{array}\right]
$$

Fig. 1.49. Program MUX-BAK-C.

(a) Prove accessibility for process P_1 of program MUX-DEK-A of Fig. 1.50. That is, show that the response formula

$$at\_\ell_2 \;\Rightarrow\; \diamondsuit\, at\_\ell_7$$

is valid over MUX-DEK-A.

(b) Prove accessibility for process P_1 of program MUX-DEK-B of Fig. 1.51. That is, show that the response formula

$$at\_\ell_2 \;\Rightarrow\; \diamondsuit\, at\_\ell_7$$

is valid over MUX-DEK-B.

Problem 1.4 (condensed form of ranking functions) page 47

In the text, it was suggested that a ranking function $\delta = (d_1, d_2)$, where d_1 and $d_2 < M$ are natural numbers, can always be replaced by the ranking function $\widehat{\delta} = M \cdot d_1 + d_2$. Show that if both $d_2 < M$ and $d_2' < M$, then

$$\widehat{\delta} \;=\; M \cdot d_1 + d_2 > \widehat{\delta}' \;=\; M \cdot d_1' + d_2' \qquad \text{iff} \qquad \delta\text{: } (d_1, d_2) \succ \delta'\text{: } (d_1', d_2').$$

Problem 1.5 (rule with relaxed premise JW2) page 52

local $y_1,\ y_2,\ t$: **integer where** $y_1 = y_2 = 0,\ t = 1$

$$
\left[\begin{array}{l}
\ell_0\!: \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\!: \textbf{noncritical} \\
\ell_2\!: y_1 := 2 \\
\ell_3\!: \textbf{while } y_2 = 2 \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_4\!: y_1 := 1 \\
\ell_5\!: \left[\begin{array}{c}\textbf{await } t = 1 \\ \textbf{or} \\ \textbf{await } y_2 = 0\end{array}\right] \\
\ell_6\!: y_1 := 2
\end{array}\right] \\
\ell_7\!: \textbf{critical} \\
\ell_8\!: t := 2 \\
\ell_9\!: y_1 := 0
\end{array}\right]
\end{array}\right]
\quad \| \quad
\left[\begin{array}{l}
m_0\!: \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
m_1\!: \textbf{noncritical} \\
m_2\!: y_2 := 2 \\
m_3\!: \textbf{while } y_1 = 2 \textbf{ do} \\
\quad \left[\begin{array}{l}
m_4\!: y_2 := 1 \\
m_5\!: \left[\begin{array}{c}\textbf{await } t = 2 \\ \textbf{or} \\ \textbf{await } y_1 = 0\end{array}\right] \\
m_6\!: y_2 := 2
\end{array}\right] \\
m_7\!: \textbf{critical} \\
m_8\!: t := 1 \\
m_9\!: y_2 := 0
\end{array}\right]
\end{array}\right]
$$

Fig. 1.50. Program MUX-DEK-A (a variant of Dekker's algorithm).

Consider a version of rule WELL-J in which premise JW2 has been replaced by the weaker premise

$$\widehat{\text{JW}2}. \qquad \rho_\tau \ \wedge\ \varphi_i \quad \rightarrow \quad q' \ \vee\ \bigvee_{j=1}^{k} \left(\varphi'_j \ \wedge\ \delta_i \succeq \delta'_j\right),$$

where $\delta_i \succeq \delta'_j$ stands for $(\delta_i \succ \delta'_j) \vee (\delta_i = \delta'_j)$. This premise requires that either q is achieved by τ, or the rank does not increase and *some* assertion φ_j (not necessarily φ_i) holds after the transition.

Show that the resulting rule is unsound. That is, show a property that satisfies premises JW1, $\widehat{\text{JW}2}$, JW3, and JW4, over a given program, and yet is invalid. This will show that persistence of helpful transitions is essential.

Problem 1.6 (integer division) page 59

Program IDIV of Fig. 1.52 accepts two positive integers in variables x and y and places in variable z their integer quotient $x\ div\ y$, and in variable w the remainder of their division $x\ mod\ y$. Prove total correctness of program IDIV, which can be specified by the response formula

$$\Theta \ \Rightarrow\ \Diamond(at\_\ell_5 \ \wedge\ x = z \cdot y + w \ \wedge\ 0 \le w < y).$$

$$\textbf{local } y_1,\ y_2, t\text{: } \textbf{integer where } y_1 = 0,\ y_2 = 0,\ t = 1$$

$$\left[\begin{array}{l}
\ell_0\text{: } \textbf{loop forever do} \\
\quad\left[\begin{array}{l}
\ell_1\text{: } \textbf{noncritical} \\
\ell_2\text{: } y_1 := 2 \\
\ell_3\text{: } \textbf{while } y_2 = 2 \textbf{ do} \\
\quad\left[\begin{array}{l}
\ell_4\text{: } \textbf{if } t = 2 \textbf{ then} \\
\quad\left[\begin{array}{l}
\ell_5\text{: } y_1 := 1 \\
\ell_6\text{: } \textbf{await } t = 1 \\
\ell_7\text{: } y_1 := 2
\end{array}\right]
\end{array}\right] \\
\ell_8\text{: } \textbf{critical} \\
\ell_9\text{: } t := 2 \\
\ell_{10}\text{: } \textbf{await } y_2 \neq 1 \\
\ell_{11}\text{: } y_1 := 0
\end{array}\right]
\end{array}\right]
\quad \| \quad
\left[\begin{array}{l}
m_0\text{: } \textbf{loop forever do} \\
\quad\left[\begin{array}{l}
m_1\text{: } \textbf{noncritical} \\
m_2\text{: } y_2 := 2 \\
m_3\text{: } \textbf{while } y_1 = 2 \textbf{ do} \\
\quad\left[\begin{array}{l}
m_4\text{: } \textbf{if } t = 1 \textbf{ then} \\
\quad\left[\begin{array}{l}
m_5\text{: } y_2 := 1 \\
m_6\text{: } \textbf{await } t = 2 \\
m_7\text{: } y_2 := 2
\end{array}\right]
\end{array}\right] \\
m_8\text{: } \textbf{critical} \\
m_9\text{: } t := 1 \\
m_{10}\text{: } \textbf{await } y_1 \neq 1 \\
m_{11}\text{: } y_2 := 0
\end{array}\right]
\end{array}\right]$$

Fig. 1.51. Program MUX-DEK-B.

This formula states that every computation of program IDIV terminates (i.e., reaches the terminal location ℓ_5) with values of z, w satisfying $x = z \cdot y + w$ and $0 \leq w < y$.

$$\begin{array}{lll}
\textbf{in} & x,\ y & : \textbf{integer where } x > 0,\ y > 0 \\
\textbf{local } t & & : \textbf{integer} \\
\textbf{out} & z,\ w & : \textbf{integer where } z = w = 0
\end{array}$$

$$\begin{array}{l}
\ell_0\text{: } t := x \\
\ell_1\text{: } \textbf{while } t > 0 \textbf{ do} \\
\qquad \ell_2\text{: } \textbf{if } w + 1 = y \\
\qquad\qquad \textbf{then } \ell_3\text{: } (z,\ w,\ t) := (z + 1,\ 0,\ t - 1) \\
\qquad\qquad \textbf{else } \ \ell_4\text{: } (z,\ w,\ t) := (z,\ w + 1,\ t - 1) \\
\ell_5\text{:}
\end{array}$$

Fig. 1.52. Program IDIV (integer division).

Problem 1.7 (greatest common divisor) page 59

Program GCDM of Fig. 1.53 accepts two positive integers in variables x_1 and x_2. It computes in z the greatest common divisor (gcd) of x_1 and x_2, and in variables w_1 and w_2 two integers which express z as a linear combination of the inputs x_1 and x_2. Prove total correctness of program GCDM, which can be stated by the response formula

$$\Theta \;\Rightarrow\; \Diamond\big(at\_\ell_6 \;\wedge\; z = gcd(x_1, x_2) \;\wedge\; z = w_1 \cdot x_1 + w_2 \cdot x_2\big).$$

$$\begin{array}{lll}
\textbf{in} & x_1,\ x_2 & : \textbf{integer where } x_1 > 0,\ x_2 > 0 \\
\textbf{local} & y_1,\ y_2,\ t_1,\ t_2,\ t_3,\ t_4,\ u & : \textbf{integer} \\
\textbf{out} & z,\ w & : \textbf{integer}
\end{array}$$

$$\begin{array}{l}
\ell_0\text{: } (y_1,\ y_2,\ t_1,\ t_2,\ t_3,\ t_4) := (x_1,\ x_2,\ 1,\ 0,\ 0,\ 1) \\
\ell_1\text{: } (y_1,\ y_2,\ u) := (y_2 \mathbin{mod} y_1,\ y_1,\ y_2 \mathbin{div} y_1) \\
\ell_2\text{: } \textbf{while } y_1 \neq 0 \textbf{ do} \\
\qquad \left[\begin{array}{l}
\ell_3\text{: } (t_1,\ t_2,\ t_3,\ t_4) := (t_2 - u \cdot t_1,\ t_1,\ t_4 - u \cdot t_3,\ t_3) \\
\ell_4\text{: } (y_1,\ y_2,\ u) := (y_2 \mathbin{mod} y_1,\ y_1,\ y_2 \mathbin{div} y_1)
\end{array}\right] \\
\ell_5\text{: } (z,\ w_1,\ w_2) := (y_2,\ t_2,\ t_3) \\
\ell_6\text{: }
\end{array}$$

Fig. 1.53. Program GCDM(greatest common divisor with multipliers).

The program uses the operation *div* of integer division and the operation *mod* which computes the remainder of an integer division. In your proof you may use the following properties of the *gcd* function which hold for every nonzero integers m and n (possibly negative):

$$\begin{aligned}
gcd(m, n) &= gcd(m - n, n) \quad \text{for every } m \neq n \\
gcd(m, m) &= |m|.
\end{aligned}$$

Problem 1.8 (computing the *gcd* and *lcm*) page 59

Program GCDLCM of Fig. 1.54 accepts two positive integers in variables x_1 and x_2. It computes in variable z their greatest common divisor and in variable w their least common multiple. Prove total correctness of program GCDLCM, which can be stated by the response formula

$$\Theta \;\Rightarrow\; \Diamond\big(at\_\ell_7 \;\wedge\; z = gcd(x_1, x_2) \;\wedge\; w = lcm(x_1, x_2)\big).$$

$$
\begin{array}{lll}
\textbf{in} & x_1,\ x_2 & : \textbf{integer where } x_1 > 0,\ x_2 > 0 \\
\textbf{local} & y_1,\ y_2,\ y_3,\ y_4 & : \textbf{integer} \\
\textbf{out} & z,\ w & : \textbf{integer}
\end{array}
$$

$$
\begin{array}{l}
\ell_0\colon\ (y_1,\ y_2,\ y_3,\ y_4) := (x_1,\ x_2,\ x_2,\ 0) \\
\ell_1\colon\ \textbf{while } y_1 \neq y_2 \textbf{ do} \\
\qquad \left[\begin{array}{l}
\ell_2\colon\ \textbf{if } y_1 > y_2 \textbf{ then} \\
\qquad \ell_3\colon\ (y_1,\ y_4) := (y_1 - y_2,\ y_3 + y_4) \\
\ell_4\colon\ \textbf{if } y_1 < y_2 \textbf{ then} \\
\qquad \ell_5\colon\ (y_2,\ y_3) := (y_2 - y_1,\ y_3 + y_4)
\end{array}\right] \\
\ell_6\colon\ (z,\ w) := (y_1,\ y_3 + y_4) \\
\ell_7\colon
\end{array}
$$

Fig. 1.54. Program GCDLCM (computing the *gcd* and *lcm*).

In your proof you may use the properties of the *gcd* function listed in Problem 1.7, and the following property of the *lcm* function:

$$lcm(m, n) \ = \ m \cdot n / gcd(m, n).$$

Problem 1.9 (set partitioning) page 59

Consider program EXCH presented in Fig. 1.55. The program accepts as input two sets of natural numbers S and T, whose initial values are S_0 and T_0, respectively.

Process P_1 repeatedly identifies and removes the maximal element in S and sends it to P_2 which places it in T. Symmetrically, P_2 identifies and removes the minimal element in T and sends it to P_1 which places it in S. The processes use the operations $\max(S)$ and $\min(T)$ which find, respectively, the maximal element in the set S and the minimal element in the set T. Show total correctness of program EXCH, which can be specified by the response formula

$$\Theta \ \Rightarrow \ \Diamond(at\_\ell_9 \ \land \ at\_m_7 \ \land \ |S| = |S_0| \ \land \ |T| = |T_0| \ \land \ S \leq T).$$

This formula states that the program terminates and on termination, sets S and T have preserved their initial sizes and that every element in S is smaller than or equal to every element in T.

Problem 1.10 (converse of Claim 1.3) page 67

Let $P :: \ [P_1 :: S_1 \| \cdots \| P_k :: S_k]$ be a program whose top-level processes communicate by shared variables and such that every program variable is owned by one

$$
\begin{array}{lll}
\textbf{in} & S_0,\ T_0 & : \textbf{set of natural where } S_0 \neq \emptyset,\ T_0 \neq \emptyset \\
\textbf{out} & S,\ T & : \textbf{set of natural where } S = S_0,\ T = T_0 \\
\textbf{local} & \alpha,\ \beta & : \textbf{channel of integer}
\end{array}
$$

$$
P_1 :: \left[\begin{array}{l}
\textbf{local } x,\ mx\text{: } \textbf{integer} \\
\ell_0\text{: } x := -1 \\
\ell_1\text{: } mx := \max(S) \\
\ell_2\text{: } \textbf{while } x < mx \textbf{ do} \\
\quad \left[\begin{array}{l}
\ell_3\text{: } \alpha \Leftarrow mx \\
\ell_4\text{: } S := S - \{mx\} \\
\ell_5\text{: } \beta \Rightarrow x \\
\ell_6\text{: } S := S \cup \{x\} \\
\ell_7\text{: } mx := \max(S)
\end{array}\right] \\
\ell_8\text{: } \alpha \Leftarrow -1 \\
\ell_9\text{: }
\end{array}\right]
\quad \| \quad
P_2 :: \left[\begin{array}{l}
\textbf{local } y,\ mn\text{: } \textbf{integer} \\
m_0\text{: } \alpha \Rightarrow y \\
m_1\text{: } \textbf{while } y \geq 0 \textbf{ do} \\
\quad \left[\begin{array}{l}
m_2\text{: } T := T \cup \{y\} \\
m_3\text{: } mn := \min(T) \\
m_4\text{: } \beta \Leftarrow mn \\
m_5\text{: } T := T - \{mn\} \\
m_6\text{: } \alpha \Rightarrow y
\end{array}\right] \\
m_7\text{: }
\end{array}\right]
$$

Fig. 1.55. Program EXCH (partitioning two sets).

of the top-level processes. Let σ be a model such that the interpretation of π is a subset of the locations of P and σ is simultaneously a modular computation of every P_i, $i = 1, \ldots, k$. Show that σ is a computation of P.

Problem 1.11 (completeness of rule COMP-B) page 68

Let $P :: [P_1 :: S_1 \| \cdots \| P_k :: S_k]$ be a program whose top-level processes communicate by shared variables and such that every program variable is owned by one of the top-level processes. Let φ be a P-valid formula. Show that the P-validity of φ can be compositionally inferred from modular validities, using rule COMP-B and temporal reasoning. This establishes the completeness of rule COMP-B for compositional verification.

A solution to this problem can be organized as follows.

- For each top-level process P_i, $i = 1, \ldots, k$, construct a formula ψ_i capturing the temporal modular semantics of P_i. That is, a model σ satisfies ψ_i iff σ is a modular computation of P_i. Argue semantically that

  $$P_i \underset{m}{\models} \psi_i,$$

  for each $i = 1, \ldots, k$.
- Use rule COMP-B and temporal reasoning to infer

$$P \;\vDash\; \psi_1 \;\wedge\; \cdots \;\wedge\; \psi_k.$$

- Argue semantically that a model σ satisfies $\psi_1 \wedge \cdots \wedge \psi_k$ iff σ is a computation of the entire program P. Therefore, if φ is P-valid, then the following general validity holds:

$$\vDash\; \psi_1 \;\wedge\; \cdots \;\wedge\; \psi_k \;\rightarrow\; \varphi.$$

- Apply temporal reasoning to $P \vDash \psi_1 \wedge \cdots \wedge \psi_k$ and $\vDash \psi_1 \wedge \cdots \wedge \psi_k \rightarrow \varphi$ to infer

$$P \;\vDash\; \varphi.$$