Chapter 3

Response Under Fairness

Chapters 1 and 2 considered proof rules for response properties that depend on *justice* for their validity. While justice is an important fairness requirement, it is only one of the two fairness requirements listed in a general fair transition system. In this chapter we consider proof rules for response properties that depend on the two *fairness* requirements, i.e., *justice* and *compassion*, for their validity.

In a way similar to Chapter 1, we first introduce a single-step rule for response under compassion, and then introduce the fair versions of CHAIN and WELL.

Let us review the basic phenomena that are modeled by the requirements of justice and compassion.

Justice

Justice represents a natural property of multi-processor systems. It represents the obvious intuition that parallel processes, implemented on separate processors, should progress independently as long as they do not need to synchronize with other processes. This intuitively obvious fact for multi-processor systems must be represented explicitly by justice in our model, due to the artifact of modeling concurrency by interleaving.

Justice excludes from the set of computations runs in which an enabled process does not progress beyond some point. Consequently, justice represents in our model the natural phenomena of independent progress in parallel components of a multi-processor system.

Since justice is associated with the general phenomenon of parallelism, it is not restricted to particular statements. In the computational model we consider here, all transitions, with two exceptions, are required to be just, i.e., are members of the justice set $\mathcal{J}$. The exceptions are the idling transition τ_I and the transition associated with the schematic *noncritical* statement.

Compassion

Compassion excludes from the set of computations runs in which one of the compassionate transitions is enabled infinitely many times but taken only finitely many times.

In comparison with justice, compassion has a much narrower scope and is associated with only a small subset of statements and their corresponding transitions. A case in point is the semaphore *request* statement.

We should bear in mind that a correct implementation of a semaphore which ensures the required compassion property must be based on either a nontrivial software protocol or a special hardware arbiter. For example, a protocol for semaphore services may place all requests in an internal queue, and allocate the semaphore on a first-come-first-serve basis.

In our high-level programming language, all these low-level details are abstracted away and the whole transaction between customer and server is represented by the single statement **request** y. The compassion requirement associated with this statement provides an abstract representation of the internal queing mechanism, replacing the *first-come-first-serve* guarantee by the weaker *infinitely-often-requested-eventually-served* guarantee.

We can view our extensive study of mutual exclusion algorithms that do not use semaphores as an investigation of ways to implement compassion using only justice. In fact, each of these algorithms can be used to implement a semaphore, which is the simplest manifestation of compassion.

Another class of statements associated with compassion requirements are the communication statements. As in the case of semaphores, the compassion assumption provides a useful abstraction that hides a more involved low-level protocol that ensures fair arbitration between processes that compete for an access to a common channel.

Compassion can also be used to model *probabilistic* behavior. Consider a program consisting of two processes that communicate over an unreliable channel. A probabilistic characterization of such a channel may specify, for example, that on the average it loses one message out of ten. This assigns a probability of 0.9 that a message sent will arrive on the other side. Assume that we have a protocol that overcomes the unreliability of the channel by repeatedly sending a message until it is acknowledged. To verify the correctness of such a protocol by the tools presented in this book, we replace the quantitative characterization

A message submitted will get across with probability 0.9

by the compassion requirement

A message submitted infinitely many times will eventually get across.

Some examples presented in this chapter illustrate the use of compassion for modeling probabilistic behavior.

In most of the chapter, we consider the simpler case of response formulas $p \Rightarrow \diamondsuit\, q$, where p and q are assertions. In the last subsection of Section 3.7, we discuss the generalization to the case that p and q are arbitrary past formulas.

In Section 3.1 we present a single-step rule that relies on the activation of a single compassionate transition.

Section 3.2 shows how to combine several applications of the just and compassionate single-step rules into a chain rule that relies on a fixed number of helpful transitions. We also introduce verification diagrams which provide a graphical representation of the chain rule.

Section 3.3 illustrates applications of the chain rule under general fairness to example programs that involve message passing.

Section 3.4 explores the use of compassion to model channels which are only eventually reliable, i.e., can only guarantee that a message that is repeatedly transmitted eventually gets through.

Section 3.5 presents a well-founded rule for response under general fairness.

Section 3.6 considers different proposed solutions to the dining philosophers problem. In the SAFETY book, we established the safety properties of these solutions. Here, we complete their formal verification by proving that most of those solutions satisfy the progress property of accessibility.

In Section 3.7 we complete the formal verification of several resource-allocation programs considered in the SAFETY book, by establishing their response properties.

Section 3.8 establishes the (relative) completeness of the well-founded rule introduced in Section 3.5 for proving all response properties of a given program. This section considers the special (but most prevalent) case of a formula $p \Rightarrow \diamondsuit\, q$ where p and q are state formulas.

Section 3.9 extends the completeness proof to the general case of a response formula $p \Rightarrow \diamondsuit\, q$ where p and q are past formulas.

Section 3.10 presents algorithms for automatic verification of response formulas $p \Rightarrow \diamondsuit\, q$ over finite-state systems, for the case that p and q are state formulas.

Finally, Section 3.11 completes the discussion of algorithmic verification of response properties by considering the general case where p and q are past formulas.

3.1 A Single-Step Rule

The rules based on justice, considered in Chapters 1 and 2, are not powerful enough to prove response properties based on compassion. To see this, consider the standard semaphore program for mutual exclusion, presented in Fig. 3.1. This program was first introduced in Fig. 1.6 of the SAFETY book.

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

$$P_1 :: \begin{bmatrix} \ell_0\text{: } \textbf{loop forever do} \\ \quad \begin{bmatrix} \ell_1\text{: } \textbf{noncritical} \\ \ell_2\text{: } \textbf{request } y \\ \ell_3\text{: } \textbf{critical} \\ \ell_4\text{: } \textbf{release } y \end{bmatrix} \end{bmatrix} \quad \Big\| \quad P_2 :: \begin{bmatrix} m_0\text{: } \textbf{loop forever do} \\ \quad \begin{bmatrix} m_1\text{: } \textbf{noncritical} \\ m_2\text{: } \textbf{request } y \\ m_3\text{: } \textbf{critical} \\ m_4\text{: } \textbf{release } y \end{bmatrix} \end{bmatrix}$$

Fig. 3.1. Program MUX-SEM (mutual exclusion by semaphores).

The main response property associated with this program is individual accessibility. Stated for process P_1, it is given by

$$\underbrace{at\_\ell_2}_{p} \;\Rightarrow\; \Diamond\, \underbrace{at\_\ell_3}_{q} \,.$$

The situation is very easy to analyze. Clearly, we can identify ℓ_2 as a helpful transition in this case, which obviously needs only one helpful step to achieve the goal at_ℓ_3.

Let us consult rule RESP-J (Fig. 1.1), which is the strongest rule for single-step response properties we have. Premises J1–J3 of this rule are satisfied for

$$\varphi = p\text{:} \quad at\_\ell_2 \qquad\qquad \tau\text{:} \quad \ell_2.$$

That is, the helpful transition ℓ_2 leads from at_ℓ_2 to at_ℓ_3, while all other transitions preserve at_ℓ_2. The premise we fail to prove is J4, which requires

$$at\_\ell_2 \;\rightarrow\; En(\ell_2),$$

or equivalently

$$at\_\ell_2 \;\rightarrow\; at\_\ell_2 \,\wedge\, y > 0.$$

This, of course, is not a valid invariant for program MUX-SEM.

Let us recall the reasons for including premise J4 in rule RESP-J. While premises J2 and J3 ensure that φ is preserved until q is achieved, and that the next τ-transition taken from a φ-state achieves q, premise J4 forces, through justice, that transition τ eventually will be taken. Justice guarantees the activation of τ only if τ is continuously enabled beyond some point. Since φ holds continuously until q is achieved, premise J4 ties the enableness of τ with φ, ensuring that τ will be continuously enabled until q is achieved. If q is not achieved, τ will be continuously enabled and never taken, violating justice with respect to τ.

Compassion is less demanding about the degree of enableness of τ that is required to ensure activation. For a compassionate transition τ, it is only necessary that τ be infinitely often enabled to guarantee its activation. Consequently, we may relax the tie between φ and the enableness of τ and, instead of requiring that τ is enabled *whenever* φ holds, only require that if φ holds then *eventually* τ should be enabled.

The Single-Step Rule

Relaxation of the last premise leads to rule RESP-C for proving single-step response under compassion (Fig. 3.2).

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

$$\begin{array}{ll} \text{C1.} & p \;\rightarrow\; q \;\vee\; \varphi \\ \text{C2.} & \{\varphi\}\; \mathcal{T}\; \{q \;\vee\; \varphi\} \\ \text{C3.} & \{\varphi\}\; \tau_h\; \{q\} \\ \text{C4.} & \varphi \;\Rightarrow\; \diamondsuit\big(\neg\varphi \;\vee\; En(\tau_h)\big) \\ \hline & p \;\Rightarrow\; \diamondsuit\, q \end{array}$$

Fig. 3.2. Rule RESP-C (single-step response under compassion).

Premise C1 requires that any p-position, that does not already satisfy q satisfies the intermediate assertion φ. Premise C2 requires that the application of any transition to a φ-state yields a state satisfying q or φ. Premise C3 requires that the application of the helpful transition τ_h to a φ-position yields a q-position. Premise C4, which distinguishes RESP-C from RESP-J, requires that if φ holds then eventually either φ will stop holding, or the helpful transition τ_h will become enabled. Note that, due to C2 and C3, the first transition that falsifies φ leads to a q-position. Consequently, C4 can also be written as

$$\varphi \;\Rightarrow\; \diamondsuit\big(q \;\vee\; En(\tau_h)\big).$$

Justification To justify the rule, consider a computation σ satisfying the four premises of the rule. Assume that p holds at some position $k \geq 0$. We wish to show that q holds at some later position $j \geq k$.

Assume, to the contrary, that q never holds beyond k. By C1, φ also holds at k. By C2 and the assumption that q does not hold beyond k, φ holds continuously beyond k.

Consider position $i_1 = k$, at which φ holds. By C4, there exists a position j_1, $j_1 \geq i_1$, at which either φ is false or τ_h is enabled. Since φ is true at all positions beyond k, τ_h is enabled at j_1. Take $i_2 = j_1 + 1$. By C4, since φ also holds at i_2, there exists a position j_2, $j_2 \geq i_2$, at which, again, τ_h is enabled. By repeating the argument we construct an infinite sequence of positions $j_1 < j_2 < \cdots$, at all of which τ_h is enabled.

Thus, τ_h is enabled at infinitely many positions in σ. Transition τ_h is not taken beyond k since, by C3, any application of τ_h to a φ-position results in a q-position, and by our assumption, there are no q-states beyond k. This violates the requirement of compassion with respect to τ_h. Thus, q must hold at some position beyond k. ∎

In applications of the rule, it is sufficient to check premise C2 for $\tau \neq \tau_h$, since C2 for $\tau = \tau_h$ is implied by C3.

It is easy to see that, for a transition τ_h that is both just and compassionate, rule RESP-C is stronger than rule RESP-J. This means that the premises of rule RESP-J imply the premises of rule RESP-C. This is because the first three premises of the two rules are identical, and due to the reflexivity and monotonicity of the $\diamondsuit$ operator,

$$\text{J4.} \quad \varphi \;\rightarrow\; En(\tau_h)$$

implies

$$\text{C4.} \quad \varphi \;\Rightarrow\; \diamondsuit\big(\neg\varphi \;\vee\; En(\tau_h)\big).$$

On the other hand, rule RESP-C has a more limited applicability than rule RESP-J. In rule RESP-J we may choose any just transition to be the helpful one. Rule RESP-C is restricted to use only compassionate transitions, which form a much smaller set than that of the just transitions.

In most applications of rule RESP-C premise C4 is established by proving the stronger statement

$$\widehat{\text{C4}}. \qquad \varphi \;\Rightarrow\; \diamondsuit\, En(\tau_h),$$

which obviously implies C4.

Rule RESP-C appears to be recursive. To prove one response property, it uses as one of its premises another response property. How is this premise to be proven? We will give several answers to this puzzling question, showing that such recursive rules do not necessarily lead to circular reasoning.

Example Let us use rule RESP-C to prove the accessibility property

$$\underbrace{at\_\ell_2}_{p} \quad \Rightarrow \quad \Diamond\, \underbrace{at\_\ell_3}_{q}$$

for program MUX-SEM of Fig. 3.1.

Take

$$\varphi = p\colon \quad at\_\ell_2 \qquad\qquad \tau_h\colon \quad \ell_2.$$

This choice identifies ℓ_2 as the helpful compassionate transition.

- Premise C1

This premise requires

$$\underbrace{at\_\ell_2}_{p} \quad \to \quad \cdots \quad \vee \quad \underbrace{at\_\ell_2}_{\varphi}\ ,$$

which is clearly valid.

- Premise C2

This premise is proven by showing

$$\rho_\tau \quad \wedge \quad \underbrace{at\_\ell_2}_{\varphi} \quad \to \quad \cdots \quad \vee \quad \underbrace{at'\_\ell_2}_{\varphi'}$$

for any $\tau \neq \ell_2$. Since while P_1 is at ℓ_2 no other transition of P_1 is enabled and no transition of P_2 can affect at_ℓ_2, the premise follows.

- Premise C3

This premise requires

$$\underbrace{\cdots \ \wedge \ at'\_\ell_3}_{\rho_{\ell_2}} \quad \wedge \quad \underbrace{at\_\ell_2}_{\varphi} \quad \to \quad \underbrace{at'\_\ell_3}_{q'}$$

which is obviously valid.

- Premise C4

This premise is established by proving

$$\underbrace{at\_\ell_2}_{\varphi} \quad \Rightarrow \quad \Diamond\Big(\cdots \quad \vee \quad \underbrace{at\_\ell_2 \ \wedge \ y > 0}_{En(\ell_2)}\Big)\ .$$

The proof of this response property using rule CHAIN-J is presented in the verification diagram of Fig. 3.3. In the proof we are aided by the invariant

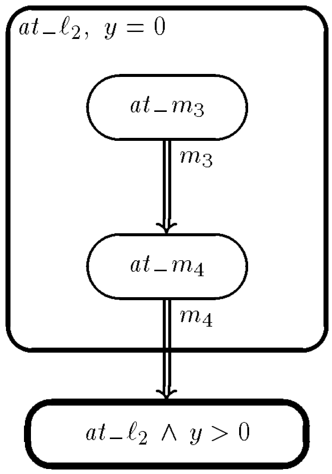


Fig. 3.3. Verification diagram for premise C4.

$$\chi_1: \quad at\_\ell_{3,4} \ + \ at\_m_{3,4} \ + \ y \ = \ 1.$$

Due to invariant χ_1, if at_ℓ_2 holds but $y \leq 0$, then $y = 0$ and P_2 must be at m_3 or m_4.

This concludes the proof of accessibility for program MUX-SEM. ▪

The example above provides a first answer to the problem of recursiveness of rule RESP-C. In some cases, the temporal premise C4 can be proven by using the J-rules.

3.2 Chain Rule

Naturally, after presenting a single-step rule for response under compassion, we should consider the compassionate version of rule CHAIN. Such a version can be obtained in the standard way, by considering a sequence of assertions $\varphi_0, \ldots, \varphi_m$ such that $\varphi_i \Rightarrow \Diamond\left(\bigvee_{j<i} \varphi_j\right)$ can be proven using rule RESP-C for each $i = 1, \ldots, m$. The conclusion of such a rule is $\left(\bigvee_{i=0}^{m} \varphi_i\right) \Rightarrow \Diamond\, \varphi_0$.

However, this version is not very useful. When tracing a chain of helpful steps, it is rarely the case that all of them are associated with compassionate transitions. The typical case is that most of the helpful steps are due to just transitions while only one or two are due to compassionate transitions. Consequently, rule CHAIN-F presented in Fig. 3.4 allows both just and compassionate helpful transitions.

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

$$\text{F1.}\quad p \;\rightarrow\; \bigvee_{j=0}^{m} \varphi_j$$

$$\left.\begin{array}{ll}
\text{F2.} & \{\varphi_i\}\ \mathcal{T}\ \left\{\bigvee_{j \le i} \varphi_j\right\} \\[2ex]
\text{F3.} & \{\varphi_i\}\ \tau_i\ \left\{\bigvee_{j < i} \varphi_j\right\} \\[2ex]
\text{F4.} & \text{If } \tau_i \in \mathcal{C}, \text{ then} \\
 & \quad \text{C4.}\quad \varphi_i \;\Rightarrow\; \diamondsuit\big(\neg\varphi_i \;\vee\; En(\tau_i)\big) \\
 & \text{Otherwise} \\
 & \quad \text{J4.}\quad \varphi_i \;\rightarrow\; En(\tau_i)
\end{array}\right\}\ \text{for } i = 1, \ldots, m$$

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

Fig. 3.4. Rule CHAIN-F (chain under fairness).

Premises F1–F3 of rule CHAIN-F are identical to the corresponding premises J1–J3 of rule CHAIN-J (Fig. 1.7).

Premise F4 of rule CHAIN-F requires either eventual or immediate enableness, according to whether τ_i is compassionate or only just. Note that if τ_i is compassionate, we may still attempt to prove for it the simpler (and stronger) version of F4, i.e., $\varphi_i \rightarrow En(\tau_i)$, because this implication obviously implies $\varphi_i \Rightarrow \diamondsuit\big(\neg\varphi_i \vee En(\tau_i)\big)$. Only if we fail to prove the simpler version, should we resort to a direct proof of the more complex version.

Justification The justification of rule CHAIN-F is similar to that of rule CHAIN-J. Assume a computation σ in which p holds at position k and no position beyond k satisfies q. From premises F1–F3 it follows that there exists a position $r \ge k$ and an index $i > 0$ such that φ_i holds continually beyond r and no φ_j, with $j < i$, holds at any such position.

Consider two cases. If τ_i, the helpful transition for φ_i, is compassionate then by F4, τ_i is enabled infinitely many times. It follows that σ does not fulfill the requirement of compassion with respect to τ_i, contradicting the assumption that σ is a computation. If τ_i is just but not compassionate, F4 implies that it is continually enabled which shows that σ violates the requirement of justice with respect to τ_i.

We conclude that the described situation is impossible, and every p-position must be followed by a q-position. ∎

In applications of the rule, it is sufficient to establish premise F2 for all $\tau \neq \tau_i$, since F2 for $\tau = \tau_i$ is implied by premise F3.

Example (producer-consumer)

We illustrate rule CHAIN-F on program PROD-CONS, presented in Fig. 3.5. This program was first introduced in Fig. 2.23 of the SAFETY book.

The response property we wish to establish for this program is accessibility, which for process *Prod* is given by

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

In the proof, we are aided by the following invariants

$$\chi_0:\quad r \geq 0 \;\wedge\; ne \geq 0 \;\wedge\; nf \geq 0$$

$$\chi_1:\quad r \,+\, at\_\ell_{4,5} \,+\, at\_m_{3,4} \;=\; 1$$

$$\chi_2:\quad ne \,+\, nf \,+\, at\_\ell_{3..6} \,+\, at\_m_{2..5} \;=\; N.$$

To prove the accessibility property we use rule CHAIN-F. Inspecting the program, it is easy to identify the intermediate stages in the progress from ℓ_2 to ℓ_4. We choose

$$\varphi_2:\ at\_\ell_2 \qquad \tau_2:\ \ell_2$$

$$\varphi_1:\ at\_\ell_3 \qquad \tau_1:\ \ell_3$$

$$\varphi_0:\ at\_\ell_4.$$

It is straightforward to verify that each τ_i, $i = 1, 2$, leads from φ_i to φ_{i-1}, and that being at φ_i, each transition τ, $\tau \neq \tau_i$ preserves φ_i.

The only nontrivial premises to verify are the enableness premises F4 for $i = 1, 2$. In this case both τ_1 and τ_2 are compassionate transitions, so we should prove eventual enableness.

- Premise F4 for $i = 2$

For this premise it suffices to prove

local $r,\ ne,\ nf$: **integer where** $r = 1,\ ne = N,\ nf = 0$
$\quad b$: **list of integer where** $b = \Lambda$

$$Prod :: \left[\begin{array}{l} \textbf{local } x\text{: } \textbf{integer} \\ \ell_0\text{: } \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} \ell_1\text{: } \textbf{produce } x \\ \ell_2\text{: } \textbf{request } ne \\ \ell_3\text{: } \textbf{request } r \\ \ell_4\text{: } b := b \bullet x \\ \ell_5\text{: } \textbf{release } r \\ \ell_6\text{: } \textbf{release } nf \end{array}\right] \end{array}\right]$$

$$\|$$

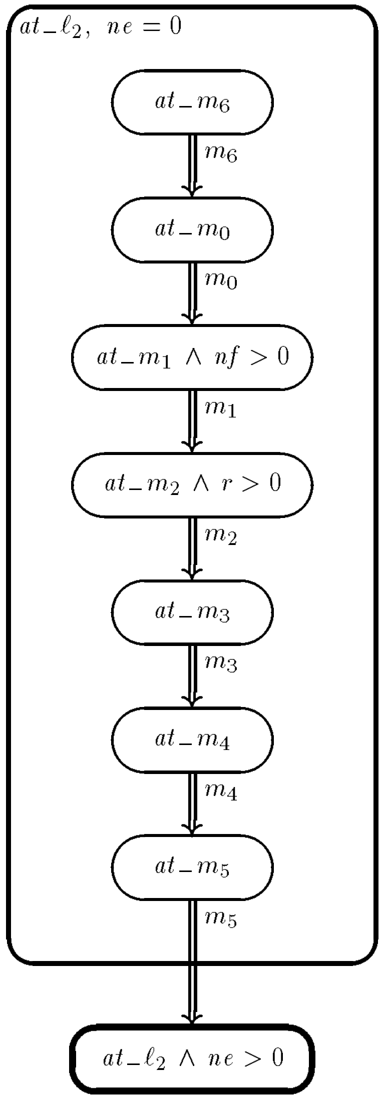
$$Cons :: \left[\begin{array}{l} \textbf{local } y\text{: } \textbf{integer} \\ m_0\text{: } \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} m_1\text{: } \textbf{request } nf \\ m_2\text{: } \textbf{request } r \\ m_3\text{: } (y,\ b) := \big(hd(b),\ tl(b)\big) \\ m_4\text{: } \textbf{release } r \\ m_5\text{: } \textbf{release } ne \\ m_6\text{: } \textbf{consume } y \end{array}\right] \end{array}\right]$$

Fig. 3.5. Program PROD-CONS (producer-consumer).

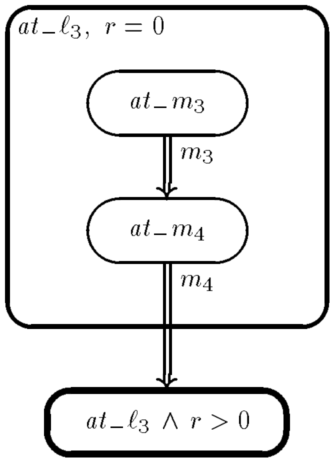
$$\underbrace{at\_\ell_2}_{\varphi_2} \;\Rightarrow\; \Diamond\Big(\cdots \;\vee\; \underbrace{at\_\ell_2 \;\wedge\; ne > 0}_{En(\ell_2)}\Big)\ .$$

To prove this response property we use rule CHAIN-J. The proof is presented in the CHAIN diagram of Fig. 3.6(a). Note that we use the invariant $ne \geq 0$ to infer that if $ne \leq 0$ then in fact $ne = 0$. Invariant χ_2 is used to deduce that, when $ne = 0$ and $at\_\ell_2 \wedge at\_m_1$ holds, then $nf = N > 0$. Invariant χ_1 is used to deduce that, when $at\_\ell_2 \wedge at\_m_2$ holds, then $r = 1 > 0$.

An interesting point is that, even though m_1 and m_2 are compassionate transitions, we included them in a proof by rule CHAIN-J. This is possible because, under the assumption that P_1 is frozen at ℓ_2 and $ne = 0$, these transitions are immediately enabled when control is at their respective locations.



(a) Proof of premise F4 for $i = 2$.



(b) Proof of premise F4 for $i = 1$.

Fig. 3.6. CHAIN diagrams.

- Premise F4 for $i = 1$

This premise is proven by showing

$$\underbrace{at\_\ell_3}_{\varphi_1} \;\Rightarrow\; \diamondsuit\Big(\cdots \;\vee\; \underbrace{at\_\ell_3 \;\wedge\; r>0}_{En(\ell_3)}\Big)\,.$$

The proof is based again on an application of rule CHAIN-J, and is presented in the CHAIN diagram of Fig. 3.6(b). We use the invariant $r \geq 0$ to infer that if $r \leq 0$ then $r = 0$. Then, we use χ_1 to deduce that if $r = 0$ and P_1 is at ℓ_3, then P_2 can only be at m_3 or at m_4.

This concludes the proof of the accessibility property

$$at\_\ell_2 \;\Rightarrow\; \diamondsuit\, at\_\ell_4.$$

In **Problem 3.2**, we request the reader to prove that the accessibility property is valid even over a weaker version of the program, in which the transitions corresponding to the semaphore *request* statements, are taken to be just, rather than compassionate. ▟

CHAIN-F *Diagrams*

Diagrams can also be used to provide a concise representation of a proof by rule CHAIN-F. The main extension over CHAIN diagrams is the introduction of a third type of an edge — a *solid* edge, where a solid edge will correspond to a helpful transition which is compassionate. Consequently, edges connecting nodes (assertions) in a diagram can be single (-line), double, or solid.

CHAIN-F Diagrams

A verification diagram is said to be an CHAIN-F *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 requirement:

- If a single edge connects node φ_i to node φ_j, then $i \geq j$.
- If a double or a solid edge connects φ_i to φ_j, then $i > j$.
- Every node φ_i, $i > 0$, has at least one departing edge which is either double or solid. The transition labeling this edge is identified as helpful for assertion φ_i.

Verification Conditions for CHAIN-F Diagrams

Let φ be a nonterminal node and $\varphi_1, \ldots, \varphi_k$, $k \geq 0$, be the τ-successors of φ.

- If τ labels a single edge, we associate with φ and τ the verification condition

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

- If τ labels a double or solid edge, we associate with φ and τ the verification condition

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

- If τ labels a double edge departing from φ, we associate with φ and τ the requirement

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

- If τ labels a solid edge departing from φ, we associate with φ and τ the response formula

$$\varphi \Rightarrow \diamondsuit(\neg\varphi \lor En(\tau)).$$

We extend the notion of verification conditions to include also the enabling requirement associated with double edges and the response formulas associated with solid edges.

Valid CHAIN-F Diagrams

A CHAIN-F diagram is said to be *P-valid* if the first three types of verification conditions associated with its nodes and transitions are P-state valid, and the response formula conditions, associated with solid edges, are P-valid.

The consequences of having a valid CHAIN-F diagram are stated in the following claim.

Claim 3.1 (fair chain)

A P-valid CHAIN-F 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 following implication:

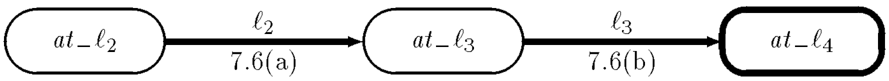
$$p \rightarrow \bigvee_{j=0}^{m} \varphi_j \qquad \text{and} \qquad \varphi_0 \rightarrow q$$

then we can conclude the P-validity of

$$p \Rightarrow \diamondsuit q.$$

Example (producer consumer)

The proof of the accessibility property of program PROD-CONS (Fig. 3.5) can be represented by the CHAIN-F diagram



Since each transition labeling a solid edge needs a subproof that establishes the validity of the compassionate version of premise F5, we often attach a reference to such edges, which points to the diagram for the subproof. Thus, in the above diagram, we referred to CHAIN diagram 3.6(a) as establishing the eventual enableness of ℓ_2, and to CHAIN diagram 3.6(b) as establishing the eventual enableness of ℓ_3. ▟

Example (multiple resource acquisition)

In Fig. 2.12 of the SAFETY book we introduced program RES-3 as an example of a program consisting of several processes that need several resources in order to perform their critical activity. We reproduce this program in Fig. 3.7.

$$\textbf{local } r_1,\ r_2,\ r_3\textbf{: integer where } r_1 = r_2 = r_3 = 1$$

$$P_1 :: \left[\begin{array}{l} \ell_0\textbf{: loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\textbf{: noncritical} \\ \ell_2\textbf{: request } (r_2;\ \ell_3)\textbf{: request } r_3 \\ \ell_4\textbf{: critical} \\ \ell_5\textbf{: release } (r_2;\ \ell_6)\textbf{: release } r_3 \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} m_0\textbf{: loop forever do} \\ \quad \left[\begin{array}{l} m_1\textbf{: noncritical} \\ m_2\textbf{: request } (r_1;\ m_3)\textbf{: request } r_3 \\ m_4\textbf{: critical} \\ m_5\textbf{: release } (r_1;\ m_6)\textbf{: release } r_3 \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_3 :: \left[\begin{array}{l} k_0\textbf{: loop forever do} \\ \quad \left[\begin{array}{l} k_1\textbf{: noncritical} \\ k_2\textbf{: request } (r_1;\ k_3)\textbf{: request } r_2 \\ k_4\textbf{: critical} \\ k_5\textbf{: release } (r_1;\ k_6)\textbf{: release } r_2 \end{array}\right] \end{array}\right]$$

Fig. 3.7. Program RES-3 — three processes needing three resources.

In Section 2.3 of the SAFETY book, we established the safety property of mutual exclusion for program RES-3. The proof was based on the following invariants

$$\chi_0: \quad r_1 \geq 0 \;\wedge\; r_2 \geq 0 \;\wedge\; r_3 \geq 0$$

$$\chi_1: \quad r_1 + at\_m_{3..5} + at\_k_{3..5} \;=\; 1$$

$$\chi_2: \quad r_2 + at\_\ell_{3..5} + at\_k_{4..6} \;=\; 1$$

$$\chi_3: \quad r_3 + at\_\ell_{4..6} + at\_m_{4..6} \;=\; 1.$$

It is easy to see that these invariants ensure pairwise exclusion between the processes. For example, to show exclusion between P_1 and P_3, we examine the invariant χ_2, corresponding to the resource r_2 common to P_1 and P_3. From χ_0 and χ_2 we obtain

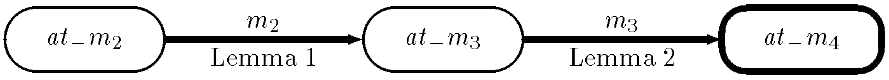
$$at\_\ell_4 + at\_k_4 \;\leq\; at\_\ell_{3..5} + at\_k_{4..6} \;\leq\; 1,$$

which shows that P_1 and P_3 cannot be at their critical sections at the same time.

Here we prove the response property of accessibility. Stated for P_2, it is given by

$$at\_m_2 \;\Rightarrow\; \diamondsuit\, at\_m_4$$

A proof of this property by rule CHAIN-F is presented in the following diagram



To complete the proof we need to show the enableness premise for m_2 and m_3, which is done in Lemmas 1 and 2.

Lemma 1 $\quad at\_m_2 \;\Rightarrow\; \diamondsuit(at\_m_2 \;\wedge\; r_1 > 0)$

Examining invariant χ_1 we see that, with P_2 being at m_2, the only way for r_1 to be nonpositive is that $r_1 = 0$ and P_3 is at $k_{3..5}$. We therefore have to follow P_3 until it exits $k_{3..5}$, at which point k_5 sets r_1 back to 1. This development is shown in the CHAIN-F diagram of Fig. 3.8.

Of the three helpful transitions identified in this diagram, immediate enableness of k_4 and k_5 follows from assertions at_k_4 and at_k_5 respectively. The situation is different with k_3 which also needs $r_2 > 0$ to proceed. Since k_3 is a compassionate transition, it is sufficient to prove eventual enableness, which is proven in Lemma 1.1.

Lemma 1.1 $\quad at\_m_2 \;\wedge\; at\_k_3 \;\wedge\; r_1 = 0 \;\Rightarrow\; \diamondsuit(at\_k_3 \;\wedge\; r_2 > 0)$

Examining invariant χ_2, we see that with P_3 at k_3, the only way for r_2 to be nonpositive is that $r_2 = 0$ and P_1 is at $\ell_{3..5}$. The diagram of Fig. 3.9 follows P_1

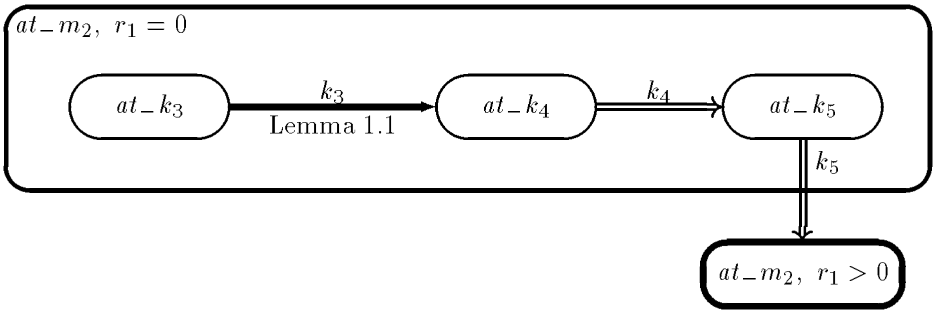


Fig. 3.8. CHAIN-F diagram for Lemma 1.

until it executes ℓ_5, at which point r_2 becomes positive. Notice that since P_2 is waiting at m_2 with $r_1 = 0$, and P_3 is waiting at k_3 with $r_2 = 0$, they are both disabled, and the only process that can save the day is P_1. The CHAIN diagram of Fig. 3.9 shows that indeed it does.

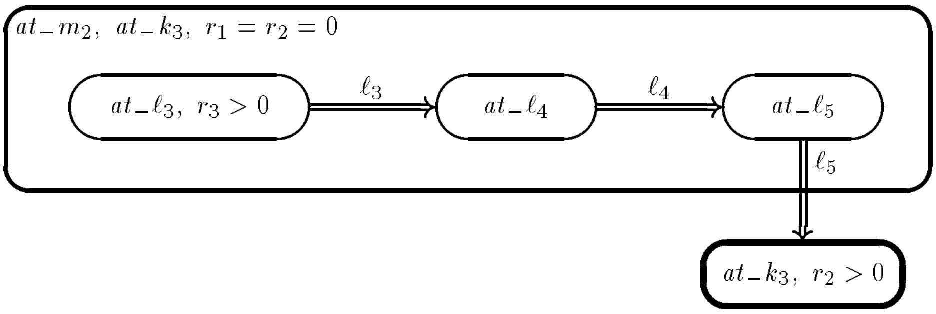


Fig. 3.9. CHAIN diagram for Lemma 1.1.

Note that due to χ_3, while P_1 is at ℓ_3 and P_2 is at m_2, r_3 is positive, which makes ℓ_3 immediately enabled. Thus, even though ℓ_3 is a compassionate transition we can prove for it immediate enableness.

Lemma 2 $\quad at\_m_3 \;\Rightarrow\; \Diamond(at\_m_3 \;\wedge\; r_3 > 0)$

This lemma proves the second enableness premise for the accessibility property. While P_2 is at m_3, the only way for r_3 to be nonpositive, according to χ_3, is that $r_3 = 0$, and P_1 is at $\ell_{4..6}$. The CHAIN diagram of Fig. 3.10 traces the progress of P_1 through $\ell_{4..6}$ until transition ℓ_6 increases r_3 to 1.

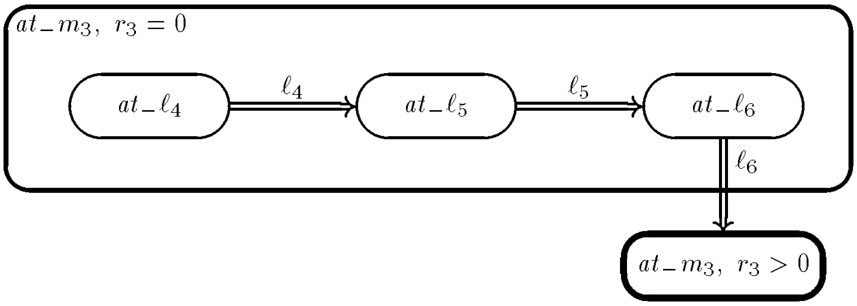


Fig. 3.10. CHAIN diagram for Lemma 2.

This concludes the proof of the lemmas, and establishes accessibility for P_2. Similarly, we can establish accessibility for P_1 and P_3. ▪

Recursiveness of the Fair Rules

The proof of accessibility for P_2 in the example above provides a second answer to the problem of recursiveness of the F-family of rules.

The recursiveness problem is that the enableness premise of the F-rules is a response formula, similar to the conclusions of these rules. The question, therefore, is how we escape circular reasoning trying to prove a formula using a rule in which one of the premises is not simpler than the conclusion.

The first answer we gave, which is also illustrated in the proof of Lemma 1, is that to prove a property by an F-rule, it is often possible to prove its eventual enableness premise F4 by J-rules. For example, the enableness premise for k_3, needed in the proof of Lemma 1, is proven in Lemma 1.1 by rule CHAIN-J (Fig. 3.9).

The second answer, which is illustrated in the complete proof of the accessibility property, is that even if we need to pursue several levels of proof, all of which use an F-rule, we are guaranteed that on each level we have to consider one process less than on the previous level.

Consider the proof of the accessibility property for program RES-3. It is mainly based on the progress of P_2 from m_2 to m_4. To complete the proof we need at m_2 an enableness premise that guarantees that the transition m_2 eventually becomes enabled. If it happens to be currently enabled, all the better. So the difficult case is to show that if P_2 is at m_2 and the transition at m_2 is currently *disabled*, it eventually becomes enabled. For that we use Lemma 1.

However, note that in all the subsequent situations considered in Lemma 1 or any of its subproofs, P_2 is disabled. This implies that, when dealing with this case, we have one less process to consider. Lemma 1, recursively, uses an F-rule to follow the progress of P_3 from k_3 to k_6. In all situations arising throughout this progress, P_2 is continuously neutralized (frozen). Considering the situation of P_3 at k_3, we realize that the proof of another eventual enableness is called for, namely the eventual enableness of k_3. For that we use Lemma 1.1.

Since the interesting case for Lemma 1.1 is when k_3 is disabled, we enter this lemma with the assumption that two processes, P_2 and P_3 are disabled. Thus, Lemma 1.1 has only process P_1 to work with. And P_1 better help itself for all the enableness properties it needs, since P_2 and P_3 are paralyzed and cannot offer any help.

In general, a recursive rule or procedure is acceptable if we can identify a well-founded measure that decreases on any recursive invocation. The discussion above identifies this measure, in our case, as the number of processes which are still potentially enabled, since any additional level in the proof tree freezes an additional process. It is obvious that, when we reach a single remaining process, the property is valid only if it can be proven by the J-rules, since no other process is available to eventually improve the enableness status of transitions.

3.3 Compassion in Message Passing

The previous section considered compassionate transitions associated with semaphore statements. Here we consider communication statements, which are the other statements associated with compassion requirements. We present several examples of programs that use message passing and analyze their response properties, using rules RESP-C and CHAIN-F.

As examples, we present two algorithms for coordinating mutual exclusion by message passing which are based on compassion. We analyze these algorithms, and verify that they guarantee exclusion and individual accessibility.

Synchronous Message Passing

The first algorithm we consider is program MUX-SYNCH of Fig. 3.11. The program consists of two customer processes P_1, P_2, and an arbiter process A. The arbiter process accepts requests for entering the critical section from both processes. These requests are represented by T-messages on channels α_1, α_2. The arbiter chooses nondeterministically to communicate with one of them. This communication grants the selected customer process a permission to enter its critical section. After the customer exits from its critical section it returns this permission by sending an F-message on its α-channel to the arbiter that keeps waiting at k_2 or k_3, respectively, for this releasing communication.

We start by considering mutual exclusion. This can be established by the

local $\alpha_1,\ \alpha_2$: **channel of boolean**

$$P_1 :: \left[\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} \ell_1\text{: } \textbf{noncritical} \\ \ell_2\text{: } \alpha_1 \Leftarrow \text{T} \\ \ell_3\text{: } \textbf{critical} \\ \ell_4\text{: } \alpha_1 \Leftarrow \text{F} \end{array}\right] \end{array}\right]$$

$$\|$$

$$A :: \left[\begin{array}{l} \textbf{local } y\text{: } \textbf{boolean} \\ k_0\text{: } \textbf{loop forever do} \\ \qquad k_1\text{: } \left[\begin{array}{l} k_1^a\text{: } \alpha_1 \ \Rightarrow\ y;\ k_2\text{: } \alpha_1 \Rightarrow y \\ \quad \textbf{or} \\ k_1^b\text{: } \alpha_2 \ \Rightarrow\ y;\ k_3\text{: } \alpha_2 \Rightarrow y \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} m_0\text{: } \textbf{loop forever do} \\ \qquad \left[\begin{array}{l} m_1\text{: } \textbf{noncritical} \\ m_2\text{: } \alpha_2 \Leftarrow \text{T} \\ m_3\text{: } \textbf{critical} \\ m_4\text{: } \alpha_2 \Leftarrow \text{F} \end{array}\right] \end{array}\right]$$

Fig. 3.11. Program MUX-SYNCH (mutual exclusion by synchronous communication).

following invariants

$$\chi_1\text{:}\quad at\_\ell_{3,4} \ \leftrightarrow\ at\_k_2$$

$$\chi_2\text{:}\quad at\_m_{3,4} \ \leftrightarrow\ at\_k_3.$$

From χ_1 and χ_2 we can infer the invariant

$$\neg(at\_\ell_{3,4}\ \wedge\ at\_m_{3,4}),$$

based on process A's inability to be simultaneously at k_2 and at k_3.

Next, we prove accessibility for P_1. Accessibility for P_1 is stated by

$$\underbrace{at\_\ell_2}_{p} \;\Rightarrow\; \Diamond\, \underbrace{at\_\ell_3}_{q} \;.$$

It calls for a single application of rule RESP-C with

$$\varphi = p\colon \quad at\_\ell_2 \qquad\qquad \tau_h\colon \quad \langle \ell_2,\ k_1^a\rangle.$$

It is obvious that premises C1, C2, and C3 hold for this choice. Needing more attention is premise C4 requiring

$$\psi_1\colon \quad \underbrace{at\_\ell_2}_{\varphi} \;\Rightarrow\; \Diamond\Big(\cdots \;\vee\; \underbrace{at\_\ell_2 \;\wedge\; at\_k_1}_{En(\langle \ell_2, k_1^a\rangle)}\Big) \;.$$

This property is proved by rule CHAIN-J as shown in the CHAIN diagram of Fig. 3.12.

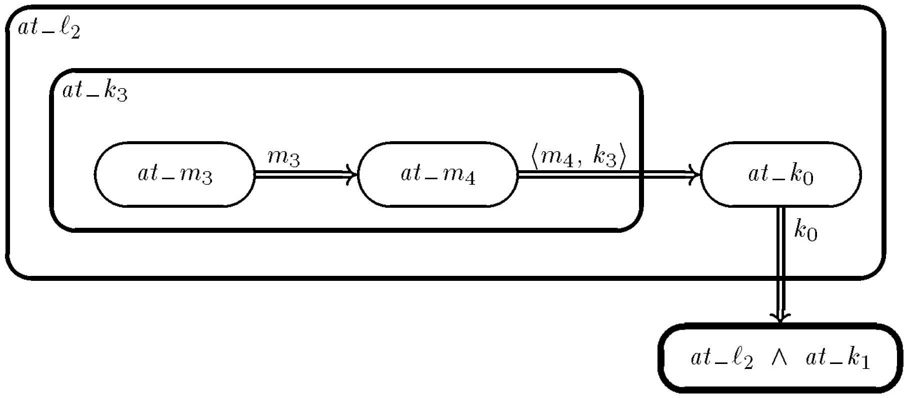


Fig. 3.12. CHAIN diagram for ψ_1.

The proof uses invariant χ_1 to infer that when P_1 is at ℓ_2, A cannot be at k_2, and invariant χ_2 to infer that when A is at k_3, P_2 can only be at m_3 or at m_4.

Accessibility for P_2 is proven in a completely symmetric way.

Asynchronous Message Passing

Program MUX-ASYNCH of Fig. 3.13 coordinates mutual exclusion by using asynchronous communication. The program uses channels which allow many readers but only one writer.

The arbiter process A starts by loading channel α with a single message, representing a permission to enter the critical section. The customer processes P_1 and P_2 compete on this single permission at locations ℓ_2 and m_2, respectively.

$$\textbf{local } \alpha,\ \beta_1,\ \beta_2\textbf{: channel } [..] \textbf{ of boolean}$$

$$P_1 :: \left[\begin{array}{l} \textbf{local } x_1\textbf{: boolean} \\ \ell_0\textbf{: loop forever do} \\ \qquad \left[\begin{array}{l} \ell_1\textbf{: noncritical} \\ \ell_2\text{: } \alpha \Rightarrow x_1 \\ \ell_3\textbf{: critical} \\ \ell_4\text{: } \beta_1 \Leftarrow \mathrm{T} \end{array}\right] \end{array}\right]$$

$$\|$$

$$A :: \left[\begin{array}{l} \textbf{local } y\textbf{: boolean} \\ k_0\textbf{: loop forever do} \\ \qquad \left[\begin{array}{l} k_1\text{: } \alpha \Leftarrow \mathrm{T} \\ k_2\text{: } \left[\begin{array}{l} k_2^a\text{: } \beta_1 \Rightarrow y \\ \quad \textbf{or} \\ k_2^b\text{: } \beta_2 \Rightarrow y \end{array}\right] \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} \textbf{local } x_2\textbf{: boolean} \\ m_0\textbf{: loop forever do} \\ \qquad \left[\begin{array}{l} m_1\textbf{: noncritical} \\ m_2\text{: } \alpha \Rightarrow x_2 \\ m_3\textbf{: critical} \\ m_4\text{: } \beta_2 \Leftarrow \mathrm{T} \end{array}\right] \end{array}\right]$$

Fig. 3.13. Program MUX-ASYNCH (mutual exclusion by asynchronous communication).

Whoever succeeds in reading the message from channel α proceeds to enter the critical section. On finishing, this process sends back a release message via β_1 or β_2, respectively. Meanwhile, process A keeps waiting at k_2 to collect this release message. On receiving the message, process A loops back and reloads channel α with a new permission.

To prove exclusion, we establish the invariant

$$\chi_1: \quad at\_k_{0,1} + |\alpha| + at\_\ell_{3,4} + at\_m_{3,4} + |\beta_1| + |\beta_2| = 1.$$

Clearly, this invariant implies that $at\_\ell_{3,4} + at\_m_{3,4} \leq 1$.

Accessibility for P_1 is expressed by

$$\underbrace{at\_\ell_2}_{p} \;\Rightarrow\; \Diamond\, \underbrace{at\_\ell_3}_{q}\;.$$

This can be proven by rule RESP-C, taking

$$\varphi = p\colon\ at\_\ell_2 \qquad \tau_h\colon\ \ell_2.$$

The first three premises C1–C3 obviously hold. It only remains to show premise C4, the eventual enableness of ℓ_2. This is expressed by

$$\psi_2\colon\quad \underbrace{at\_\ell_2}_{\varphi} \;\Rightarrow\; \Diamond\Big(\cdots \;\vee\; \underbrace{at\_\ell_2 \;\wedge\; |\alpha| > 0}_{En(\ell_2)}\Big)\;.$$

A CHAIN-J proof of this property is presented in the CHAIN diagram of Fig. 3.14. The proof uses invariant χ_1 to infer that when $|\alpha| = 0$ then either A is at $k_{0,1}$, or it is at k_2 with $|\beta_2| = 1$, or at k_2 with $|\beta_2| = 0$ and P_2 at $m_{3,4}$.

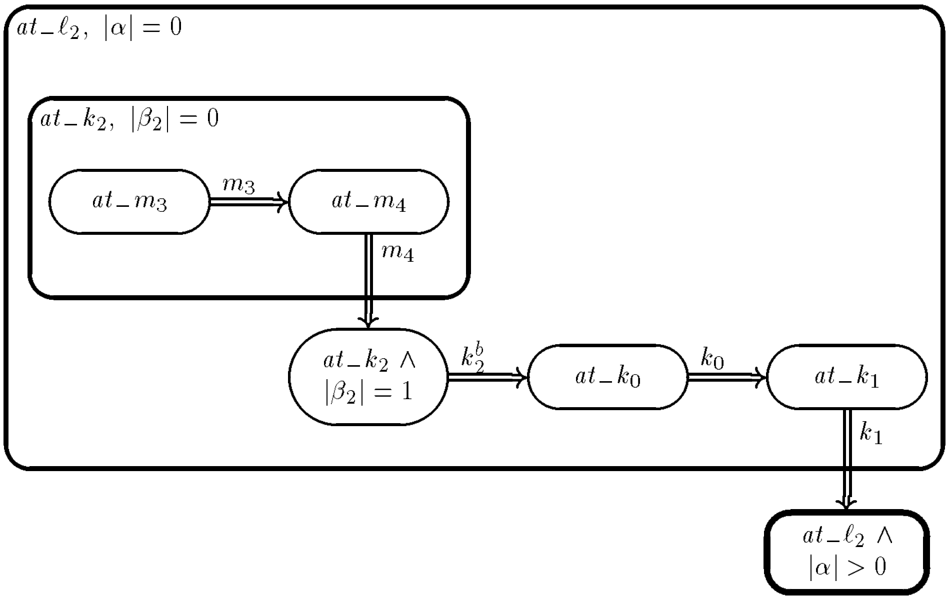


Fig. 3.14. CHAIN diagram for ψ_2.

3.4 Modeling Eventual Reliability

The preceding sections used compassion as an abstraction for a protocol that is implemented either by special hardware or by a software procedure. Our two main examples were semaphore and communication statements.

There are additional contexts and applications where compassion provides a useful abstraction. We introduce here one such application, in which compassion is used to model *eventual reliability* of a hardware component of the system.

Modeling Faulty Channels

Consider program REL-TRANS of Fig. 3.15. This program consists of process P_1 that sends messages $1, 2, \ldots$ on channel α, and process P_2 that reads these messages from channel α into y.

$$\textbf{local } \alpha\text{: } \textbf{channel } [1..] \textbf{ of integer}$$

$$P_1 :: \begin{bmatrix} \textbf{local } x\text{: } \textbf{integer where } x = 0 \\ \ell_0\text{: } \textbf{loop forever do} \\ \qquad \begin{bmatrix} \ell_1\text{: } x := x + 1 \\ \ell_2\text{: } \alpha \Leftarrow x \end{bmatrix} \end{bmatrix} \quad \| \quad P_2 :: \begin{bmatrix} \textbf{local integer } y \\ m_0\text{: } \textbf{loop forever do} \\ \qquad m_1\text{: } \alpha \Rightarrow y \end{bmatrix}$$

Fig. 3.15. Program REL-TRANS (reliable transmission).

It is a simple matter to verify that y assumes the values $1, 2, \ldots$.

Such verification depends, however, on the implicit assumption that channel α is reliable. This assumption is reflected in the definition of the transitions associated with the communication statements. For program REL-TRANS, these definitions imply that execution of ℓ_2: $\alpha \Leftarrow x$ appends x to the end of list α, and the message remains in α until removed by execution of m_1: $\alpha \Rightarrow y$.

In some cases, we may wish to study situations in which channel α is not reliable and may either lose some messages or transmit them with corrupted values. There are two possible approaches to modeling such faulty channels.

The first approach associates with communication statements over faulty channels different types of transitions. These transitions should represent the potentially faulty behavior of such channels.

Another approach, which is the one adopted here, is not to change the association of transitions to communication statements but to modify the program, replacing each *send* statement by an appropriate selection statements. We will illustrate this approach for two types of faults.

Modeling Lossy Channels

The first fault we model is that of a channel that may lose some of its messages. Program LOSSY-TRANS of Fig. 3.16 represents a modified version of program REL-TRANS, which takes into account the possibility of lost messages.

$$\textbf{local } \alpha\text{: } \textbf{channel } [1..] \textbf{ of integer}$$

$$P_1 :: \left[\begin{array}{l} \textbf{local } x\text{: } \textbf{integer where } x = 0 \\ \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: } x := x + 1 \\ \ell_2\text{: } \left[\begin{array}{l} \ell_2^g\text{: } \alpha \Leftarrow x \\ \quad \textbf{or} \\ \ell_2^s\text{: } \textbf{skip} \end{array}\right] \end{array}\right] \end{array}\right] \quad \| \quad P_2 :: \left[\begin{array}{l} \textbf{local integer } y \\ m_0\text{: } \textbf{loop forever do} \\ \quad m_1\text{: } \alpha \Rightarrow y \end{array}\right]$$

Fig. 3.16. Program LOSSY-TRANS (transmission with lossy channels).

Program LOSSY-TRANS is obtained by replacing statement ℓ_2: $\alpha \Leftarrow x$ of program REL-TRANS by the selection statement

$$[\ell_2^g\text{: } \alpha \Leftarrow x] \ \textbf{ or } \ [\ell_2^s\text{: } \textbf{skip}].$$

This implies that, whenever process P_1 is ready to send x over channel α, there is a nondeterministic choice between taking the *good transition* ℓ_2^g which actually sends x on α, or taking the *skip* transition ℓ_2^s which does nothing and models a loss of the message. Thus, computations of program LOSSY-TRANS faithfully represent the situation that some messages are lost, and therefore y may assume only a subset of the values $1, 2, \ldots$.

Such a transformation is intended to model operations on an abstract faulty asynchronous channel $\widehat{\alpha}$. Taking either ℓ_2^g or ℓ_2^s represents sending x on $\widehat{\alpha}$: Taking ℓ_2^g represents a successful transmission on $\widehat{\alpha}$, while taking ℓ_2^s represents a loss of a message. Transition m_1 represents receiving a message from $\widehat{\alpha}$.

It is important to realize that the faulty channel $\widehat{\alpha}$ modeled by this transformation is *eventually reliable*. This means that, if infinitely many messages are sent, channel $\widehat{\alpha}$ cannot lose *all* of them from a certain point on. This excludes computations in which infinitely many messages are sent on $\widehat{\alpha}$ but all but finitely many of them are lost.

Eventual reliability of our model follows from the requirement of compassion for the communication statement ℓ_2^g: $\alpha \Leftarrow x$. If P_1 attempts to send a message infinitely many times (i.e., visits ℓ_2 infinitely many times), then transition ℓ_2^g is enabled infinitely many times and, by compassion, must eventually be taken. It follows that, eventually, some x will be placed on channel α, and process P_2 will read it.

Modeling Corrupt Channels

Another type of fault that may be displayed by unreliable (but eventually reliable) channels is corruption of messages. This fault transmits a sent message but corrupts its value. In program CORR-TRANS of Fig. 3.17 we model a channel that may lose some messages and corrupt some of the others but eventually also transmits some correct messages. The selection statement in ℓ_2 contains three substatements. Statement ℓ_2^g represents faithful transmission; ℓ_2^s represents a loss of a message; and statement ℓ_2^c represents transmission of a corrupted value: 0 instead of x.

$$\textbf{local } \alpha\text{: } \textbf{channel } [1..] \textbf{ of integer}$$

$$P_1 :: \left[\begin{array}{l} \textbf{local } x\text{: } \textbf{integer} \\ \quad \textbf{where } x=0 \\ \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: } x := x+1 \\ \ell_2\text{: } \left[\begin{array}{l} \ell_2^g\text{: } \alpha \Leftarrow x \\ \quad \textbf{or} \\ \ell_2^s\text{: } \textbf{skip} \\ \quad \textbf{or} \\ \ell_2^c\text{: } \alpha \Leftarrow 0 \end{array}\right] \end{array}\right] \end{array}\right] \quad \| \quad P_2 :: \left[\begin{array}{l} \textbf{local integer } y \\ m_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1\text{: } y := 0 \\ m_2\text{: } \textbf{while } y=0 \textbf{ do} \\ \qquad m_3\text{: } \alpha \Rightarrow y \end{array}\right] \end{array}\right]$$

Fig. 3.17. Program CORR-TRANS (transmission with corrupting channels).

The receiving process P_2 in this program is aware of the possibility of corruption, and is programmed to keep receiving until it gets a nonzero message.

Again, due to compassion, the modeled channel is eventually reliable, excluding computations in which, from a certain point on, all messages are either lost or corrupted. Note, that a side effect of this modeling is that it also implies that some messages must eventually be correpted.

This approach to modeling faulty channels is also applicable to synchronous communication, using the same transformations.

Example (alternating bit protocol)

Consider program ALTER of Fig. 3.18. This program is designed to ensure reliable communication over a faulty channel. We assume a particular type of fault, by which each message is either transmitted intact or corrupted in a noticeable way, i.e., no messages are lost. The program uses synchronous channel *trans* to transmit messages from P_1 to P_2, and synchronous channel *ack* to send acknowledgements from P_2 to P_1.

Each message sent over channel *trans* is a record with three components: pt, d, and ct. As we will see, pt serves as an identification of the message, d is the data that should be reliably transmitted from P_1 to P_2, and ct is a boolean field indicating whether the message has been transmitted correctly (correct transmission) or corrupted by the faulty channel.

A message sent over channel *ack* is also a record with the two components pa and ca. Field pa identifies the message that is acknowledged. Boolean field ca indicates whether the message is transmitted correctly (correct acknowledgement) or corrupted.

The possibility of corruption of a transmitted message is represented in Fig. 3.18 as a nondeterministic but fair (due to compassion) choice of transmitting the intended message with an additional bit of either T or F value. The choice of T for the additional bit represents a successful transmission, and then the other fields of the message are reliably valid. The case of F for the added bit, represents a failed transmission, in which the other fields of the message are unreliable, except for the F bit which identifies the message as a corrupt one.

It is important to realize that Fig. 3.18 contains a mixture of a program for the protocol and additional transitions such as ℓ_1^c and m_4^c, which model the potentially erratic behavior of the eventually reliable channels. The protocol, which is the program that will run in an actual application, has the statements

$$\ell_1\colon\ trans \Leftarrow \langle S,\ X[i],\ \mathrm{T}\rangle \qquad m_4\colon\ ack \Leftarrow \langle R,\ \mathrm{T}\rangle$$

at locations ℓ_1 and m_4. The *selection* statements appearing at these locations in Fig. 3.18 are intended to model the behavior of *trans* and *ack* as eventually reliable channels.

- **The Protocol**

The main idea in the protocol is that it maintains an acknowledgement channel, called *ack*, in which the receiver reports whether the message has arrived correctly, or has been corrupted. In both cases, the message in the *ack* channel is a request for a next transmission. In the case that the last message has been correctly received, the request is for the next message. In the case that the last

$$
\begin{array}{lll}
\textbf{in} & X & : \textbf{array } [1..] \textbf{ of integer} \\
\textbf{local} & i,\ j & : \textbf{integer where } i = j = 1 \\
 & \multicolumn{2}{l}{s,\ r,\ S,\ R,\ gt,\ ga\text{: } \textbf{boolean where } S = R = \mathrm{T}} \\
 & \multicolumn{2}{l}{\mathit{trans}\text{: } \textbf{channel of } \langle pt\text{: } \textbf{boolean},\ d\text{: } \textbf{integer},\ ct\text{: } \textbf{boolean}\rangle} \\
 & \mathit{ack} & : \textbf{channel of } \langle pa\text{: } \textbf{boolean},\ ca\text{: } \textbf{boolean}\rangle \\
\textbf{out} & Y & : \textbf{array } [1..] \textbf{ of integer}
\end{array}
$$

$$
P_1 :: \left[\begin{array}{l}
\ell_0\text{: } \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\text{: } \left[\begin{array}{l}
\ell_1^g\text{: } \mathit{trans} \Leftarrow \langle S,\ X[i],\ \mathrm{T}\rangle \\
\quad \textbf{or} \\
\ell_1^c\text{: } \mathit{trans} \Leftarrow \langle -,\ -,\ \mathrm{F}\rangle
\end{array}\right] \\
\ell_2\text{: } \mathit{ack} \Rightarrow \langle r,\ ga\rangle \\
\ell_3\text{: } \textbf{if } ga \wedge r \neq S \textbf{ then} \\
\qquad \ell_4\text{: } (i,\ S) := (i+1,\ \neg S)
\end{array}\right]
\end{array}\right]
$$

$$||$$

$$
P_2 :: \left[\begin{array}{l}
m_0\text{: } \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
m_1\text{: } \mathit{trans} \Rightarrow \langle s,\ Y[j],\ gt\rangle \\
m_2\text{: } \textbf{if } gt \wedge s = R \textbf{ then} \\
\qquad m_3\text{: } (j,\ R) := (j+1,\ \neg R) \\
m_4\text{: } \left[\begin{array}{l}
m_4^g\text{: } \mathit{ack} \Leftarrow \langle R,\ \mathrm{T}\rangle \\
\quad \textbf{or} \\
m_4^c\text{: } \mathit{ack} \Leftarrow \langle -,\ \mathrm{F}\rangle
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Fig. 3.18. Program ALTER (alternating bit protocol with synchronous communication).

message has been incorrectly received, the request is for a retransmission of the last message. Corruption is also possible in the acknowledgement message. Consequently, it can be that the last message has been correctly received, but the sender is unaware of this fact, and keeps retransmitting the old message until a correct acknowledgement arrives. This possibility is also handled by the protocol.

The interesting point about this protocol is that, under the above assumptions about the characteristics of the fault, only one bit of information, represented by a boolean value, is necessary to identify whether the sender is transmitting the old message or a new message, and whether the receiver is asking for a transmission of a new message or for retransmission of an old one.

- **The Parity Bit**

It is possible to consider a simpler but less space-efficient algorithm in which fields *pt* and *pa* contain the full sequence number of the message with each message and each acknowledgement, i.e., i and j, respectively. In this simpler algorithm, the receiver knows immediately which message it is receiving, and the sender immediately recognizes which message is being acknowledged.

However, since messages are transmitted and acknowledged in sequence, and none are lost (by assumption), it is clear that the possibilities for confusion about which message is being transmitted or acknowledged are limited and can be resolved by one bit. When P_2 is awaiting message $X[j]$, and some message is successfully received, it can either be $X[j]$ or $X[j-1]$. Similarly, when P_1 is awaiting a request for $i+1$ (which indirectly acknowledges message i), it can receive a request either for $i+1$ or for i. The latter is a request for a retransmission of the old message.

Therefore, it is sufficient to enclose with each message the *parity bit* of the full sequence number (T for odd numbers, F for even ones). This will resolve the ambiguity between j and $j-1$, and between $i+1$ and i.

Indeed, initially $i = j = 1$ and the corresponding parity bits, S in P_1 (the sender parity bit) and R in P_2 (the receiver parity bit) are both T. Process P_1, when sending $X[i]$, includes S with it, which is the parity bit of i. When P_2 receives a message in m_1, it checks at m_2 for corruption and whether the sequence number of the received message, represented by its parity bit, matches P_2's expectations of receiving $X[j]$. If the message is not corrupt ($gt = \text{T}$) and the received parity bit matches R, then this is the expected message.

Consequently, both j and R are advanced in m_3 to the next sequence number and its corresponding parity bit. Otherwise j and R retain their old values.

Next, P_2 sends in m_4 a request for the next transmission which names $X[j]$ as the requested message. This is represented by sending R, the parity bit of j. In the case j has not been advanced, this is a request for retransmission of $X[j]$. In case j has been advanced, this is a request for the next message.

In ℓ_2, P_1 reads this request. In ℓ_3 it checks for a correct reception of the request message, and whether the request is for the next message, tested by $r \neq S$. If it is a request for a new message, then both i and S are advanced in ℓ_4. Otherwise they stay the same.

- **A Safety Property**

An appropriate safety property we wish to establish is that

> Messages approved by the receiver are identical to the original mes-

sages sent by the sender.

For convenience, we have represented the list of messages transmitted by the sender as an input array X, and the list of messages approved by the receiver as an output array Y. These arrays are not an integral part of the algorithm, and are only used for the purpose of the specification and verification of the program.

The main assertion, whose invariance implies the safety property described above, is

$$\psi_1: \quad Y[1..j{-}1] \;=\; X[1..j{-}1].$$

This assertion states that the segment of the array Y consisting of positions $1, \ldots, j{-}1$ is identical to the corresponding segment of the array X. Since at any point in the program for P_2, j denotes the sequence number of the message that P_2 attempts to receive, we can assume that it has already approved all the preceding messages, currently stored in $Y[1..j{-}1]$.

The proof of invariant ψ_1 uses the following auxiliary invariants:

$$I_1: \quad (S = R \,\wedge\, i = j) \,\vee\, (S \neq R \,\wedge\, i+1 = j)$$

$$I_2: \quad at\_\ell_3 \,\wedge\, ga \;\rightarrow\; r = R$$

$$I_3: \quad at\_m_2 \,\wedge\, gt \;\rightarrow\; s = S$$

$$I_4: \quad at\_\ell_4 \;\rightarrow\; S \neq R$$

$$I_5: \quad at\_m_3 \;\rightarrow\; S = R$$

$$I_6: \quad at\_\ell_2 \;\leftrightarrow\; at\_m_{2..4}.$$

In **Problem 3.1**, the reader is requested to prove the invariant ψ_1.

An alternative proof of invariant ψ_1 may be based on the assertion diagram of Fig. 3.19 which summarizes the states that may appear in a computation of program ALTER.

- **Proving Progress**

As previously explained, the nondeterministic selections at ℓ_1 and m_4 are not statements that actually exist in the protocol, but are intended to represent the unreliability of channels *trans* and *ack*. They present the possibility that instead of receiving a good record, containing $\langle S, X[i], \mathrm{T}\rangle$ as prepared by P_1, process P_2 may receive a faulty record with unpredictable values for the first two fields. The only alleviating assumption we make is that P_2 can recognize whether it has received a valid or faulty message. This is represented by the last field in the message, which is T for a valid message and F for a faulty one. The same assumptions are made about channel *ack*, which explains the nondeterministic selection at m_4.

The main response property that is natural for this problem is that any message that P_1 attempts to send is eventually correctly received by P_2. In view

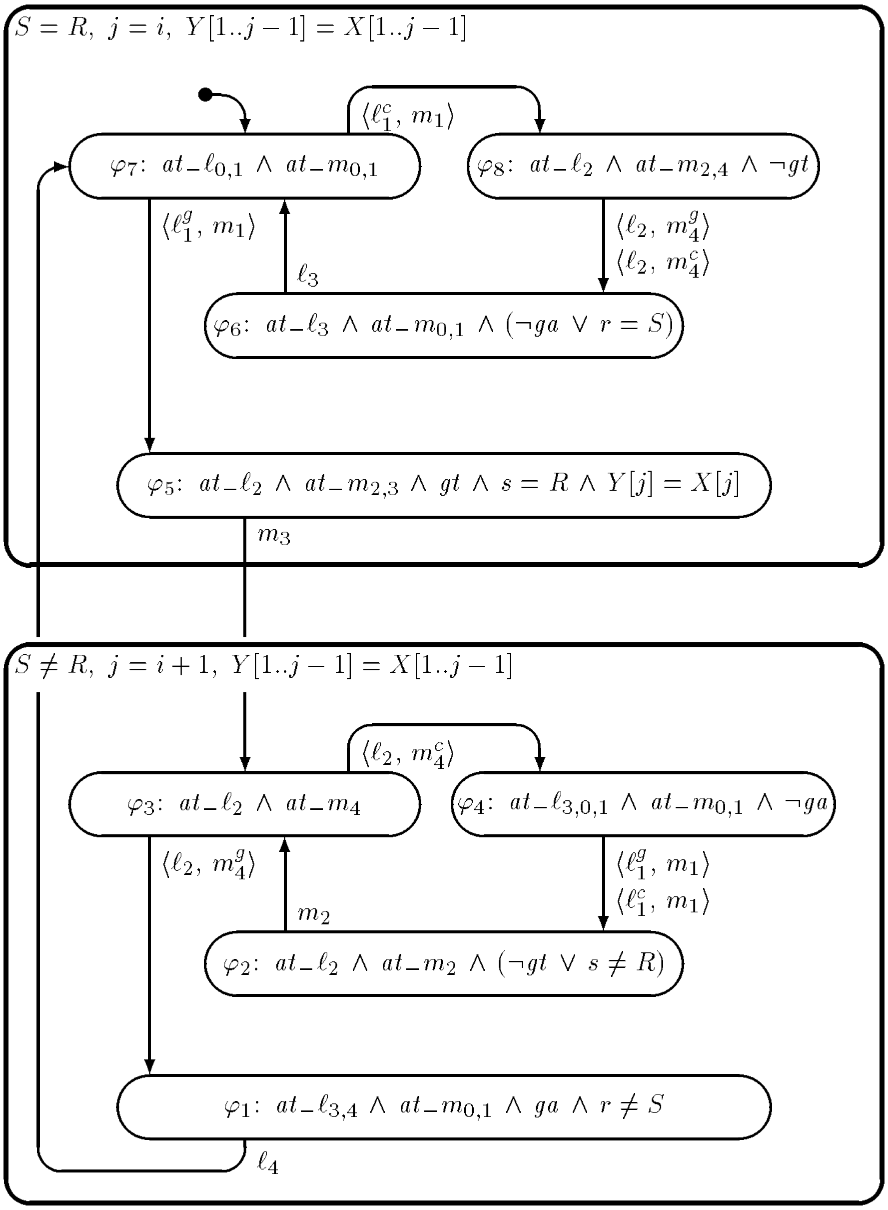


Fig. 3.19. Possible states of program ALTER.

of the safety property proved above, it is sufficient to show that j eventually increases. This can be expressed by the response property

$$\psi_2\colon \quad j = u \;\Rightarrow\; \diamondsuit(j = u + 1),$$

using rigid variable u to record the current value of j.

By invariant I_1, either $j = i$ or $j = i + 1$. Considering the two cases, it is possible to decompose ψ_2 into the two response properties

$$\psi_3\colon \quad i = j = u \;\;\Rightarrow\;\; \diamondsuit(i + 1 = j = u + 1)$$
$$\psi_4\colon \quad i + 1 = j = u \;\;\Rightarrow\;\; \diamondsuit(i = j = u)$$

and prove them separately.

Consulting Fig. 3.19, we see that the behavior of the program is partitioned into two alternating phases. The first phase, occupying the top of the diagram, is characterized by $j = i$. In this phase process P_1 repeatedly attempts to send message $X[i]$ to P_2. A successful attempt will lead to φ_5, which some steps later terminates the phase. Any failed attempt causes the system to traverse the cycle φ_7, φ_8, and φ_6 and return to φ_7. This cycle cannot repeat indefinitely since each visit to φ_7 involves a state in which the compassionate transition $\langle \ell_1^g, m_1\rangle$ is enabled. By compassion, the system must eventually get to φ_5 and then proceed to φ_3. Response formula ψ_3 states the termination of the upper phase.

The second phase, occupying the bottom half of the diagram, is characterized by $j = i + 1$. In this phase, process P_2 attempts to acknowledge correct reception of message $X[j] = X[i + 1]$. Each failed attempt traverses the cycle φ_3, φ_4, φ_2, returning to φ_3 once. By compassion, eventually $\langle \ell_2, m_4^g\rangle$ must be taken, leading to φ_1 and the eventual termination of this phase, as required by ψ_1.

To summarize, it cannot be that one of the processes sends infinitely many messages and all of them, from a certain point, arrive corrupted. This implies that, since P_1 persistently sends message $m[u]$ until correctly acknowledged, eventually this message will arrive uncorrupted and P_2 will increase j to $u + 1$. Following this correct arrival, process P_2 persistently sends an acknowledgement for $m[u]$ (requesting $m[i + 1]$) until a correct $m[u + 1]$ arrives. Eventually, one of these acknowledgements will arrive correctly and P_1 will increase i to $u + 1$.

We proceed to substantiate this intuition by formally proving ψ_2. As suggested, we first establish

$$\psi_3\colon \quad (i = j = u) \;\Rightarrow\; \diamondsuit(i + 1 = j = u + 1).$$

The proof of this property, using rule CHAIN-F, is presented in the CHAIN-F diagram of Fig. 3.20.

It is easy to verify premises F1–F4 of rule CHAIN-F for assertions φ_1 and φ_2, appearing in Fig. 3.20.

It only remains to check eventual enableness of $\langle \ell_1^g, m_1\rangle$ from φ_3-states.

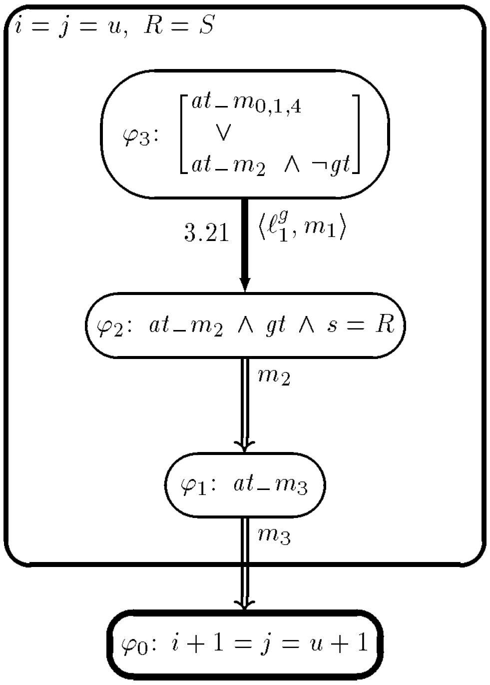


Fig. 3.20. Top-level CHAIN-F diagram for ψ_3.

This means that we have to show

$$at\_m_{0,1,4} \ \vee\ (at\_m_2 \ \wedge\ \neg gt) \ \Rightarrow\ \diamondsuit(at\_\ell_1 \ \wedge\ at\_m_1).$$

This is proven by CHAIN-J as presented in the CHAIN diagram of Fig. 3.21. The proof uses I_6 to infer that when P_2 is at m_2 or at m_4, P_1 must be at ℓ_2. It also uses I_2 and I_3 to infer that, while P_2 is at m_1 and $S = R$, P_1 can be only at ℓ_0, at ℓ_1, or at ℓ_3 with $\neg ga \vee r = S$.

This concludes the proof of the property

$$\psi_3\colon \quad i = j = u \ \Rightarrow\ \diamondsuit(i + 1 = j = u + 1).$$

The second response property ensures that eventually P_1 will also increase counter i

$$\psi_4\colon \quad i + 1 = j = u \ \Rightarrow\ \diamondsuit(i = j = u).$$

The proof of this property is presented in the CHAIN-F diagram of Fig. 3.22.

Eventual enableness of $\langle \ell_4, m_5 \rangle$ is proven by the CHAIN diagram of Fig. 3.23.

Combining properties ψ_3 and ψ_4, we obtain the main response property of

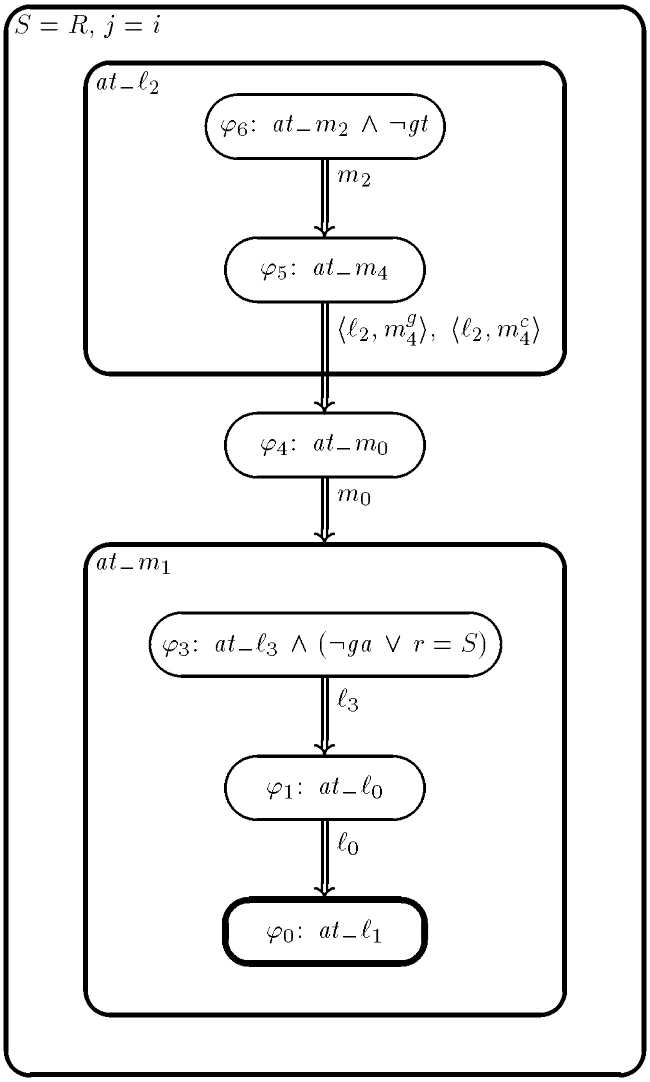


Fig. 3.21. CHAIN diagram for eventual enableness of $\langle \ell_1^g,\ m_1 \rangle$.

program ALTER

$$\psi_2: \quad j = u \;\Rightarrow\; \Diamond(j = u + 1). \qquad \blacksquare$$

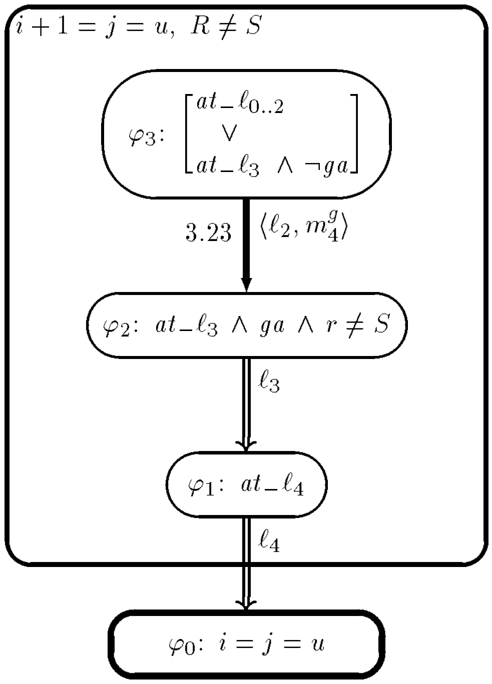


Fig. 3.22. Top-level CHAIN-F diagram for ψ_4.

3.5 *Well-Founded Rule*

The next rule we consider is the fair version of rule WELL-J (Fig. 1.26). This version can be obtained by replacing the temporal premise W2 of rule WELL (Fig. 1.23) by the appropriate premises of rule RESP-C for the case that τ_i is compassionate and the appropriate premises of rule RESP-J for a just τ_i. This leads to rule WELL-F (Fig. 3.24).

In applications of the rule, it is sufficient to establish premise FW2 for all $\tau \neq \tau_i$, since FW2 for $\tau = \tau_i$ is implied by FW3.

A verification diagram corresponding to rule WELL-F is called a RANK-F diagram.

Example (Producer Consumer)

Consider program PROD-CONS of Fig. 3.5. Using rule CHAIN-F, we proved accessibility for the producer, i.e., $at\_\ell_2 \Rightarrow \diamond\, at\_\ell_4$. However, the main response property of this program is that any value produced by the producer is eventually consumed by the consumer. This property can be written as

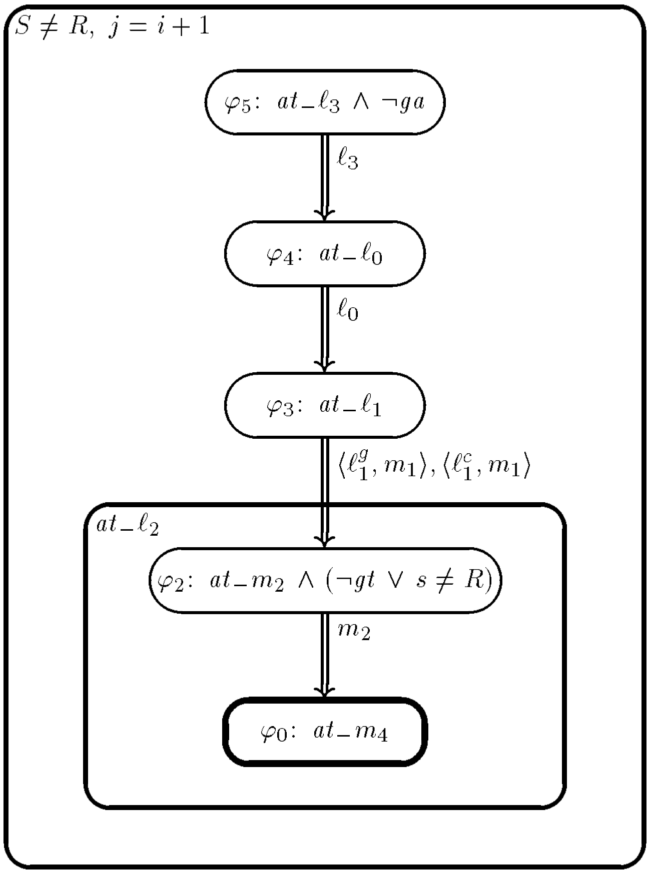


Fig. 3.23. CHAIN diagram for eventual enableness of $\langle \ell_2, m_4^g \rangle$.

$$prod(u) \;\Rightarrow\; \diamondsuit\, cons(u),$$

where we define

$$prod(u)\colon\ at\_\ell_2 \;\wedge\; x = u,$$

$$cons(u)\colon\ at\_m_6 \;\wedge\; y = u.$$

These two state formulas characterize, respectively, the situation that process *Prod* has just produced the value u at statement ℓ_1, and the situation that process *Cons* is about to consume the value u at statement m_6. Substituting the definitions for *prod* and *cons*, we obtain

$$\underbrace{at\_\ell_2 \;\wedge\; x = u}_{p} \;\Rightarrow\; \diamondsuit\, \underbrace{at\_m_6 \;\wedge\; y = u}_{q}\ .$$

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

$$\text{FW1.} \quad p \;\rightarrow\; \bigvee_{j=0}^{m} \varphi_j$$

$$\left.\begin{array}{l}
\text{FW2.} \quad \rho_\tau \;\wedge\; \varphi_i \;\rightarrow\; \begin{bmatrix} \displaystyle\bigvee_{j=0}^{m} (\varphi'_j \;\wedge\; \delta_i \succ \delta'_j) \\ \vee \;\; (\varphi'_i \;\wedge\; \delta_i = \delta'_i) \end{bmatrix} \\
\qquad\qquad\qquad\qquad\qquad \text{for every } \tau \in \mathcal{T} \\
\text{FW3.} \quad \rho_{\tau_i} \;\wedge\; \varphi_i \;\rightarrow\; \displaystyle\bigvee_{j=0}^{m} (\varphi'_j \;\wedge\; \delta_i \succ \delta'_j) \\
\text{FW4.} \quad \text{If } \tau_i \in \mathcal{C}\text{, then} \\
\qquad\quad \text{CW4.} \quad \varphi_i \;\Rightarrow\; \diamondsuit\big(\neg\varphi_i \;\vee\; En(\tau_i)\big) \\
\qquad \text{Otherwise} \\
\qquad\quad \text{JW4.} \quad \varphi_i \;\rightarrow\; En(\tau_i)
\end{array}\right\} \text{ for } i = 1, \ldots, m$$

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

Fig. 3.24. Rule WELL-F (well-founded rule under fairness).

We break the proof into three lemmas.

Lemma A $\quad at\_\ell_2 \;\wedge\; x = u \;\;\Rightarrow\;\; \diamondsuit(u \in b)$

This lemma states that if u is produced, it eventually will be placed in buffer b. The proof of this lemma closely resembles the proof of accessibility that we have proven before, and therefore will be omitted.

Lemma B $\quad u \in b \;\;\Rightarrow\;\; \diamondsuit(at\_m_4 \;\wedge\; y = u)$

This lemma claims that if message u is somewhere in the buffer, then eventually it will be read by process *Cons* and placed in y. The proof of this lemma is presented below.

Lemma C $\quad at\_m_4 \ \wedge \ y = u \ \Rightarrow \ \Diamond(at\_m_6 \ \wedge \ y = u)$

This lemma completes the journey of u, by following *Cons* from m_4 to m_6. It is provable by a simple application of CHAIN-J, and the proof is omitted.

- *Proof of Lemma B*

  The proof of Lemma B is assisted by the invariants

$$\chi_0: \quad r \geq 0 \ \wedge \ ne \geq 0 \ \wedge \ nf \geq 0$$

$$\chi_1: \quad r + at\_\ell_{4,5} + at\_m_{3,4} = 1$$

$$\chi_2: \quad ne + nf + at\_\ell_{3..6} + at\_m_{2..5} = N.$$

The proof is based on rule WELL-F and is presented in the RANK-F diagram of Fig. 3.25. It uses intermediate assertions $\varphi_1, \ldots, \varphi_7$. As the well-founded domain we take the lexicographic product $\mathbb{N} \times [0..7]$. The ranking function for assertion φ_i, $i = 1, \ldots, 7$, is given by

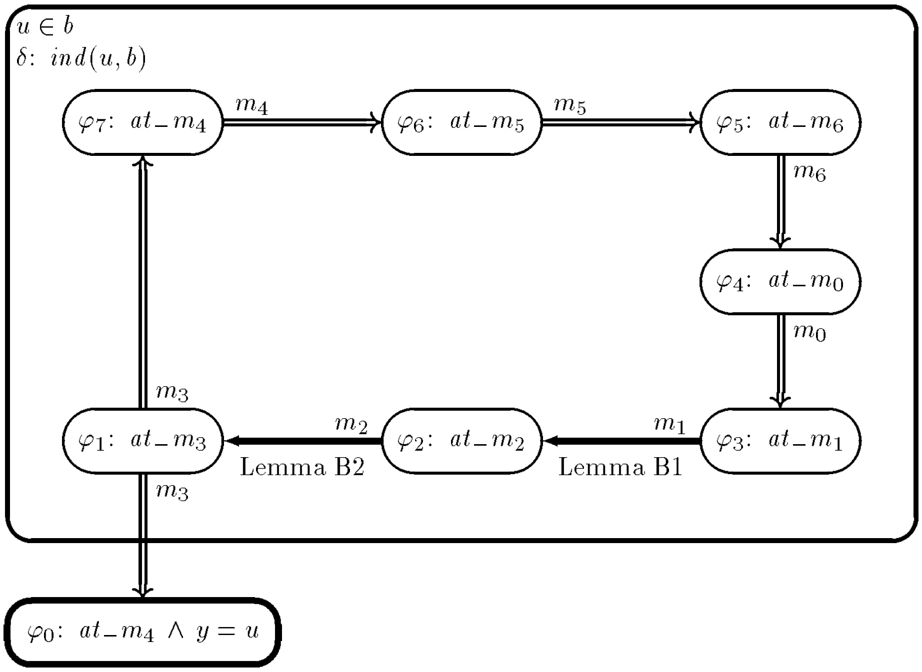


Fig. 3.25. RANK-F diagram for eventual transmission of messages.

$$\delta_i: \quad \big(ind(u, b),\ i\big),$$

where $ind(u, b)$ is defined as

$$ind(u,\ b) \ = \ min\{\ i \geq 1 \mid b[i] = u\ \}.$$

That is, $ind(u, b)$ is the smallest index i, $i \geq 1$, such that $b[i] = u$. Clearly, the value of $ind(u, b)$ keeps decreasing as *Cons* removes elements from b. We rely on the obvious implication

$$u \in b \quad \rightarrow \quad \exists i \geq 1\colon\ b[i] = u.$$

We adopt the convention by which if $u \notin b$ then $ind(u, b) = 0$. It follows that $ind(u, b) \geq 0$ in all cases.

It is not difficult to see that premises FW1–FW3 are satisfied by the assertions appearing in the diagram. In particular, we observe that no transition, except for m_3, can change the value of $ind(u, b)$, under the assumption of $u \in b$. This also holds for ℓ_4. Also, it is not difficult to see that the full ranking functions δ_i: $\big(ind(u, b), i\big)$ decrease on any helpful transition, unless the goal φ_0: $at\_m_4 \ \wedge\ y = u$ is achieved. The transition m_3, either achieves φ_0 in the case that $ind(u, b) = 1$, or maintains $u \in b$, with $ind(u, b') < ind(u, b)$. All other helpful transitions preserve $ind(u, b)$, and lead from the rank $\big(ind(u, b), i\big)$ to the rank $\big(ind(u, b), i - 1\big)$, for some i, $i > 0$.

Let us consider premise FW4. Most of the helpful transitions are just transitions for which we can easily establish JW4. The exceptions are the compassionate transitions m_1 and m_2 for which we will establish CW4.

To establish CW4, it is sufficient to show the eventual enableness of m_2 and m_3. These are proven by the two following lemmas.

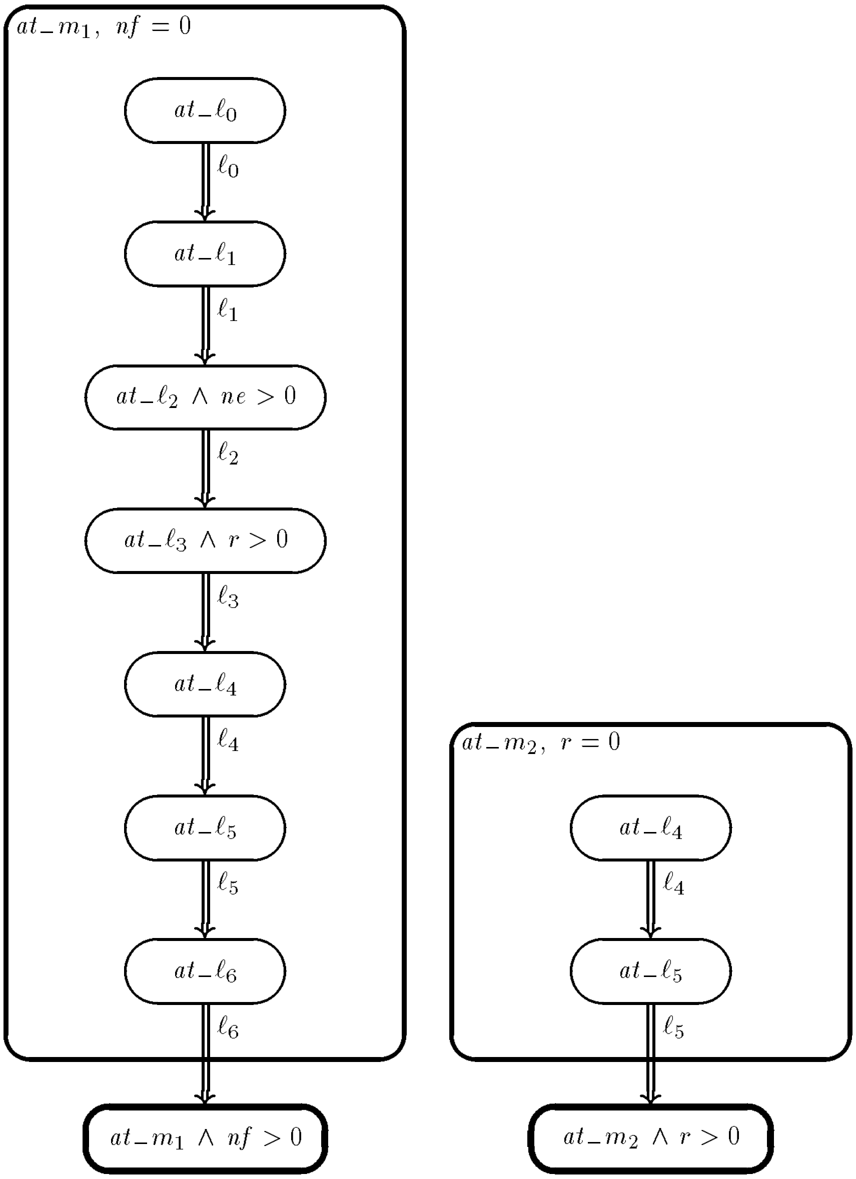
Lemma B1 $\quad at\_m_1 \ \Rightarrow\ \diamondsuit(at\_m_1 \ \wedge\ nf > 0)$

The lemma is proven by rule CHAIN-J, as presented in the CHAIN diagram of Fig. 3.26(a). Clearly if nf is not positive it must equal 0. We use invariant χ_2 and the fact that P_2 is at m_1 and $nf = 0$, to deduce that while P_1 is at ℓ_2, $ne = N > 0$. Similarly, we use χ_1 to deduce that while P_1 is at ℓ_3 and P_2 is at m_1, r is positive.

Lemma B2 $\quad at\_m_2 \ \Rightarrow\ \diamondsuit(at\_m_2 \ \wedge\ r > 0)$

The lemma is proven by rule CHAIN-J, as presented in the CHAIN diagram of Fig. 3.26(b). According to χ_0 and χ_1, if r is nonpositive while P_2 is at m_2, then r must equal 0, and P_1 must reside at ℓ_4 or at ℓ_5.

This concludes the proof of Lemma B, as well as the proof of the response property that ensures that any value produced by *Prod* eventually gets consumed by *Cons*. ∎



(a) Proof of Lemma B1. (b) Proof of Lemma B2.

Fig. 3.26. CHAIN diagrams.

3.6 The Dining Philosophers Problem

In Section 2.3 of the SAFETY book, we presented the dining philosophers problem and studied three proposed solutions to the problem given by programs DINE (Fig. 2.15), DINE-CONTR (Fig. 2.16), and DINE-EXCL (Fig. 2.17). In that section, we established the safety property of chopstick exclusion, by which no two adjacent processes (which share a chopstick) can reside in their critical section at the same time.

Here, we wish to study the progress property of these solutions, which is the property of accessibility. This property states that every philosopher who becomes hungry (exits its noncritical section) will eventually eat (enter its critical section).

As shown in Section 2.3 of the SAFETY book, program DINE may deadlock, which means that no accessibility property is valid for it. It remains to consider the two other programs: DINE-CONTR and DINE-EXCL.

Accessibility for Program DINE-CONTR

In Fig. 3.27 we reproduce program DINE-CONTR, first presented in Fig. 2.16 of the SAFETY book.

For this program, we established the following invariant assertions:

$$\varphi_0[j]:\quad c[j] \;\geq\; 0, \text{ for every } j \in [1..M]$$

$$\varphi_1[j]:\quad at\_\ell_{4..6}[j] \;+\; at\_\ell_{3..5}[j+1] \;+\; c[j+1] \;=\; 1, \text{ for every } j \in [1..M{-}2]$$

$$\varphi_2\quad:\quad at\_\ell_{4..6}[M{-}1] \;+\; at\_\ell_{4..6}[M] \;+\; c[M] \;=\; 1$$

$$\varphi_3\quad:\quad at\_\ell_{3..5}[M] \;+\; at\_\ell_{3..5}[1] \;+\; c[1] \;=\; 1.$$

We will show that the accessibility property

$$A[j]:\quad at\_\ell_2[j] \;\Rightarrow\; \diamondsuit\, at\_\ell_4[j]$$

is valid over program DINE-CONTR for each $j \in [1..M]$.

The accessibility property $A[j]$ can be decomposed into the two properties:

$$A_{2,3}[j]:\quad at\_\ell_2[j] \;\Rightarrow\; \diamondsuit\, at\_\ell_3[j]$$

$$A_{3,4}[j]:\quad at\_\ell_3[j] \;\Rightarrow\; \diamondsuit\, at\_\ell_4[j].$$

Obviously, the validity of $A_{2,3}[j]$ and $A_{3,4}[j]$ implies the validity of $A[j]$, over program DINE-CONTR. We intend to establish properties $A_{2,3}$ and $A_{3,4}$ in the following order:

- First, we will establish $A_{2,3}[j]$ for every $j \in [2..M{-}1]$.

$$
\begin{array}{l}
\textbf{in} \quad M\colon \textbf{integer where } M \geq 2 \\
\textbf{local } c \ \colon \textbf{array } [1..M] \textbf{ of integer where } c = 1 \\
\\
\displaystyle\mathop{\Big\|}_{j=1}^{M-1} P[j] :: \left[\begin{array}{l}
\ell_0\colon \textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
\ell_1\colon \textbf{noncritical} \\
\ell_2\colon \textbf{request } c[j] \\
\ell_3\colon \textbf{request } c[j+1] \\
\ell_4\colon \textbf{critical} \\
\ell_5\colon \textbf{release } c[j] \\
\ell_6\colon \textbf{release } c[j+1]
\end{array}\right]
\end{array}\right] \\
\\
\quad \| \\
\\
P[M] :: \left[\begin{array}{l}
\ell_0\colon \textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
\ell_1\colon \textbf{noncritical} \\
\ell_2\colon \textbf{request } c[1] \\
\ell_3\colon \textbf{request } c[M] \\
\ell_4\colon \textbf{critical} \\
\ell_5\colon \textbf{release } c[1] \\
\ell_6\colon \textbf{release } c[M]
\end{array}\right]
\end{array}\right]
\end{array}
$$

Fig. 3.27. Program DINE-CONTR: solution with one contrary philosopher.

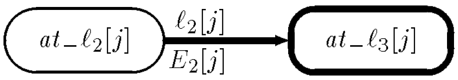
- Then, we establish by induction on decreasing j the properties $A_{3,4}[j]$ for $j = M, M{-}1, \ldots, 1$.
- Finally, we establish $A_{2,3}[1]$ and $A_{2,3}[M]$.

Proving $A_{2,3}[j]$ for $j \in [2..M{-}1]$

Let $j \in [2..M{-}1]$. The property

$$A_{2,3}[j]\colon \quad at\_\ell_2[j] \ \Rightarrow \ \Diamond\, at\_\ell_3[j]$$

is proven by the following CHAIN-F diagram



The proof relies on the compassionate transition $\ell_2[j]$. As required for valid diagrams involving compassionate transitions, we have to establish the eventual enableness of $\ell_2[j]$ stated by the formula

$$E_2[j]: \quad at\_\ell_2[j] \;\;\Rightarrow\;\; \diamondsuit\, \underbrace{at\_\ell_2[j] \;\wedge\; c[j] > 0}_{En(\ell_2[j])}\ .$$

Property $E_2[j]$ is proven by the CHAIN diagram of Fig. 3.28.

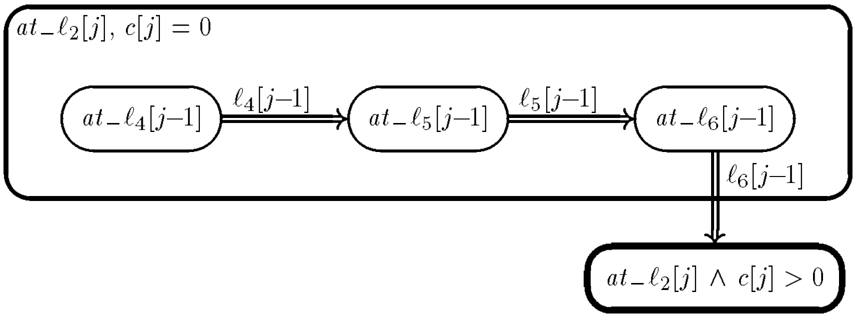


Fig. 3.28. CHAIN diagram for $E_2[j]$.

This proof relies on assertion $\varphi_0[j]$: $c[j] \geq 0$ to infer that $\neg\big(c[j] > 0\big)$ implies $c[j] = 0$. Then, we use assertion

$$\varphi_1[j-1]: \quad at\_\ell_{4..6}[j-1] \;+\; at\_\ell_{3..5}[j] \;+\; c[j] \;\;=\;\; 1$$

to infer that $at\_\ell_2[j] \wedge c[j] = 0$ implies $at\_\ell_{4..6}[j-1]$.

Proving $A_{3,4}[j]$ for $j = M, M{-}1, \ldots, 1$

Next, we establish the property

$$A_{3,4}[j]: \quad at\_\ell_3[j] \;\Rightarrow\; \diamondsuit\, at\_\ell_4[j]$$

by induction on decreasing $j = M, M{-}1, \ldots, 1$.

- *Induction Basis*

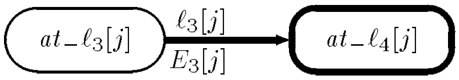
  For the induction basis, we prove

  $$A_{3,4}[j]: \quad at\_\ell_3[j] \;\Rightarrow\; \diamondsuit\, at\_\ell_4[j]$$

for $j \in \{M{-}1, M\}$. We can establish $A_{3,4}$ for both $M{-}1$ and M in one proof because both processes compete on $c[M]$ at location ℓ_3 and release $c[M]$ at location ℓ_6.

Let j denote one of the indices in $\{M{-}1, M\}$, and k denote the *other* index. Thus, if $j = M{-}1$ then $k = M$ and if $j = M$ then $k = M{-}1$.

Property $A_{3,4}[j]$ is proven by the following CHAIN-F diagram:



The diagram requires establishing the eventual enableness of $\ell_3[j]$ given by

$$E_3[j]: \quad at\_\ell_3[j] \;\Rightarrow\; \diamondsuit\big(at\_\ell_3[j] \;\wedge\; c[M] > 0\big).$$

This is established by the CHAIN diagram of Fig. 3.29.

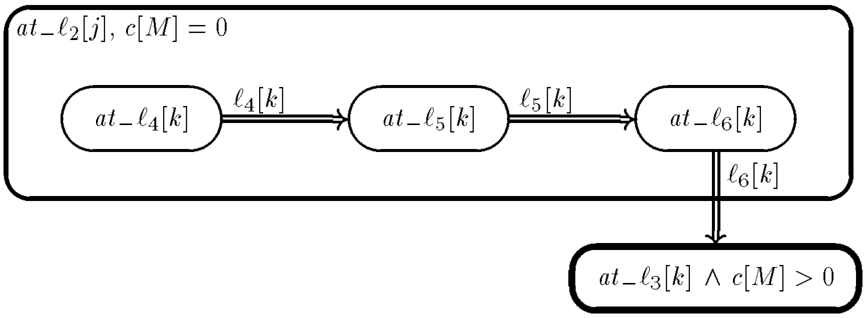


Fig. 3.29. CHAIN diagram for $E_3[j]$.

The proof uses $\varphi_0[M]$: $c[M] \geq 0$ to infer $c[M] = 0$ from $\neg\big(c[M] > 0\big)$ and the invariant

$$\varphi_2: \quad at\_\ell_{4..6}[M{-}1] \;+\; at\_\ell_{4..6}[M] \;+\; c[M] \;=\; 1$$

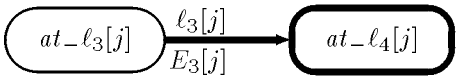
to infer $at\_\ell_{4..6}[M{-}1]$ from $at\_\ell_3[M] \wedge c[M] = 0$, and to infer $at\_\ell_{4..6}[M]$ from $at\_\ell_3[M{-}1] \wedge c[M] = 0$.

- *Induction Step*

We show that $A_{3,4}[j+1]$ implies $A_{3,4}[j]$ for each $j \in [1..M{-}2]$. Assume that $A_{3,4}[j+1]$ has been established. Property

$$A_{3,4}[j]: \quad at\_\ell_3[j] \;\Rightarrow\; \diamondsuit\, at\_\ell_4[j]$$

is proven by the following CHAIN-F diagram:



The required eventual enableness condition

$$E_3[j]: \quad at\_\ell_3[j] \;\Rightarrow\; \diamondsuit\big(at\_\ell_3[j] \;\wedge\; c[j+1] > 0\big)$$

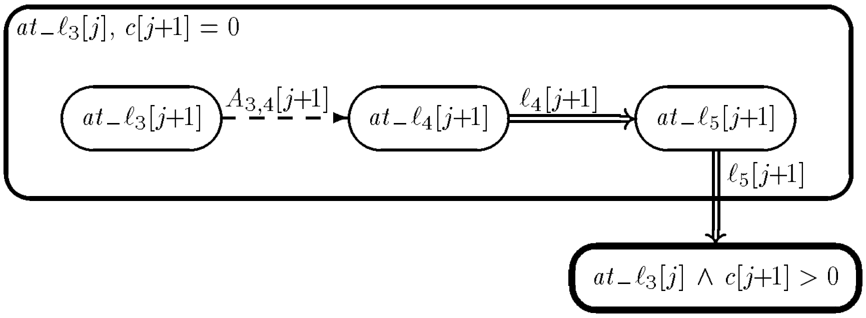


Fig. 3.30. CHAIN diagram for $E_3[j]$ (for the inductive step).

is proven by the CHAIN diagram of Fig. 3.30.

The proof uses assertion $\varphi_0[j+1]$: $c[j+1] \geq 0$ to infer $c[j+1] = 0$ from $\neg\big(c[j+1] > 0\big)$ and assertion

$$\varphi_1[j]: \quad at\_\ell_{4..6}[j] \; + \; at\_\ell_{3..5}[j+1] \; + \; c[j+1] \;\; = \;\; 1$$

to infer $at\_\ell_{3..5}[j+1]$ from $at\_\ell_3[j] \wedge c[j+1] = 0$.

It uses the assumption $A_{3,4}[j+1]$ to ensure that if $P[j+1]$ is at ℓ_3 it will eventually arrive to ℓ_4. In the diagram, this is represented by the dashed edge labeled by $A_{2,3}[j]$.

This concludes the proof of $A_{3,4}[j]$ for all $j \in [1..M]$.

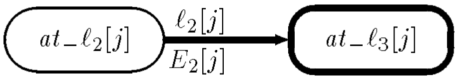
Proving $A_{2,3}[j]$ for $j \in \{1, M\}$

Finally, we prove

$$A_{2,3}[j]: \quad at\_\ell_2[j] \;\Rightarrow\; \Diamond\, at\_\ell_3[j]$$

for $j \in \{1, M\}$. Again, it is possible to present a single proof for these two indices since both $P[1]$ and $P[M]$ compete for $c[1]$ at location ℓ_2 and release $c[1]$ at location ℓ_5. As before, let j denote one of the two indices in $\{1, M\}$ and k denote the other index.

The proof of $A_{2,3}[j]$ is established by the following diagram:



The proof is completed by establishing the eventual enableness of $\ell_2[j]$, given by

$$E_2[j]: \quad at\_\ell_2[j] \quad \Rightarrow \quad \diamondsuit \underbrace{at\_\ell_2[j] \ \wedge \ c[1] > 0}_{En(\ell_2[j])} .$$

This property is proved by the CHAIN diagram of Fig. 3.31.

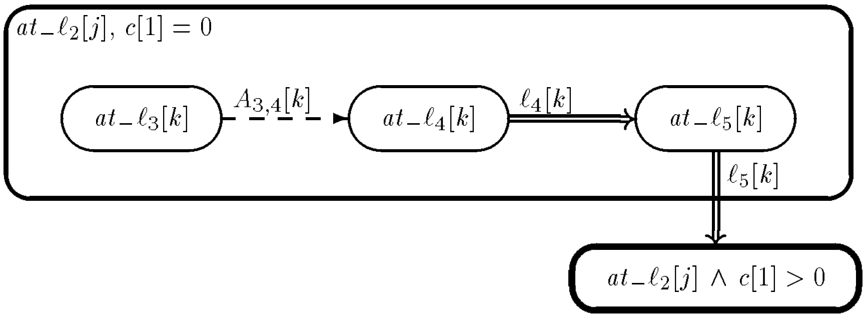


Fig. 3.31. CHAIN diagram for $E_2[j]$ for $j \in \{1, M\}$.

The proof uses assertion $\varphi_0[1]$: $c[1] \geq 0$ to infer $c[1] = 0$ from $\neg(c[1] > 0)$. It uses assertion

$$\varphi_3: \quad at\_\ell_{3..5}[M] \ + \ at\_\ell_{3..5}[1] \ + \ c[1] \ = \ 1$$

to infer $at\_\ell_{3..5}[M]$ from $at\_\ell_2[1] \wedge c[1] = 0$, and to infer $at\_\ell_{3..5}[1]$ from $at\_\ell_2[M] \wedge c[1] = 0$.

This concludes the proof of accessibility for all processes of program DINE-CONTR.

Accessibility for Program DINE-EXCL

In Fig. 3.32 we reproduce program DINE-EXCL, first presented in Fig. 2.17 of the SAFETY book. In contrast to program DINE-CONTR in which process $P[M]$ has a different program than all other processes, processes in program DINE-EXCL have identical programs. Symmetry is broken in program DINE-EXCL by the semaphore r which admits at most $M-1$ processes to the location range $\ell_{3..8}$.

In Section 2.3 of the SAFETY book, we established for program DINE-EXCL the following invariants

$$\varphi_0[j]: \quad c[j] \ \geq \ 0 \text{ for all } j \in [1..M]$$

$$\varphi_1[j]: \quad at\_\ell_{5..7}[j] \ + \ at\_\ell_{4..6}[j \oplus_M 1] \ + \ c[j \oplus_M 1] \ = \ 1 \text{ for all } j \in [1..M]$$

$$\varphi_2 \ : \ r \ \geq \ 0$$

$$\varphi_3 \ : \ N_{3..8} + r \ = \ M-1.$$

$$
\begin{array}{ll}
\textbf{in} & M\text{: } \textbf{integer where } M \geq 2 \\
\textbf{local} & c \text{ : } \textbf{array } [1..M] \textbf{ integer where } c = 1 \\
 & r \text{ : } \textbf{integer where } r = M - 1
\end{array}
$$

$$
\overset{M}{\underset{j=1}{\|}} P[j] :: \left[\begin{array}{l}
\ell_0\text{: } \textbf{loop forever do} \\
\quad \left[\begin{array}{l}
\ell_1\text{: } \textbf{noncritical} \\
\ell_2\text{: } \textbf{request } r \\
\ell_3\text{: } \textbf{request } c[j] \\
\ell_4\text{: } \textbf{request } c[j \oplus_M 1] \\
\ell_5\text{: } \textbf{critical} \\
\ell_6\text{: } \textbf{release } c[j] \\
\ell_7\text{: } \textbf{release } c[j \oplus_M 1] \\
\ell_8\text{: } \textbf{release } r
\end{array}\right]
\end{array}\right]
$$

Fig. 3.32. Program DINE-EXCL: one philosopher stays out.

Here we estabish the accessibility property

$$A[j]: \quad at\_\ell_2[j] \;\Rightarrow\; \Diamond\, at\_\ell_5[j].$$

As before, we break the accessibility property into three steps

$$
\begin{array}{ll}
A_{2,3}[j]: & at\_\ell_2[j] \;\Rightarrow\; \Diamond\, at\_\ell_3[j] \\
A_{3,4}[j]: & at\_\ell_3[j] \;\Rightarrow\; \Diamond\, at\_\ell_4[j] \\
A_{4,5}[j]: & at\_\ell_4[j] \;\Rightarrow\; \Diamond\, at\_\ell_5[j].
\end{array}
$$

We establish $A_{3,4}[j]$ first, followed by $A_{4,5}[j]$, followed by $A_{2,3}[j]$.

Proving $A_{3,4}[j]$

The response property

$$A_{3,4}[j]: \quad at\_\ell_3[j] \;\Rightarrow\; \Diamond\, at\_\ell_4[j]$$

is proven by the following CHAIN-F diagram:

$$\boxed{at\_\ell_3[j]} \xrightarrow[E_3[j]]{\ell_3[j]} \boxed{at\_\ell_4[j]}$$

Eventual enableness of compassionate transition $\ell_3[j]$ is stated by

$$E_3[j]: \quad at\_\ell_3[j] \;\Rightarrow\; \Diamond\, \underbrace{at\_\ell_3[j] \;\wedge\; c[j] > 0}_{En(\ell_3[j])}$$

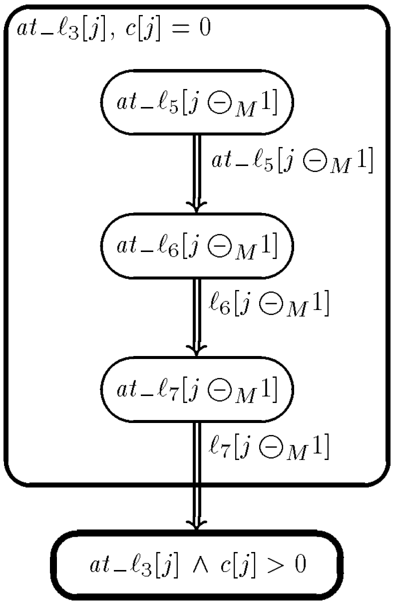


Fig. 3.33. CHAIN diagram of $E_3[j]$ (for program DINE-EXCL).

and is proven in the CHAIN diagram of Fig. 3.33.

The proof uses $\varphi_0[j]$: $c[j] \geq 0$ to infer $c[j] = 0$ from $\neg\big(c[j] > 0\big)$ and assertion

$$\varphi_1[j \ominus_M 1]: \quad at\_\ell_{5..7}[j \ominus_M 1] + at\_\ell_{4..6}[j] + c[j] = 1$$

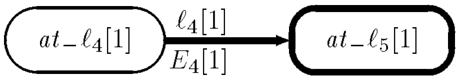
to infer $at\_\ell_{5..7}[j \ominus_M 1]$ from $at\_\ell_3[j] \wedge c[j] = 0$.

Proving $A_{4,5}[j]$

Since processes in program DINE-EXCL are fully symmetric it is sufficient to prove $A_{4,5}$ for a particular value of j, say $j = 1$

$$A_{4,5}[1]: \quad at\_\ell_4[1] \Rightarrow \Diamond\, at\_\ell_5[1].$$

The proof of $A_{4,5}[1]$ is presented in the following CHAIN-F diagram:



The required eventual enableness of $\ell_4[1]$ is stated by

$$E_4[1]: \quad at\_\ell_4[1] \Rightarrow \Diamond\, \underbrace{at\_\ell_4[1] \wedge c[2] > 0}_{En(\ell_4[1])}$$

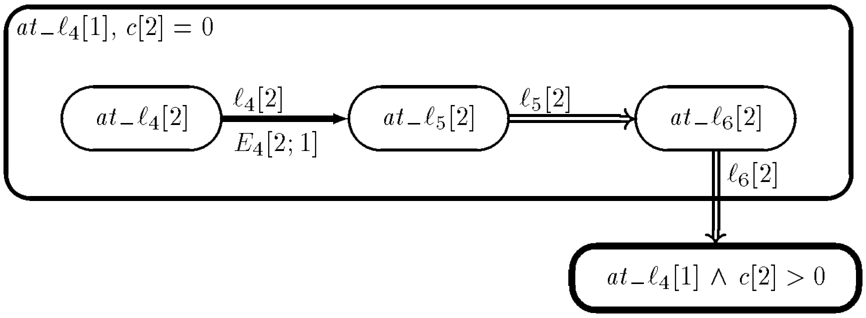


Fig. 3.34. CHAIN-F diagram of $E_4[1]$.

and can be proven by the CHAIN-F diagram of Fig. 3.34.

In this proof we use $\varphi_0[2]$ and $\varphi_1[1]$ to infer that $at\_\ell_4[1] \wedge \neg\big(c[2] > 0\big)$ implies $at\_\ell_{4..6}[2]$. The eventual enableness proof obligation for $\ell_4[2]$ is given by

$$E_4[2;1]:\quad at\_\ell_4[1] \ \wedge\ at\_\ell_4[2] \ \ \Rightarrow\ \ \diamondsuit\, \underbrace{at\_\ell_4[2] \ \wedge\ c[3]>0}_{En(\ell_4[2])}\ .$$

We denote it by $E_4[2;1]$ because it claims the eventual enableness of $\ell_4[2]$ while $P[1]$ is also at ℓ_4.

Obviously, a proof of $E_4[2;1]$ will analyze the locations of process $P[3]$ and rely on an eventual enableness requirement given by

$$E_4[3;1,2]:\quad at\_\ell_4[1] \ \wedge\ at\_\ell_4[2] \ \wedge\ at\_\ell_4[3] \ \ \Rightarrow\ \ \diamondsuit\, \underbrace{at\_\ell_4[3] \ \wedge\ c[4]>0}_{En(\ell_4[3])}\ .$$

Assume we constructed this sequence of verification diagrams reaching process $P[M{-}2]$. For this process, we need to establish

$$E_4[M{-}2;1,2,\ldots,M{-}3]:\quad at\_\ell_4[1] \ \wedge\ \cdots\ \wedge\ at\_\ell_4[M{-}2] \ \ \Rightarrow$$
$$\diamondsuit\, \underbrace{at\_\ell_4[M{-}2] \ \wedge\ c[M{-}1]>0}_{En(\ell_4[M-2])}\ .$$

This property can be established by the CHAIN-F diagram of Fig. 3.35.

The eventual enableness requirement is given by

$$E_4[M{-}1;1,\ldots,M{-}2]:\quad at\_\ell_4[1] \ \wedge\ \cdots\ \wedge\ at\_\ell_4[M{-}1] \ \ \Rightarrow$$
$$\diamondsuit\, \underbrace{at\_\ell_4[M{-}1] \ \wedge\ c[M]>0}_{En(\ell_4[M-1])}\ .$$

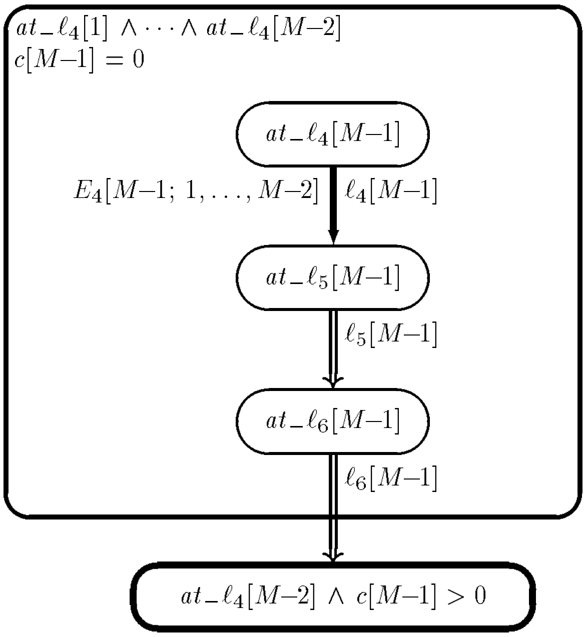


Fig. 3.35. CHAIN-F diagram of $E_4[M-2; 1, 2, \ldots; M-3]$.

However, in view of φ_2 and φ_3, $N_{3..8} \leq M-1$, which means that $at\_\ell_4[1] \wedge \cdots \wedge at\_\ell_4[M-1]$ implies $at\_\ell_{0..2}[M]$. Considering $\varphi_1[M-1]$, we see that $at\_\ell_4[M-1] \wedge at\_\ell_{0..2}[M]$ implies $c[M] = 1$. We conclude that the implication

$$at\_\ell_4[1] \;\wedge\; \cdots \;\wedge\; at\_\ell_4[M-1] \;\;\rightarrow\;\; at\_\ell_4[M-1] \;\wedge\; c[M] > 0$$

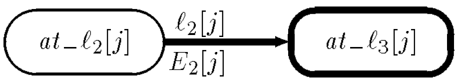
is P-state valid, implying the validity of $E_4[M-1;\, 1, \ldots, M-2]$.

Proving $A_{2,3}[j]$

Finally, we prove the response property

$$A_{2,3}[j]: \quad at\_\ell_2[j] \;\Rightarrow\; \diamondsuit\, at\_\ell_3[j].$$

This property can be proven by the CHAIN-F diagram:



It remains to discharge the proof obligation

$$E_2[j]: \quad at\_\ell_2[j] \;\Rightarrow\; \diamondsuit\big(at\_\ell_2[j] \;\wedge\; r > 0\big).$$

Obviously, it suffices to prove the entailment starting from a state which satisfies $\neg(r > 0)$

$$at\_\ell_2[j] \ \wedge \ r = 0 \ \Rightarrow \ \diamondsuit\big(at\_\ell_2[j] \ \wedge \ r > 0\big).$$

By φ_2 and φ_3, it follows that

$$at\_\ell_2[j] \ \wedge \ r = 0 \ \rightarrow \ \exists k \in [1..M]\colon \ at\_\ell_2[j] \ \wedge \ at\_\ell_{3..8}[k],$$

stating that if r is not positive, some process $P[k]$ must be executing at some location in the range $[\ell_3..\ell_8]$.

Therefore, it is sufficient to prove the property

$$at\_\ell_2[j] \ \wedge \ at\_\ell_{3..8}[k] \ \Rightarrow \ \diamondsuit\big(at\_\ell_2[j] \ \wedge \ r > 0\big).$$

This proof is presented in the CHAIN diagram of Fig. 3.36.

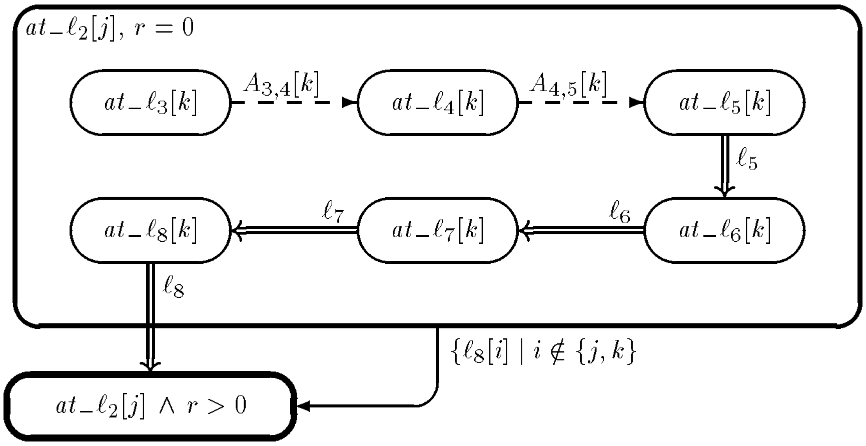


Fig. 3.36. CHAIN diagram for $E_2[j]$.

This concludes the proof of accessibility for program DINE-EXCL.

3.7 Allocation of Quantifiable Resources

One of the problems that requires coordination between concurrent processes is the management of limited resources. There are different types of resources in a system. Some resources, such as the ownership of a particular printer or a shared variable, form one indivisible unit that must be granted in its entirety to a single process at a time. Others, such as disk or memory space, consist of a large number of identical units, which may be granted to several processes at the same time, provided the sum of granted quantities never exceeds a given available total.

Typically, before entering a particular activity, to which we refer as "a critical section," a process requests a certain quantity of units, and must wait for this quantity to be granted before entering the critical section.

Most of our previous examples considered resources of the atomic type. We now focus our attention on the allocation of resources of the second, *quantifiable*, type. To set a general framework, we consider $M > 0$ processes $P_1, \ldots, P_M$, and a single resource having $N > 0$ units. Assume that an array $X[1..M]$ represents the needs the processes in units of the resource. That is, process P_i needs $X[i]$ units of the resource in order to perform correctly its critical activity. Naturally, we have to assume that

$$X[i] \leq N \quad \text{for every} \quad i = 1, \ldots, M,$$

but this is the only assumption we make. This assumption states that there are sufficiently many units of the resource to satisfy the needs of each process separately.

In the following, we use our notation for parameterized programs, by which L_a denotes the set of indices of processes that currently execute at ℓ_a and $L_{a..b}$ denotes the set of indices of processes currently executing at any of $\ell_a, \ell_{a+1}, \ldots, \ell_b$. Similarly, $N_a = |L_a|$ and $N_{a..b} = |L_{a..b}|$.

A First Approximation

A first approximation to an algorithm that manages the allocation of a quantifiable resource is presented in parameterized program INADEQUATE of Fig. 3.37. In the critical section of $P[i]$, $X[i]$ units of the resource are used. The program uses the generalized semaphore statements

$$\textbf{request}\ (r,\ C),$$

whose effect is equivalent to

$$\langle \textbf{await}\ r \geq C;\ r := r - C \rangle,$$

and

$$\textbf{release}\ (r,\ C),$$

whose effect is equivalent to

$$r\ :=\ r + C.$$

The **request** (r, C) statement is associated with a compassion requirement, prescribing that it is impossible for the statement to be enabled infinitely many times, but executed only finitely many times.

- *Safety*

$$
\begin{array}{lll}
\textbf{in} & M,\ N & : \textbf{integer where } M > 0,\ N > 0 \\
 & X & : \textbf{array } [1..M] \textbf{ of integer where } \forall i \in [1..M]\colon X[i] \le N \\
\textbf{local} & r & : \textbf{integer where } r = N
\end{array}
$$

$$
\overset{M}{\underset{i=1}{\Big\|}}\ P[i] :: \left[\begin{array}{l}
\ell_0\colon \textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
\ell_1\colon \textbf{noncritical} \\
\ell_2\colon \textbf{request } (r,\ X[i]) \\
\ell_3\colon \textbf{critical} \\
\ell_4\colon \textbf{release } (r,\ X[i])
\end{array}\right]
\end{array}\right]
$$

Fig. 3.37. Program INADEQUATE (inadequate allocation of quantifiable resource).

It is easy to see that the program of Fig. 3.37 has the right safety property of exclusion. Exclusion in this case means that the sum of allocated units of the resource never exceeds N. This can be expressed by

$$\left(\sum_{i \in L_3} X[i]\right) \le N.$$

This formula takes the sum of all $X[i]$ for processes $P[i]$ that are currently at ℓ_3, and requires that the sum never exceeds N.

This safety property can be deduced from the two invariants

$$\chi_0\colon \quad r \ge 0$$

$$\chi_1\colon \quad \left(\sum_{i \in L_{3,4}} X[i]\right) + r = N.$$

It is clear that execution of $\ell_2[i]$ increments the sum by $X[i]$ and decrements r by the same amount. Similarly, transition $\ell_4[i]$ leaving the range $\ell_{3,4}$, decrements the sum and increments r by $X[i]$.

- *Individual accessibility not guaranteed*

The program of Fig. 3.37 does not ensure accessibility for each process wishing to enter ℓ_3, that is, it does not satisfy

$$at\_\ell_2[i] \;\Rightarrow\; \Diamond\, at\_\ell_3[i] \qquad \text{for every } i,\ 1 \le i \le M.$$

To show this, consider the particular case of three processes ($M = 3$), with $N = 2$ and the array of resource requirements given by $X[1] = X[2] = 1$, $X[3] = 2$.

Consider the computation

$$\Big\langle \pi\colon \{\ell_0[1],\ \ell_0[2],\ \ell_0[3]\},\ r\colon 2\Big\rangle \ \cdots \xrightarrow{P[1],P[2],P[3]}$$
$$\cdots \Big\langle \pi\colon \{\ell_2[1],\ \ell_2[2],\ \ell_2[3]\},\ r\colon 2\Big\rangle \ \cdots \xrightarrow{P[1],P[2]} \cdots$$

$$\Big\langle \pi\colon \{\ell_3[1],\ \ell_3[2],\ \ell_2[3]\},\ r\colon 0\Big\rangle \ \cdots \xrightarrow{P[1]}$$
$$\cdots \Big\langle \pi\colon \{\ell_2[1],\ \ell_3[2],\ \ell_2[3]\},\ r\colon 1\Big\rangle \ \cdots \xrightarrow{P[1]} \cdots$$

$$\Big\langle \pi\colon \{\ell_3[1],\ \ell_3[2],\ \ell_2[3]\},\ r\colon 0\Big\rangle \ \cdots \xrightarrow{P[2]} \cdots \Big\langle \pi\colon \{\ell_3[1],\ \ell_2[2],\ \ell_2[3]\},\ r\colon 1\Big\rangle \ \cdots \xrightarrow{P[2]}$$
$$\cdots \Big\langle \pi\colon \{\ell_3[1],\ \ell_3[2],\ \ell_2[3]\},\ r\colon 0\Big\rangle \ \cdots .$$

This computation consists of a finite prefix reaching the state

$$\Big\langle \pi\colon \{\ell_3[1],\ \ell_3[2],\ \ell_2[3]\},\ r\colon 0\Big\rangle,$$

followed by an infinite repetition of a segment. The segment starts with both $P[1]$ and $P[2]$ at ℓ_3. Then $P[1]$ performs a complete cycle and ends back at ℓ_3, followed by a similar cycle performed by $P[2]$. Throughout the segment, $P[3]$ waits at ℓ_2. The computation is compassionate with respect to $P[3]$, even though $P[3]$ continuously waits at ℓ_2. This is because transition $\ell_2[3]$ is never enabled in the infinitely repeating segment, having $r \leq 1$ at all states of this segment.

This computation illustrates the possibility of $P[1]$ and $P[2]$ conspiring against $P[3]$, leaving it stranded at ℓ_2.

In **Problem 3.3** the reader is requested to prove the weaker property of communal accessibility $N_2 > 0 \Rightarrow \diamondsuit(N_3 > 0)$ for program INADEQUATE.

An Improved Solution

To remedy the fault of the previous solution, we must be able to assign priorities to the different processes. If the assignment of priorities is fair and the process with highest priority is guaranteed to enter its critical section, then individual accessibility will be ensured. We can look at the possession of the highest priority as another resource, atomic this time, and implement its allocation by another semaphore.

This leads to program Q-RES of Fig. 3.38. This program uses semaphore variable p to grant high priority to precisely one process. The process that gains this priority is allowed to make a request of the quantifiable resource r, and waits until this request is granted. While this process is waiting, no other process can make a request to r. Consequently, the only possible changes to r while this process is waiting at ℓ_3 are caused by processes that exit their critical section and increment r by $X[i]$. Eventually r must become greater or equal to the quantity

$$
\begin{array}{ll}
\textbf{in} & M,\ N: \textbf{integer where } M > 0,\ N > 0 \\
 & X \quad : \textbf{array } [1..M] \textbf{ of integer where } \forall i \in [1..M]\colon X[i] \le N \\
\textbf{local} & r,\ p \quad : \textbf{integer where } r = N,\ p = 1
\end{array}
$$

$$
\overset{M}{\underset{i=1}{\Big\|}} P[i] :: \left[\begin{array}{l}
\ell_0\colon \textbf{loop forever do} \\
\qquad \left[\begin{array}{l}
\ell_1\colon \textbf{noncritical} \\
\ell_2\colon \textbf{request } p \\
\ell_3\colon \textbf{request } \big(r,\ X[i]\big) \\
\ell_4\colon \textbf{release } p \\
\ell_5\colon \textbf{critical} \\
\ell_6\colon \textbf{release } \big(r,\ X[i]\big)
\end{array}\right]
\end{array}\right]
$$

Fig. 3.38. Program Q-RES (allocation of quantifiable resource).

requested by the waiting process. At that stage, the waiting process is granted its request and relinquishes its high priority at ℓ_4 on the way to the critical section.

Let us verify the correctness of program Q-RES.

- *Safety*

  Exclusion requires the invariant

$$\left(\sum_{i \in L_5} X[i]\right) \le N.$$

This can be established by the invariants

$$
\begin{aligned}
\chi_0&\colon \quad r \ge 0 \ \wedge\ p \ge 0 \\
\chi_1&\colon \quad N_{3,4} + p \ = \ 1 \\
\chi_2&\colon \quad \left(\sum_{i \in L_{4..6}} X[i]\right) + r \ = \ N.
\end{aligned}
$$

In particular, exclusion is deducible from χ_2 and $r \ge 0$.

- *Individual accessibility*

  To show accessibility, we prove

$$at\_\ell_2[i] \ \Rightarrow\ \Diamond\, at\_\ell_5[i].$$

The CHAIN-F diagram of Fig. 3.39 presents the proof of this property by rule CHAIN-F.

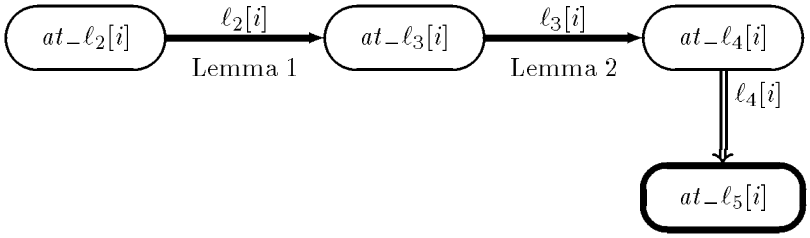


Fig. 3.39. CHAIN-F diagram for accessibility.

The more interesting part of this diagram is the proof of eventual enableness of transitions $\ell_2[i]$ and $\ell_3[i]$. This is proved by Lemmas 1 and 2.

Lemma 1 $\quad at\_\ell_2[i] \;\Rightarrow\; \diamondsuit\big(at\_\ell_2[i] \;\wedge\; p > 0\big)$

The nontrivial case is when $p = 0$ while $P[i]$ is at $\ell_2[i]$. Due to χ_1, this is possible only if $at\_\ell_{3,4}[j]$ for some $j \in [1..M]$.

Therefore, we prove

$$at\_\ell_2[i] \;\wedge\; at\_\ell_{3,4}[j] \;\;\Rightarrow\;\; \diamondsuit\big(at\_\ell_2[i] \;\wedge\; p > 0\big).$$

This is proven by the CHAIN-F diagram of Fig. 3.40.

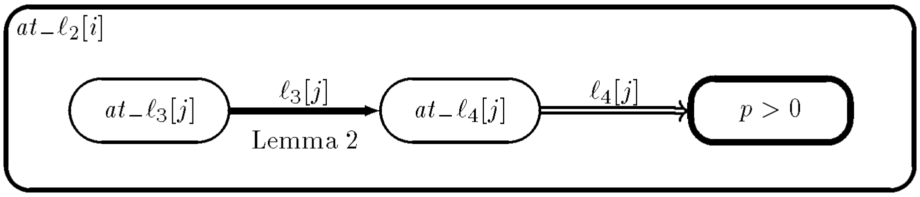


Fig. 3.40. CHAIN-F for Lemma 1.

Note that this proof also relies on Lemma 2 to provide the eventual enableness of $\ell_3[j]$.

Lemma 2 $\quad at\_\ell_3[i] \;\Rightarrow\; \diamondsuit\big(at\_\ell_3[i] \;\wedge\; r \geq X[i]\big)$

The proof of this lemma requires the use of the well-founded rule. The case we have to deal with is $at\_\ell_3[i] \wedge r < X[i]$. Since $X[i] \leq N$, it follows that $r < N$. According to χ_2, this can hold only if $N_{4..6} > 0$. Since $at\_\ell_3[i]$ and χ_1 imply $N_4 = 0$, it follows that $N_{5,6} > 0$.

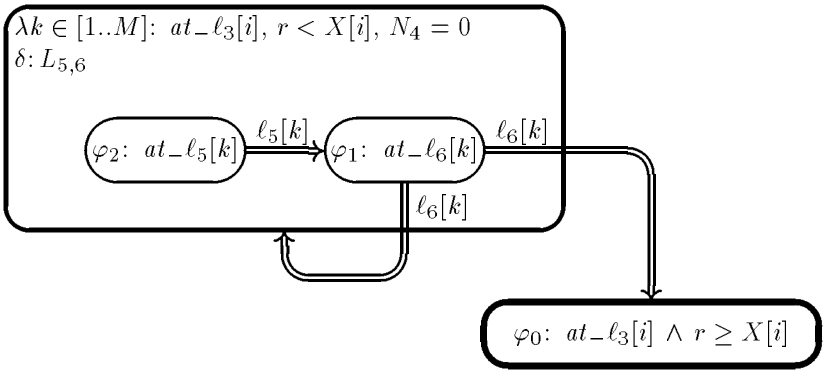


Fig. 3.41. CHAIN diagram for Lemma 2.

The lemma is proved by rule WELL-JP (Fig. 2.2), as shown in the CHAIN diagram of Fig. 3.41.

The double edge connecting φ_2: $at\_\ell_5[k]$ to φ_1: $at\_\ell_6[k]$ represents a transition of process $P[k]$ from ℓ_5 to ℓ_6. The two edges departing from φ_1: $at\_\ell_6[k]$ represent a movement of process $P[k]$ from ℓ_6 back to ℓ_0. Such movement decreases $L_{5,6}$ and increases r. If, as a result of the increase, r becomes greater or equal to $X[i]$, we have reached the goal φ_0. This possibility is represented by the edge connecting φ_1 to φ_0. In the other case, r is smaller than $X[i]$ even after the increase, and by the previous argument $r < X[i]$ and $at\_\ell_3[i]$ imply $L_{5,6} \neq \emptyset$, which means that, for some other $k \in [1..M]$, either $at\_\ell_5[k]$ or $at\_\ell_6[k]$ holds. This is represented by the edge connecting φ_1 to the compound node containing both φ_2 and φ_1.

Note that as long as $r < X[i]$, $P[i]$ cannot move from ℓ_3 to ℓ_4, and therefore N_4 remains zero and $N_{5,6}$ cannot increase.

This concludes the proof of accessibility. We remind the reader that most of the solutions to the *readers-writers* problem (Fig. 2.26) can be viewed as a special case of the quantifiable resource allocation. In these solutions, $M = N$ and $X[i] = 1$ for a reader, and $X[i] = M$ for a writer.

Consequently, the program in Fig. 3.38 also provides a solution to the *readers-writers* problem that ensures individual accessibility.

Response with Past Subformulas

Up to this point, we have only considered response formulas $p \Rightarrow \Diamond q$, in which p and q are assertions. The generalization to the case that p or q are past formulas is straightforward. As indicated in Section 1.6, this generalization replaces intermediate assertions by intermediate past-formulas, and implications by entailments.

In Fig. 3.42, we present the general version of rule WELL-F. The other F-rules can be generalized in a similar way.

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

FW1. $p \;\Rightarrow\; \bigvee_{j=0}^{m} \varphi_j$

For $i = 1, \ldots, m$:

FW2. $\rho_\tau \;\wedge\; \varphi_i \;\Rightarrow\; \left[\begin{array}{l} \bigvee_{j=1}^{m} (\varphi'_j \;\wedge\; \delta_i \succ \delta'_j) \\ \vee\;\; (\varphi'_i \;\wedge\; \delta_i = \delta'_i) \end{array}\right]$ for every $\tau \in \mathcal{T}$

FW3. $\rho_{\tau_i} \;\wedge\; \varphi_i \;\rightarrow\; \bigvee_{j=0}^{m} (\varphi'_j \;\wedge\; \delta_i \succ \delta'_j)$

FW4. If $\tau_i \in \mathcal{C}$, then

CW4. $\varphi_i \;\Rightarrow\; \diamondsuit\big(\neg\varphi_i \;\vee\; En(\tau_i)\big)$

Otherwise

JW4. $\varphi_i \;\Rightarrow\; En(\tau_i)$

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

Fig. 3.42. General version of rule WELL-F.

** 3.8 Completeness: State Response

In this and the following section we will show that rule WELL-F (Fig. 3.42) is complete for proving P-validity of a given response formula

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

We follow a strategy similar to the one taken in Chapters 1 and 4 of the SAFETY book and divide the proof into two steps. In the first step, covered in this section, we show WELL-F to be complete for state response formulas, i.e., the case that p and q are assertions. In a second step, covered in the next section, we use the statification techniques introduced in Chapter 4 of the SAFETY book to show that completeness for the assertional case implies completeness for the general case in which p and q are past formulas.

The precise statement of completeness for state response formulas is given by the following theorem.

Theorem 3.2 (completeness of rule WELL-F, state response)

For every two assertions p and q such that

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

is P-valid, there exist
assertions $q = \varphi_0, \varphi_1, \ldots, \varphi_m$,
transitions $\tau_1, \ldots, \tau_m \in \mathcal{J} \cup \mathcal{C}$,
a well-founded domain $\{\mathcal{A}, \succ\}$, and
ranking functions $\delta_0, \ldots, \delta_m\colon\ S \mapsto \mathcal{A}$,
such that the premises of rule WELL-F (Fig. 3.24) are provable from state validities.

We prove the theorem in this section.

Preliminaries

Let P be a fair transition system with justice set $\mathcal{J}$ and compassion set $\mathcal{C}$. We refer to the union of $\mathcal{J}$ and $\mathcal{C}$ as the *fairness set* of P and denote it by $\mathcal{F} = \mathcal{J} \cup \mathcal{C}$. Assume that $\mathcal{F} = \{\tau_1, \ldots, \tau_m\}$.

For the case that p and q are assertions, we will show that for every p and q there exist intermediate assertions φ_i, one for each fair transition $\tau_i \in \mathcal{F}$, $i = 1, \ldots, m$, a well-founded domain $(\mathcal{A}, \succ)$, and ranking functions $\delta_1, \ldots, \delta_m$ such that premises FW1–FW3 and subpremise JW4 are satisfied as P-state valid implications, and subpremise CW4 is P-valid. From the corollary to the completeness theorem of state invariances (Section 2.5 of the SAFETY book) it follows that premises FW1–FW3 and subpremise JW4 are provable from state validities. Subpremise CW4 will be shown to be provable from state validities in the course of the completeness proof.

An interesting conclusion of the completeness proof is that, while a first-order language over the integers and additional data domains is adequate for expressing all the constructs required by the safety rules, it is no longer adequate for expressing the necessary response constructs. Several extensions to the assertion language have been suggested. We will use the extension that allows notions such as ordinals and fixpoints to write formulas that quantify over predicates.

In the construction of ranking functions, we intend to construct a single ranking function δ: $\mathcal{S} \mapsto \mathcal{A}$, and take $\delta_1 = \cdots = \delta_m = \delta$.

The Kernel of the Intermediate Assertions

The intermediate assertions $\varphi_1, \ldots, \varphi_m$ we intend to construct have the common form

$$\varphi_i: \quad \varphi \ \land \ \psi_i, \quad \text{for } i = 1, \ldots, m.$$

Assertion φ is a common conjunct appearing in all φ_i's, to which we refer as the *kernel* of $\varphi_1, \ldots, \varphi_m$.

Here, we define the common assertion φ and argue that it can be expressed in a first-order assertion language.

We define a $(P\text{-})$*segment* to be a finite sequence of states $[s_a, \ldots, s_b]$ such that $a \leq b$ and for every i, $a \leq i < b$, s_{i+1} is a τ-successor of s_i for some transition τ of P. A segment $[s_a, \ldots, s_b]$ is called $(P\text{-})$*accessible* if state s_a is P-accessible. Obviously, if $[s_a, \ldots, s_b]$ is accessible then there exists a P-computation containing $[s_a, \ldots, s_b]$ as a segment. A segment $[s_a, \ldots, s_b]$ is called q-*free* if no s_i, $a \leq i \leq b$, satisfies q.

We say that $[s_a, \ldots, s_b]$ is a (p, q)-*segment* if it is an accessible q-free segment such that $s_a \Vdash p$. Based on this notion, we define an assertion Q such that

$$s \Vdash Q \quad \leftrightarrow \quad \text{there exists a } (p,q)\text{-segment } [s_a, \ldots, s_b] \text{ where } s_b = s.$$

Thus, Q characterizes all states that can appear as last states in a (p, q)-segment. The definition implies that s satisfies Q if and only if there exists a prefix $[s_0, \ldots, s_b]$ of a P-computation, such that $s_b = s$ and there exists a position a, $0 \leq a \leq b$, such that p holds at a and no position i, $a \leq i \leq b$, satisfies q.

Our interest in Q is motivated by the following claim.

Claim 3.3 (characterizes q-pending states)

Assume that $p \Rightarrow \diamondsuit\, q$ is P-valid and that σ is a P-computation such that Q holds at position $i \geq 0$. Then q holds at some position $k \geq i$ (in fact $k > i$).
Or equivalently:
if $p \Rightarrow \diamondsuit\, q$ is P-valid then so is the response formula $Q \Rightarrow \diamondsuit\, q$.

Thus, Q characterizes all states that must have a following q in all computations.

Justification To justify the claim, let $s = s_i$ be the state appearing at position i of σ. Consider the infinite sequence $s_{i+1}, s_{i+2}, \ldots$ obtained by removing the prefix $s_0, \ldots, s_i$ from σ. By definition of Q, there exists a prefix $\widehat{s}_0, \ldots, \widehat{s}_b$ such that $\widehat{s}_b = s$, $\widehat{s}_a \Vdash p$ for some a, $0 \le a \le b$, and $\widehat{s}_j \not\models q$ for all j, $a \le j \le b$. Consider the spliced sequence

$$\widetilde{\sigma}: \quad \widehat{s}_0,\ \ldots,\ \widehat{s}_b,\ s_{i+1},\ s_{i+2},\ \ldots.$$

Since $\widehat{s}_b = s = s_i$, this sequence is obviously a computation and p holds at position a of $\widetilde{\sigma}$. Since $p \Rightarrow \Diamond q$ is P-valid there exists some position $r \ge a$ such that q holds at position r of $\widetilde{\sigma}$. Since q cannot hold at any j, $a \le j \le b$, it follows that $r > b$. It follows that q holds at position k of the original computation σ, where $k = r - b + i > i$. ∎

Using the techniques of Section 2.5 of the SAFETY book, we can write a first-order formula $\varphi(V)$ using dynamic arrays which expresses Q. We take $\varphi = Q$ to be the common part of the intermediate assertions.

Example Consider program UP-DOWN of Fig. 3.43 (see also Fig. 1.27). The response property

$$\underbrace{y = 10}_{p} \quad \Rightarrow \quad \Diamond\, \underbrace{y = 5}_{q}$$

is valid over this program.

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

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

Fig. 3.43. Program UP-DOWN.

The kernel assertion Q, relative to p: $y = 10$ and q: $y = 5$, can be given by

$$Q: \quad acc_P \ \wedge\ y > 5,$$

where

$$acc_P\text{:} \quad (at\_m_0 \wedge at\_\ell_{0,1} \wedge x = 0 \ \ \vee\ \ at\_m_1 \wedge at\_\ell_{0..4} \wedge\ x = 1)\ \wedge$$
$$y \ge at\_\ell_3 \ \wedge\ (at\_\ell_4 \ \rightarrow\ y = 0)$$

characterizes all the accessible states for this program.

Note that if we observe a state s in which $y = 6$ in a computation σ, it does not necessarily imply that a prefix containing a $(y = 10)$-position is part of that computation. The value 6 may well be the highest value attained by y in σ. We are only guaranteed that there exists some prefix (possibly not present in σ) containing a $(y = 10)$-position and ending in our $(y = 6)$-state without encountering a $(y = 5)$-position in between. However, the preceding claim guarantees that every $(y = 6)$-position is followed by a $(y = 5)$-position. ▪

We denote by $\|Q\|$ the set of all states satisfying Q. A segment σ: $[s_a, \ldots, s_b]$ is called a *Q-segment* if $s_i \vDash Q$ for every $i \in [a..b]$.

A Primary Order

The identification of the ranking functions, mapping states into a well-founded domain, is based on the identification of a well-founded ordering over the states of the program.

For a segment σ: $[s_a, \ldots, s_b]$, we say that σ *leads* from s_a to s_b. Transition $\tau_i \in \mathcal{F}$ is said to be *gratified* in a segment σ: $[s_a, \ldots, s_b]$ if

- τ_i is taken in σ, or
- $\tau_i \in \mathcal{J} - \mathcal{C}$ and τ_i is disabled on some s_j, $j \in [a..b]$.

We denote by $grat(\sigma)$ the index set of the transitions that are gratified in σ

$$grat(\sigma): \quad \{i \mid \tau_i \in \mathcal{F} \text{ is gratified in } \sigma\}.$$

A segment σ is called *fair* if it gratifies all fair transitions, i.e., $grat(\sigma) = [1..m]$. Note that a compassionate transition $\tau \in \mathcal{C}$ is considered to be gratified in σ if it is taken in σ. In this, the compassionate transitions differ from just transitions which can be gratified also by being disabled somewhere in σ.

Example Consider program UP-DOWN of Fig. 3.43. The set of fair transitions for this program consists of the just transitions ℓ_0, ℓ_1, ℓ_2, ℓ_3, and m_0. The segment

$$\sigma_1: \quad \langle \pi{:}\, \{\ell_0, m_0\},\ x{:}\, 0,\ y{:}\, 5\rangle,\ \langle \pi{:}\, \{\ell_1, m_0\},\ x{:}\, 0,\ y{:}\, 5\rangle$$

is not fair. Segment σ_1 gratifies transitions ℓ_0, ℓ_1, ℓ_2, and ℓ_3 but does not gratify m_0. Note that ℓ_0–ℓ_3 are gratified since each has a state within σ_1 on which the transition is disabled.

On the other hand, the following extension of σ_1,

$$\sigma_2: \quad \langle \pi{:}\, \{\ell_0, m_0\},\ x{:}\, 0,\ y{:}\, 5\rangle,\ \langle \pi{:}\, \{\ell_1, m_0\},\ x{:}\, 0,\ y{:}\, 5\rangle,\ \langle \pi{:}\, \{\ell_1, m_1\},\ x{:}\, 1,\ y{:}\, 5\rangle$$

is fair since it gratifies all fair transitions. ▪

We define a binary relation $\sqsupset$ between states in $\|Q\|$ as follows:

$$s_1 \sqsupset s_2 \quad \text{iff} \quad \text{there exists a fair } Q\text{-segment leading from } s_1 \text{ to } s_2.$$

Claim 3.4 (well-foundedness of $\sqsupset$)

The relation $\sqsupset$ is a well-founded order over $\|Q\|$.

Justification We start by showing that $\sqsupset$ is a well-founded relation over $\|Q\|$.

Assume to the contrary, that there exists an infinite sequence of states $s^1, s^2, \ldots \in \|Q\|$, such that

$$s^1 \ \sqsupset \ s^2 \ \sqsupset \ \cdots.$$

By the definition of Q and the meaning of $\sqsupset$, there exists an infinite sequence of states as follows

$$\sigma: \quad s_0, \ \ldots, \ \underbrace{s_a, \ \ldots, \ \overbrace{s^1}^{}}_{q\text{-free}}\!\!\!\!\overbrace{, \ \ldots, \ s^2}^{q\text{-free and fair}}, \ \underbrace{\ldots, \ s^3}_{q\text{-free and fair}}, \ \ldots$$

where $s_a \vDash p$. This sequence satisfies the fairness requirement since in each segment $[s^i, \ldots, s^{i+1}]$ each transition $\tau_i \in \mathcal{C}$ is either taken at least once or is disabled on all states in the segment, and each transition $\tau_i \in \mathcal{J} - \mathcal{C}$ is either taken or disabled on some state in the segment. On the other hand, p holds at position a of σ and q does not hold at any position $j \geq a$, in violation of our assumption that $p \Rightarrow \diamondsuit\, q$ is P-valid.

Next, we show that $\sqsupset$ is an ordering relation over $\|Q\|$. Relation $\sqsupset$ is anti-symmetric. This is because the existence of s_1, s_2 such that $s_1 \sqsupset s_2$ and $s_2 \sqsupset s_1$ gives rise to the infinitely descending sequence

$$s_1 \ \sqsupset \ s_2 \ \sqsupset \ s_1 \ \sqsupset \ s_1 \ \sqsupset \ \cdots,$$

which violates the just-proven well-foundedness of $\sqsupset$. Irreflexivity is proven in the same way by taking $s_1 = s_2$. Transitivity follows from the definition of $\sqsupset$. Namely, if there exists a fair Q-segment $s_a, \ldots, s_b$ and a fair Q-segment $s_b, \ldots, s_c$, then the Q-segment $s_a, \ldots, s_b, \ldots, s_c$ obtained by dove-tailing these two segments is fair and establishes $s_a \sqsupset s_c$. ∎

Example Consider program SKIP2 presented in Fig. 3.44.

The set of fair (just) transitions for this program is:

$$\mathcal{F}: \quad \{\tau_1: \ \ell_1, \qquad \tau_2: \ \ell_2^{\mathrm{E}}, \qquad \tau_3: \ \ell_3, \qquad \tau_4: \ \ell_4, \qquad \tau_5: \ \ell_2^{\mathrm{X}}\},$$

where ℓ_2^{E} and ℓ_2^{X} are the entry and exit transitions from the cooperation statement ℓ_2.

The response property we investigate for program SKIP2 is

$$\begin{bmatrix} \ell_1\colon \ \textbf{skip} \\ \ell_2\colon \ \Big[[\ell_3\colon \ \textbf{skip};\ \ell_5\colon\ ] \quad \| \quad [\ell_4\colon \ \textbf{skip};\ \ell_6\colon\ ] \Big] \\ \ell_7\colon \end{bmatrix}$$

Fig. 3.44. Program SKIP2.

$$\underbrace{at\_\ell_3 \ \wedge \ at\_\ell_4}_{p} \quad \Rightarrow \quad \diamondsuit \underbrace{at\_\ell_7}_{q} \ .$$

The corresponding assertion Q should characterize all the states which can be reached from accessible states satisfying p: $at\_\ell_3 \wedge at\_\ell_4$ without passing through a state satisfying q: at_ℓ_7. Therefore, Q is given by

$$Q\colon \quad at\_\ell_{3,5} \ \wedge \ at\_\ell_{4,6}.$$

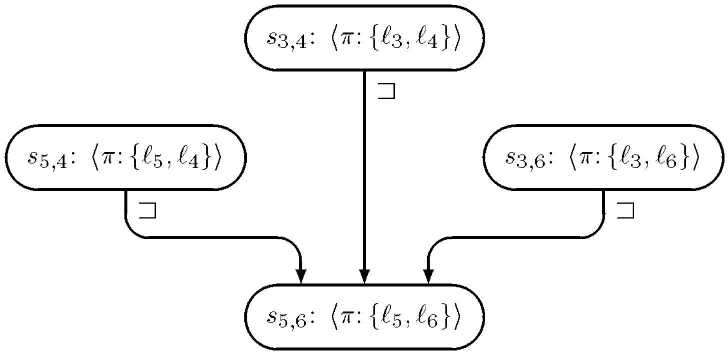


Fig. 3.45. Q-states of program SKIP2 related by $\sqsupset$.

In Fig. 3.45, we present the four Q-states of this program and their ordering by relation $\sqsupset$. For example, segment $[s_{5,4}, s_{5,6}]$ is fair, i.e., gratifies all fair transitions τ_1–τ_5: it gratifies τ_1, τ_2, τ_3, and τ_5 (i.e., ℓ_1, ℓ_2^E, ℓ_3, and ℓ_2^X) because they are all disabled on $s_{5,4}$; it gratifies $\tau_4\colon \ell_4$ by taking this transition in the step from $s_{5,4}$ to $s_{5,6}$.

On the other hand, the segment $[s_{3,4}, s_{5,4}]$ is not fair since it does not gratify ℓ_4, which is enabled on all states in the segment but not taken there. This is why there is no $\sqsupset$-labeled edge connecting $s_{3,4}$ to $s_{5,4}$ in Fig. 3.45, nor is there a $\sqsupset$-labeled edge from $s_{3,4}$ to $s_{3,6}$. ▟

The Primary Order is Too Coarse

Order $\sqsupset$ is a good first approximation to the order we need. It is well-founded and measures progress in the sense that when the system moves from state s_1 to state s_2 such that $s_1 \sqsupset s_2$, a progress step has been taken. This is because the system cannot take infinitely many such steps.

Unfortunately, the order $\sqsupset$ only provides a coarse measure of progress. Reconsider program SKIP2 presented in Fig. 3.44 and the $\sqsupset$-ordering of its Q-states in Fig. 3.45.

Clearly, every step leading from one Q-state to another represents progress towards the goal q: at_ℓ_7. However, the order $\sqsupset$ identifies only segments leading to $s_{5,6}$ as observable progress. In particular, it does not identify the moves from $s_{3,4}$ to either $s_{5,4}$ or $s_{3,6}$ as observable progress. It is possible to conclude that $\sqsupset$ successfully measures coarse progress but is insensitive to finer progress.

As a first step in the refinement of $\sqsupset$, we find a ranking function into a well-founded domain, whose ordering is compatible with $\sqsupset$.

Mapping into the Ordinals

As a first approximation of the well-founded domain $(\mathcal{A}, \succ)$, we take $(\mathcal{O}, >)$, where $\mathcal{O}$ is the domain of the *ordinals* and $>$ is the natural order over the ordinals.

We remind the reader that the ordinals are obtained by extending the sequence of natural numbers beyond the finite naturals. There are very few facts about the ordinals that are required in our proof here. Following is a list of some of these properties:

- Every natural number is an ordinal. We refer to the naturals as the *finite ordinals*.
- There is a total order $<$ on the ordinals which generalizes the "smaller than" ordering over the naturals.
- Some ordinals β are the immediate successors of another ordinal α, and we write $\beta = \alpha + 1$. Others are *limit ordinals* and are defined as the smallest ordinal greater than (least upper bound of) all ordinals in a given set. For example, ω is the limit ordinal obtained as the least upper bound of all finite ordinals, and is usually described as the first infinite ordinal. The ordinal $\omega + 1$ is a non-limit ordinal and is the immediate successor of ω.
- The order $>$ ("greater than") is well-founded over the ordinals.
- There exists an induction principle over the ordinals, called *transfinite induction*, which generalizes the complete induction principle over the naturals. It can be stated as

$$\forall\alpha\Big(\big(\forall\beta<\alpha\colon\varphi(\beta)\big)\rightarrow\varphi(\alpha)\Big)\quad\rightarrow\quad\forall\gamma\colon\varphi(\gamma).$$

This principle claims that if we can prove the induction step then the property φ holds for every ordinal. The induction step requires showing, for every α, that if $\varphi(\beta)$ for all $\beta < \alpha$ then φ holds for ordinal α.

Ordinals are not so mysterious as they may sometimes appear to be. In fact, we have already introduced and extensively used structures that are isomorphic to the low infinite ordinals.

A lexicographic pair of natural numbers (m, n) can be faithfully considered as an isomorphic representation of the ordinal $\omega \cdot m + n$. To see this, we observe that lexicographic pairs of the form $(0, n)$ are isomorphic to the naturals, with $(0, n)$ corresponding to n. The pair $(1, 0)$, like ω, is the smallest pair which is bigger than all the pairs in the infinite set $\big\{(0, n) \mid n \geq 0\big\}$.

Our proof relies on the following general theorem.

Theorem 3.5 (embedding within the ordinals)

Let $\sqsupset$ be a well-founded ordering over a set S. Then, there exists a (ranking) function $\delta_\pi\colon S \mapsto \mathcal{O}$ such that:

(a) $s \sqsupset s'$ implies $\delta_\pi(s) > \delta_\pi(s')$.

(b) If $\delta_\pi(s) = \beta$ and $\alpha < \beta$, then there exists an element s' such that $\delta_\pi(s') = \alpha$ and $s \sqsupset s'$.

(c) If $s' \sqsupset s''$ implies $s \sqsupset s''$ for every $s'' \in S$, then $\delta_\pi(s) \geq \delta_\pi(s')$.

The theorem can be proven, using a transfinite construction that assigns a value $\delta_\pi(s)$ to an element s, based on the values $\delta_\pi(s')$ assigned to all $s' \sqsubset s$. The constrruction is defined as follows:

- $\delta_\pi(s) = 0$ for all $\sqsupset$-minimal elements of S, i.e., elements s such that there exists no $s' \sqsubset s$.
- For all other $s \in S$, $\delta_\pi(s)$ is the least upper bound of $\delta_\pi(s') + 1$ for all $s' \sqsubset s$.

Example Consider system LEX presented in Fig. 3.46. Transition τ_1 of this system sets x to zero and sets y to an arbitrary positive integer value. Transition τ_1 can be taken only once since it sets x to 0 which disables τ_1 for any future activation. Transition τ_2 can be taken as long as $y > 0$ and always sets y to a nonnegative value which is smaller than its value before the transition. Obviously, any computation of this systems takes τ_1 once and takes τ_2 finitely many times until it terminates in the state $\langle x\colon 0,\ y\colon 0\rangle$.

Assume we wish to prove for this system the following response property.

$$\underbrace{x = y = 1}_{p=\Theta}\quad\Rightarrow\quad\Diamond\Big(\underbrace{x = y = 0}_{q}\Big)\ .$$

V: $\{x, y : \textbf{integer}\}$

Θ: $x = y = 1$

$\mathcal{T}$: $\{\tau_I, \tau_1, \tau_2\}$ where $\rho_{\tau_1}: x > 0 \ \wedge \ x' = 0 \ \wedge \ y' > 0$

$\qquad\qquad\qquad\qquad \rho_{\tau_2}: x = 0 \ \wedge \ y > 0 \ \wedge \ x' = x \ \wedge \ 0 \leq y' < y$

$\mathcal{J}$: $\{\tau_1, \tau_2\}$

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

Fig. 3.46. System LEX.

States of this system consist of pairs of integers (m, n), where m and n are the current values of x and y, respectively. Clearly, the set of Q-states is given by

$$\|Q\|: \quad \{(1,1)\} \ \cup \ \{(0,n) \mid n > 0\}.$$

Consider the primary ordering $\sqsupset$, holding between (m_1, n_1) and $(m_2, n_2) \in \|Q\|$ if there exists a fair Q-segment connecting (m_1, n_1) to (m_2, n_2). It can be shown that the order $\sqsupset$ is given by

$$(m_1, n_1) \sqsupset (m_2, n_2): \quad (m_1, n_1) \succ (m_2, n_2) \ \wedge \ m_2 = 0 \ \wedge \ n_2 > 0,$$

where $\succ$ is the lexicographic ordering over $\|Q\|$. The condition $m_2 = 0$ follows from the observation that any fair segment must either take τ_1 which sets x to 0, implying $m_2 = 0$, or have τ_1 disabled on some state of the segment, which also implies $m_2 = 0$. The consequence of this condition is that any two states of the form $(1, n_1)$ and $(1, n_2)$ are unrelated by $\sqsupset$. The condition $n_2 > 0$ excludes from consideration the state $(0, 0)$ which satisfies q.

Applying the described bottom-up construction of the mapping δ_π, we observe the following:

- $(0, 1)$ is the least element of $\|Q\|$, hence $\delta_\pi\big((0,1)\big) = 0$.
- $\delta_\pi\big((0,2)\big)$ is the least upper bound of $\delta_\pi\big((m,n)\big) + 1$ for all $(m,n) \prec (0,2)$. Since the only element smaller than $(0, 2)$ is $(0, 1)$, we compute

  $$\delta_\pi\big((0,2)\big) \ = \ \textit{least-upper-bound}\big(\{1\}\big) \ = \ 1.$$

  In a similar way, we can show that

  $$\delta_\pi\big((0, n+1)\big) \ = \ n$$

  for every natural n.
- Next consider $\delta_\pi\big((1,1)\big)$. State $(1, 1)$ dominates all states $(0, n)$. Consequently, denoting *least-upper-bound* by *l.u.b*, we obtain

$$\delta_\pi\big((1,1)\big) \;=\; l.u.b\Big(\big\{\delta_\pi((0,n+1))+1 \mid n\in\mathbb{N}\big\}\Big) \;=\; lub\{n+1 \mid n\in\mathbb{N}\} \;=\; \omega.$$

Let us return to the general case of the primary order $\sqsupset$ holding between states s and s' if there exists a fair Q-segment connecting s to s'. Applying the embedding theorem to $\sqsupset$, we obtain a mapping δ_π from the program states to the ordinals $\mathcal{O}$.

The following statement identifies a property satisfied by the mapping δ_π.

Property P0 If $s\in \|Q\|$ and s' is a P-successor of state s, not satisfying q, then $s'\in\|Q\|$ and $\delta_\pi(s)\geq\delta_\pi(s')$.

Justification From the definition of Q it follows that if s', a P-successor of some $s\in\|Q\|$, does not satisfy q it is reachable by a (p,q)-prefix and, therefore, belongs to $\|Q\|$. Let s'' be such that $s'\sqsupset s''$. By definition of $\sqsupset$, there exists a fair Q-segment σ: $[s',\ldots,s'']$. Obviously, the segment $[s,s',\ldots,s'']$ is also a fair Q-segment, implying $s\sqsupset s''$. Thus, $s'\sqsupset s''$ implies $s\sqsupset s''$ for every s''. By clause (c) of the embedding theorem, it follows that $\delta_\pi(s)\geq\delta_\pi(s')$. ∎

The Secondary Ranking

We can view the mapping δ_π as a first approximation to our desired ranking function. Unfortunately, it did not remedy our complaint about $\sqsupset$ being insensitive to fine progress. For example, the mapping δ_π, corresponding to the fundamental order presented in Fig. 3.45, is the following.

$$\delta_\pi(s_{3,4})=\delta_\pi(s_{5,4})=\delta_\pi(s_{3,6})=1,\quad \delta_\pi(s_{5,6})=0.$$

This primary rank is presented in Fig. 3.47.

As we see, there is no observable progress (according to the measure δ_π) when the program moves from $s_{3,4}$ to $s_{5,4}$ or from $s_{3,4}$ to $s_{3,6}$.

We proceed to define a secondary ranking which, together with δ_π, will form our final ranking function. Looking at the diagram of Fig. 3.47, we see that the three states $s_{3,4}$, $s_{5,4}$, and $s_{3,6}$ are situated on a plateau defined by $\delta_\pi=1$. However, in the order of execution, $s_{5,4}$ and $s_{3,6}$ are "closer to the edge" of the plateau than state $s_{3,4}$. How can we measure this closeness? A possible measure could be the "size" of possible paths that are all located within the tableau. Thus, $s_{3,4}$ is further away from the edge of the plateau since it has a path of length 2, traversing states that all have $\delta_\pi=1$. States $s_{5,4}$ and $s_{3,6}$ are closer to the edge since they only have paths of length 1.

In the simple example shown here, we could take the "size" of the path to be simply its length. However, in many cases this is not the right measure, since

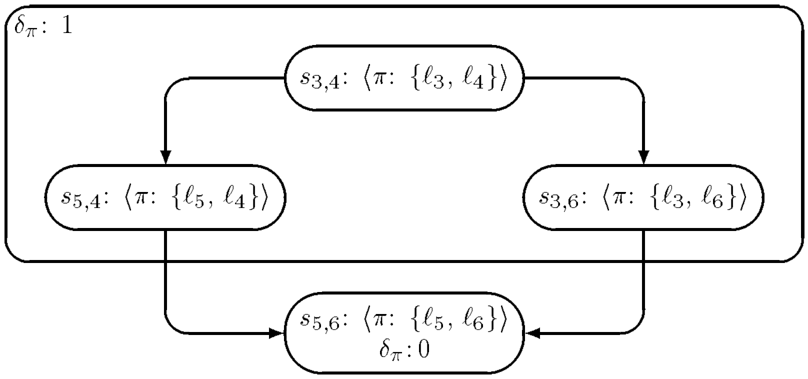


Fig. 3.47. Q-states of program SKIP2 ranked by δ_π.

states may have infinite paths all with the same value of δ_π. The relevant "size" of a path in our case is the number of transitions it gratifies.

For a segment σ, we define the *deficit* of σ, denoted by $\Delta(\sigma)$, to be the smallest natural number $i \geq 1$ such that $i \notin \mathit{grat}(\sigma)$, i.e., τ_i is not gratified by σ. If σ is fair we define $\Delta(\sigma) = m+1$.

We use the deficit as the measure of the "size" of a segment. Note that it is related to the number of transitions gratified by a segment, since if $\mathit{grat}(\sigma) \subseteq \mathit{grat}(\sigma')$ then $\Delta(\sigma) \leq \Delta(\sigma')$.

A (Q-)segment σ: $[s_a, \ldots, s_b]$ is called *leveled* if $\delta_\pi(s_a) = \delta_\pi(s_{a+1}) = \cdots = \delta_\pi(s_b)$. This corresponds to the previously discussed notion of a segment that is fully contained in a plateau, defined by the value of $\delta_\pi(s_a)$ which is common to all states in the segment. We define the *height* of a state s, denoted by $h(s)$, by

$$h(s) \ = \ \max\Big\{\Delta\big([s = s_a, \ldots, s_b]\big) \mid \delta(s_a) = \cdots = \delta(s_b)\Big\}.$$

That is, $h(s)$ is the maximal deficit over all leveled segments originating at s. This corresponds to the intuition of the distance from the edge of the plateau along leveled segments.

Example Consider the ranking by δ_π on the states presented in Fig. 3.47. Segments $[s_{3,4}, s_{5,4}]$ and $[s_{3,4}, s_{3,6}]$ are leveled. Segments $[s_{5,4}, s_{5,6}]$ and $[s_{3,4}, s_{5,4}, s_{5,6}]$ are not leveled.

There are three leveled Q-segments originating at $s_{3,4}$: $[s_{3,4}]$, $[s_{3,4}, s_{5,4}]$, and $[s_{3,4}, s_{3,6}]$. Segment $[s_{3,4}]$ gratifies all pair transitions except for $\tau_3\colon \ell_3$ and $\tau_4\colon \ell_4$. Consequently, its deficit is $\Delta\big([s_{3,4}]\big) = 3$. Segment $[s_{3,4}, s_{5,4}]$ gratifies all fair transitions except for $\tau_4\colon \ell_4$. Consequently, its deficit is $\Delta\big([s_{3,4}, s_{5,4}]\big) = 4$. Segment $[s_{3,4}, s_{3,6}]$ gratifies all fair transitions except for $\tau_3\colon \ell_3$. Consequently

$\Delta([s_{3,4}, s_{3,6}]) = 3$. Taking the maximum over $\{3, 4\}$, we obtain the height of $s_{3,4}$ as $h(s_{3,4}) = 4$.

The only leveled Q-segment originating at $s_{5,4}$ is the singleton segment $[s_{5,4}]$ which gratifies all fair transitions except for $\tau_4 \colon \ell_4$ and $\tau_5 \colon \ell_2^X$. Consequently, $h(s_{5,4}) = 4$.

In a similar way, $h(s_{3,6}) = 3$ and $h(s_{5,6}) = 5$. ■

The Final Ranking

The final ranking δ that is intended to satisfy premises FW2 and FW3 combines the primary ranking δ_π with the secondary ranking induced by comparing state heights.

As the well-founded domain, we take the lexicographic product

$$(\mathcal{A},\ \succ)\colon\quad (\mathcal{O},\ >) \times \big([1..m+1],\ >\big).$$

Thus, elements of $\mathcal{A}$ are pairs (α, k), where $\alpha \in \mathcal{O}$ is an ordinal and $k \in [1..m+1]$. The ranking function, mapping states into elements of $\mathcal{A}$, is given by

$$\delta(s)\colon\quad \big(\delta_\pi(s),\ h(s)\big).$$

This definition of $\delta(s)$ applies only to states which satisfies Q. To complete the definition, we define

$$\delta(s) \ = \ (0, 0),$$

for all states that do not satisfy Q. An immediate consequence of this definition is that

$$\delta(s_1) \ \succ \ \delta(s_2)$$

for every $s_1 \in \|Q\|$ and $s_2 \notin \|Q\|$. This is because $\delta_0(s) \geq 0$ and $h(s) > 0$ for every $s \in \|Q\|$.

Rule WELL-F allows us to have a different ranking function δ_i for each $i = 0, 1, \ldots, m$. However, in the proof of completeness we do not use this option, and take $\delta_i = \delta$ for all $i = 0, \ldots, m$.

Example We return to program SKIP2. In Fig. 3.48 we present the ordering between the states as determined by the complete rank δ.

As we see in the diagram, states $s_{3,4}$ and $s_{5,4}$ have equal ranks $(1, 4)$. This is because they both have $\delta_\pi = 1$ and have height 4. State $s_{3,4}$ is higher than $s_{3,6}$ because $\delta_\pi(s_{3,4}) = \delta_\pi(s_{3,6}) = 1$ but $h(s_{3,4}) = 4 > 3 = h(s_{3,6})$. States $s_{5,4}$ and $s_{3,6}$ are higher than $s_{5,6}$ because $\delta(s_{5,4}) = (1, 4) > (0, 5) = \delta_\pi(s_{5,6})$ and $\delta(s_{3,6}) = (1, 3) \succ (0, 5) = \delta(s_{5,6})$.

Thus, the full ranking δ identifies the step from $s_{3,4}$ to $s_{3,6}$ as observable

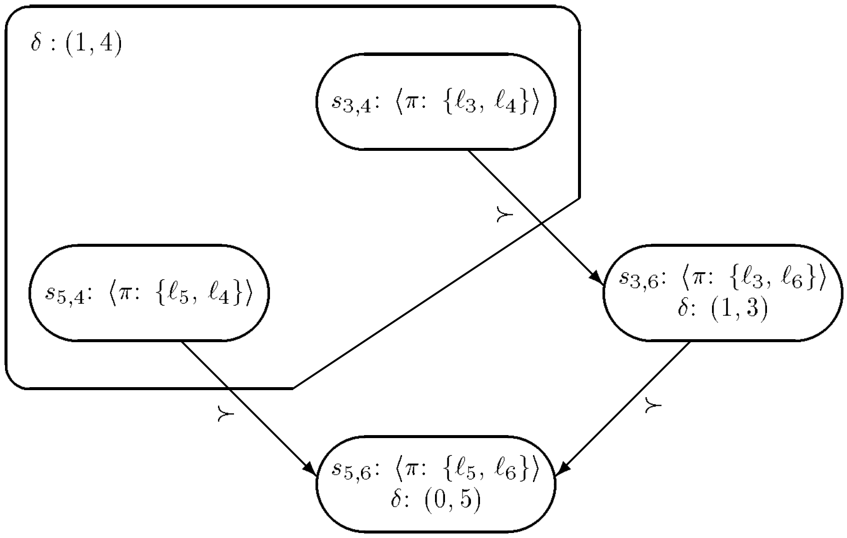


Fig. 3.48. States of program SKIP2 ranked by δ.

progress but not the step from $s_{3,4}$ to $s_{5,4}$. The reason for this lack of symmetry will become apparent later. ▟

There are several properties that the defined constructs satisfy.

Property P1 If $s \in \|Q\|$ then $h(s) \leq m$.

This is because $h(s) = m + 1$ implies the existence of a leveled Q-segment $s = s_a, \ldots, s_b$ which gratifies all the fair transitions. However, in this case, $s_a, \ldots, s_b$ is fair and therefore $s_a \sqsupset s_b$, contradicting the assumption that $s_a, \ldots, s_b$ is leveled.

Property P2 If $s, s' \in \|Q\|$ and s' is a successor of s then $\delta(s) \succeq \delta(s')$.

Property P0 already established $\delta_\pi(s) \geq \delta_\pi(s')$. If $\delta_\pi(s) > \delta_\pi(s')$ we are done.

It remains to consider the case $\delta_\pi(s) = \delta_\pi(s')$ and show that $h(s) \geq h(s')$. Let $h(s') = j$. By definition of the height h, there exists a leveled Q-segment σ: $[s'..s_b]$ such that $\Delta(\sigma) = j$. The segment $\widetilde{\sigma}$: $s, s', \ldots, s_b$ is a leveled $\big(\delta_\pi(s) = \delta_\pi(s')\big)$ Q-segment originating at s. Since $\widetilde{\sigma}$ extends σ, $grat(\widetilde{\sigma}) \supseteq grat(\sigma)$ and therefore $\Delta(\widetilde{\sigma}) \geq \Delta(\sigma) = j$. Since $h(s)$ is the maximal deficit over all leveled Q-segments originating at s, and $\widetilde{\sigma}$ is one of them, it follows that $h(s) \geq \Delta(\widetilde{\sigma}) \geq j = h(s')$, leading to the required inequality.

Property P3 If $s, s' \in \|Q\|$, $h(s) = i$, and s' is a τ_i-successor of s, then $\delta(s) \succ \delta(s')$.

By P2, $\delta(s) \succeq \delta(s')$. If $\delta_\pi(s) > \delta_\pi(s')$, we are done. Therefore, consider the case $\delta_\pi(s) = \delta_\pi(s')$ and let $h(s') = j$. Since $\delta(s) \succeq \delta(s')$, it follows that $i \geq j$. It remains to show that $i \neq j$. Assume, to the contrary, that $i = j$. As before, let σ: $[s', \ldots, s_b]$ be a leveled Q-segment realizing the deficit $\Delta(\sigma) = j$. This means that σ gratifies all fair transitions τ_k, $k < j$, but does not gratify τ_j. Consider again the extended segment $\widetilde{\sigma}$: $[s, s', \ldots, s_b]$. This segment gratifies all the transitions gratified by σ and, in addition, also gratifies $\tau_i = \tau_j$ since τ_i is taken in the step from s to s'. It follows that $\Delta(\widetilde{\sigma}) > i$. This contradicts the definition of $h(s)$ as being the maximal deficit of all leveled Q-segments originating at s.

The Intermediate Assertions

In the next step of the proof, we construct the intermediate assertions $\varphi_1, \ldots, \varphi_m$, required by rule WELL-F. For each $i = 1, \ldots, m$, we define

$$\varphi_i: \quad Q \ \land \ h = i.$$

Thus, φ_i is satisfied by all Q-states whose height is precisely i.

To complete the definition, we take

$$\varphi_0: \ q.$$

The following property is a consequence of the definition of the intermediate assertions.

Property P4 If $s \in \|Q\|$, then s satisfies one of $\varphi_1, \ldots, \varphi_m$.

To prove this statement, let s be a state satisfying Q. It has a defined height $h(s) = j$ which, by P1, is in the range $j \in [1..m]$. Consequently, s satisfies φ_j: $Q \land h = j$.

Example For the response property $at\_\ell_3 \land at\_\ell_4 \Rightarrow \diamondsuit\, at\_\ell_7$ of program SKIP2, we obtain

$$\begin{array}{lll}
\varphi_0 = q: & at_\ell_7 & \\
\varphi_1: & \underbrace{at_\ell_{3,5} \ \land \ at_\ell_{4,6}}_{Q} \land h = 1 & \sim \quad \mathrm{F} \\
\varphi_2: & at_\ell_{3,5} \ \land \ at_\ell_{4,6} \ \land \ h = 2 & \sim \quad \mathrm{F} \\
\varphi_3: & at_\ell_{3,5} \ \land \ at_\ell_{4,6} \ \land \ h = 3 & \sim \quad at_\ell_3 \ \land \ at_\ell_6 \\
\varphi_4: & at_\ell_{3,5} \ \land \ at_\ell_{4,6} \ \land \ h = 4 & \sim \quad at_\ell_{3,5} \ \land \ at_\ell_4
\end{array}$$

$$\varphi_5 \quad : \quad at\_\ell_{3,5} \ \wedge \ at\_\ell_{4,6} \ \wedge \ h = 5 \quad \sim \quad at\_\ell_5 \ \wedge \ at\_\ell_6.$$

The assertions are simplified based on the full knowledge of the four Q-states and their heights. ▟

Claim 3.3 implies that the response formula $Q \Rightarrow \Diamond q$ is P-valid. Since φ_i: $Q \wedge h = i$ implies Q we conclude that for every $i = 1, \ldots, m$, the response formula

$$\varphi_i \ \Rightarrow \ \Diamond q$$

is also P-valid. ▟

Proving the Premises

The construction of $\varphi_0, \ldots, \varphi_m$ concludes the development of all the constructs needed for rule WELL-F.

We proceed to show that all premises are satisfied by these constructs. All premises, excluding CW4, are implications $\varphi \rightarrow \psi$ that have to be shown as P-state valid. To prove this, it is sufficient to show that any P-accessible state satisfying φ also satisfies ψ.

- Premise FW1. This premise requires

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

Let s be an accessible state satisfying p. If s satisfies $q = \varphi_0$, we are done. Otherwise, s satisfies assertion Q since it is reachable by the singleton (p, q)-segment $[s]$. By P4, s satisfies φ_j: $Q \wedge h = j$ for some $j \in [1..m]$.

- Premise FW2. This premise requires

$$\rho_\tau \ \wedge \ \varphi_i \quad \rightarrow \quad \bigvee_{j=0}^{m} (\varphi'_j \ \wedge \ \delta \succ \delta') \ \vee \ (\varphi'_i \ \wedge \ \delta = \delta').$$

Let s be a state satisfying φ_i: $Q \wedge h = i$ and let s' be a τ-successor of s. If s' satisfies $q = \varphi_0$, we are done. Otherwise, s' satisfies Q, then, by P4, φ'_j holds for some $j \in [1..m]$ and, by P2, $\delta(s) \succeq \delta(s')$ which means that either $\delta_\pi(s) > \delta_\pi(s')$ or $\delta_\pi(s) = \delta_\pi(s')$ and $h(s) \geq h(s')$. If $\delta_\pi(s) > \delta_\pi(s')$ or $\delta_\pi(s) = \delta_\pi(s')$ and $h(s) > h(s')$, then $\delta(s) \succ \delta(s')$. Otherwise, $\delta_\pi(s) = \delta_\pi(s')$ and $h(s) = h(s')$ which implies $\delta(s) = \delta(s')$ and that $\varphi'_i = \varphi_i(s') = Q(s') \wedge h(s') = i$ holds.

- Premise FW3. This premise requires

$$\rho_{\tau_i} \ \wedge \ \varphi_i \quad \rightarrow \quad \bigvee_{j=0}^{m} (\varphi'_j \ \wedge \ \delta \succ \delta').$$

Let s be a state satisfying φ_i: $Q \wedge h = i$ and s' a τ_i-successor of s. If s' satisfies $q = \varphi_0$, we are done. Otherwise s' satisfies Q and, therefore, by P4, φ'_j for some $j \in [1..m]$. Also, by P3, $\delta(s) \succ \delta(s')$.

Premise FW4 consists of the two subpremises: CW4 and JW4. We consider JW4 first.

- Subpremise JW4. This subpremise requires

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

Let s be a state satisfying φ_i: $Q \wedge h = i$. By the definition of h, there exists a leveled Q-segment σ: $[s, \ldots, s_b]$ whose deficit is $\Delta(\sigma) = i$. This implies that τ_i is not gratified in σ.

Transition τ_i must be enabled on s because otherwise, τ_i would have been gratified in σ.

- Subpremise CW4. This subpremise requires a proof of

$$\varphi_i \;\Rightarrow\; \diamondsuit\big(\neg\varphi_i \;\vee\; En(\tau_i)\big).$$

All other premises considered previously were implications of state formulas. Since our statement of completeness is relative to state validities it was enough to show that premises FW2, FW3, and subpremise JW4 are state valid and that premise FW1 is P-state valid.

Subpremise CW4 is a temporal formula. Consequently, we have to show that its proof can be syntactically reduced, perhaps using rule WELL-F again, to state and P-state validities.

We begin by showing that $\varphi_i \Rightarrow \diamondsuit\big(\neg\varphi_i \vee En(\tau_i)\big)$ is P^{-i}-valid where P^{-i} is the fair transition system obtained from P by removing transition τ_i from the compassion set $\mathcal{C}_P$. This is motivated by the observation that, if we are currently at position j where τ_i is disabled and are working towards the first position $k > j$ where τ_i becomes enabled, then τ_i cannot be taken between j and k. Therefore, the computation segment between j and k is also a computation segment of the simpler system P^{-i}.

Assume to the contrary, that $\varphi_i \Rightarrow \diamondsuit\big(\neg\varphi_i \vee En(\tau_i)\big)$ is not P^{-i}-valid. Then, there exists a computation σ of P^{-i} and a position $j \geq 0$, such that φ_i holds and τ_i is disabled at all positions $k \geq j$. Since τ_i is continually disabled beyond j, σ is also a computation of the complete system P. In particular, σ is fair with respect to τ_i which is never taken due to the continual disableness of τ_i. Furthermore, φ_i implies Q which implies $\neg q$. It follows that q never holds beyond j leading to the fact that $Q \Rightarrow \diamondsuit\, q$ does not hold on σ, a computation of P. This contradicts a corollary of Claim 3.3. We therefore conclude that $\varphi_i \Rightarrow \diamondsuit\big(\neg\varphi_i \vee En(\tau_i)\big)$ is P^{-i}-valid.

Next, we show that subpremise CW4 is not only P^{-i}-valid but can be proven from state validities. This is the time to explain to the reader that the completeness proof is done by induction on κ, the number of compassionate transitions.

If $\kappa = 0$ then we are never called to establish subpremise CW4. Therefore all necessary premises and subpremises are state formulas and it is sufficient to show that they are state or P-state valid as we have done in the previous discussion.

If $\kappa > 0$ then system P^{-i} has only $\kappa - 1 < \kappa$ compassionate transitions since τ_i is compassionate. By the induction hypothesis, rule WELL-F is complete for such systems. It follows that the response formula $\varphi_i \Rightarrow \diamondsuit\big(\neg\varphi_i \vee En(\tau_i)\big)$, which has been shown to be P^{-i}-valid, is provable by a (recursive use of) rule WELL-F.

This concludes the proof of completeness of rule WELL-F for proving response formulas $p \Rightarrow \diamondsuit\, q$ for the case that p and q are assertions.

Example We illustrate the generation of the constructs needed for rule WELL-F on an example. In particular, we wish to illustrate the principle of inductive reduction by which compassionate transitions are removed one at a time.

Consider program SIMPLE-MUX presented in Fig. 3.49.

$$\textbf{local } y\text{: } \textbf{integer where } y = 1$$

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

Fig. 3.49. Program SIMPLE-MUX.

This program can be viewed as a simplified version of program MUX-SEM presented in Fig. 3.1.

The property in which we are interested is

$$\psi\text{:}\quad \underbrace{at\_m_0}_{p} \;\Rightarrow\; \diamondsuit\, \underbrace{at\_m_1}_{q}\,,$$

claiming that process P_2 eventually terminates. This formula is obviously valid over SIMPLE-MUX. We follow the process outlined in the completeness proof to show how formula ψ can be verified over SIMPLE-MUX, using rule WELL-F.

Constructing Assertion Q

In this simple case, we can explicitly enumerate the set of accessible states. In Fig. 3.50 we present a state-transition graph which displays the accessible states of program SIMPLE-MUX and the transitions between them.

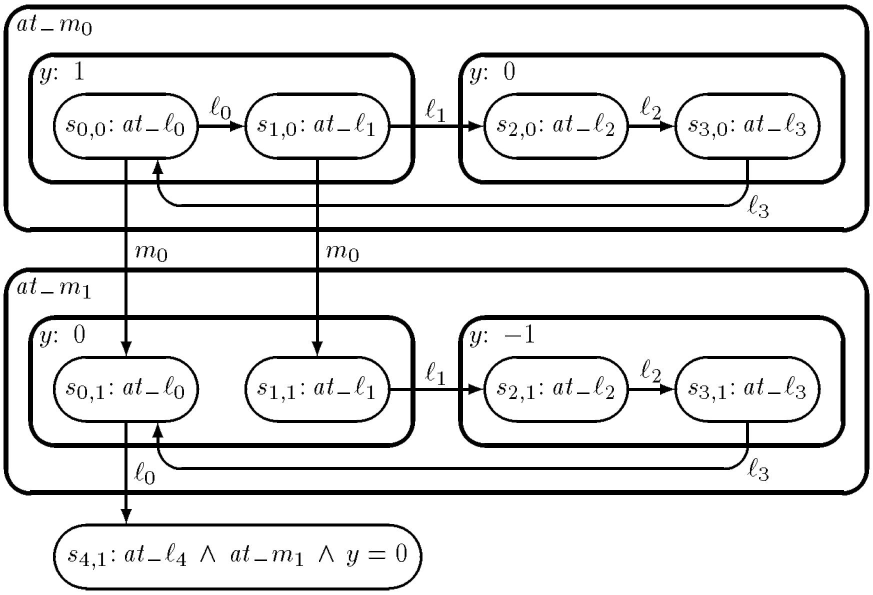


Fig. 3.50. A state-transition graph for program SIMPLE-MUX.

Assertion Q characterizes all the states which are reachable from an accessible (p: at_m_0)-state by a (q: at_m_1)-free segment. This includes the states

$$\|Q\|: \quad \{s_{0,0},\ s_{1,0},\ s_{2,0},\ s_{3,0}\}.$$

Therefore, we can take for Q the assertion

$$Q: \quad (at\_\ell_{0,1} \wedge y = 1 \vee at\_\ell_{2,3} \wedge y = 0) \wedge at\_m_0.$$

Constructs $\sqsupset$, δ_π, h, δ, φ_i, and τ_i

The fair transitions of program SIMPLE-MUX consist of the just transitions ℓ_0, ℓ_1, ℓ_2, ℓ_3 and the compassionate transition m_0. We list them in the following order:

$$\mathcal{F}: \quad \{\tau^1\colon \ell_0,\ \ \tau^2\colon \ell_1,\ \ \tau^3\colon \ell_2,\ \ \tau^4\colon \ell_3,\ \ \tau^5\colon m_0\}.$$

Examining the graph of Fig. 3.50, where the set $\|Q\|$ occupies the top row of states, we observe that there is no Q-segment which gratifies all the five fair transitions.

In particular, no Q-segment gratifies $\tau^5\colon m_0$. It follows that the order $\sqsupset$ is empty, i.e.,

$$\sqsupset\colon\ \emptyset.$$

This makes the task of computing δ_π trivial, leading to

$$\delta_\pi(s) = 0 \quad \text{for all } s \in \|Q\|.$$

Next, we compute the height function $h(s)$. Since $\delta_\pi(s) = 0$ for all Q-states, there is a leveled Q-segment departing from each Q-state which gratifies the transitions $\{\ell_0, \ell_1, \ell_2, \ell_3\}$. The deficit of such a segment is 5, corresponding to the ungratified transition $\tau^5\colon m_0$. We conclude that

$$h(s) = 5 \quad \text{for all } s \in \|Q\|.$$

It follows that the full ranking function is

$$\delta(s) = (0, 5) \quad \text{for all } s \in \|Q\|.$$

Consequently, there exists a single intermediate assertion $\varphi = Q$ with a corresponding helpful transition $\tau = \tau^5\colon m_0$.

It is not difficult to check that all the premises of rule WELL-F, except CW4, are P-state valid. It only remains to prove

$$\text{CW4.}\qquad \underbrace{(at\_\ell_{0,1} \ \wedge\ y = 1 \ \vee\ at\_\ell_{2,3} \ \wedge\ y = 0) \ \wedge\ at\_m_0}_{\varphi = Q} \ \Rightarrow \ \diamondsuit\Big(\neg Q \ \vee\ \underbrace{at\_m_0 \ \wedge\ y = 1}_{En(m_0)}\Big).$$

Proving CW4

To prove CW4, we reapply rule WELL-F with

$$\begin{array}{lcl} p^{-5} = Q\colon & & (at\_\ell_{0,1} \ \wedge\ y = 1 \ \vee\ at\_\ell_{2,3} \ \wedge\ y = 0) \ \wedge\ at\_m_0 \\ q^{-5} & : & \neg Q \ \vee\ (at\_m_0 \ \wedge\ y = 1) \end{array}$$

over program SIMPLE-MUX^{-5} which is identical to program SIMPLE-MUX, except that $\tau^5\colon m_0$ is no longer considered compassionate. The removal of m_0 from the compassion list does not change the behavior of the program in the segments of interest, but has an effect on the auxiliary constructs.

Assertion Q^{-5}

Assertion Q^{-5} should characterize all states which are reachable from an accessible p^{-5}-state by a q^{-5}-free segment. It is obvious that

$$\|Q^{-5}\| \ = \ \{s_{2,0},\ s_{3,0}\}.$$

Consequently,

$$Q^{-5}: \quad at\_\ell_{2,3} \ \wedge \ at\_m_0 \ \wedge \ y = 0.$$

Constructs $\sqsupset^{-5}$, δ_π^{-5}, h^{-5}, δ^{-5}, φ_i^{-5}, and τ_i^{-5}

Consider the q^{-5}-free segments departing from Q^{-5}-states. State $s_{2,0}$ initiates the segment $[s_{2,0}, s_{3,0}]$ which gratifies all the remaining four transitions ℓ_0, ℓ_1, ℓ_2, and ℓ_3. Consequently, we set

$$\sqsupset^{-5} \ = \ (s_{2,0}, \ s_{3,0})$$

which implies $s_{2,0} \sqsupset^{-5} s_{3,0}$.

Computing δ_π^{-5}, we get

$$\delta_\pi^{-5}(s_{2,0}) \ = \ 1 \qquad \text{and} \qquad \delta_\pi^{-5}(s_{3,0}) \ = \ 0.$$

It follows that $s_{2,0}$ has $[s_{2,0}]$ as the only departing leveled Q-segment with deficit 3, corresponding to the ungratified transition $\tau^3\colon \ell_2$. State $s_{3,0}$ has the departing leveled Q-segment $[s_{3,0}]$ with deficit 4, corresponding to $\tau^4\colon \ell_3$.

Consequently, we obtain two intermediate assertions with the following associated constructs:

i	φ_i^{-5}	δ^{-5}	Helpful transition τ_i^{-5}
1	$at\_\ell_2 \ \wedge \ at\_m_0 \ \wedge \ y = 0$	$(1, 3)$	ℓ_2
2	$at\_\ell_3 \ \wedge \ at\_m_0 \ \wedge \ y = 0$	$(0, 4)$	ℓ_3

It is not difficult to check that all premises of rule WELL-F are satisfied for this choice. Since the two helpful transitions are just, it is only necessary to check subpremise JW4. ▪

In **Problem 3.4**, we ask the reader to apply the construction outlined in the completeness proof to the full mutual-exclusion program MUX-SEM of Fig. 3.1.

Expressing the Constructs by a Formal Language

The completeness proof specified the necessary constructs by semantic means. We now show that all constructs can be expressed within an appropriate assertional language. We use some of the notation introduced in Section 2.5 of the SAFETY book.

The Kernel Assertion Q

Recall that a state s satisfies assertion Q if there exists a P-prefix $[s_1, \ldots, s_n]$, such that

- $s_n = s$, and
- for some $k \in [1..n]$, s_k satisfies p, and
- for all $i \in [k..n]$, s_i does not satisfy q.

We use the notations introduced in Section 2.5 of the SAFETY book to express Q for the case that the system variables of program P, $\overline{y} = \{y_1, \ldots, y_r\}$, range over data domains that are either natural numbers or encodable as natural numbers. We use a two-dimensional dynamic array $a \in \mathcal{A}^2(\mathbb{N})$ to encode sequences of states intended to represent prefixes and segments.

The complete formula is given by

$$Q(\overline{y}): \quad \exists n > 0\colon \exists a \in \mathcal{A}^2(\mathbb{N})\colon \Big(reach(\overline{y}) \;\wedge\; \exists k \in [1..n]\colon \big(p(a[k]) \;\wedge\; q\text{-}free\big)\Big),$$

where

$$reach(\overline{y}): \quad |a| = (n, r) \;\wedge\; \Theta\big(a[1]\big) \;\wedge\; a[n] = \overline{y} \;\wedge\; evolve(a, n)$$

$$evolve(a, n): \quad \forall i (1 \le i < n)\colon \bigvee_{\tau \in \mathcal{T}^-} \rho_\tau\big(a[i], a[i+1]\big)$$

$$q\text{-}free: \quad \forall i \in [k..n]\colon \neg q\big(a[i]\big).$$

The free variables in Q are the list of system variables $\overline{y} = y_1, \ldots, y_r$, representing the current state.

Segments and the Transitions they Gratify

In the following discussion we use two-dimensional dynamic arrays $b \in \mathcal{A}^2(\mathbb{N})$ of shape $|b| = (n, r)$ to represent segments.

The formula $Qseg(b, n, \overline{u})$ states that b encodes a Q-segment of length n, originating at state $\overline{u}$

$$Qseg(b, n, \overline{u}): \quad |b| = (n, r) \;\wedge\; b(1) = \overline{u} \;\wedge\; evolve(b, n) \;\wedge\; \forall i \in [1..n]\colon Q\big(b[i]\big).$$

The formula $gratify(b, i)$ states that segment b gratifies transition τ_i:

$$\begin{aligned} gratify(b, i): \quad & \exists j \in [1..n{-}1]\colon \rho_{\tau_i}\big(b[j], b[j+1]\big) \;\vee \\ & \Big(\tau_i \in \mathcal{C} \;\wedge\; \forall j \in [1..n]\colon \neg En\big(\tau_i, b[j]\big)\Big) \;\vee \\ & \Big(\tau_i \in \mathcal{J} - \mathcal{C} \;\wedge\; \exists j \in [1..n]\colon \neg En\big(\tau_i, b[j]\big)\Big), \end{aligned}$$

where

$$En(\tau_i, \overline{y}): \quad \exists \overline{y}'\colon \rho_{\tau_i}(\overline{y}, \overline{y}').$$

The formula $fair(b)$ states that segment b is fair, i.e., gratifies all fair transitions

$$fair(b):\quad \bigwedge_{i=1}^{m} grat(b,i).$$

The Primary Order $\sqsupset$

Recall that $s_1 \sqsupset s_2$ if there exists a fair Q-segment leading from s_1 to s_2. Representing the state-variables at s_1 and at s_2 by $\overline{u}$ and $\overline{v}$, respectively, we define the corresponding formula $\overline{u} \sqsupset \overline{v}$

$$\overline{u} \sqsupset \overline{v}:\quad \exists n > 0:\ \exists b \in \mathcal{A}^2(\mathbb{N}):\ \Big(Qseg(b,n,\overline{u})\ \wedge\ b[n] = \overline{v}\ \wedge\ fair(b)\Big).$$

The Primary Ranking δ_π

When we attempt to express the primary ranking δ_π, we find that the assertional language we used so far, a first-order language over the integers, is no longer adequate. Our failure to express the ranking δ_π by a first-order language is not accidental and there are theoretical arguments which shows that some extensions to the first-order language are necessary.

The extension we choose is one which leads to the most natural formalization of the construction of δ_π. We allow *least fixed points* over recursive function definitions. With this extension we can express the ranking δ_π as the least fixed point of the recursive equation

$$F(\overline{u}) \quad = \quad lub\big\{F(\overline{v}) + 1 \mid \overline{v} \sqsubset \overline{u}\big\}.$$

Note that if $\overline{u}$ represents a $\sqsupset$-minimal state, then the least upper bound is taken over an empty set. The least ordinal which is bigger than all elements of the empty set is 0. This shows that $\delta_\pi(s) = 0$ for all $\sqsupset$-minimal states s.

Expressing the Height of a State

The formula $Lseg(b,n,\overline{u})$ identifies array b as representing a leveled Q-segment of length n, originating at state s, which is represented by system variables $\overline{u}$.

$$Lseg(b,n,\overline{u}):\quad Qseg(b,n,\overline{u})\ \wedge\ \forall i \in [1..n{-}1]:\ \delta_\pi\big(a[i]\big) = \delta_\pi\big(a[i+1]\big).$$

The formula $has\text{-}def(\overline{u},j)$ states the existence of a leveled Q-segment originating at $\overline{u}$ with deficit $j \in [1..m]$

$$has\text{-}def(\overline{u},j):\quad \exists n > 0\, \exists b \in \mathcal{A}^2(\mathbb{N}):\ Lseg(b,n,\overline{u})\ \wedge\ \neg gratify(b,j)\ \wedge\ \bigwedge_{k<j} gratify(b,k).$$

That is, τ_j is not gratified in b but every τ_k, $1 \le k < j$, is gratified in b.

The height function $h(\overline{u})$ can now be written as follows

$$h(\overline{u}):\quad max\big\{j \in [1..m] \mid has\text{-}def(\overline{u},j)\big\}.$$

This formula computes the maximal deficit over all leveled Q-segments departing from $\overline{u}$.

The Final Rank and Assertions

The final rank δ is given by

$$\delta(\overline{u}): \quad \big(\delta_\pi(\overline{u}),\ h(\overline{u})\big).$$

The intermediate assertions $\varphi_i(\overline{u})$, $i = 1, \ldots, m$, can be expressed by the following

$$\varphi_i(\overline{u}): \quad Q \ \land \ h(\overline{u}) = i.$$

** *3.9 Completeness: General Response*

In the general case, we consider a response formula

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

where p and q are past formulas. In this case, the intermediate formulas $\varphi_1, \ldots, \varphi_m$ are also past formulas and all premises are to be written as entailments $\Rightarrow$. Note that the ranking function δ may also depend on the past.

Example As our running example for the general case, consider system INC presented in Fig. 3.51 (see also Fig. 4.2 of the SAFETY book).

$$V: \quad \{x\colon \textbf{integer}\}$$
$$\Theta: \quad x \ = \ 0$$
$$\mathcal{T}: \quad \{\tau_I, \tau\} \text{ where } \rho_\tau\colon\ x' = x + 1$$
$$\mathcal{J}: \quad \{\tau\}$$
$$\mathcal{C}: \quad \{\ \}$$

Fig. 3.51. System INC.

We wish to establish for it the response property

$$\diamondminus(x = 5) \ \Rightarrow \ \diamondsuit\, \diamondminus(x = 10).$$

This property states that every position that has $x = 5$ in its past is followed by a position that has $x = 10$ in its past. ▪

The strategy we intend to follow in proving completeness for the general case is identical to the one followed in Section 4.9 of the SAFETY book, where we established completeness for past invariances.

Namely, given a response formula $p \Rightarrow \diamondsuit q$, where p and q are past formulas, and a program P, we construct statified versions of p, q, and P, denoted by $\widehat{p}$, $\widehat{q}$, and $\widehat{P}$, respectively. Assuming that $p \Rightarrow \diamondsuit q$ is P-valid, we conclude that $\widehat{p} \Rightarrow \diamondsuit \widehat{q}$ is $\widehat{P}$-valid. We invoke the proof of completeness for response over state formulas to construct intermediate assertions $\widehat{\varphi}_1, \ldots, \widehat{\varphi}_m$ and ranking $\widehat{\delta}$, satisfying premises FW1, FW2, FW3, and FW4 of rule WELL-F. As a last step we unstatify $\widehat{\varphi}_1, \ldots, \widehat{\varphi}_m$, and $\widehat{\delta}$ to obtain past formulas $\varphi_1, \ldots, \varphi_m$ and a (possibly) past dependent ranking δ.

Since these steps are very similar to the corresponding steps taken in Section 4.9 of the SAFETY book, we provide only a sketch and concentrate on a few minor differences between the two cases.

Statifying p, q, and Program P

The base for statification in Section 4.9 of the SAFETY book was the closure Φ_ψ which contains all subformulas of ψ whose principal operator is a past operator. Here, we need to statify both p and q at the same time. Consequently, we base statification on the set $\Phi_{p,q}$ which contains all subformulas of p and q with a past principal operator.

Using $\Phi_{p,q}$ we statify p, q, and P.

Example Consider property $\underbrace{\diamondminus(x=5)}_{p} \Rightarrow \diamondsuit \underbrace{\diamondminus(x=10)}_{q}$ for system INC. The closure of p and q is given by

$$\Phi_{p,q}: \quad \{\diamondminus(x=5),\ \diamondminus(x=10)\}.$$

The statification transformation can be defined as

$$stat(\varphi): \quad \varphi[b_5/\diamondminus(x=5),\ b_{10}/\diamondminus(x=10)].$$

Consequently, we have $\widehat{p}$: b_5 and $\widehat{q}$: b_{10}.

The statified system $\widehat{\text{INC}}$ is presented in Fig. 3.52.

Thus, we have reduced the task of proving $\diamondminus(x=5) \Rightarrow \diamondsuit \diamondminus(x=10)$ over INC to the task of proving $b_5 \Rightarrow \diamondsuit b_{10}$ over $\widehat{\text{INC}}$. ▪

Invoking the Completeness Proof for State Response

Faced with the task of verifying $\widehat{p} \Rightarrow \diamondsuit \widehat{q}$ over $\widehat{P}$, we invoke the completeness proof for state response formulas. This invocation results in intermediate assertions $\widehat{\varphi}_1, \ldots, \widehat{\varphi}_m$, and ranking $\widehat{\delta}$ satisfying premises FW1, FW2, FW3, and FW4 of rule WELL-F.

$\widehat{V}$: $\{x$: **integer**; b_5, b_{10}: **boolean**$\}$

$\widehat{\Theta}$: $x = 0 \ \wedge \ b_5 = b_{10} = \text{F}$

$\widehat{\mathcal{T}}$: $\{\widehat{\tau}_I, \widehat{\tau}\}$ with transition relation (after simplification)
$\rho_{\widehat{\tau}}$: $x' = x + 1 \quad \wedge \quad b'_5 = (x = 4 \ \vee \ b_5) \quad \wedge \quad b'_{10} = (x = 9 \ \vee \ b_{10})$

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

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

Fig. 3.52. Statified system $\widehat{\text{INC}}$.

Example Applying the completeness proof to property $b_5 \Rightarrow \diamondsuit\, b_{10}$ over system $\widehat{\text{INC}}$ may result in a single intermediate assertion $\widehat{\varphi}$ and the ranking $\widehat{\delta}$

$$\widehat{\varphi}: \quad x \geq 0 \ \wedge \ (b_5 \ \leftrightarrow \ x \geq 5) \ \wedge \ (b_{10} \ \leftrightarrow \ x \geq 10) \ \wedge \ (b_5 \ \wedge \ \neg b_{10})$$

$$\widehat{\delta}(x, b_5, b_{10}): \ \text{ if } \widehat{\varphi} \text{ then } (9 - x, 1) \text{ else } (0, 0).$$

Note that if we were doing a manual proof of the considered property, it would have been sufficient to take $5 \leq x < 10$ for $\widehat{\varphi}$ and $max(9 - x, 0)$ for $\widehat{\delta}(x)$. The constructs $\widehat{\varphi}$ and $\widehat{\delta}$ displayed above are (equivalent to) the ones that are actually constructed by the completeness proof. For example, they only allow accessible states in $\widehat{\varphi}$. ▪

Besides providing the constructs $\widehat{\varphi}_1, \ldots, \widehat{\varphi}_m$, and $\widehat{\delta}$, the state-response completeness proof provided us with a detailed deductive proof of $\widehat{p} \Rightarrow \diamondsuit\, \widehat{q}$ over program $\widehat{P}$. This proof consists of repeated applications of rule WELL-F, which establish the $\widehat{P}$-state validity of various instances of premises FW1, FW2, FW3, and subpremise JW4, all of which are state implications. Consequently, the steps in the proof of $\widehat{P} \vDash \widehat{p} \Rightarrow \diamondsuit\, \widehat{q}$ alternate between establishing the $\widehat{P}$-state validity of a state implication and invocations of rule WELL-F.

Unstatifying

As the final step, we unstatify the constructs $\widehat{\varphi}_1, \ldots, \widehat{\varphi}_m$, and $\widehat{\delta}$ into $\varphi_1, \ldots, \varphi_m$, and δ. Subsequently, we show that each of the premises, previously shown to be a $\widehat{P}$-state valid implication, is transformed into an entailment of past formulas which can be shown to be P-valid using inverse statification, rule IGEN, and entailment reasoning. This step is very similar to the corresponding step in Section 4.9 of the SAFETY book. Details are therefore omitted here.

Example Unstatifying the obtained constructs for system $\widehat{\text{INC}}$, we obtain the following

$$\varphi: \quad x \geq 10 \ \wedge \ \big(\diamondminus(x=5) \ \leftrightarrow \ x \geq 5\big) \ \wedge \ \big(\diamondminus(x=10) \ \leftrightarrow \ x \geq 10\big) \ \wedge$$
$$\big(\diamondminus(x=5) \ \wedge \ \neg\diamondminus(x=10)\big)$$

$$\delta(x): \quad \text{if } \varphi \text{ then } (9-x, 1) \text{ else } (0,0).$$ ∎

3.10 Finite-State Algorithmic Verification: State Response

In this and the next section we present an algorithm for checking whether a finite-state program satisfies a response property $p \Rightarrow \diamondsuit q$. Consistent with our general strategy, we consider in this section the case that p and q are assertions. In the next section, we study the extensions necessary to handle the more general case in which p and q may be general past formulas.

The algorithm searches for a counter-example. That is, it looks for a computation σ and a position $j \geq 0$ such that p holds at j but q holds at no position $k \geq j$.

Even for finite-state programs, a computation is an infinite object and, usually, there are uncountably many different computations. Consequently, we cannot simply enumerate all possible computations and check each of them for being a counter-example. Instead, we will analyze the state-transition graph G_P, which is always finite for a finite-state program P. Recall that G_P is a graph containing all the P-accessible states as nodes, and edges connecting s_1 to s_2 if s_2 is a τ-successor of s_1 for some $\tau \in \mathcal{T}$. In this section, we only consider states appearing in G_P. Therefore, any reference to a state s implies that s is P-accessible.

Let σ be a computation. Define $Inf(\sigma)$, called the *infinity set* of σ, to be the set of states that appear at infinitely many positions of σ. Since there are only finitely many P-accessible states, there exists a position $f \geq 0$ in σ such that

$$s \in Inf(\sigma) \quad \text{if and only if} \quad s = s_j \text{ for some position } j \geq f \text{ in } \sigma.$$

Recall that a subgraph $S \subseteq G_P$ is called a *strongly connected subgraph* (SCS) if, for every two distinct states $s_1, s_2 \in S$, there exists a path in S, leading from s_1 to s_2.

Since $Inf(\sigma)$ contains only P-accessible states, we can view it as a subgraph of G_P, called the subgraph *induced* by σ.

Claim 3.6 (from computations to subgraphs)

$Inf(\sigma)$ is an SCS.

Justification Let $s, \widehat{s} \in Inf(\sigma)$ be two different states. Since $s \in Inf(\sigma)$, there exists a position $j \geq f$ such that $s_j = s$. As $\widehat{s}$ also appears infinitely many times in σ, there exists a later position $k > j$ such that $s_k = \widehat{s}$. Consider the segment $s{=}s_j, s_{j+1}, \ldots, s_k{=}\widehat{s}$ in σ. For every $i \in [j..k]$, $i \geq f$. Consequently, $s_i \in \mathit{Inf}(\sigma)$ for every $i \in [j..k]$ and the path $[s_j, \ldots, s_k]$ leading from s to $\widehat{s}$ is contained in $\mathit{Inf}(\sigma)$. ∎

Example Consider system INC2 presented in Fig. 3.53.

V: $\{x\colon [0..7]\}$

Θ: $x = 0$

$\mathcal{T}$: $\{\tau_I, \tau_1, \tau_2, \tau_3\}$ where
$\rho_{\tau_1}\colon\ x' = x \oplus_8 1$
$\rho_{\tau_2}\colon\ \mathit{even}(x)\ \wedge\ x' = x \oplus_8 2$
$\rho_{\tau_3}\colon\ \mathit{odd}(x)\ \wedge\ x' = x \oplus_8 2$

$\mathcal{J}$: $\{\tau_1, \tau_2\}$

$\mathcal{C}$: $\{\tau_3\}$

Fig. 3.53. System INC2.

In Fig. 3.54, we present the state transition graph for INC2.

Consider the computation

$$\sigma\colon\quad s_0[s_2,\ s_4,\ s_6,\ s_0]^k\ [s_2,\ s_3,\ s_5, s_7,\ s_0]^\omega$$

which consists of s_0, followed by $k \geq 0$ repetitions of the segment $[s_2, s_4, s_6, s_0]$, followed by infinitely many repetitions of the segment $[s_2, s_3, s_5, s_7, s_0]$. Independent of how large k is,

$$\mathit{Inf}(\sigma)\ =\ \{s_0,\ s_2,\ s_3,\ s_5,\ s_7\}$$

since these are the states that appear infinitely many times in σ.

Fair Subgraphs

Not every SCS can be obtained as the infinity set $\mathit{Inf}(\sigma)$ of a computation σ. Consider, for example, the set S: $\{s_0, s_2, s_4, s_6\}$ of Fig. 3.54. This set cannot be the infinity set of a computation. This is because every sequence σ such that $\mathit{Inf}(\sigma) = S$ has transition τ_1 continuously enabled and never taken beyond position f (the position such that $s_j \in \mathit{Inf}(\sigma)$ for every $j \geq f$). Thus, any run σ such that $\mathit{Inf}(\sigma) = S$ is necessarily unjust towards transition τ_1.

Consequently, we characterize those SCS's that can be $\mathit{Inf}(\sigma)$ for some computation σ.

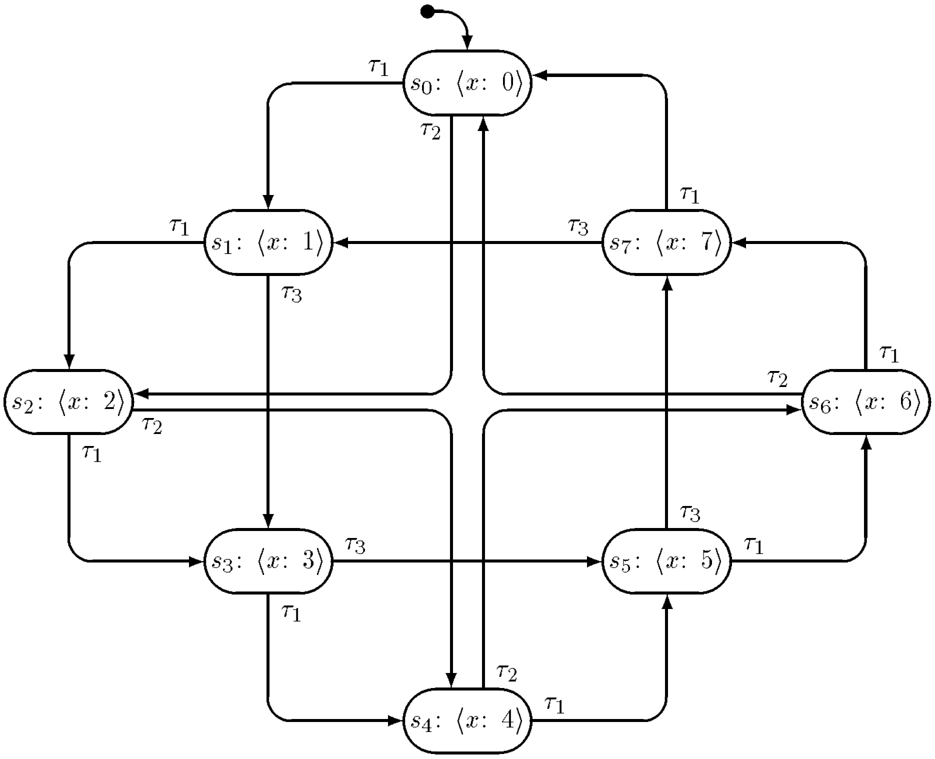


Fig. 3.54. State transition graph for INC2.

Let S be a subgraph of G_P and τ a transition. We say that τ is *taken in S* if S contains two states, s_1 and s_2, such that s_1 is connected by a directed edge to s_2 and $s_2 \in \tau(s_1)$. Transition τ is said to be *enabled* (*disabled*) *in S* if there exists an $s \in S$ such that τ is enabled (disabled) on s.

A subgraph S is called *just* if every just transition $\tau \in \mathcal{J}$ is either taken in S or is disabled in S. For example, subgraph $\{s_0, s_1, s_3, s_5, s_7\}$ of Fig. 3.54 is just. It is just towards τ_1 because $s_1 \in \tau_1(s_0)$. It is just towards τ_2 since τ_2 is disabled on s_1.

As previously observed, subgraph $\{s_0, s_2, s_4, s_6\}$ is not just towards τ_1, which is enabled on all of its states but not taken.

A subgraph S is called *compassionate* if every compassionate transition $\tau \in \mathcal{T}$ is either taken in S or is not enabled in S. *Not enabled* means that τ is disabled on all states $s \in S$.

For example, subgraph $\{s_0, s_2, s_4, s_6\}$ is compassionate but not just. It fulfills its obligations to the only compassionate transition τ_3, since τ_3 is disabled on all states in the subgraph. On the other hand, subgraph $\{s_0, s_1, s_2, s_4, s_6\}$ is just but not compassionate. This is because τ_3 is enabled on state s_1 of the subgraph but is not taken in the subgraph.

A subgraph is called *fair* if it is both just and compassionate. Thus, subgraph $\{s_0, s_1, s_3, s_5, s_7\}$ of Fig. 3.54 is fair.

Traversing Cycle

Let S be an SCS of G_P. A path $\sigma\colon [s_a, \ldots, s_b]$ in the subgraph S is called a *traversing cycle* of S if it satisfies the following requirements:

- $s_a = s_b$.
- For every $s \in S$, $s = s_i$ for some $i \in [a..b]$.
- For every pair of states $s, \widehat{s} \in S$ that are connected by an edge, there exists an $i \in [a..b{-}1]$ such that $s = s_i$ and $\widehat{s} = s_{i+1}$.

Thus, σ visits every state $s \in S$ at least once and traverses every edge connecting two states in S at least once.

Consider, for example, subgraph $\{s_0, s_1, s_2, s_4, s_6\}$. The shortest traversing cycle originating at s_0 is

$$s_0,\ s_1,\ s_2,\ s_4,\ s_6,\ s_0,\ s_2,\ s_4,\ s_6,\ s_0.$$

Note that we have to go twice around the subgraph in order to traverse both edge $\langle s_0, s_1\rangle$ and $\langle s_0, s_2\rangle$.

It is not too difficult to see that every SCS with n nodes has a traversing cycle whose size has a worst case complexity of $O(n^3)$.

The following claim establishes a correspondence between computations and fair subgraphs.

Claim 3.7 (fair SCS's and computations)

An SCS is the infinity set of a computation iff it is fair.

Justification Let σ be a computation and $S = \mathit{Inf}(\sigma)$ its infinity set. We will show that S is fair.

Consider a just transition $\tau \in \mathcal{J}$. Being a computation, σ is just towards τ which means that either τ is disabled on a state s and $s = s_i$ for infinitely many i's, or there exists a pair of states $\widehat{s} \in \tau(\widetilde{s})$ such that $\widetilde{s} = s_i$ and $\widehat{s} = s_{i+1}$ for infinitely many i's. In the first case, $s \in S$ and, therefore, τ is disabled in S. In the second case, both $\widetilde{s}, \widehat{s} \in S$ and τ is taken in S. We conclude that S is just.

Next, consider a compassionate transition $\tau \in \mathcal{T}$. As σ is compassionate towards τ, either τ is continually disabled from some point on, or τ is taken infinitely many times in σ. In the first case, τ is disabled on all states in $S = \mathit{Inf}(\sigma)$. In the second case, S contains a pair of states $\widehat{s} \in \tau(\widetilde{s})$, which means that τ is taken in S. We conclude that S is compassionate.

Consequently, $S = \mathit{Inf}(\sigma)$ is fair.

In the other direction, consider a fair strongly connected subgraph S. Let it have the traversing cycle

$$s_a,\ \ldots,\ s_b\ =\ s_a.$$

Since all states in G_P are accessible, there exists a path

$$s_0,\ s_1,\ \ldots,\ s_a$$

leading from some initial state s_0 to s_a.

Consider the state sequence

$$\sigma:\quad s_0,\ \ldots,\ s_a,\ [s_{a+1},\ \ldots,\ s_b]^{\omega}$$

which consists of the initial prefix $s_0, \ldots, s_a$ followed by endless repetition of the segment $s_{a+1}, \ldots, s_b$. It is not difficult to verify that σ is a computation of program P. In particular, the fact that σ satisfies the requirements of justice and compassion follows from the fairness of S and the property that $s_a, \ldots, s_b$ visits every state in S and traverses every edge in S at least once. ∎

For example, an INC2-computation σ such that $\mathit{Inf}(\sigma) = \{s_0, s_1, s_3, s_5, s_7\}$ can be given by

$$\sigma:\quad s_0,\ [s_1,\ s_3,\ s_5,\ s_7,\ s_1,\ s_3,\ s_5,\ s_7,\ s_0]^{\omega}.$$

Computation σ satisfies the fairness requirement by taking τ_1 and τ_3 infinitely many times. Just transition τ_2 is not taken at all, but this is allowed since it is infinitely often disabled — at all visits to states s_1, s_3, s_5, and s_7.

Subgraphs Corresponding to Counter-Examples

The correspondence between computations and fair SCS's raises the possibility that instead of examining uncountably many computations we may have to inspect only the fair subgraphs of G_P, whose number is finite.

A path $[s_a, \ldots, s_b]$ in graph G_P is called q-free, if s_i satisfies q for no $i \in [a..b]$. A state s is called *q-pending* if there exists a q-free path $[s_a, \ldots, s_b]$ such that $s_b = s$ and s_a satisfies p.

Assume that property $p \Rightarrow \diamondsuit\, q$ is P-valid. An important property of q-pending states is that if state s_i in a computation of P is q-pending it must be followed by a later state s_j, $j > i$, satisfying q. The reason for this is that these states characterize situations in which we have already observed an occurrence of p but not yet the corresponding q, promised by the P-validity of $p \Rightarrow \diamondsuit q$. In that respect, they generalize the family of p-states, since for all of these states there is still a pending occurrence of q.

Example Consider the response property

$$\underbrace{x = 0}_{p} \quad \Rightarrow \quad \diamondsuit \underbrace{5 \leq x \leq 6}_{q}$$

for system INC2 (Fig. 3.53). The set of q-pending states for this choice of p and q is

$$\{s_0,\ s_1,\ s_2,\ s_3,\ s_4\}.$$

States s_5 and s_6 are not included since they satisfy q. State s_7 is not included since every path connecting s_0 to s_7 must pass through either s_5 or s_6. ▟

When we want to check whether $p \Rightarrow \diamondsuit q$ is P-valid, we refer to a computation σ that does not satisfy $p \Rightarrow \diamondsuit q$ as a *counter-example*. It is not difficult to see that

σ is a counter-example iff
all states from a certain position on are q-pending iff
all states in $Inf(\sigma)$ are q-pending.

Since we are interested in counter-examples, we define the *Q-graph* of program P, denoted Q_P, to be the subgraph of G_P consisting of the q-pending states in G_P. The following algorithm describes an efficient calculation of Q_P, given G_P.

Algorithm Q-GRAPH (calculation of Q_P)

Let Q_P be a copy of G_P.

Remove from Q_P all q-states.

Remove from Q_P all states not reachable from a p-state.

Example Let us calculate Q_P for system INC2 for p: $x = 0$ and q: $5 \leq x \leq 6$. As a first step, we remove the q-states s_5 and s_6. Next we identify the p-reachable states by successive marking. This procedure marks states according to the sequence s_0, s_1, s_2, s_3, s_4. State s_7 remains unmarked and is removed. We remain with Q_P: $\{s_0, s_1, s_2, s_3, s_4\}$. ▟

The preceding discussion and sequence of claims lead to a conclusion that is summarized by the following proposition.

Claim 3.8 (checking for validity)

The graph Q_P contains a fair SCS iff the property $p \Rightarrow \diamondsuit q$ is not P-valid.

Justification Assume that Q_P contains a fair SCS S. Following the construction described in Claim 3.7, there exists a P-computation σ such that $Inf(\sigma) = S$. Since all but finitely many states in σ are q-pending, σ is a counter-example which violates $p \Rightarrow \diamondsuit q$.

In the other direction, if σ is a counter-example which violates $p \Rightarrow \diamondsuit q$ then $S = Inf(\sigma)$ is a fair SCS of G_P consisting of q-pending states. Consequently, it is also a subgraph of Q_P. ∎

Example Consider the property

$$\varphi: \quad \underbrace{x = 0}_{p} \quad \Rightarrow \quad \diamondsuit \underbrace{x = 4}_{q}$$

for program INC2 (Fig. 3.53). We wish to check whether property φ is P-valid. Calculation of Q_P shows that all states, excluding s_4, are q-pending. Consequently, by Claim 3.8, φ is valid over INC2 iff there does not exist a fair SCS of $Q_P = G_P - \{s_4\}$. Obviously, $\{s_0, s_1, s_3, s_5, s_7\}$ is such a subgraph. We conclude that $(x = 0) \Rightarrow \diamondsuit(x = 4)$ is not valid over INC2 and can actually point to the computation

$$\sigma: \quad [s_0,\ s_1,\ s_3,\ s_5,\ s_7]^\omega$$

as a counter-example.

On the other hand, let us consider another property

$$\psi: \quad \underbrace{x = 0}_{p} \quad \Rightarrow \quad \diamondsuit \underbrace{odd(x)}_{q} \ .$$

Calculating Q_P we obtain Q_P: $\{s_0, s_2, s_4, s_6\}$. The only SCS of Q_P is Q_P itself. However, Q_P is not fair since the just transition τ_1 is enabled on all states of Q_P but is not taken in Q_P.

We conclude that $(x = 0) \Rightarrow \diamondsuit\ odd(x)$ is valid over INC2. ▟

In principle, we can consider the problem solved. Since Q_P contains only finitely many subgraphs, and each can be checked for being strongly connected and fair in a finite number of steps, we can suggest an algorithm based on this procedure.

Unfortunately, the resulting algorithm will not be very efficient, since a graph of size n may contain an exponential number of SCS's. We therefore study a more efficient algorithm.

An Efficient Algorithm

Let S be a strongly connected graph. We wish to check whether S is fair or contains a fair SCS.

It is straightforward to check whether S is just. For each $\tau \in \mathcal{J}$, we check that τ is disabled in S or taken in S. If S is not just then no subgraph of S can be just. To see this, we observe that if some subgraph $S' \subseteq S$ treats τ justly, that is, τ is disabled or taken in S', then so does S. Consequently, if $S' \subseteq S$ is just then so is S and, equivalently, if S is unjust it cannot contain a just subgraph.

Assume we found S to be just. Next, we check whether S is compassionate, i.e., if it is the case that every compassionate transition $\tau \in \mathcal{C}$ is either taken in S or is disabled on all states of S. If S is compassionate, we conclude that it is fair. However, if S is not compassionate it may still, unlike the justice case, contain a compassionate subgraph.

For example, considering graph G_P of Fig. 3.54, the subgraph S: $\{s_0, s_1, s_2, s_4, s_6\}$ is not compassionate towards τ_3 which is enabled on s_1 but not taken in S. On the other hand, the subgraph of S, S': $\{s_0, s_2, s_4, s_6\}$ is compassionate since τ_3 is disabled on all of its states.

If S was found to be uncompassionate, there exists some transition $\tau_k \in \mathcal{C}$ which is not taken in S but is enabled on some states in S. Let $EN(\tau_k, S)$ denote the set of S-states on which τ_k is enabled. Clearly, no compassionate subgraph $S' \subseteq S$ can contain any of these states. Therefore, it is safe to remove $EN(\tau_k, S)$ from our consideration. Let $U = S - EN(\tau_k, S)$. We should check whether U contains a fair SCS. Removing the set $EN(\tau_k, S)$ from S may lead to a subgraph U which is no longer strongly connected. Therefore, before proceeding, we decompose U into a sequence of maximal strongly connected subgraphs (MSCS's) $U_1, \ldots, U_k$. There exist several efficient algorithms for such decomposition, and we refer by the generic name DECOMPOSE to any of them. We used such an algorithm already in Section 3.6 of the SAFETY book. We proceed to test recursively each U_i, $i = 1, \ldots, k$, checking whether it contains a fair SCS.

We summarize these steps in algorithm FAIR-SUB which accepts as input an SCS S and produces as an output a fair SCS $S' \subseteq S$ if one exits, or the empty set $\emptyset$ if S does not contain a fair SCS.

Algorithm FAIR-SUB (fair SCS's)

Recursive Algorithm *fair-sub*(S: subgraph) **returns** subgraph

If S is not just **return** $\emptyset$ — failure

If S is compassionate **return** S — S is fair

Otherwise, there exists $\tau \in \mathcal{C}$ such that τ is not taken in S and $EN(\tau, S) \neq \emptyset$

Let $U = S - EN(\tau, S)$

Decompose U into MSCS's $U_1, \ldots, U_k$

For each $i = 1, \ldots, k$ **do**
  if $V_i = \textit{fair-sub}(U_i) \neq \emptyset$ **then return** V_i — fair subgraph

end-for
return $\emptyset$. — no fair subgraphs.

Justification It is not difficult to see that if *fair-sub*(S) returns a nonempty subgraph $S' \subseteq S$ then S' is fair.

We concentrate on showing that if S contains a fair subgraph $S' \subseteq S$ then *fair-sub*(S) returns some fair subgraph $S'' \subseteq S$, which need not be identical to S'. We prove this by induction on $n = |S|$, the number of states in S.

For $|S| = n = 1$, S can contain a fair subgraph $S' \subseteq S$ only if $S' = S$ itself is fair. For this case, the first two tests in *fair-sub* will find that S is both just and compassionate and return it as the result.

Assume that the inductive claim is true for all subgraphs of size not exceeding n, and consider a subgraph S of size $n+1$ that contains a fair subgraph $S' \subseteq S$. If S itself is fair then *fair-sub* will return S after the first two tests. Otherwise, the algorithm will find S to be just but not compassionate and identify a compassionate transition $\tau \in \mathcal{C}$ which is not taken in S and such that $En(\tau, S) \neq \emptyset$. Subgraph S' being fair, it must treat τ with compassion, which implies that S' cannot contain any state $s \in En(\tau, S)$. Consequently, S' is contained in $U = S - En(\tau, S)$. Decomposing U into MSCS's $U_1, \ldots, U_k$, we observe that S', which is strongly connected, must be fully contained in U_j, for some $j \in [1..k]$. Since $|U_j| \leq |S - En(\sigma, S)| < n + 1$, the induction hypothesis holds for U_j which contains a fair subgraph. Consequently, if the algorithm does not return a nonempty result for some $i < j$, it will invoke *fair -sub*(U_i) and obtain, by the induction hypothesis, a nonempty fair subgraph V_i as a result. This V_i is the result of *fair -sub*(S). ∎

Example Consider the state-transition graph presented in Fig. 3.55.

This graph represents a finite-state fair transition system whose fairness sets are

$$\mathcal{J}\colon \{\tau_1\}$$
$$\mathcal{C}\colon \{\tau_2,\ \tau_3\}.$$

We wish to check whether the property

$$\underbrace{x = 0}_{p} \quad \Rightarrow \quad \Diamond \underbrace{x = 5}_{q}$$

is valid over system ROUND. As a first step, we identify the set of q-pending states, which is $Q_P\colon \{s_0, s_1, s_2, s_3, s_4\}$. Since Q_P is strongly connected we can apply Algorithm FAIR-SUB. We find that Q_P is just but not compassionate. In particular it does not treat τ_3 with compassion. Transition τ_3 is enabled on s_3 but not taken in Q_P. Consequently, we consider $U = Q_P - En(\tau, Q_P) = Q_P - \{s_3\} = \{s_0, s_1, s_2, s_4\}$. Subgraph U is not strongly connected and it is decomposed, using

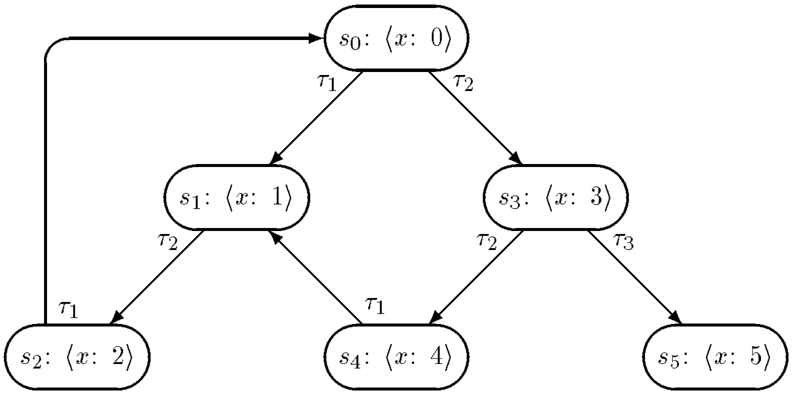


Fig. 3.55. System ROUND.

Algorithm DECOMPOSE, into the MSCS's U_1: $\{s_4\}$ and U_2: $\{s_0, s_1, s_2\}$.

In the next step, we apply FAIR-SUB to U_1: $\{s_4\}$. This subgraph is not just since τ_1 is enabled on all of its states but not taken in U_1. Consequently, *fair-sub*$(\{s_4\})$ returns $\emptyset$, denoting a failure to locate a fair SCS within U_1.

Applying FAIR-SUB to U_2: $\{s_0, s_1, s_2\}$ we observe that this subgraph is both just and compassionate and is therefore fair. It is compassionate towards τ_3 by having τ_3 disabled on all states of U_2.

We therefore conclude that property $x = 0 \Rightarrow \Diamond(x = 5)$ is not valid over system ROUND. A counter-example is provided by any sequence that endlessly traverses U_2. For example, σ: $[s_0, s_1, s_2]^\omega$ is a computation not satisfying $x = 0 \Rightarrow \Diamond(x = 5)$. ▟

We can now combine all the constituents into algorithm STATE-RESP for checking validity of a response formula over a finite state system.

Algorithm STATE-RESP (algorithmic verification of state formulas)

To check whether formula φ: $p \Rightarrow \Diamond\, q$ is valid over finite-state system P,

- Construct the state-transition graph G_P.
- Construct Q_P, the graph of q-pending states, using Algorithm Q-GRAPH.
- Decompose Q_P into MSCS's $S_1, \ldots, S_t$.
- For each $i = 1, \ldots, t$, apply *fair-sub*(S_i).
  If any of these applications returns with a nonempty result, formula φ is not P-valid. A counter-example can be constructed from the returned

fair subgraph.
If all applications return the empty set as result, formula φ is P-valid.

Example (checking accessibility for MUX-SEM)

Let us consider the finite-state program MUX-SEM of Fig. 3.56 (see also Fig. 3.1). We wish to model-check whether it satisfies the property of accessibility

$$\textbf{local } y\text{: } \textbf{integer where } y = 1$$

$$P_1 :: \left[\begin{array}{l} \ell_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_1\text{: } \textbf{noncritical} \\ \ell_2\text{: } \textbf{request } y \\ \ell_3\text{: } \textbf{critical} \\ \ell_4\text{: } \textbf{release } y \end{array}\right] \end{array}\right] \quad \| \quad P_2 :: \left[\begin{array}{l} m_0\text{: } \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1\text{: } \textbf{noncritical} \\ m_2\text{: } \textbf{request } y \\ m_3\text{: } \textbf{critical} \\ m_4\text{: } \textbf{release } y \end{array}\right] \end{array}\right]$$

Fig. 3.56. Program MUX-SEM (mutual exclusion by semaphores).

$$\underbrace{at\_\ell_2}_{p} \quad \Rightarrow \quad \diamondsuit\, \underbrace{at\_\ell_3}_{q} \ .$$

We already model-checked this program for the safety property of mutual exclusion in Section 2.6 of the SAFETY book. During this check we constructed the state-transition graph for MUX-SEM as presented in Fig. 2.27 of the SAFETY book.

Following algorithm STATE-RESP, we construct Q_P, the graph of q-pending states for p: at_ℓ_2 and q: at_ℓ_3. This graph is presented in Fig. 3.57.

Analyzing the strongly connected graph Q_P, we find that it is just but not compassionate, since the compassionate transition ℓ_2 is enabled on states s_0, s_1, and s_2. Removing these states, we are left with two SCS's $\{m_3\}$ and $\{m_4\}$. Neither of these subgraphs is just. Transition m_3 is enabled but not taken in $\{m_3\}$ and transition m_4 is enabled but not taken in $\{m_4\}$.

Thus Q_P contains no fair SCS. We conclude that $at\_\ell_2 \Rightarrow \diamondsuit\, at\_\ell_3$ is valid over program MUX-SEM. ▪

Example (checking accessibility for MUX-PET1)

Consider program MUX-PET1 of Fig. 3.58 (see also Fig. 1.13). In Section 2.6 of the SAFETY book we analyzed its safety property of mutual exclusion and constructed in Fig. 2.28 of the SAFETY book its state-transition graph.

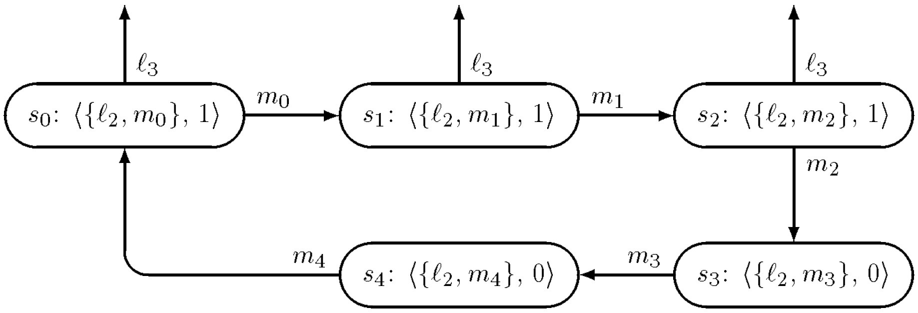


Fig. 3.57. Graph Q_P of q-pending states of program MUX-SEM.

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

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

$$||$$

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

Fig. 3.58. Program MUX-PET1 (Peterson's algorithm) — version 1.

Here we wish to check the response property of accessibility

$$\underbrace{at\_\ell_2}_{p} \quad \Rightarrow \quad \diamondsuit\, \underbrace{at\_\ell_4}_{q}$$

over program MUX-PET1.

In Fig. 3.59 we present graph Q_P of q-pending states for p: at_ℓ_2 and q: at_ℓ_4 as extracted from the graph of Fig. 2.28 of the SAFETY book. Each node in this graph is a triple (i, j, k) representing a state in which π: $\{\ell_i, m_j\}$ and s: k.

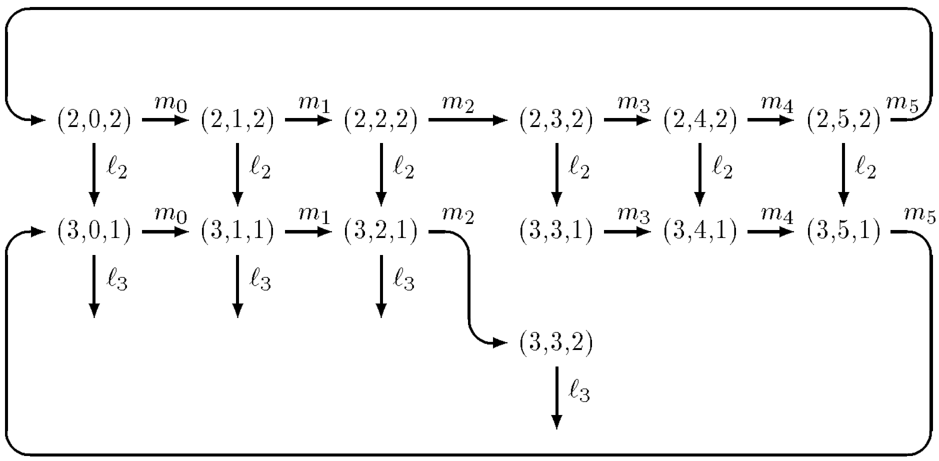


Fig. 3.59. Graph Q_P of q-pending states for MUX-PET1.

Decomposing Q_P into a sequence of MSCS's, we obtain the following list

$$S_1: \{(2,0,2),\ (2,1,2),\ (2,2,2),\ (2,3,2),\ (2,4,2),\ (2,5,2)\},$$
$$S_2: \{(3,3,1)\},\quad S_3: \{(3,4,1)\},\quad S_4: \{(3,5,1)\},\quad S_5: \{(3,0,1)\},$$
$$S_6: \{(3,1,1)\},\quad S_7: \{(3,2,1)\},\quad S_8: \{(3,3,2)\}.$$

None of these subgraphs is just. Subgraph S_1 is not just because ℓ_2 is enabled on all of its states and not taken in S_1. Subgraphs S_2, S_3, and S_4 are unjust because transitions m_3, m_4, and m_5 are enabled but not taken in them. Each of S_5, S_6, S_7, or S_8 is unjust because transition ℓ_3 is enabled but not taken in it.

We conclude that the accessibility property

$$at\_\ell_2 \ \Rightarrow\ \Diamond\, at\_\ell_4$$

is valid over program MUX-PET1. ▪

3.11 Finite-State Algorithmic Verification: General Response

Algorithm STATE-RESP can be applied for algorithmic verification of a formula

$$\psi:\quad p \ \Rightarrow\ \diamondsuit\, q$$

over a finite-state program P for the case that p and q are assertions. Here, we consider the more general case that p and q are past formulas. Algorithmic verification of general response formulas is based on extending algorithm STATE-RESP to analyze the reachable atoms graph $\mathcal{A}_{(P,\psi)}$ instead of the state-transition graph G_P.

Recall from Section 4.10 of the SAFETY book that $\mathcal{A}_{(P,\psi)}$ is a graph of atoms where each atom $\alpha \in \mathcal{A}$ has the form α: $\langle s;\, \varphi_1^{b_1}, \ldots, \varphi_n^{b_n}\rangle$ consisting of a state s and assignment of truth-values to each past formula φ_i belonging to the past-closure Φ_ψ of formula ψ.

Example Consider system INC2 presented in Fig. 3.54 and the response property

$$\psi:\quad \underbrace{x = 0}_{p} \ \ \Rightarrow\ \ \diamondsuit\Big(\underbrace{even(x)\ \wedge\ \ominus\, odd(x)}_{q}\Big)$$

claiming that any state satisfying $x = 0$ is followed by an even state (a state in which x is even) which is immediately preceded by an odd state.

The past-closure Φ_ψ for this case is

$$\Phi_\psi:\quad \big\{\ominus\, odd(x)\big\}$$

since $\ominus\, odd(x)$ is the only subformula of ψ with a principal past operator. Consequently, we construct in Fig. 3.60 the graph $\mathcal{A}_{(P,\psi)}$ of reachable atoms, using the algorithm presented in Section 4.10 of the SAFETY book.

Given an atom $\alpha \in \mathcal{A}_{(P,\psi)}$, we can evaluate formulas p and q on α since α provides an interpretation (T or F) to each subformula of p or q. Consequently, we refer to an atom α as a *p-atom* or a *q-atom* if the evaluation of p or q on α yields T. In analogy with the assertional case, we say that a path $\alpha_a, \ldots, \alpha_b$ in $\mathcal{A}_{(P,\psi)}$ is q-free if no α_i, $i \in [a, b]$ is a q-atom. An atom $\alpha \in \mathcal{A}_{(P,\psi)}$ is called *q-pending* if there exists a q-free atom path $\alpha_a, \ldots, \alpha_b = \alpha$ such that α_a is a p-atom. We define $Q_{\mathcal{A}}$ to be the subgraph of all q-pending atoms in $\mathcal{A}_{(P,\psi)}$. It can be calculated by an algorithm analogous to Q-GRAPH replacing all references to states, p-states, and q-states, by atoms, p-atoms, and q-atoms.

Example In Fig. 3.61 we present $Q_{\mathcal{A}}$ the subgraph of q-pending atoms for system INC2, p: $x = 0$, and q: $even(x) \wedge \ominus\, odd(x)$. ▟

The notions of just, compassionate, and fair subgraphs extend naturally to subgraphs of atoms.

Algorithm GENERAL-RESP generalizes algorithm STATE-RESP to the case that p and q are past formulas.

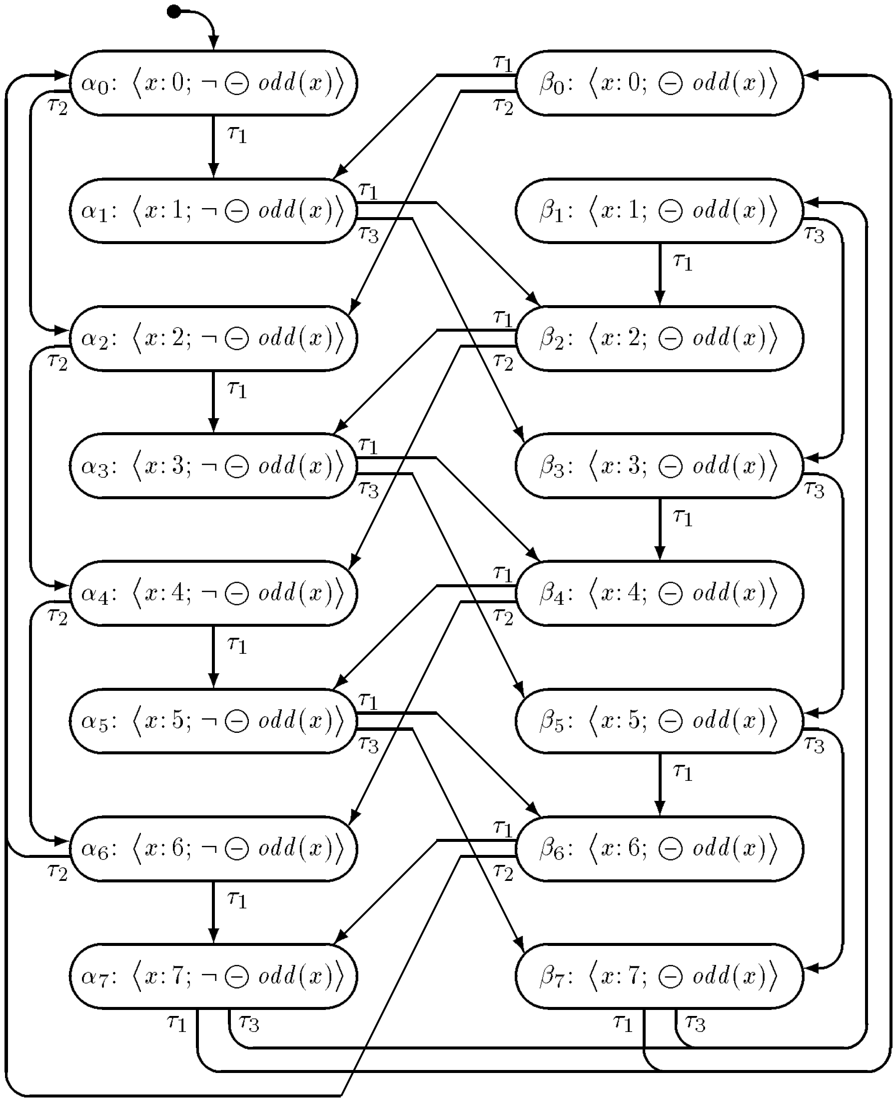


Fig. 3.60. Graph $\mathcal{A}_{(P,\psi)}$ of reachable atoms for INC2.

Algorithm GENERAL-RESP (algorithmic verification of general response formulas)

To check whether formula ψ: $p \Rightarrow \diamondsuit\, q$, where p and q are past formulas, is valid over a finite-state system P.

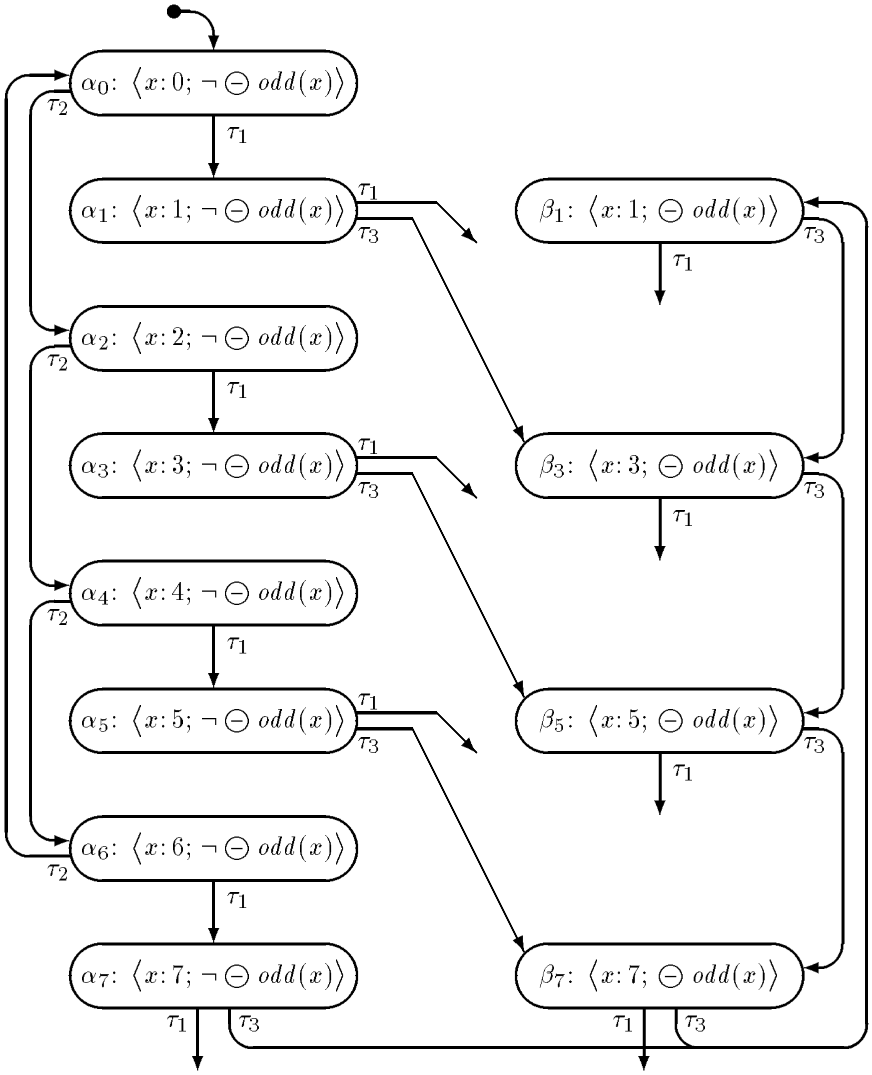


Fig. 3.61. Graph of q-pending atoms for INC2.

- Construct $\mathcal{A}_{(P,\psi)}$ the graph of reachable atoms for P and ψ.
- Calculate $Q_{\mathcal{A}}$ the graph of q-pending atoms.
- Decompose $Q_{\mathcal{A}}$ into MSCS's $A_1, \ldots, A_t$.
- For each $i = 1, \ldots, t$, apply *fair-sub* to A_i.

If any of the applications returns with a nonempty result, then formula ψ is not P-valid. A counter-example can be constructed from the returned fair subgraph.
If all applications return the empty set as a result, then formula ψ is P-valid.

Example Decomposing graph $Q_{\mathcal{A}}$ of Fig. 3.61 into MSCS's, we obtain the following list of subgraphs

$$A_1\colon \{\alpha_0, \alpha_2, \alpha_4, \alpha_6\}, \quad A_2\colon \{\alpha_1\}, \quad A_3\colon \{\alpha_3\}, \quad A_4\colon \{\alpha_5\},$$
$$A_5\colon \{\alpha_7\}, \quad A_6\colon \{\beta_1, \beta_3, \beta_5, \beta_7\}.$$

All of these subgraphs are unjust towards τ_1 since this transition is always enabled but not taken in any of these subgraphs.

We conclude that $Q_{\mathcal{A}}$ contains no fair SCS and, therefore, $x = 0 \Rightarrow \diamondsuit\big(even(x) \land \ominus\, odd(x)\big)$ is valid over system INC2. ▪

The justification of Algorithm GENERAL RESP follows precisely the same lines as the sequence of claims justifying its state counterpart, Algorithm STATE-RESP.

Problems

Problem 3.1 (program PROD-CONS without compassion) page 159

Consider program PROD-CONS (Fig. 3.5) under the assumption that the transitions corresponding to the *request* statements are taken to be just, and the compassion set is empty. Show that, in spite of this weakening assumption, the program still satisfies the response property

$$at\_\ell_2 \quad \Rightarrow \quad \diamondsuit\, at\_\ell_4 \ .$$

Problem 3.2 (invariant for program ALTER) page 176

Prove the invariant

$$Y[1..j-1] \ = \ X[1..j-1]$$

for program ALTER of Fig. 3.18. In your proof you may use invariants I_1–I_6 presented in the text. Please verify any of the invariants you use.

Problem 3.3 (communal accessibility for program INADEQUATE) page 200

Prove communal accessibility $N_2 > 0 \Rightarrow \diamondsuit(N_3 > 0)$ for program INADEQUATE (Fig. 3.37).

Problem 3.4 (using the completeness construction to verify MUX-SEM) page 224

Use the construction outlined in the completeness proof of Section 3.8 to verify accessibility for program MUX-SEM of Fig. 3.1.