Chư ơ ng 10

Soạn thông số kỹ thuật

Các hệ thống thư ờng đư ợc mô tả theo các thành phần của chúng. Trong các thông số kỹ thuật mà chúng tôi đã viết cho đến nay, các thành phần đã đư ợc biểu diễn dư ới dạng các phần riêng biệt của hành động ở trạng thái tiếp theo. Ví dụ: hệ thống FIFO đư ợc mô tả ở trang 35 đư ợc chỉ định trong mô-đun InnerFIFO ở trang 38 bằng cách biểu thị ba thành phần với các phần tách rời sau của hành động trạng thái tiếp theo:

Ngư ời gửi: tin nhắn Tin nhắn: SSend(tin nhắn)

Bộ đệm: BufRcv BufSend

Ngư ời nhận: Rrcv

Trong chư ơ ng này, chúng ta tìm hiểu cách xác định các thành phần riêng biệt và kết hợp các đặc tả của chúng để tạo thành một đặc tả hệ thống duy nhất. Hầu hết thời gian, không có ích gì khi làm điều này. Hai cách viết thông số kỹ thuật chỉ khác nhau vài dòng—một sự khác biệt nhỏ trong thông số kỹ thuật hàng trăm hoặc hàng nghìn dòng. Tuy nhiên, bạn có thể gặp phải tình huống trong đó tốt hơ n là chỉ định một hệ thống làm bố cục.

Đầu tiên, chúng ta phải hiểu ý nghĩa của việc soạn thảo thông số kỹ thuật. Chúng tôi thư ờng nói rằng công thức TLA chỉ định hành vi đúng đắn của hệ thống. Tuy nhiên, như đã giải thích ở Phần 2.3 (trang 18), một hành vi thực sự thể hiện một lịch sử có thể có của toàn bộ vũ trụ chứ không chỉ của hệ thống. Vì vậy, sẽ chính xác hơ n khi nói rằng công thức TLA chỉ định một vũ trụ trong đó hệ thống hoạt động chính xác. Xây dựng một hệ thống thực hiện đặc tả F có nghĩa là xây dựng vũ trụ sao cho nó thỏa mãn F. (May mắn thay, tính đúng đắn của hệ thống chỉ phụ thuộc vào hành vi của một phần rất nhỏ của vũ trụ và đó là phần duy nhất chúng ta phải xây dựng.) hai hệ thống có thông số kỹ thuật là F và G có nghĩa là làm cho vũ trụ thỏa mãn cả F và G, điều này cũng giống như làm cho vũ trụ thỏa mãn F G. Do đó, đặc điểm kỹ thuật về thành phần của hai hệ thống là sự kết hợp các đặc điểm kỹ thuật của chúng.

Do đó, viết một đặc tả như là sự kết hợp của các thành phần của nó có nghĩa là viết đặc tả đó như một sự kết hợp, mỗi liên kết của nó có thể được xem như là đặc tả của một thành phần. Mặc dù ý tưởng cơ bản rất đơn giản nhưng các chi tiết không phải lúc nào cũng rõ ràng. Để đơn giản hóa việc trình bày, tôi bắt đầu bằng cách chỉ xem xét các đặc tính an toàn, bỏ qua sự sống động và phần lớn bỏ qua việc che giấu. Sự sống và sự ẩn náu được thảo luận trong Phần 10.6.

10.1 Soạn thảo hai thông số kỹ thuật

Hãy quay trở lại với đồng hồ giờ đơn giản một lần nữa, không có yêu cầu về tính sống động hoặc thời gian thực. Trong Chương 2, chúng ta đã chỉ định một chiếc đồng hồ như vậy có màn hình được biểu thị bằng biến hr . Chúng ta có thể viết đặc tả đó như

```
(giờ  1 . . 12)  [HCN (giờ )]giờ
```

trong đó HCN được xác định bởi

```
HCN (h)  = h = (h % 12) + 1
```

Bây giờ hãy viết đặc tả TwoClocks của một hệ thống bao gồm hai đồng hồ giờ riêng biệt, có màn hình được biểu thị bằng các biến x và y. (Hai đồng hồ không được đồng bộ hóa và hoàn toàn độc lập với nhau.) Chúng ta chỉ có thể định nghĩa TwoClocks là sự kết hợp của hai thông số kỹ thuật đồng hồ

```
Hai đồng hồ  =  (x  1 . . 12)  [HCN (x )]x  (y
                    1 . . 12)  [HCN (y)]y
```

Phép tính sau đây cho thấy cách chúng ta có thể viết lại TwoClocks ở dạng thông thường dưới dạng đặc tả "nguyên khối" với một hành động trạng thái tiếp theo duy nhất:[1]

```
Hai đồng hồ
  ≡  (x  1 . . 12)  (y  1 . . 12)
       [HCN (x )]x    [HCN (y)]y
  ≡  (x  1 . . 12)  (y  1 . . 12)        Bởi vì  (F  G) ≡ ( F)  ( G).
       ( [HCN (x )]x  [HCN (y)]y )
  ≡  (x  1 . . 12)  (y  1 . . 12)        Theo định nghĩa của [. . .]x và [. . .]y .
       (  HN (x )  x = x
          HCN (y)  y = y )
```

[1] Tính toán này không chính thức vì nó chứa các công thức không phải là TLA hợp pháp—cụ thể là các công thức có dạng A trong đó A là một hành động không có dạng cú pháp [B]v . Tuy nhiên, nó có thể được thực hiện một cách nghiêm ngặt.

```
≡   (x   1 . . 12)   (y   1 . . 12)
        (   HCN (x )   HCN (y)
            HCN (x )   (y = y)
            HCN (y)   ( x = x )
            (x = x )   (y = y) )
```

Bởi vì:

```
A1          A1   B1
A2     ≡    A1   B2
B1          A2   B1
B2          A2   B2
```

```
≡   (x   1 . . 12)   (y   1 . . 12)
        [   HCN (x )   HCN (y)
            HCN (x )   (y = y)
            HCN (y)   ( x = x ) ]x,      y
```

Theo định nghĩa của [. . .]x, y .

Như vậy, TwoClocks tư ơ ng đư ơ ng với Init [TCNxt]x, action y trạng thái tiếp theo ở đâu TCNxt là

```
TCnxt   =   HCN (x )   HCN (y)   HCN
              (x )   (y = y)   HCN
              (y)   (x = x )
```

Hành động ở trạng thái tiếp theo này khác với hành động mà chúng ta quen viết vì HCN (x) HCN (y) không liên tục, biểu thị sự tiến lên đồng thời của hai màn hình. Trong các thông số kỹ thuật mà chúng tôi đã viết cho đến nay, các thành phần khác nhau không bao giờ hoạt động đồng thời.

Cho đến bây giờ, chúng ta vẫn đang viết cái đư ợc gọi là các đặc tả xen kẽ. Trong đặc tả kỹ thuật xen kẽ, mỗi bư ớc chỉ thể hiện một hoạt động của một thành phần. Ví dụ: trong đặc tả FIFO của chúng tôi, một bư ớc (không bị lặp) thể hiện hành động của ngư ời gửi, bộ đệm hoặc ngư ời nhận. Để có một thuật ngữ tốt hơ n, chúng tôi mô tả là không xen kẽ một đặc tả, giống như TwoClocks, cho phép hai thành phần thực hiện hành động đồng thời.

Giả sử chúng ta muốn viết một đặc tả đan xen của hệ thống hai đồng hồ dư ới dạng kết hợp của hai đặc tả thành phần. Có một cách là thay các tác dụng ở trạng thái tiếp theo HCN (x) và HCN (y) của hai thành phần đó bằng hai tác dụng HNx và HCNy sao cho khi thực hiện phép tính tư ơ ng tự như trên, ta đư ợc:

```
(x   1 . . 12)   [HCNx ]x          ≡   (x   1 . . 12)   (y   1 . . 12)
(y   1 . . 12)   [HCNy]y                   [   HCNx   (y = y)
                                            HCNy   (x = x ) ]x,      y
```

Từ tính toán trên, chúng ta thấy rằng sự tư ơ ng đư ơ ng này xảy ra nếu thỏa mãn ba điều kiện sau: (i) HCNx suy ra HCN (x), (ii) HNy suy ra HCN (y), và (iii) HCNx HCNy suy ra x = x hoặc y = y. (Điều kiện (iii) ngụ ý rằng HCNx HCy phân biệt của tác dụng ở trạng thái tiếp theo đư ợc gộp bởi một trong các phân ly HNx (y = y) và HCNy (x = x).) Cách thông thư ờng

Việc thỏa mãn các điều kiện này là để hành động ở trạng thái tiếp theo của mỗi đồng hồ khẳng định rằng màn hình của đồng hồ kia không thay đổi. Chúng tôi làm điều này bằng cách xác định

HNx = HCN (x) (y = y) HCy = HCN (y) (x = x)

Một cách khác để viết một đặc tả xen kẽ chỉ đơn giản là không cho phép những thay đổi đồng thời đối với cả hai màn hình đồng hồ. Chúng ta có thể làm điều này bằng cách lấy công thức làm đặc tả

TwoClocks [(x = x) (y = y)]x, y

Liên từ thứ hai khẳng định rằng bất kỳ bước nào cũng phải giữ nguyên x hoặc y (hoặc cả hai).

Mọi thứ chúng tôi đã làm cho hệ thống hai đồng hồ đều khái quát cho bất kỳ hệ thống nào gồm hai thành phần. Tính toán tương tự như trên cho thấy nếu

(v1 = v1) (v2 = v2) ≡ (v = v) Điều này khẳng định rằng v không thay đổi nếu cả v1 và v2 đều như vậy.

sau đó

$$\begin{array}{lll} I1 & [N1]v1 & \equiv \\ I2 & [N2]v2 & \end{array} \qquad \begin{array}{l} I1 \quad I2 \\ [\quad N1 \quad N2 \\ \quad N1 \quad (v2 = v2) \\ \quad N2 \quad (v1 = v1)\ ]v \end{array} \tag{10.1}$$

đối với bất kỳ vị từ trạng thái I1 và I2 nào và mọi hành động N1 và N2. Vế trái của sự tương đương này biểu thị sự hợp thành của hai đặc tả thành phần nếu vk là một bộ chứa các biến mô tả thành phần k, với k = 1, 2 và v là bộ của tất cả các biến.

Các công thức tương đương trong (10.1) biểu thị đặc tính kỹ thuật xen kẽ nếu phân tách đầu tiên trong hành động trạng thái tiếp theo ở vế phải là dư thừa nên có thể loại bỏ nó. Đây là trường hợp nếu N1 N2 ngụ ý rằng v1 hoặc v2 không thay đổi.
Cách thông thường để đảm bảo rằng điều kiện này được thỏa mãn là xác định từng Nk sao cho nó ngụ ý rằng bộ dữ liệu của thành phần khác không thay đổi. Một cách khác để có được đặc tả xen kẽ là kết hợp công thức [(v1 = v1) (v2 = v2)]v .

10.2 Soạn nhiều thông số kỹ thuật

Chúng ta có thể khái quát hóa (10.1) thành thành phần của bất kỳ tập hợp thành phần C nào. Bởi vì lượng hóa phổ quát khái quát hóa sự kết hợp, quy tắc sau đây là sự khái quát hóa của (10.1):

Quy tắc thành phần Cho mọi tập hợp C ,

(k C : vk = vk) ≡ (v = v) Điều này khẳng định rằng v không thay đổi nếu tất cả các vk đều như vậy.

sau đó

(k C : Ik [Nk]vk) ≡

k C : Ik

k C : Nk (i C \ {k} : vi = vi)

i, j C : (i = j) Ni Nj Fij v

đối với một số hành động Fij .

Điểm gián đoạn thứ hai của hành động ở trạng thái tiếp theo là dư thừa, và chúng ta có một đặc tả xen kẽ, nếu mỗi Ni ngụ ý rằng vj không thay đổi, với mọi j = i. Tuy nhiên, để giữ được điều này, Ni phải đề cập đến vj cho các thành phần j ngoài i. Bạn có thể phản đối cách tiếp cận này—hoặc trên cơ sở triết học, vì bạn cảm thấy rằng đặc tả của một thành phần không nên đề cập đến trạng thái của thành phần khác hoặc vì việc đề cập đến các biến của thành phần khác sẽ làm phức tạp thêm đặc tả của thành phần đó. Một cách tiếp cận khác chỉ đơn giản là khẳng định sự xen kẽ. Bạn có thể làm điều này bằng cách kết hợp công thức sau, công thức này cho biết mọi i và j có i = j: không có vj , bước nào thay đổi cả vi và

[k C : i C \ {k} : vi = vi]v

Liên kết này có thể được xem như một điều kiện toàn cục, không gắn với bất kỳ đặc tả nào của thành phần.

Để vk bên trái của kết luận của Quy tắc tổng hợp thể hiện sự hợp thành của các thành phần riêng biệt, vk không cần phải bao gồm các biến riêng biệt. Chúng có thể chứa các "phần" khác nhau của cùng một biến mô tả các thành phần khác nhau. Ví dụ: hệ thống của chúng tôi có thể bao gồm một bộ Đồng hồ gồm các đồng hồ riêng biệt, độc lập, trong đó màn hình của đồng hồ k được mô tả bằng giá trị hr [k]. Khi đó vk sẽ bằng hr [k]. Thật dễ dàng để chỉ định một dãy đồng hồ như vậy làm thành phần. Sử dụng định nghĩa HCN ở trang 136 ở trên, chúng ta có thể viết thông số kỹ thuật như sau:

(10.2) Mảng đồng hồ = k Đồng hồ : (hr [k] 1 . . 12) [HCN (hr [k])]hr[k]

Đây là đặc tính không xen kẽ vì nó cho phép các bước đồng thời của các đồng hồ khác nhau.

Giả sử chúng ta muốn sử dụng Quy tắc Thành phần để thể hiện ClockArray dưới dạng đặc tả nguyên khối. Chúng ta sẽ thay thế v bằng gì? Ý nghĩ đầu tiên của chúng ta là thay hr cho v. Tuy nhiên, giả thuyết của quy tắc yêu cầu v phải được giữ nguyên nếu hr [k] không thay đổi, với mọi k Đồng hồ. Tuy nhiên, như đã giải thích ở Phần 6.5 trên trang 72, việc chỉ định giá trị của hr [k] cho mọi k Clock không chỉ định giá trị của hr . Nó thậm chí không ngụ ý rằng hr là một hàm. Chúng ta phải thay thế v bằng hàm hrfcn được xác định bởi

(10.3) hrfcn = [k Đồng hồ giờ [k]]

Hàm hrfcn bằng hr iff hr là một hàm có miền Clock. Công thức ClockArray không hàm ý rằng hr luôn là một hàm. Nó chỉ định các giá trị có thể có của hr [k], cho tất cả k Đồng hồ, nhưng nó không chỉ định giá trị của hr. Ngay cả khi chúng ta thay đổi điều kiện ban đầu để ngụ ý rằng hr ban đầu là một hàm có miền Clock, thì công thức ClockArray sẽ không ngụ ý rằng hr luôn là một hàm. Ví dụ: nó vẫn cho phép các bước "nói lắp" giữ nguyên mỗi giờ [k] nhưng thay đổi giờ theo những cách không xác định.

Chúng ta có thể thích viết một đặc tả trong đó hr là một hàm với miền Clock. Một cách để làm điều này là kết hợp với đặc tả công thức IsFcnOn(hr , Clock), trong đó IsFcnOn(hr , Clock) khẳng định rằng hr là một hàm tùy ý với miền Clock. Toán tử IsFcnOn được xác định bởi IsFcnOn(f , S) = f = [x S f [x]]

Chúng ta có thể xem công thức IsFcnOn(hr , Clock) dưới dạng ràng buộc toàn cục đối với hr , trong khi giá trị của hr [k] cho mỗi thành phần k được mô tả theo thông số kỹ thuật của thành phần đó.

Bây giờ, giả sử chúng ta muốn viết một đặc tả đan xen của mảng đồng hồ dưới dạng thành phần đặc tả của từng đồng hồ. Nói chung, sự kết hợp trong Quy tắc Thành phần là một đặc tả xen kẽ nếu mỗi Nk ngụ ý rằng vi không thay đổi, với mọi i = k. Vì vậy, chúng ta muốn hành động ở trạng thái tiếp theo Nk của đồng hồ k ngụ ý rằng hr [i] không thay đổi đối với mọi đồng hồ i khác với k.
Cách rõ ràng nhất để làm điều này là xác định Nk bằng

giờ [k] = (hr [k] % 12) + 1
i Đồng hồ \ {k} : giờ [i] = giờ [i]

Chúng ta có thể diễn đạt công thức này gọn gàng hơn bằng cách sử dụng cấu trúc ngoại trừ. Cấu trúc ngoại trừ này chỉ áp dụng cho các hàm, vì vậy chúng ta phải chọn có yêu cầu hr là một hàm hay không. Nếu hr là một hàm thì ta có thể cho Nk bằng hr (10.4) = [hr ngoại trừ ![k] = (hr [k]

% 12) + 1]

struct được giải thích trong Phần 5.2 trên trang 48.

Như đã lưu ý ở trên, chúng ta có thể đảm bảo rằng hr là một hàm bằng cách kết hợp công thức IsFcnOn(hr , Clock) với thông số kỹ thuật. Một cách khác là xác định hàm trạng thái hrfcn theo (10.3) ở trang trước và đặt N (k) bằng

hrfcn = [hrfcn ngoại trừ ![k] = (hr [k] % 12) + 1]

Thông số kỹ thuật chỉ là một công thức toán học; như chúng ta đã thấy trước đây, thường có nhiều cách viết công thức tương đương. Cái nào bạn chọn thường là vấn đề về hương vị.

10.3 FIFO

Bây giờ chúng ta hãy xác định FIFO, được mô tả trong Chương 4, là thành phần của ba thành phần của nó—Người gửi, Bộ đệm và Người nhận. Chúng tôi bắt đầu với

đặc tả bên trong, trong đó biến q xuất hiện - nghĩa là q không bị ẩn.
Đầu tiên, chúng tôi quyết định phần nào của trạng thái mô tả từng thành phần. Các biến vào và ra là các kênh. Hãy nhớ lại rằng mô-đun Kênh (trang 30) chỉ định kênh chan là bản ghi với các thành phần val, rdy và ack. Hành động Gửi, gửi một giá trị, sửa đổi các thành phần val và rdy; Hành động Rcv nhận một giá trị sẽ sửa đổi thành phần ack. Vì vậy, trạng thái của các thành phần được mô tả bằng các hàm trạng thái sau:

Người gửi: in.val, in.rdy

Đệm: in.ack, q, out.val, out.rdy

Người nhận: out.ack

Thật không may, chúng tôi không thể sử dụng lại các định nghĩa từ mô-đun InnerFIFO ở trang 38 vì lý do sau. Biến q, được ẩn trong đặc tả cuối cùng, là một phần trạng thái bên trong của thành phần Bộ đệm. Do đó, nó không nên xuất hiện trong thông số kỹ thuật của thành phần Người gửi hoặc Người nhận.
Các hành động của Người gửi và Người nhận được xác định trong mô-đun InnerFIFO đều đề cập đến q, vì vậy chúng tôi không thể sử dụng chúng. Do đó chúng tôi sẽ không bận tâm đến việc sử dụng lại mô-đun đó. Tuy nhiên, thay vì bắt đầu hoàn toàn từ đầu, chúng ta có thể sử dụng các hành động Gửi và Rcv từ mô-đun Kênh trên trang 30 để mô tả các thay đổi vào và ra.

Hãy viết một đặc tả không xen kẽ. Khi đó, các hành động ở trạng thái tiếp theo của các thành phần cũng giống như các hành động phân tách tương ứng của hành động Tiếp theo trong mô-đun InnerFIFO, ngoại trừ việc chúng không đề cập đến các phần trạng thái thuộc về các thành phần khác. Chúng chứa các hành động Gửi và Rcv, được khởi tạo từ mô-đun Kênh, sử dụng cấu trúc ngoại trừ. Như đã lưu ý ở trên, chúng ta chỉ có thể áp dụng ngoại trừ cho các chức năng—và cho các bản ghi nằm trong Mục 5.2 về chức năng. Do đó, chúng tôi thêm vào đặc tả của mình liên từ

trang 48 giải thích tại sao bản ghi lại là chức năng.

(IsChannel(vào) IsChannel(out))

trong đó IsChannel(c) xác nhận rằng c là một kênh—đó là bản ghi có các trường val, ack và rdy. Vì bản ghi có các trường val, ack và rdy là một hàm có miền là {“val”, “ack”, “rdy”} nên chúng ta có thể định nghĩa IsChannel(c) bằng IsFcnOn(c, {“val” , “ack”, “rdy”}). Tuy nhiên, thật dễ dàng để xác định trực tiếp công thức IsChannel(c) bằng

IsChannel(c) = c = [ack c.ack, val c.val, rdy c.rdy]

Khi viết đặc tả này, chúng ta gặp phải vấn đề tương tự như trong đặc tả FIFO ban đầu của chúng ta là đưa biến q và sau đó ẩn nó. Trong Chương 4, chúng tôi đã giải quyết vấn đề này bằng cách giới thiệu q trong một mô-đun InnerFIFO riêng biệt, được khởi tạo bởi mô-đun FIFO xác định đặc tả cuối cùng. Về cơ bản, chúng tôi thực hiện điều tương tự ở đây, ngoại trừ việc chúng tôi giới thiệu q trong mô hình con

thay vì trong một mô-đun hoàn toàn riêng biệt. Tất cả các ký hiệu được khai báo và xác định tại thời điểm mô hình con xuất hiện đều có thể được sử dụng trong đó. Bản thân mô-đun con có thể được khởi tạo trong mô-đun chứa ở bất kỳ đâu sau khi nó xuất hiện.
(Các mô-đun con được sử dụng trong thông số kỹ thuật RealTimeHourClock và RTMemory ở trang 121 và 126 của Chương 9.)

Có một vấn đề nhỏ cần được giải quyết trước khi chúng ta có thể viết một đặc tả tổng hợp của FIFO—làm thế nào để chỉ định các vị từ ban đầu. Điều hợp lý là vị từ ban đầu trong đặc tả của từng thành phần sẽ chỉ định các giá trị ban đầu của phần trạng thái của nó. Tuy nhiên, điều kiện ban đầu bao gồm các yêu cầu in.ack = in.rdy và out.ack = out.rdy, mỗi yêu cầu liên quan đến trạng thái ban đầu của hai thành phần khác nhau. (Các yêu cầu này được nêu trong mô-đun InnerFIFO bằng các liên từ InChan !Init và OutChan !Init của tiền từ ban đầu Init.) Có ba cách thể hiện một yêu cầu liên quan đến các trạng thái ban đầu của nhiều thành phần:

- Khẳng định điều đó với điều kiện ban đầu của tất cả các thành phần. Mặc dù sym-
  số liệu, điều này có vẻ dư thừa không cần thiết.
- Tự ý gán yêu cầu cho một trong các thành phần. Điều này gợi ý một cách trực quan rằng chúng ta đang giao cho thành phần đó trách nhiệm đảm bảo rằng yêu cầu được đáp ứng.
- Khẳng định yêu cầu dưới dạng kết hợp tách biệt với một trong các thông số kỹ thuật của thành phần. Điều này gợi ý một cách trực quan rằng đó là một giả định về cách các thành phần được kết hợp với nhau chứ không phải là yêu cầu của một trong hai thành phần.

Khi chúng ta viết một đặc tả hệ thống mở, như được mô tả trong Phần 10.7 dưới đây, những gợi ý trực quan của hai cách tiếp cận cuối cùng có thể được chuyển thành các yêu cầu hình thức. Tôi đã thực hiện cách tiếp cận cuối cùng và thêm

```
(in.ack = in.rdy)    (out.ack = out.rdy)
```

như một điều kiện riêng biệt. Thông số kỹ thuật đầy đủ có trong mô-đun CompositeFIFO của Hình 10.1 trên trang tiếp theo. Thông số công thức của mô-đun này là thông số kỹ thuật không xen kẽ; ví dụ: nó cho phép một bước duy nhất vừa là bước InChan !Send (người gửi gửi một giá trị) vừa là bước OutChan !Rcv (người nhận xác nhận một giá trị). Do đó, nó không tương đương với thông số kỹ thuật xen kẽ Spec của mô-đun FIFO ở trang 41, không cho phép thực hiện bước như vậy.

10.4 Thành phần với trạng thái chia sẻ

Cho đến nay, chúng ta đang xem xét các thành phần trạng thái rời rạc—những thành phần trong đó các thành phần được thể hiện bằng các phần rời rạc của trạng thái, và một thành phần

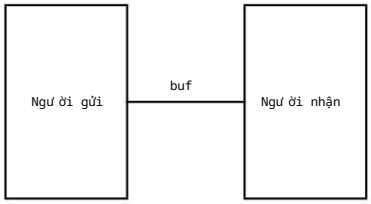
```
mô-đun CompositeFIFO
mở rộng Naturals, Chuỗi hằng
số Thông báo biến
vào, ra
InChan   = instance Channel with Data    Message, chan    in
OutChan   = instance Channel with Data    Message, chan    out
SenderInit   = (in.rdy    {0, 1})    (in.val    Tin nhắn)                    của ngư ời gửi
Ngư ời gửi   = SenderInit      [    msg    Tin nhắn : InChan !Send(msg) ]in.val, in.rdy    sự chỉ rõ.
  mô-đun InnerBuf
  biến q
  BufferInit   =    in.ack    {0, 1}                                          Nội bộ của bộ đệm
                      q                                                      đặc điểm kỹ thuật, với q
                  =    (out.rdy    {0, 1})    (out.val    Tin nhắn)          hiển thị.
  BufRcv   =    InChan !Rcv    q
                 = Nối(q, in.val)    không
                 thay đổi out.val, out.rdy BufSend
    =    q =    OutChan !
                 Send(Head(q))    q = Tail(q)
                    in.ack không
                 thay đổi
  InnerBuffer   = BufferInit      [BufRcv    BufSend]in.ack,      q, out.val, out.rdy
Buf (q)   = instance InnerBuf                                    Đặc tả bên ngoài của Bộ đệm với q bị ẩn.
Buffer   =    q : Buf (q)!InnerBuffer Bộ
thu ban đầu   = out.ack    {0, 1}                                Ngư ời nhận
Bộ thu   = Bộ thuInit      [OutChan !Rcv]out.ack                 sự chỉ rõ.
IsChannel(c)   =    c = [ack    c.ack, val    c.val, rdy    c.rdy]
Spec   =      (IsChannel(in)    IsChannel(out))                  Khẳng định rằng vào và ra luôn là bản ghi.
              (in.ack = in.rdy)    (out.ack = out.rdy)           Liên quan đến trạng thái ban đầu của các thành phần khác nhau.
           Ngư ời gửi    Bộ đệm    Ngư ời nhận                   Kết hợp ba thông số kỹ thuật.
```

Hình 10.1: Đặc tả tổng hợp không xen kẽ của FIFO.

Hành động ở trạng thái tiếp theo của nent chỉ mô tả những thay đổi đối với một phần trạng thái của nó.2 Bây giờ chúng ta hãy xem xét trường hợp khi điều này có thể không thể thực hiện được.

10.4.1 Thay đổi trạng thái rõ ràng

Trước tiên, chúng tôi kiểm tra tình huống trong đó một phần trạng thái không thể được phân chia giữa các thành phần khác nhau, nhưng sự thay đổi trạng thái mà mỗi thành phần thực hiện được mô tả hoàn toàn bằng đặc tả. Để làm ví dụ, hãy
một lần nữa hãy xem xét Người gửi và Người nhận giao tiếp với bộ đệm FIFO.
Trong hệ thống mà chúng ta đã nghiên cứu ở Chương 4, việc gửi hoặc nhận một giá trị cần có hai các bước. Ví dụ: Người gửi thực hiện bước Gửi để gửi một giá trị và nó phải
sau đó đợi cho đến khi bộ đệm thực hiện bước Rcv trước khi nó có thể gửi một giá trị khác.
Chúng tôi đơn giản hóa hệ thống bằng cách thay thế thành phần Buffer bằng một biến buf
có giá trị là chuỗi các giá trị được Người gửi gửi nhưng chưa nhận được
bởi Người nhận. Điều này thay thế hệ thống ba thành phần được mô tả ở trang 35
với cái hai thành phần này:



Người gửi gửi một giá trị bằng cách thêm nó vào cuối buf ; Người nhận nhận được một giá trị bằng cách xóa nó khỏi phần đầu của buf .

Nói chung, Người gửi thực hiện một số tính toán để tạo ra các giá trị mà nó gửi và Bộ thu thực hiện một số tính toán trên các giá trị mà nó gửi nhận được. Trạng thái hệ thống bao gồm buf và hai bộ s và r của các biến mô tả trạng thái Người gửi và Người nhận. Trong đặc tả nguyên khối,
Hành động ở trạng thái tiếp theo của hệ thống là sự phân tách Sndr ,Rcvr trong đó Sndr và Rcvr mô tả các bước được thực hiện bởi Người gửi và Người nhận tương ứng. Những hành động này là Được định nghĩa bởi

Sndr =
buf = Nối thêm(buf , . . .)
SCom
không thay đổi r
STính toán
không thay đổi buf , r

Rcv =
buf =
buf = Đuôi(buf)
RCom
không thay đổi s
RTính toán
không thay đổi buf , s

[2] Trong một bố cục xen kẽ, một đặc tả thành phần có thể khẳng định rằng trạng thái của các thành phần khác không thay đổi.

đối với một số hành động SComm, SCompute, RComm và RCompute. Để đơn giản, chúng tôi giả định rằng cả Sndr và Rcvr đều không cho phép các hành động giật hình, do đó SCompute thay đổi s và RCompute thay đổi r . Bây giờ chúng ta viết đặc tả dưới dạng thành phần của các đặc tả riêng biệt của Người gửi và Người nhận.

Việc tách vị ngữ ban đầu rất đơn giản. Các điều kiện ban đầu trên s thuộc vị từ ban đầu của Người gửi; những cái trên r thuộc về vị từ ban đầu của Người nhận; và điều kiện ban đầu buf = có thể được gán tùy ý cho một trong hai điều kiện đó.

Bây giờ hãy xem xét các hành động trạng thái tiếp theo NS và NR của Người gửi và Các thành phần máy thu. Bí quyết là xác định chúng bằng cách

NS = Sndr (σ (s = s)) NR = Rcvr (ρ (r = r))

trong đó σ và ρ là các hành động chỉ chứa biến buf . Hãy coi σ là mô tả những thay đổi có thể có đối với buf mà không phải do Người gửi gây ra và ρ là mô tả những thay đổi có thể có đối với buf không phải do Người nhận gây ra. Do đó, NS cho phép bất kỳ bước nào là bước Sndr hoặc bước không thay đổi và là một thay đổi đối với buf mà không thể "đổ lỗi" cho Người gửi.

Giả sử σ và ρ thỏa mãn ba điều kiện sau:

- d : (buf = Nối(buf , d)) ρ
  Bước thêm giá trị vào buf không phải do Người nhận gây ra.
- (buf =) (buf = Tail(buf)) σ Bước
  loại bỏ một giá trị khỏi phần đầu của buf không phải do Người gửi gây ra.
- (σ ρ) (buf = buf)
  Một bước do cả Người gửi và Người nhận đều không thể thay đổi buf .

Sử dụng các mối quan hệ rõ ràng như 3

(buf = buf) (buf =) (buf = Tail(buf)) ≡ sai

một phép tính giống như phép tính mà chúng tôi rút ra (10.1) cho thấy

[NS]buf , $_S$ [NR]buf , $_r$ ≡ [Sndr Rcvr]buf , s, r

Như vậy NS và NR là các hành động trạng thái tiếp theo phù hợp cho các thành phần nếu ta chọn σ và ρ thỏa mãn 3 điều kiện trên. Có sự tự do đáng kể trong sự lựa chọn đó. Các lựa chọn mạnh nhất có thể có của σ và ρ là những lựa chọn mô tả chính xác những thay đổi được thành phần khác cho phép:

σ = (buf =) (buf = Đuôi(buf)) =

ρ d : buf = Nối(buf , d)

3Các mối quan hệ này chỉ đúng nếu buf là một dãy. Việc tính toán chặt chẽ đòi hỏi phải sử dụng của một bất biến để khẳng định rằng buf thực sự là một dãy.

Chúng ta có thể làm suy yếu các định nghĩa này theo bất kỳ cách nào chúng ta muốn, miễn là chúng ta duy trì điều kiện σ ρ ngụ ý rằng buf không thay đổi. Ví dụ: chúng ta có thể định nghĩa σ như trên và đặt ρ bằng €σ . Sự lựa chọn là vấn đề của hương vị.

Tôi đã mô tả đặc tả đan xen của hệ thống Người gửi/Người nhận. Bây giờ hãy xem xét một đặc tả không xen kẽ—một đặc tả cho phép các bước trong đó cả Người gửi và Người nhận đều thực hiện tính toán. Nói cách khác, chúng ta muốn đặc tả cho phép các bước SCompute RCompute không thay đổi bf. Đặt SSndr là hành động giống như Sndr ngoại trừ nó không đề cập đến r

, và để RRcvr được định nghĩa tương tự. Sau đó chúng tôi có

Sndr ≡ SSndr (r = r) Rcvr ≡ RRcvr (s = s)

Một đặc tả không xen kẽ nguyên khối có hành động ở trạng thái tiếp theo

Sndr Rcvr (SSndr RRcvr (buf = buf))

Nó là sự kết hợp của các đặc tả thành phần có các hành động ở trạng thái tiếp theo NS và NR được xác định bởi

NS = SSndr (σ (s = s)) NR = RRcvr (ρ (r = r))

trong đó σ và ρ như trên.

Tình huống hai quá trình này khái quát hóa thành thành phần của bất kỳ tập hợp thành phần C nào có chung một biến hoặc bộ biến w. Trường hợp xen kẽ khái quát hóa quy tắc sau, trong đó Nk là hành động trạng thái tiếp theo của thành phần k, hành động µk mô tả tất cả các thay đổi đối với w được quy cho một số thành phần khác ngoài k, bộ vk mô tả trạng thái riêng tư của k, và v là bộ dữ liệu được hình thành bởi tất cả vk:

Quy tắc thành phần trạng thái chia sẻ Bốn điều kiện

1. (k C : vk = vk) ≡ (v = v)

   v không thay đổi nếu trạng thái riêng vk của mọi thành phần không thay đổi.

2. i, k C : Nk (i = k) (vi = vi)

   Hành động ở trạng thái tiếp theo của bất kỳ thành phần k nào sẽ giữ nguyên trạng thái riêng vi của tất cả các thành phần i khác.

3. i, k C : Nk (w = w) (i = k) µi

   Một bước của bất kỳ thành phần k nào thay đổi w là bước µi đối với bất kỳ thành phần i nào khác.

4. (k C : µk) ≡ (w = w)

   Một bước được gây ra bởi không có thành phần nào nếu nó không thay đổi w.

bao hàm, ngụ ý

(k C : Ik [Nk (µk (vk = vk))]w, vk)

≡ (k C : Ik) [k C : Nk]w, v

Giả định 2 khẳng định rằng chúng ta có đặc tả đan xen. Nếu chúng ta loại bỏ giả định đó thì vế phải của kết luận có thể không phải là một đặc tả hợp lý, vì Nk rời rạc có thể cho phép các bư ớc trong đó một biến của một số thành phần khác giả định các giá trị tùy ý. Tuy nhiên, nếu mỗi Nk xác định chính xác các giá trị mới của trạng thái riêng vk của thành phần k thì vk bên trái sẽ là một đặc tả hợp lý, mặc dù có thể là một đặc tả không xen kẽ (và không nhất thiết phải tư ơ ng đư ơ ng với vk bên phải).

10.4.2 Thành phần với hành động chung

Hãy xem xét bộ nhớ tuyến tính hóa của Chư ơ ng 5. Như đư ợc hiển thị trong hình ở trang 45, nó là một hệ thống bao gồm một tập hợp các bộ xử lý, bộ nhớ và giao diện đư ợc biểu thị bằng biến memInt. Bây giờ chúng ta coi nó là một hệ thống hai thành phần, trong đó bộ xử lý tạo thành một thành phần, đư ợc gọi là môi trư ờng và bộ nhớ là thành phần còn lại. Bây giờ chúng ta hãy bỏ qua việc ẩn và chỉ xem xét thông số kỹ thuật bên trong, với tất cả các biến có thể nhìn thấy đư ợc. Chúng tôi muốn viết đặc tả dư ới dạng

(10.5) (IE [NE]vE) (IM [NM]vM)

trong đó E đề cập đến thành phần môi trư ờng (bộ xử lý) và M đề cập đến thành phần bộ nhớ. Bộ vE của các biến bao gồm memInt và các biến của thành phần môi trư ờng; bộ vM bao gồm memInt và các biến của thành phần bộ nhớ. Ta phải chọn các công thức IE, NE, v.v. sao cho (10.5), ẩn các biến nội, tư ơ ng đư ơ ng với thông số bộ nhớ Spec của module Memory ở trang 53.

Trong đặc tả bộ nhớ, giao tiếp giữa môi trư ờng và bộ nhớ đư ợc mô tả bằng một hành động có dạng

Gửi(p, d, memInt, memInt) hoặc Trả lời(p, d, memInt, memInt)

trong đó Gửi và Trả lời là các toán tử không xác định đư ợc khai báo trong mô-đun MemoryInterface (trang 48). Thông số kỹ thuật không nói gì về giá trị thực của memInt. Vì vậy, chúng ta không những không biết cách chia memInt thành hai phần, mỗi phần chỉ đư ợc thay đổi bởi một trong các thành phần, mà chúng ta thậm chí còn không biết chính xác memInt thay đổi như thế nào.

Mẹo để viết đặc tả dư ới dạng bố cục là đặt các hành động Gửi và Trả lời vào các hành động trạng thái tiếp theo của cả hai thành phần. Chúng tôi thể hiện việc gửi một giá trị qua memInt dư ới dạng một hành động chung đư ợc thực hiện bởi cả bộ nhớ và môi trư ờng. Các hành động ở trạng thái tiếp theo có dạng sau:

NM = p Proc : MRqst(p) MRsp(p) MInternal(p)

NE = p Proc : ERqst(p) ERsp(p)

trong đó bước MRqst(p) ERqst(p) thể hiện việc gửi yêu cầu của bộ xử lý p (một phần của môi trường) tới bộ nhớ, bước MRsp(p) ERsp(p) thể hiện việc gửi phản hồi của bộ nhớ tới bộ xử lý p và bước MInternal(p) là bước bên trong của thành phần bộ nhớ thực hiện yêu cầu.
(Không có bước nội bộ của môi trường.)

Việc gửi phản hồi được điều khiển bởi bộ nhớ, bộ nhớ sẽ chọn giá trị nào được gửi và thời điểm gửi. Do đó, điều kiện kích hoạt và giá trị được gửi được chỉ định bởi hành động MRsp(p). Hãy coi các biến bên trong của thành phần bộ nhớ giống các biến mem, ctl và buf như trong đặc tả bộ nhớ nguyên khối bên trong của mô-đun InternalMemory trên trang 52 và 53. Sau đó, chúng ta có thể đặt MRsp(p) giống như hành động Rsp(p) được xác định trong mô-đun đó. Hành động ERsp(p) phải luôn được bật và nó phải cho phép gửi mọi phản hồi pháp lý. Phản hồi pháp lý là một phần tử của Val hoặc giá trị đặc biệt NoVal, vì vậy chúng ta có thể định nghĩa ERsp(p) bằng4

```
            rsp    Val    {NoVal} : Trả lời(p, rsp, memInt, memInt )
        . . .
```

ở đâu có “. . .” mô tả các giá trị mới của các biến nội bộ của môi trường.

Việc gửi yêu cầu được kiểm soát bởi môi trường, môi trường này chọn giá trị nào được gửi và thời điểm gửi. Do đó, điều kiện kích hoạt phải là một phần của hành động ERqst(p). Trong thông số kỹ thuật nguyên khối của mô-đun Bộ nhớ trong, điều kiện kích hoạt đó là ctl[p] = “rdy”. Tuy nhiên, nếu ctl là biến nội bộ của bộ nhớ thì nó cũng không thể xuất hiện trong đặc tả môi trường. Do đó, chúng tôi phải thêm một biến mới có giá trị cho biết liệu bộ xử lý có được phép gửi yêu cầu mới hay không. Hãy sử dụng biến Boolean rdy, trong đó rdy[p] là đúng nếu bộ xử lý p có thể gửi yêu cầu. Giá trị của rdy[p] được đặt thành sai khi p gửi yêu cầu và được đặt lại thành đúng khi phản hồi tương ứng tới p được gửi. Do đó, chúng ta có thể định nghĩa ERqst(p) và hoàn thành định nghĩa của ERsp(p) như sau:

```
ERqst(p)   =    rdy[p]
                    req    MReq : Gửi(p, req, memInt, memInt )
                rdy = [rdy ngoại trừ ![p] = false]

ERsp(p)    =      rsp    Val    {NoVal} :
                    Trả lời(p, rsp, memInt, memInt )
                 rdy = [rdy ngoại trừ ![p] = true]
```

Hành động MRqst(p) của bộ nhớ giống như hành động Req(p) của mô-đun Bộ nhớ trong, ngoại trừ việc không có điều kiện kích hoạt ctl[p] = “rdy”.

4Giới hạn trên là không cần thiết. Chúng ta có thể để bộ xử lý chấp nhận bất kỳ giá trị nào, không chỉ giá trị hợp pháp, bằng cách lấy rsp : Reply(p, rsp, memInt, memInt) làm liên từ đầu tiên. Tuy nhiên, nhìn chung sẽ tốt hơn nếu sử dụng các bộ định lượng giới hạn khi có thể.

Cuối cùng, hành động bên trong của bộ nhớ MInternal(p) giống với Do(p)
hoạt động của mô-đun InternalMemory.

Phần còn lại của đặc điểm kỹ thuật là dễ dàng. Các bộ vE và vM lần lượt là memInt, rdy
và memInt, mem, ctl, buf . Xác định các vị từ ban đầu IE
và IM rất đơn giản, ngoại trừ việc quyết định đặt địa chỉ ban đầu ở đâu
điều kiện memInt initMemInt. Chúng ta có thể đặt nó trong IE hoặc IM ở cả hai, ,
hoặc trong một liên kết riêng biệt không thuộc về đặc điểm kỹ thuật của thành phần nào. Hãy
đặt nó vào IM, sau đó bằng vị từ ban đầu IInit từ
Mô-đun bộ nhớ trong. Đặc tả môi trường cuối cùng có được bằng cách ẩn rdy trong đặc tả bên
trong của nó; đặc tả thành phần bộ nhớ cuối cùng
thu được bằng cách ẩn mem, ctl và buf trong đặc tả bên trong của nó. Thông số kỹ thuật đầy
đủ xuất hiện trong Hình 10.2 ở trang tiếp theo. Tôi đã không làm phiền
để xác định IM , MRsp(p) hoặc MInternal(p), vì chúng bằng IInit, Rsp(p) và
Do(p) từ mô-đun InternalMemory tương ứng.

Những gì chúng ta vừa làm cho hệ thống bộ nhớ môi trường khái quát hóa một cách tự
nhiên các thông số kỹ thuật hoạt động chung của bất kỳ hệ thống hai thành phần nào trong đó phần
của trạng thái không thể được coi là thuộc về một trong hai thành phần. Nó cũng khái quát
hóa các hệ thống trong đó bất kỳ số lượng thành phần nào cũng chia sẻ một phần của
tình trạng. Ví dụ: giả sử chúng ta muốn viết một đặc tả tổng hợp của
hệ thống bộ nhớ tuyến tính hóa trong đó mỗi bộ xử lý là một thành phần riêng biệt.
Thông số kỹ thuật của thành phần bộ nhớ sẽ giống như trước đây. Các
hành động trạng thái tiếp theo của bộ xử lý p bây giờ sẽ là

ERqst(p) ERsp(p) OtherProc(p)

trong đó ERqst(p) và ERsp(p) giống như trên và bước OtherProc(p)
thể hiện việc gửi yêu cầu hoặc phản hồi tới một số bộ xử lý khác
hơn p. Hành động OtherProc(p) thể hiện sự tham gia của p vào hành động chung bởi
mà bộ xử lý q khác giao tiếp với thành phần bộ nhớ. Nó là
được xác định bằng

```
q  Proc \ {p} :    req  MReq : Gửi(q, req, memInt, memInt )
                   rsp  Val  {NoVal} :
                     Trả lời(q, rsp, memInt, memInt )
```

Ví dụ này khá ngớ ngẩn vì mỗi bộ xử lý phải tham gia vào các hoạt động giao tiếp chỉ liên
quan đến các thành phần khác. Sẽ tốt hơn nếu
thay đổi giao diện để biến memInt thành một mảng, với giao tiếp giữa
bộ xử lý p và bộ nhớ được biểu thị bằng sự thay đổi đối với memInt[p]. Một cách hợp lý
Ví dụ sẽ yêu cầu một hành động chung thể hiện sự tương tác thực sự giữa
tất cả các thành phần—ví dụ: hoạt động đồng bộ hóa rào cản trong đó
các thành phần đợi cho đến khi chúng sẵn sàng rồi mới thực hiện đồng bộ hóa
bước cùng nhau.

```
┌──────────────────────── mô-đun JointActionMemory ────────────────────────┐
mở rộng mô-đun
  ┌──────────────MemoryInterface InnerEnvironmentComponent ──────────────┐
   biến số

   IE   = rdy = [p    Proc    true]

   ERqst(p)   =    rdy[p]
                       req    MReq : Gửi(p, req, memInt, memInt )
                  rdy = [rdy ngoại trừ ![p] = false]

   ERsp(p)   =      rsp    Val    {NoVal} : Trả lời(p, rsp, memInt, memInt )
                   rdy = [rdy ngoại trừ ![p] = true]

   NE   =    p    Proc : ERqst(p)    ERsp(p)

   IESpec   = IE      [NE]memInt, rdy
  └──────────────────────────────────────────────────────────────────────┘

  ┌──────────────────────── mô-đun InnerMemoryComponent ─────────────────┐
   mở rộng InternalMemory

   MRqst(p)   =      req    MReq :    Gửi(p, req, memInt, memInt )
                                     buf = [buf ngoại trừ ![p] = req]
                                        ctl = [ctl ngoại trừ ![p] = “ bận"]
                    bộ nhớ không thay đổi

   NM   =    p    Proc : MRqst(p)    Do(p)    Rsp(p)

   IMSpec   = IInit      [NM ]memInt, mem, ctl, buf
  └──────────────────────────────────────────────────────────────────────┘

IEnv(rdy)   = instance InnerEnvironmentComponent IMem(mem,

ctl, buf )   = instance InnerMemoryComponent Spec   =      rdy :

IEnv(rdy)!IESpec      mem, ctl, buf :
           IMem(mem, ctl, buf )! IMSpec
└──────────────────────────────────────────────────────────────────────────┘
```

Hình 10.2: Đặc tả hoạt động chung của bộ nhớ tuyến tính hóa.

10.5 Đánh giá ngắn gọn

Ý tưởng cơ bản của việc soạn thảo các đặc tả rất đơn giản: một đặc tả tổng hợp là sự kết hợp của các công thức, mỗi công thức có thể được coi là đặc tả của một thành phần riêng biệt. Chương này đã trình bày một số kỹ thuật để viết một đặc tả dưới dạng một bố cục. Trước khi đi xa hơn, chúng ta hãy xem xét các kỹ thuật này.

10.5.1 Phân loại thành phần

Chúng ta đã thấy ba cách khác nhau để phân loại các thông số kỹ thuật tổng hợp:

Từ xen kẽ là tiêu chuẩn; không có thuật ngữ chung cho ngư ời khác các khái niệm.

Xen kẽ và không xen kẽ. Một đặc tả xen kẽ là một trong
: mà mỗi bư ớc (không nói lắp) có thể đư ợc quy cho chính xác một thành phần. Đặc tả không xen kẽ cho phép các bư ớc thể hiện các hoạt động đồng thời của hai hoặc nhiều thành phần khác nhau.

Trạng thái rời rạc so với trạng thái chia sẻ. Một đặc tả trạng thái rời rạc là một trong
: mà trạng thái có thể đư ợc phân chia, với mỗi phần thuộc về một phần riêng biệt thành phần. Một phần của trạng thái có thể là một biến v, hoặc một "phần" của nó biến chẳng hạn như vc hoặc v[c] đối với một số c cố định. Bất kỳ thay đổi nào đối với một thành phần một phần của trạng thái đư ợc quy cho thành phần đó. Trong đặc tả trạng thái chia sẻ, một số phần của trạng thái có thể đư ợc thay đổi theo các bư ớc đư ợc quy cho nhiều hơ n. hơ n một thành phần.

Hành động chung và hành động riêng biệt. Đặc tả hành động chung là đặc tả không xen kẽ trong
: đó phải xảy ra một số bư ớc đư ợc quy cho một thành phần đồng thời với một bư ớc đư ợc quy cho thành phần khác. Đặc tả hành động riêng biệt đơ n giản là đặc tả không phải là đặc tả hành động chung.

Đây là những cách phân loại thông số kỹ thuật độc lập, ngoại trừ việc đặc tả hành động chung phải không xen kẽ.

10.5.2 Việc xen kẽ đư ợc xem xét lại

Chúng ta nên viết các đặc tả xen kẽ hay không đan xen? Chúng ta có thể cố gắng
Hãy trả lời câu hỏi này bằng cách hỏi: liệu các thành phần khác nhau có thực sự thực hiện đư ợc các bư ớc đồng thời không? Tuy nhiên, câu hỏi này không có ý nghĩa. Một bư ớc là một phép toán trừu tư ợng; các thành phần thực thực hiện các hoạt động đòi hỏi một lư ợng hữu hạn thời gian. Các hoạt động đư ợc thực hiện bởi hai thành phần khác nhau có thể trùng lặp về mặt thời gian. Chúng ta có thể tự do biểu diễn tình huống vật lý này bằng một biểu tư ợng đồng thời duy nhất. bư ớc của hai thành phần hoặc với hai bư ớc riêng biệt. Trong trư ờng hợp sau, đặc điểm kỹ thuật thư ờng cho phép hai bư ớc xảy ra theo một trong hai thứ tự. (Nếu hai các hoạt động phải xảy ra đồng thời, khi đó chúng ta đã viết một đặc tả hành động chung.) Viết xen kẽ hay không xen kẽ là tùy thuộc vào bạn sự chỉ rõ. Bạn nên chọn cái nào thuận tiện hơ n.

Sự lựa chọn không hoàn toàn tùy tiện nếu bạn muốn một đặc tả này triển khai một đặc tả khác. Nói chung, một đặc tả không xen kẽ sẽ không thực hiện một xen kẽ một vì đặc tả không xen kẽ sẽ cho phép các hành động đồng thời mà đặc tả xen kẽ cấm. Vì vậy, nếu bạn muốn viết một đặc tả cấp thấp hơ n thực hiện một đặc tả xen kẽ cấp cao hơ n, thì bạn sẽ phải sử dụng một đặc tả xen kẽ. Như chúng ta đã thấy, thật dễ dàng

để biến một đặc tả không xen kẽ thành một đặc tả xen kẽ bằng cách kết hợp một giả định xen kẽ.

10.5.3 Hành động chung được xem xét lại

Lý do viết một đặc tả tổng hợp là để tách biệt các đặc tả của các thành phần khác nhau. Sự pha trộn các hành động từ các thành phần khác nhau trong đặc tả hành động chung sẽ phá hủy sự tách biệt này. Vậy tại sao chúng ta phải viết một đặc điểm kỹ thuật như vậy?

Các đặc tả hành động chung xuất hiện thường xuyên nhất trong các mô tả rất trừu tượng về giao tiếp giữa các thành phần. Khi viết một đặc tả tổng hợp của bộ nhớ có thể điều chỉnh được, chúng tôi buộc phải sử dụng các hành động chung vì tính chất trừu tượng của giao diện. Trong các hệ thống thực, sự giao tiếp xảy ra khi một thành phần thay đổi trạng thái và thành phần khác sau đó sẽ quan sát sự thay đổi đó. Giao diện được mô tả bởi mô-đun MemoryInterface trừu tượng hóa hai bước đó, thay thế chúng bằng một cái duy nhất tượng trưng cho sự giao tiếp tức thời— một điều hư cấu không tồn tại trong thế giới thực. Vì mỗi thành phần phải ghi nhớ rằng giao tiếp đã xảy ra nên bước giao tiếp duy nhất phải thay đổi trạng thái riêng tư của cả hai thành phần. Đó là lý do tại sao chúng tôi không thể sử dụng cách tiếp cận của Mục 10.4.1 (trang 144), trong đó yêu cầu mọi thay đổi đối với giao diện được chia sẻ, hãy thay đổi trạng thái không chia sẻ của chỉ một thành phần.

Giao diện bộ nhớ trừu tượng đơn giản hóa đặc tả, cho phép giao tiếp được biểu diễn dưới dạng một bước thay vì hai bước. Nhưng sự đơn giản hóa này phải trả giá bằng việc làm mờ đi sự khác biệt giữa hai thành phần. Nếu chúng ta làm lu mờ sự khác biệt này, sẽ không có ý nghĩa gì nếu viết đặc tả dưới dạng sự kết hợp của các đặc tả thành phần riêng biệt. Như ví dụ về hệ thống bộ nhớ minh họa, phân rã hệ thống thành các thành phần riêng biệt giao tiếp với các hành động chung có thể yêu cầu đưa ra các biến bổ sung. Có thể có đôi khi có thể là một lý do chính đáng để thêm loại phức tạp này vào một đặc tả, nhưng điều đó tất nhiên không nên được thực hiện.

10.6 Sự sống động và ẩn náu

10.6.1 Độ sống động và đóng máy

Cho đến nay, việc thảo luận về bố cục đã bỏ qua tính sống động. Trong tổng hợp thông số kỹ thuật, thường dễ dàng xác định tính sống động bằng cách đặt các điều kiện công bằng cho hành động của từng thành phần riêng lẻ. Ví dụ, để chỉ định một dãy đồng hồ luôn tích tắc mãi mãi, chúng tôi sẽ sửa đổi thông số kỹ thuật

ClockArray của (10.2) trên trang 139 bằng

k Đồng hồ :

(hr [k] 1 . . 12) [HCN (hr [k])]hr[k] WFhr[k](HCN (hr [k]))

Khi viết công thức công bằng yếu hay mạnh cho hành động A của thành phần c, sẽ nảy sinh câu hỏi chỉ số dư ới phải là gì. Rõ ràng các lựa chọn là (i) bộ v mô tả toàn bộ trạng thái đặc tả và (ii) tuple v c mô tả trạng thái của thành phần đó. Sự lựa chọn chỉ có thể quan trọng nếu phần an toàn của thông số kỹ thuật cho phép hệ thống đạt đến một số trạng thái trong đó Một bư ớc có thể khiến v c không thay đổi khi thay đổi v. Mặc dù khó xảy ra như ng điều này có thể có thể hình dung đư ợc là trư ờng hợp trong đặc tả hành động chung. Nếu vậy thì có lẽ chúng ta không muốn điều kiện công bằng đư ợc thỏa mãn bằng một bư ớc rời khỏi thành phần trạng thái không thay đổi, vì vậy chúng ta sẽ sử dụng chỉ số dư ới v c.

Các điều kiện công bằng cho thông số kỹ thuật tổng hợp đặt ra một câu hỏi quan trọng: nếu mỗi thông số kỹ thuật thành phần đư ợc đóng bằng máy thì thông số tổng hợp có đư ợc xác định không? ification nhất thiết phải đóng máy? Giả sử chúng ta viết đặc tả như

k C : Sk Lk Lk, mỗi cặp S là con, ở trang 111.

trong Phần 8.9.2

điểm nối của Sk và L sự kết hợp của Lk nên thông số kỹ thuật bằng, S L. Sự kết hợp của các thuộc tính an toàn là thuộc tính an toàn,5 nên S là thuộc tính an toàn tài sản. Do đó, ta có thể hỏi cặp S, L có đư ợc đóng bằng máy hay không.

Nói chung S, L không cần phải đóng máy. Tuy nhiên, đối với một bố cục đan xen thì thư ờng là như vậy. Các đặc tính sống động thư ờng đư ợc thể hiện dư ới dạng kết hợp tính chất công bằng yếu và mạnh của hành động. Như đã nêu ở trang 112, một thông số kỹ thuật sẽ bị đóng máy nếu thuộc tính sống động của nó là sự kết hợp của tính công bằng thuộc tính cho các hành động phụ của hành động ở trạng thái tiếp theo. Trong tổ hợp xen kẽ, mỗi Sk thư ờng có dạng I k [Nk]vk trong đó v k thỏa mãn giả thuyết về Quy Tắc Thành Phần (trang 138), và mỗi Nk hàm ý vi = vi , với mọi thứ i trong C \ {k}. Trong trư ờng hợp này, Quy tắc Thành phần ngụ ý rằng một hành động phụ của Nk cũng là một tập con của hành động trạng thái tiếp theo của S. Do đó, nếu chúng ta viết một kết cấu xen kẽ theo cách thông thư ờng và ta viết thông số thành phần đóng bằng máy theo cách thông thư ờng thì thông số tổ hợp là máy đóng cửa.

Để có đư ợc một bố cục không xen kẽ đư ợc đóng kín bằng máy không phải là điều dễ dàng— đặc biệt là với một thành phần hành động chung. Chúng tôi thực sự đã thấy một ví dụ của đặc tính kỹ thuật tác động chung trong đó mỗi bộ phận đư ợc đóng bằng máy như ng thành phần thì không. Trong Chư ơ ng 9, chúng ta đã viết đặc tả thời gian thực bằng kết hợp một hoặc nhiều công thức RTBound và công thức RTnow với một công thức không tính thời gian sự chỉ rõ. Một ví dụ bệnh lý sau đây là công thức (9.2) ở trang 131:

HC RTBound(hr = hr 1, hr , 0, 3600) RTnow(hr)

HC là thông số kỹ thuật của đồng hồ giờ từ Chư ơ ng 2.

5Hãy nhớ lại rằng đặc tính an toàn là đặc tính bị vi phạm bởi một hành vi nếu nó bị vi phạm vào một thời điểm nào đó. điểm cụ thể trong hành vi. Một hành vi vi phạm sự kết hợp các thuộc tính an toàn Sk iff nó vi phạm một số Sk cụ thể , và Sk bị vi phạm tại một số điểm cụ thể.

Chúng ta có thể xem công thức này là sự kết hợp của ba thông số kỹ thuật thành phần:

1. HC chỉ định đồng hồ, được biểu thị bằng biến hr .
2. RTBound(hr = hr 1, hr , 0, 3600) chỉ định một bộ đếm thời gian, được biểu thị bằng biến hẹn giờ ẩn (được định lượng hiện tại).
3. RTnow(hr) chỉ định thời gian thực, được biểu thị bằng biến bây giờ.

Công thức này là một chế phẩm tác động chung, với hai loại tác động chung:

- Hoạt động chung của thành phần thứ nhất và thứ hai làm thay đổi cả giờ và đồng hồ bấm giờ.
- Hành động chung của thành phần thứ hai và thứ ba làm thay đổi cả hẹn giờ và bây giờ.

Hai thông số kỹ thuật đầu tiên được đóng máy một cách tầm thường vì chúng khẳng định không điều kiện sống động, vì vậy tính chất sống động của chúng là đúng. Thông số kỹ thuật thứ ba Thuộc tính an toàn khẳng định rằng bây giờ là số thực chỉ được thay đổi theo bước tăng nó lên và giữ nguyên hr; tài sản sống động của nó là New Zealand khẳng định rằng bây giờ tăng không giới hạn. Bất kỳ hành vi hữu hạn nào thỏa mãn tính chất an toàn có thể dễ dàng được mở rộng đến một hành vi vô hạn thỏa mãn toàn bộ đặc tả, vì vậy thông số kỹ thuật thứ ba cũng được đóng bằng máy. Tuy nhiên, như chúng tôi quan sát thấy ở Mục 9.4, thông số tổng hợp là Zeno, nghĩa là không phải máy đóng cửa.

10.6.2 Ẩn

Giả sử chúng ta có thể viết đặc tả S là thành phần của hai thành phần thông số kỹ thuật S_1 và S Có thể viết h : S, thông số S với biến h 2. ẩn, như một thành phần—nghĩa là, như sự kết hợp của hai thành phần riêng biệt thông số kỹ thuật? Nếu h đại diện cho trạng thái được truy cập bởi cả hai thành phần thì câu trả lời là không. Nếu hai thành phần giao tiếp thông qua một phần nào đó của trạng thái, thì phần đó của trạng thái không thể trở thành nội bộ của trạng thái riêng biệt các thành phần.

Trường hợp đơn giản nhất trong đó h không đại diện cho trạng thái chia sẻ là khi nó chỉ xảy ra ở một trong các thông số kỹ thuật của thành phần—ví dụ: S 2. Nếu h không xảy ra trong S_1 thì sự tương đương

$$(\quad h : S_1 \quad S_2) \equiv S_1 \quad (\quad h : S_2)$$

cung cấp sự phân hủy mong muốn.

Bây giờ giả sử rằng h xuất hiện trong cả hai đặc tả thành phần, nhưng không đại diện cho trạng thái được truy cập bởi cả hai thành phần. Điều này chỉ có thể xảy ra nếu "các phần" khác nhau của h xuất hiện trong hai đặc tả thành phần. Ví dụ,

h có thể là một bản ghi có các thành phần h.c1 và h.c2, trong đó S h.c1 1 chỉ đề cập đến và S 2 chỉ đề cập đến h.c2. Trong trư ờng hợp này,

chúng ta có (h : S 1 S 2) ≡ (h1 : T1)

(h2 : T2) trong đó T1 thu đư ợc từ S 1 bằng cách thay biến h1 cho biểu thức h. c1 và T2 đư ợc định nghĩa tư ơ ng tự. Tất nhiên chúng ta có thể sử dụng bất kỳ biến nào thay cho h1 và h2; cụ thể là chúng ta có thể thay thế cả hai bằng cùng một biến.

Chúng ta có thể khái quát hóa kết quả này như sau với thành phần của bất kỳ hữu hạn nào số 6 của các thành phần:

Quy tắc ẩn thành phần Nếu biến h không xuất hiện trong và Si thu đư ợc từ Ti bằng , cách thay h[i] cho q, thì công thức Ti

(h : i C : Si) ≡ (i C : q : Ti)

với mọi tập hữu hạn C .

Giả sử h không xuất hiện trong Ti có nghĩa là biến h chỉ xuất hiện trong công thức Si trong biểu thức h[i]. Điều này lại hàm ý rằng thành phần i C : Si không xác định giá trị của h mà chỉ xác định các thành phần h[i] của nó đối với i C . Như đã lư u ý trong Phần 10.2 trên trang 138, chúng ta có thể làm cho thông số kỹ thuật tổng hợp xác định giá trị của h bằng cách kết hợp công thức IsFcnOn(h, C) với nó, trong đó IsFcnOn đư ợc xác định trên trang 140. Các giả thuyết về Ẩn thành phần Quy tắc ngụ ý

(h : IsFcnOn(h, C) i C : Si) ≡ (i C : q : Ti)

Bây giờ hãy xem xét trư ờng hợp phổ biến trong đó i C : Si là một thành phần đan xen, trong đó mỗi đặc tả Si mô tả các thay đổi đối với h[i] và khẳng định rằng các bư ớc của thành phần i giữ nguyên h[j] đối với j = i. Chúng ta không thể áp dụng Quy tắc Ẩn Thành phần vì Si phải đề cập đến các thành phần khác của h ngoài h[i]. Ví dụ: nó có thể chứa biểu thức có dạng h (10.6) = [h ngoại trừ ![i] = exp]

trong đó đề cập đến tất cả h. Tuy nhiên, chúng ta có thể chuyển Si thành đặc tả Si chỉ mô tả những thay đổi của h[i] và không đư a ra khẳng định nào về các thành phần khác. Ví dụ, chúng ta có thể thay thế (10.6) bằng h [i] = exp, và chúng ta có thể thay thế một khẳng định rằng h không thay đổi bằng khẳng định rằng h[i] không thay đổi.

Thành phần i C : Si có thể cho phép các bư ớc thay đổi hai thành phần khác nhau h[i] và h[j], trong khi giữ nguyên tất cả các biến khác, làm cho nó trở thành đặc tả không xen kẽ. Khi đó nó sẽ không tư ơ ng đư ơ ng với i C : Si đòi hỏi những thay đổi đối với h[i] , và h[j] phải đư ợc thực hiện theo các bư ớc khác nhau.

Tuy nhiên, có thể chỉ ra rằng việc ẩn h sẽ che giấu sự khác biệt này, làm cho hai thông số kỹ thuật tư ơ ng đư ơ ng nhau. Sau đó chúng ta có thể áp dụng Quy tắc ẩn thành phần với Si đư ợc thay thế bằng Si .

6 Quy tắc ẩn thành phần nói chung không đúng nếu C là tập hợp vô hạn; như ng những ví dụ không đúng là bệnh lý và không xuất hiện trong thực tế.

10.7 Thông số kỹ thuật hệ thống mở

Đặc tả mô tả sự tương tác giữa hệ thống và môi trường của nó.
Ví dụ: đặc tả bộ đệm FIFO của Chương 4 chỉ định sự tương tác giữa bộ đệm (hệ thống) và môi trường bao gồm người gửi và người nhận. Cho đến nay, tất cả các đặc tả mà chúng ta đã viết đều là các đặc tả hệ thống hoàn chỉnh, nghĩa là chúng được thỏa mãn bởi hành vi thể hiện hoạt động chính xác của cả hệ thống và môi trường của nó. Khi chúng ta viết đặc tả như thành phần của đặc tả môi trường E và đặc tả hệ thống M, nó có dạng E M .

,

Đặc tả hệ thống mở là đặc tả có thể phục vụ như một hợp đồng giữa các đặc tả hệ thống mở đôi khi được gọi là đặc tả đảm bảo tin cậy hoặc đảm bảo giả định. người sử dụng hệ thống và người thực hiện nó. Một lựa chọn rõ ràng cho đặc tả như vậy là công thức M mô tả hành vi đúng của chính thành phần hệ thống. Tuy nhiên, một đặc điểm kỹ thuật như vậy là không thể thực hiện được. Nó khẳng định rằng hệ thống hoạt động chính xác bất kể môi trường làm gì. Một hệ thống không thể hành xử như mong đợi khi đối mặt với hành vi tùy tiện của môi trường nó. Sẽ không thể xây dựng một bộ đệm thỏa mãn đặc điểm kỹ thuật của thành phần bộ đệm bất kể người gửi và người nhận đã làm gì. Ví dụ: nếu người gửi gửi một giá trị trước khi giá trị trước đó được xác nhận thì bộ đệm có thể đọc giá trị đó trong khi nó đang thay đổi, gây ra kết quả không thể đoán trước.

Hợp đồng giữa người dùng và người triển khai chỉ nên yêu cầu hệ thống hoạt động chính xác nếu môi trường thực hiện như vậy. Nếu M mô tả hành vi đúng của hệ thống và E mô tả hành vi đúng của môi trường, thì thông số kỹ thuật đó phải yêu cầu M phải đúng nếu E đúng. Điều này gợi ý rằng chúng ta lấy đặc tả hệ thống mở của mình là công thức E M, công thức này đúng nếu hệ thống hoạt động đúng hoặc môi trường hoạt động không đúng. Tuy nhiên, E M là thông số kỹ thuật quá yếu vì lý do sau. Hãy xem xét lại ví dụ về bộ đệm FIFO, trong đó M mô tả bộ đệm và E mô tả người gửi và người nhận. Giả sử bây giờ bộ đệm gửi một giá trị mới trước khi bên nhận xác nhận giá trị trước đó. Điều này có thể khiến máy thu hoạt động không chính xác, có thể sửa đổi kênh đầu ra theo cách nào đó mà thông số kỹ thuật của máy thu không cho phép. Tình huống này được mô tả bằng một hành vi trong đó cả E và M đều sai—một hành vi thỏa mãn đặc tả E M . Tuy nhiên, bộ đệm không được coi là hoạt động chính xác trong trường hợp này, vì chính lỗi của bộ đệm đã khiến bộ thu hoạt động không chính xác. Do đó, hành vi này sẽ không đáp ứng được đặc điểm kỹ thuật của bộ đệm.

Đặc tả hệ thống mở phải khẳng định rằng hệ thống hoạt động chính xác ít nhất là trong thời gian môi trường hoạt động chính xác. Để diễn đạt điều này, chúng tôi giới thiệu một toán tử thời gian mới + , trong đó E + M khẳng định rằng M vẫn đúng lâu hơn E ít nhất một bước, vẫn đúng mãi mãi nếu E đúng như vậy. Chính xác hơn một chút, E + M khẳng định rằng

- E hàm ý M .

- Nếu n trạng thái đầu tiên của một hành vi không vi phạm tính an toàn của E thì n+1 trạng thái đầu tiên không vi phạm tính chất an toàn của M đối với mọi số tự nhiên n. (Hãy nhớ lại rằng đặc tính an toàn là đặc tính mà nếu bị vi phạm thì sẽ bị vi phạm ở một điểm nhất định nào đó trong hành vi.)

Định nghĩa chính xác hơn về + xuất hiện trong Phần 16.2.4 (trang 314). Nếu E mô tả hành vi mong muốn của môi trường và M mô tả hành vi mong muốn của hệ thống, thì chúng ta lấy công thức E + M làm đặc tả hệ thống mở.

Khi chúng ta viết các đặc tả riêng biệt của các thành phần, chúng ta thường có thể chuyển đổi đặc tả hệ thống hoàn chỉnh thành đặc tả hệ thống mở bằng cách thay thế kết hợp bằng + . Điều này đòi hỏi trước tiên phải quyết định xem mỗi phần kết hợp của đặc tả hệ thống hoàn chỉnh có thuộc về đặc tả của môi trường, của hệ thống hay không thuộc về cả hai. Ví dụ: hãy xem xét đặc tả tổng hợp của bộ đệm FIFO trong mô-đun CompositeFIFO ở trang 143.
Chúng tôi coi hệ thống chỉ bao gồm bộ đệm, với người gửi và người nhận tạo thành môi trường. Đặc tả hệ thống khép kín Spec có ba liên kết chính:

Người gửi Bộ đệm Người
nhận Liên kết Người gửi và Người nhận rõ ràng là một phần của đặc tả môi trường và Bộ đệm là một phần của đặc tả hệ thống.

(in.ack = in.rdy) (out.ack = out.rdy)
Hai liên kết ban đầu này có thể được gán cho một trong hai, tùy thuộc vào thành phần nào mà chúng ta muốn đổ lỗi nếu chúng bị vi phạm. Hãy gán cho thành phần gửi trên kênh c trách nhiệm thiết lập c.ack = c.rdy ban đầu. Sau đó, chúng tôi gán in.ack = in.rdy cho môi trường và out.ack = out.rdy cho hệ thống.

(IsChannel(vào) IsChannel(out))
Công thức này không tự nhiên được quy cho hệ thống hoặc môi trường. Chúng tôi coi nó như một thuộc tính vốn có trong cách mô hình hóa hệ thống của chúng tôi, giả định rằng vào và ra là các bản ghi có các thành phần ack, val và rdy. Do đó, chúng tôi coi công thức này là một phần riêng biệt của đặc tả hoàn chỉnh, không thuộc về hệ thống hoặc môi trường.

Sau đó chúng ta có đặc tả hệ thống mở sau đây cho bộ đệm FIFO:

(IsChannel(in) IsChannel(out))
(in.ack = in.rdy) Người gửi Người nhận +
(out.ack = out.rdy) Bộ đệm

Như ví dụ này gợi ý, có rất ít sự khác biệt giữa việc viết một đặc tả hệ thống hoàn chỉnh tổng hợp và một đặc tả hệ thống mở. Hầu hết các

đặc điểm kỹ thuật không phụ thuộc vào việc chúng tôi chọn. Cả hai chỉ khác nhau ở điểm cuối cùng, khi chúng ta ghép các mảnh lại với nhau.

10.8 Tinh chỉnh giao diện

Tinh chỉnh giao diện là phương pháp thu được đặc tả cấp thấp hơn bằng cách tinh chỉnh các biến của đặc tả cấp cao hơn. Hãy bắt đầu với hai ví dụ và sau đó thảo luận về việc tinh chỉnh giao diện nói chung.

10.8.1 Đồng hồ giờ nhị phân

Khi chỉ định đồng hồ giờ, chúng tôi đã mô tả cách hiển thị của nó bằng một biến hr có giá trị (trong hành vi đáp ứng đặc tả) là một số nguyên từ 1 đến 12.
Giả sử chúng ta muốn chỉ định đồng hồ giờ nhị phân. Đây là đồng hồ giờ để sử dụng trong máy tính, trong đó màn hình bao gồm một thanh ghi bốn bit hiển thị giờ dưới dạng một trong mười hai giá trị 0001, 0010, . . . , 1100. Chúng ta có thể dễ dàng chỉ định một chiếc đồng hồ như vậy từ đầu. Nhưng giả sử chúng ta muốn mô tả nó một cách thân mật cho một người đã biết đồng hồ giờ là gì. Chúng ta có thể nói đơn giản rằng đồng hồ giờ nhị phân giống như đồng hồ giờ thông thường, ngoại trừ giá trị của màn hình được biểu thị dưới dạng nhị phân. Bây giờ chúng tôi chính thức hóa mô tả đó.

Chúng ta bắt đầu bằng việc mô tả ý nghĩa của việc một giá trị bốn bit biểu diễn một số. Có một số cách hợp lý để biểu diễn giá trị bốn bit về mặt toán học. Chúng ta có thể sử dụng chuỗi bốn phần tử, trong TLA+ là hàm có miền xác định là 1 . . 4. Tuy nhiên, một nhà toán học sẽ thấy việc biểu diễn số bit (n + 1) dưới dạng hàm từ 0 sẽ tự nhiên hơn n. . n đến {0, 1}, hàm Chúng ta cũng có thể viết b biểu thị số b[0] 2 + . . . + b[n] 2 xác định BitArrayVal(b) là giá trị số của hàm b bằng cách 0 + b[1] 2 1 N . Trong TLA+, chúng ta có thể {0, 1} là 0 . . 1.

```
BitArrayVal(b)  =
  để n       =  chọn i   Nat : miền b = 0 . . Tôi
       giá trị [i   0 . . n]  = Xác định val[i] bằng b[0]  0 + . . . + b[i]  2 ·
         nếu i = 0 thì b[0]  2  0  2 else b[i]  2 + giá trị[i   1]
  trong val[n]
```

Để chỉ định đồng hồ giờ nhị phân có màn hình được mô tả bằng các bit thay đổi, chúng ta chỉ cần nói rằng BitArrayVal(bit) thay đổi giống như cách mà thông số kỹ thuật HC của đồng hồ giờ cho phép hr thay đổi. Về mặt toán học, điều này có nghĩa là chúng ta thu được thông số kỹ thuật của đồng hồ giờ nhị phân bằng cách thay thế BitArrayVal(bits) cho biến hr trong HC. Trong TLA+, sự thay thế được thể hiện bằng câu lệnh instance. Viết

```
B  = phiên bản HourClock với giờ  BitArrayVal(bits)
```

định nghĩa (trong số những thứ khác) B !HC là công thức thu được từ HC bằng cách thay thế BitArrayVal(bits) cho hr .

Thật không may, đặc điểm kỹ thuật này là sai. Giá trị của BitArrayVal(b) chỉ được chỉ định nếu b là hàm có miền 0 . . n với một số tự nhiên n.
Chúng tôi không biết BitArrayVal({“abc”}) bằng bao nhiêu. Nó có thể bằng 7. Nếu đúng như vậy thì B !HC sẽ cho phép một hành vi trong đó giá trị ban đầu của bit là {“abc”}. Chúng ta phải loại trừ khả năng này bằng cách thay thế hr không phải BitArrayVal(bit), mà thay thế một số biểu thức HourVal(bits) có giá trị là phần tử của 1 . . 12 chỉ khi b là hàm trong [(0 . . 3) {0, 1}]. Ví dụ, chúng ta có thể viết

GiờVal(b) = nếu b [(0 . . 3) {0, 1}] thì BitArrayVal(b)
khác 99

B = phiên bản HourClock với giờ HourVal(bit)

Điều này xác định B !HC là thông số kỹ thuật mong muốn của đồng hồ giờ nhị phân.
Vì HC không hài lòng với hành vi trong đó hr từng giả định giá trị 99, nên B !HC không hài lòng với bất kỳ hành vi nào trong đó các bit từng giả định một giá trị không có trong tập hợp [(0 . . 3) {0, 1}].

Có một cách khác là sử dụng đặc tả HC của đồng hồ giờ để chỉ định đồng hồ giờ nhị phân. Thay vì thay thế giờ trong thông số kỹ thuật đồng hồ giờ, trước tiên chúng tôi chỉ định một hệ thống bao gồm cả đồng hồ giờ và đồng hồ giờ nhị phân có cùng thời gian và sau đó chúng tôi ẩn đồng hồ giờ. Đặc tả này có dạng

(10.7) giờ : IR HC

trong đó IR là một công thức tạm thời đúng, các bit iff luôn là giá trị bốn bit biểu thị giá trị của hr. Công thức này khẳng định rằng các bit là biểu diễn của hr dưới dạng một mảng bốn bit, đối với một số lựa chọn giá trị cho hr thỏa mãn HC.
Sử dụng định nghĩa của HourVal ở trên, chúng ta có thể định nghĩa IR đơn giản bằng (h = HourVal(b)).

Nếu HC được xác định như trong mô-đun HourClock thì (10.7) không thể xuất hiện trong thông số kỹ thuật TLA+. Để HC được xác định trong ngữ cảnh của công thức, biến hr phải được khai báo trong ngữ cảnh đó. Nếu hr đã được khai báo thì nó không thể được sử dụng làm biến ràng buộc của bộ định lượng . Như thường lệ, vấn đề này được giải quyết bằng cách khởi tạo tham số. Thông số BHC TLA+ hoàn chỉnh của đồng hồ giờ nhị phân xuất hiện trong mô-đun BinaryHourClock của Hình 10.3 trên trang tiếp theo.

10.8.2 Tinh chỉnh kênh

Ví dụ thứ hai của chúng tôi về sàng lọc giao diện, hãy xem xét một hệ thống tương tác với môi trường của nó bằng cách gửi các số từ 1 đến 12 qua một kênh.
Chúng tôi tinh chỉnh nó thành một hệ thống cấp thấp hơn giống nhau, ngoại trừ việc nó gửi một số

mô-đun Giờ nhị phânĐồng hồ

mở rộng Naturals

bit thay đổi

H (hr) = instance HourClock

BitArrayVal(b) = để n = chọn i Nat : miền b = 0 . . Tôi

giá trị [i 0 . . n] = Xác định val[i] bằng b[0] 2^0 + . . . + b[i] 2^{i} ·

nếu i = 0 thì b[0] 2 ^{0} 2 else (b[i] 2^{i}) + val[i 1]

trong val[n]

GiờVal(b) = nếu b [(0 . . 3) {0, 1}] thì BitArrayVal(b) else
99

IR(b, h) = (h = HourVal(b))

BHC = giờ : IR(bit, giờ) H (giờ)!HC

Hình 10.3: Đặc điểm kỹ thuật của đồng hồ giờ nhị phân.

dưới dạng một chuỗi bốn bit. Mỗi bit được gửi riêng biệt, bắt đầu từ bit ngoài cùng bên trái (quan trọng nhất). Ví dụ: để gửi số 5, hệ thống cấp thấp hơn sẽ gửi chuỗi bit 0, 1, 0, 1. Chúng tôi chỉ định cả hai kênh bằng mô-đun Kênh của Hình 3.2 trên trang 30, vì vậy mỗi giá trị được gửi phải là được xác nhận trước khi cái tiếp theo có thể được gửi đi.

Giả sử HSpec là đặc tả của hệ thống cấp cao hơn và kênh của nó được biểu thị bằng biến h. Đặt l là biến đại diện cho kênh cấp thấp hơn. Chúng tôi viết đặc tả của hệ thống cấp thấp hơn như

(10.8) h : IR HSpec

trong đó IR chỉ định trình tự các giá trị được gửi qua h dưới dạng hàm của các giá trị được gửi qua l. Việc gửi bit thứ tư trên l được hiểu là gửi số đầy đủ trên h; xác nhận tiếp theo trên l được hiểu là gửi xác nhận trên h; và bất kỳ bước nào khác được hiểu là bước không thay đổi h.

Để xác định IR, chúng tôi khởi tạo Kênh mô-đun cho từng kênh:

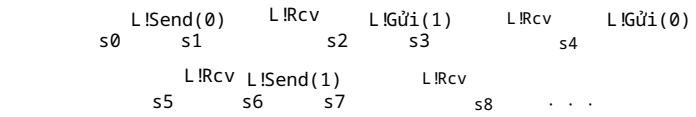
Dữ liệu là tập hợp các giá trị có thể được gửi qua kênh.

H = instance Kênh có chan h, Dữ liệu 1 . . 12 L = instance

Channel with chan l, Data {0, 1}

Do đó, việc gửi một giá trị d qua kênh l được biểu thị bằng bước L!Send(d) và xác nhận việc nhận một giá trị trên kênh h được biểu thị bằng bước H !Rcv.
Hành vi sau đây thể hiện việc gửi và xác nhận số 5, trong đó tôi có

bỏ qua tất cả các bước không thay đổi l:



Hành vi này sẽ thỏa mãn IR iff s6 s7 là bước H !Send(5), s7 s8 là bước H !Rcv và tất cả các bước khác không thay đổi h.

Chúng tôi muốn đảm bảo rằng (10.8) không được thỏa mãn trừ khi l đại diện cho kênh cấp thấp hơn chính xác - ví dụ: (10.8) sẽ sai nếu l được đặt thành một giá trị kỳ lạ nào đó. Do đó, chúng ta sẽ định nghĩa IR sao cho nếu chuỗi các giá trị được giả định bởi l không đại diện cho kênh mà các bit được gửi và xác nhận, thì chuỗi các giá trị của h không thể hiện hành vi đúng của kênh có các số từ 1 đến 12 được gửi đi. Công thức HSpec, và do đó (10.8), khi đó sẽ sai đối với hành vi như vậy.

Công thức IR sẽ có dạng chuẩn cho đặc tả TLA, với điều kiện ban đầu và hành động ở trạng thái tiếp theo. Tuy nhiên, nó chỉ định h là hàm của l; nó không hạn chế l. Do đó, điều kiện ban đầu không chỉ định giá trị ban đầu của l và hành động trạng thái tiếp theo không chỉ định giá trị của l. (Giá trị của l bị ràng buộc ngầm bởi IR, nó khẳng định mối quan hệ giữa các giá trị của h và l, cùng với HSpec liên hợp trong (10.8), ràng buộc giá trị của h.) Đối với hành động trạng thái tiếp theo để chỉ định giá trị được gửi trên h, chúng ta cần một biến nội bộ ghi nhớ những gì đã được gửi trên l kể từ số hoàn chỉnh cuối cùng. Chúng ta để biến bitSent chứa chuỗi bit đã gửi cho số hiện tại. Để thuận tiện, bitSent chứa chuỗi bit theo thứ tự ngược lại, với bit được gửi gần đây nhất ở đầu. Điều này có nghĩa là bit thứ tự cao của số được gửi đầu tiên nằm ở đuôi của bitSent.

Định nghĩa về IR xuất hiện trong mô-đun ChannelRefinement của Hình 10.4 trên trang tiếp theo. Trước tiên, mô-đun xác định ErrorVal là một giá trị tùy ý không phải là giá trị hợp pháp của h. Tiếp theo là định nghĩa của hàm BitSeqToNat. Nếu s là một chuỗi bit thì BitSeqToNat[s] là giá trị số của nó được hiểu là số nhị phân có bit thứ tự thấp nằm ở đầu của s. Ví dụ: BitSeqToNat[0, 1, 1] bằng 6. Sau đó là hai phiên bản của mô-đun Channel.

Sau đây là mô-đun con xác định thông số kỹ thuật bên trong—mô -đun con Việc sử dụng mô-đun con để xác định với biến nội bộ bitSent hiển thị. Vị từ ban đầu của đặc tả nội bộ Init khẳng định rằng nếu l có giá trị ban đầu hợp pháp thì h có thể có bất kỳ giá trị ban đầu hợp pháp nào; nếu không thì h có giá trị không hợp lệ. Ban đầu bitSent là chuỗi trống. Hành động ở trạng thái tiếp theo của đặc tả nội bộ là sự phân tách của ba hành động:

thông số kỹ thuật bên trong đã được giới thiệu trong đặc tả kỹ thuật đồng hồ giờ thời gian thực của Phần 9.1.

SendBit Bước SendBit là bước trong đó một bit được gửi trên l. Nếu bitSent có ít hơn ba phần tử, nghĩa là ít hơn ba bit đã được gửi, thì bit đó sẽ được thêm vào phần đầu của bitSent và h

Tinh chỉnh kênh mô-đun

Mô-đun này xác định sàng lọc giao diện từ kênh h cấp cao hơn, qua đó các số trong 1 . . 12 được gửi đến kênh cấp thấp hơn l trong đó một số được gửi dưới dạng một chuỗi gồm bốn bit, mỗi bit được nhận biết riêng biệt. (Xem mô-đun Kênh trong Hình 3.2 trên trang 30.) Công thức IR đúng nếu chuỗi các giá trị được h giả định thể hiện chế độ xem cấp cao hơn về chuỗi các giá trị được gửi trên l. Nếu chuỗi các giá trị được giả định bởi l không thể hiện việc gửi và xác nhận các bit thì h sẽ nhận một giá trị không hợp lệ.

```
mở rộng các biến Naturals,

Sequences h, l

ErrorVal  = chọn v : v  / [val : 1 . . 12, thứ : {0, 1}, ack : {0, 1}]

BitSeqToNat[s  Seq({0, 1})]  = BitSeqToNat[b0, b1, b2, b3 ] = b0 + 2  (b1 + 2  (b2 + 2  b3))

  nếu s = thì 0 khác Đầu + 2  BitSeqToNat[Đuôi]

H  = instance Kênh có chan  h, Dữ liệu  1 . . 12 L  =

instance Channel with chan  l, Data  {0, 1}
```

H là kênh gửi các số trong 1 . . 12; L là kênh gửi chút ít.

mô-đun bên trong

```
  bit biếnSentSent Chuỗi các bit được gửi cho đến nay cho số hiện tại.

  Ban đầu  =  bitSent
           =  nếu L!Init thì H !Init
                    khác h = ErrorVal

  SendBit  =  b  {0,
                1} :  L!
                Send(b)  nếu
                    Len(bitsSent) < 3 thì  bitSent = b  bitSent
                                không thay đổi h
                    else  bitSent =
                                H !Send(BitSeqToNat[b  bitSent])

  RcvBit  =  L!Rcv
             if bitSent = then H !Rcv khác không
                                thay đổi h
                bit không thay đổi Đã gửi

  Lỗi  =  l = l
             и ((  b  {0, 1} : L!Send(b))  L!
           Rcv)  h = ErrorVal

  Tiếp theo  = SendBit  RcvBit  Lỗi

InnerIR  = Ban đầu  [Tiếp theo]l, h, bitSent

I(bitsSent)  = instance Inner IR

  =  bitSent : I(bitsSent)!InnerIR
```

Xác định giá trị ban đầu của h là hàm của l.

Việc gửi một trong ba bit đầu tiên trên l sẽ thêm nó vào trước bitSent và giữ nguyên h; gửi bit thứ tư đặt lại bitSent và gửi số đầy đủ trên h.

Một hành động Rcv trên l gây ra một hành động Rcv trên h nếu nó diễn ra sau khi gửi bit thứ tư .

Một hành động bất hợp pháp trên l đặt h thành ErrorVal.

Hình 10.4: Tinh chỉnh kênh.

được giữ nguyên không thay đổi. Mặt khác, giá trị được biểu thị bằng bốn bit đã gửi cho đến nay, bao gồm cả bit hiện tại, sẽ được gửi trên h và bitSent được đặt lại thành .

RcvBit Bước RcvBit là bước trong đó một xác nhận được gửi trên l. Nó thể hiện việc gửi một xác nhận trên hiff, đây là một xác nhận của bit thứ tư , điều này đúng nếu bitSent là chuỗi trống.

Lỗi Bước Lỗi là bước trong đó xảy ra thay đổi bất hợp pháp đối với l. Nó đặt h thành một giá trị bất hợp pháp.

Đặc tả bên trong InnerIR có dạng thông thường. (Không có yêu cầu về tính sống động.) Sau đó, mô-đun bên ngoài khởi tạo mô-đun con bên trong với tham số là bitSent và nó xác định IR bằng InnerIR với bitSent bị ẩn.

Bây giờ, giả sử chúng ta có một mô-đun HigherSpec xác định đặc tả HSpec của một hệ thống giao tiếp bằng cách gửi các số từ 1 đến 12 qua kênh hchan. Như sau, chúng ta thu được đặc tả LSpec cấp thấp hơn trong đó các số được gửi dưới dạng chuỗi bit trên kênh lchan. Đầu tiên chúng ta khai báo lchan và tất cả các biến, hằng của mô-đun HigherSpec ngoại trừ hchan.
Sau đó chúng tôi viết

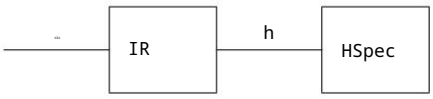
```
HS(hchan)   = instance HigherSpec CR(h)
  = instance ChannelRefinement with l    lchan LSpec   =
h : CR(h)!IR    HS(h)!HSpec
```

10.8.3 Tinh chỉnh giao diện nói chung

Trong các ví dụ về đồng hồ nhị phân và sàng lọc kênh, chúng tôi đã xác định đặc tả LSpec cấp thấp hơn theo HSpec cấp cao hơn như

(10.9) LSpec = h : IR HSpec

trong đó h là biến tự do của HSpec và IR là mối quan hệ giữa h và biến cấp thấp l của LSpec. Chúng ta có thể xem thông số kỹ thuật bên trong IR HSpec dưới dạng thành phần của hai thành phần, như được hiển thị ở đây:



Chúng ta có thể coi IR là đặc tả của một thành phần chuyển đổi hành vi cấp thấp hơn của l thành hành vi cấp cao hơn của h. Công thức IR được gọi là sàng lọc giao diện.

Trong cả hai ví dụ, việc sàng lọc giao diện đều độc lập với đặc tả hệ thống. Nó chỉ phụ thuộc vào sự thể hiện của giao diện - nghĩa là vào cách sự tương tác giữa hệ thống và môi trường của nó được thể hiện. Nói chung, để sàng lọc giao diện IR độc lập với hệ thống bằng cách sử dụng giao diện, nó sẽ mô tả hành vi của biến giao diện cấp cao hơn h cho bất kỳ hành vi nào của biến cấp thấp l. Nói cách khác, với bất kỳ chuỗi nào các giá trị của l, cần phải có một chuỗi giá trị nào đó của h thỏa mãn IR. Cái này được biểu thị bằng toán học bằng yêu cầu công thức h : IR phải hợp lệ—nghĩa là đúng cho mọi hành vi.

Cho đến nay, tôi đã thảo luận về việc tinh chỉnh một biến giao diện h bằng một biến l. Điều này khái quát một cách rõ ràng đến việc sàng lọc một bộ sưu tập của các biến cấp cao hơn h1, . . . , hn theo các biến l lm. Giao diện sàng lọc IR xác định các giá trị của hi theo các giá trị của lj và có lẽ còn có những biến số khác nữa. Công thức (10.9) được thay thế bằng

LSpec = h1, . . . , hn : IR HSpec

Một loại sàng lọc giao diện đặc biệt đơn giản là sàng lọc dữ liệu, trong IR có dạng P, trong đó P là vị từ trạng thái biểu thị giá trị của các biến cấp cao hơn h1, . . . , hn là hàm của các giá trị của các biến cấp thấp hơn l ừm. Việc sàng lọc giao diện trong đồng hồ nhị phân của chúng tôi 1, . . . , đặc tả là sàng lọc dữ liệu, trong đó P là vị từ hr = HourVal(bits). Một ví dụ khác, hai thông số kỹ thuật của giao diện kênh không đồng bộ trong Chương 3 có thể được lấy từ nhau bằng cách tinh chỉnh giao diện. Thông số kỹ thuật của mô-đun Kênh (trang 30) tương đương với thông số kỹ thuật thu được dưới dạng sàng lọc dữ liệu của thông số kỹ thuật Spec của Mô-đun AsynchInterface (trang 27) bằng cách đặt P bằng

(10.10) chan = [val val, rdy rdy, ack ack]

Công thức này khẳng định rằng chan là một bản ghi có trường val là giá trị của biến val, trường rdy là giá trị của biến rdy và trường ack là giá trị giá trị của biến ack. Ngược lại, thông số kỹ thuật của AsynchInterface mô-đun tương đương với việc sàng lọc dữ liệu về thông số kỹ thuật của Kênh mô-đun. Trong trường hợp này, việc xác định vị từ trạng thái P hơi phức tạp. Rõ ràng lựa chọn là đặt P là công thức GoodVals được xác định bởi

GoodVals = val = chan.val
rdy = chan.rdy
ack = chan.ack

Tuy nhiên, điều này có thể khẳng định rằng val, rdy và ack có giá trị tốt ngay cả khi chan có giá trị không hợp lệ—ví dụ: nếu đó là bản ghi có nhiều hơn ba trường. Thay vào đó, chúng ta đặt P bằng

if chan [val : Data, rdy : {0, 1}, ack : {0, 1}] thì GoodVals
khác BadVals

trong đó BadVals khẳng định rằng val, rdy và ack có một số giá trị không hợp lệ—
nghĩa là các giá trị không thể có trong một hành vi thỏa mãn công thức Spec của mô-
đun AsynchInterface. (Chúng ta không cần thủ thuật như vậy khi định nghĩa chan là
hàm của val, rdy và ack vì (10.10) ngụ ý rằng giá trị của chan là hợp pháp nếu giá
trị của cả ba biến val, rdy và ack đều hợp pháp.)

Tinh chỉnh dữ liệu là hình thức sàng lọc giao diện đơn giản nhất. Trong sàng
lọc giao diện phức tạp hơn, giá trị của các biến cấp cao hơn không thể được biểu
thị dưới dạng hàm của các giá trị hiện tại của các biến cấp thấp hơn. Trong ví dụ
sàng lọc kênh ở Mục 10.8.2, số được gửi trên kênh cấp cao hơn phụ thuộc vào giá
trị của các bit đã được gửi trước đó trên kênh cấp thấp hơn, không chỉ phụ thuộc
vào trạng thái hiện tại của kênh cấp thấp hơn.

Chúng tôi thường tinh chỉnh cả hệ thống và giao diện của nó cùng một lúc. Ví
dụ: chúng ta có thể triển khai thông số kỹ thuật H của hệ thống giao tiếp bằng cách
gửi số qua kênh với thông số kỹ thuật cấp thấp hơn LImpl của hệ thống gửi các bit
riêng lẻ. Trong trường hợp này, LImpl không được lấy từ HSpec bằng cách sàng lọc
giao diện. Đúng hơn, LImpl triển khai một số đặc tả LSpec thu được từ HSpec bằng
IR tinh chỉnh giao diện. Trong trường hợp đó, chúng tôi nói rằng LImpl triển khai
HSpec theo IR sàng lọc giao diện.

10.8.4 Thông số kỹ thuật hệ thống mở

Cho đến nay, chúng ta đã xem xét việc sàng lọc giao diện cho các thông số kỹ thuật
của hệ thống hoàn chỉnh. Bây giờ chúng ta hãy xem điều gì sẽ xảy ra nếu đặc tả HSpec
ở mức cao hơn là loại đặc tả hệ thống mở được thảo luận trong Phần 10.7 ở trên. Để
đơn giản, chúng tôi xem xét việc sàng lọc một biến giao diện cấp cao hơn h bằng
một biến cấp độ thấp hơn l. Việc khái quát hóa cho nhiều biến hơn sẽ rõ ràng.

Trước tiên hãy giả sử rằng HSpec là một thuộc tính an toàn, không có điều kiện sống động.
Như đã giải thích trong Phần 10.7, mỗi đặc tả đều thay đổi thành h đối với hệ thống
hoặc môi trường. Do đó, bất kỳ thay đổi nào đối với biến giao diện cấp thấp l tạo
ra thay đổi đối với h đều được quy cho hệ thống hoặc môi trường. Một thay đổi xấu
đối với h được cho là do môi trường khiến HSpec đúng; một thay đổi xấu được cho
là do hệ thống khiến HSpec sai.
Do đó, (10.9) định nghĩa LSpec là một đặc tả hệ thống mở. Để đây là một thông số kỹ
thuật hợp lý, IR sàng lọc giao diện phải đảm bảo rằng cách thức thay đổi l được quy
cho hệ thống hoặc môi trường là hợp lý.

Nếu HSpec chứa các điều kiện sống động thì việc sàng lọc giao diện có thể tinh
tế hơn. Giả sử IR là sàng lọc giao diện được xác định trong mô-đun ChannelRefinement
của Hình 10.4 trên trang 162 và giả sử rằng HSpec yêu cầu hệ thống cuối cùng gửi một
số số trên h. Hãy xem xét một hành vi trong đó hệ thống gửi bit đầu tiên của một số
trên l, nhưng môi trường không bao giờ thừa nhận nó. Theo IR sàng lọc giao diện,
hành vi này được hiểu là hành vi trong đó h không bao giờ thay đổi. Hành vi như
vậy không thỏa mãn điều kiện sống

của HSpec. Do đó, nếu LSpec được xác định bởi (10.9), thì việc môi trường không thực hiện được điều gì đó có thể khiến LSpec bị vi phạm mà không phải do lỗi của hệ thống.

Trong ví dụ này, chúng tôi muốn môi trường bị lỗi nếu nó khiến hệ thống tạm dừng do không xác nhận bất kỳ bit nào trong ba bit đầu tiên của số được hệ thống gửi. (Việc xác nhận bit thứ tư được IR hiểu là sự xác nhận giá trị được gửi trên h, do đó, việc đổ lỗi cho sự vắng mặt của nó sẽ được gán chính xác cho môi trường.) Đặt lỗi cho môi trường có nghĩa là làm cho LSpec đúng. Chúng ta có thể đạt được điều này bằng cách sửa đổi (10.9) để xác định LSpec như sau:

(10.11) LSpec = Độ sống h : IR HSpec

trong đó Liveness là một công thức yêu cầu bất kỳ bit nào được gửi trên l, ngoại trừ bit cuối cùng của một số, cuối cùng phải được xác nhận. Tuy nhiên, nếu l được đặt thành một giá trị không hợp lệ thì chúng ta muốn phần an toàn của đặc tả xác định ai chịu trách nhiệm. Vì vậy, chúng tôi muốn Liveness là đúng trong trường hợp này.

Chúng tôi xác định Liveness theo các biến bên trong h và bitSent, có liên quan đến l theo công thức InnerIR từ mô-đun con Inner của mô-đun ChannelRefinement. (Hãy nhớ rằng l phải là biến tự do duy nhất của LSpec.) Hành động xác nhận việc nhận một trong ba bit đầu tiên của số là RcvBit (bitsSent =). Tính công bằng yếu kém của hành động này khẳng định rằng những xác nhận cần thiết cuối cùng phải được gửi đi. Đối với trường hợp các giá trị không hợp lệ, hãy nhớ lại rằng việc gửi một giá trị không hợp lệ lên l sẽ khiến h bằng ErrorVal.

Chúng tôi muốn Liveness là đúng nếu điều này từng xảy ra, nghĩa là nếu cuối cùng nó cũng xảy ra. Do đó, chúng tôi thêm định nghĩa sau vào mô-đun con Bên trong của mô-đun ChannelRefinement:

```
Sự sống bên trong = IR bên trong
                        WFl, h, bitSent (RcvBit (bitsSent = )) (h =
                    ErrorVal)
```

Để xác định Liveness, chúng ta phải ẩn h và bitSent trong InnerLiveness. Chúng ta có thể làm điều này, trong bối cảnh l được khai báo, như sau:

```
ICR(h) = instance ChannelRefinement Liveness =
    h, bitSent : ICR(h)!I(bitsSent)!InnerLiveness
```

Bây giờ, giả sử chính môi trường gửi các số qua h và hệ thống phải xác nhận việc nhận chúng rồi xử lý chúng theo một cách nào đó. Trong trường hợp này, chúng tôi muốn việc không thừa nhận một chút là lỗi hệ thống. Vì vậy, LSpec sẽ sai nếu Liveness là sai. Sau đó, đặc điểm kỹ thuật phải được

```
LSpec = Độ sống động ( h : IR HSpec)
```

Vì h không xuất hiện tự do trong Liveness nên định nghĩa này tương đương với

```
LSpec = h : Độ sống động IR HSpec
```

có dạng (10.9) nếu IR sàng lọc giao diện của (10.9) đư ợc coi là Liveness IR. Nói cách khác, chúng ta có thể biến điều kiện sống động thành một phần của quá trình sàng lọc giao diện. (Trong trư ờng hợp này, chúng ta có thể đơ n giản hóa định nghĩa bằng cách thêm trực tiếp tính sống động vào InnerIR.)

Nói chung, nếu HSpec là một đặc tả hệ thống mở mô tả tính sống động cũng như độ an toàn thì việc sàng lọc giao diện có thể phải có dạng công thức (10.11). Cả Độ sống và điều kiện độ sống của IR có thể phụ thuộc vào những thay đổi nào đối với biến giao diện cấp thấp l đư ợc quy cho hệ thống và thay đổi nào cho môi trư ờng. Đối với việc sàng lọc kênh, điều này có nghĩa là chúng sẽ phụ thuộc vào việc hệ thống hoặc môi trư ờng có gửi các giá trị trên kênh hay không.

10.9 Bạn có nên sáng tác không?

Khi chỉ định một hệ thống, chúng ta nên viết một đặc tả nguyên khối với một hành động trạng thái tiếp theo duy nhất, một thành phần hệ thống đóng là sự kết hợp các đặc tả của các thành phần riêng lẻ hay một đặc tả hệ thống mở? Câu trả lời là: nó thư ờng tạo ra rất ít sự khác biệt. Đối với một hệ thống thực, việc định nghĩa hành động của các thành phần sẽ mất hàng trăm hoặc hàng nghìn dòng. Các dạng đặc tả khác nhau chỉ khác nhau ở một vài dòng trong đó chúng ta tập hợp các vị từ ban đầu và hành động ở trạng thái tiếp theo thành công thức cuối cùng.

Nếu bạn đang viết một đặc tả từ đầu, có lẽ tốt hơ n nên viết một đặc tả nguyên khối. Nó thư ờng dễ hiểu hơ n. Tất nhiên, có những trư ờng hợp ngoại lệ. Chúng tôi viết một đặc tả thời gian thực dư ới dạng sự kết hợp của một đặc tả không xác định thời gian và các ràng buộc về thời gian; việc mô tả các thay đổi đối với các biến hệ thống và bộ định thời bằng một hành động trạng thái tiếp theo thư ờng làm cho thông số kỹ thuật khó hiểu hơ n.

Viết một đặc tả tổng hợp có thể hợp lý khi bạn bắt đầu từ một đặc tả hiện có. Nếu bạn đã có thông số kỹ thuật của một thành phần, bạn có thể muốn viết thông số kỹ thuật riêng của thành phần kia và soạn thảo hai thông số kỹ thuật. Nếu bạn có đặc tả cấp cao hơ n, bạn có thể muốn viết một phiên bản cấp thấp hơ n làm sàng lọc giao diện. Tuy nhiên, đây là những tình huống khá hiếm. Hơ n nữa, có thể dễ dàng sửa đổi đặc tả ban đầu hoặc sử dụng lại nó theo cách khác. Ví dụ: thay vì kết hợp một thành phần mới với đặc tả của thành phần hiện có, bạn có thể chỉ cần đư a định nghĩa về hành động trạng thái tiếp theo của thành phần hiện có, với một câu lệnh mở rộng hoặc câu lệnh phiên bản, như một phần của đặc tả mới.

Thành phần cung cấp một cách mới để viết đặc tả hệ thống hoàn chỉnh; nó không thay đổi đặc điểm kỹ thuật. Do đó, việc lựa chọn giữa thông số kỹ thuật tổng hợp và thông số kỹ thuật nguyên khối cuối cùng là vấn đề sở thích. Các bố cục trạng thái rời rạc thư ờng đơ n giản và không có vấn đề gì. Các tác phẩm ở trạng thái chia sẻ có thể phức tạp và cần đư ợc chăm sóc.

Các đặc tả hệ thống mở đưa ra một loại đặc tả khác về mặt toán học. Đặc tả hệ thống đóng E M và đối tác hệ thống mở E + M của nó là không tương đương. Nếu chúng ta thực sự muốn một đặc tả đóng vai trò như một hợp đồng pháp lý giữa người dùng và người triển khai thì chúng ta phải viết một đặc tả hệ thống mở. Chúng ta cũng cần các đặc tả hệ thống mở nếu chúng ta muốn xác định và lý giải về các hệ thống được xây dựng bằng cách kết hợp các thành phần sẵn có với các đặc tả có sẵn. Tất cả những gì chúng ta có thể giả định về một thành phần như vậy là nó đáp ứng hợp đồng giữa người xây dựng hệ thống và nhà cung cấp và hợp đồng như vậy chỉ có thể được chính thức hóa dưới dạng đặc tả hệ thống mở. Tuy nhiên, bạn khó có thể gặp phải những thông số kỹ thuật thành phần sẵn có vào đầu thế kỷ XXI. Trong tương lai gần, các đặc tả hệ thống mở có thể chỉ được quan tâm về mặt lý thuyết.