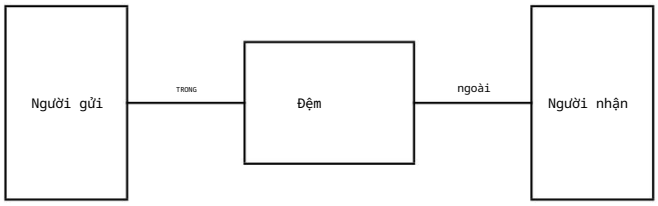
Chương 4

một FIFO

Ví dụ tiếp theo của chúng ta là bộ đệm FIFO, gọi tắt là FIFO—một thiết bị với tiến trình bên gửi truyền một chuỗi các giá trị đến bên nhận. Người gửi và bộ thu sử dụng hai kênh vào và ra để giao tiếp với bộ đệm:



Các giá trị được gửi vào và ra bằng giao thức không đồng bộ được chỉ định bởi Mô-đun kênh của Hình 3.2 trên trang 30. Thông số kỹ thuật của hệ thống sẽ cho phép hành vi với bốn loại bước không nói lắp: Các bước Gửi và Rcv trên cả hai kênh vào và kênh ra.

4.1 Đặc điểm kỹ thuật bên trong

Đặc tả của FIFO trước tiên mở rộng các mô-đun Naturals và Sequences. Mô-đun Sequences xác định các hoạt động trên các chuỗi hữu hạn. Chúng tôi đại diện cho một dãy hữu hạn dưới dạng một bộ nên dãy ba số 3, 2, 1 là bộ ba 3, 2, 1. Mô-đun Trình tự xác định các toán tử sau trên trình tự:

Seq(S) Tập hợp tất cả các dãy phần tử của tập S. Ví dụ: 3, 7 là một phần tử của Seq(Nat).

Head(s) Phần tử đầu tiên của dãy s. Ví dụ: Đầu(3, 7) bằng 3.

(Các) đuôi Đuôi của chuỗi s, bao gồm s đã bị loại bỏ phần đầu.
Ví dụ: Đuôi(3, 7) bằng 7.

Nối(s, e) Chuỗi thu được bằng cách thêm phần tử e vào đuôi của chuỗi s. Ví dụ:
Nối (3, 7, 3) bằng 3, 7, 3.

s t Trình tự thu được bằng cách nối các chuỗi s và t. Ví dụ: 3, 7 3
bằng 3, 7, 3. (Chúng tôi nhập trong ascii là \o.)

Len(s) Độ dài của dãy s. Ví dụ: Len(3, 7) bằng 2.

Đặc tả của FIFO tiếp tục bằng cách khai báo Thông báo không đổi, đại diện cho tập hợp tất cả các thông báo có thể được gửi.1 Sau đó, nó khai báo các biến.
Có ba biến: in và out, đại diện cho các kênh và biến thứ ba q đại diện cho hàng đợi các tin nhắn được lưu vào bộ đệm. Giá trị của q là chuỗi các tin nhắn đã được gửi bởi người gửi nhưng chưa được người nhận nhận. (Phần 4.3 nói thêm về biến q bổ sung này.)

Chúng tôi muốn sử dụng các định nghĩa trong mô-đun Kênh để chỉ định các hoạt động trên các kênh vào và ra. Điều này yêu cầu hai phiên bản của mô-đun đó—một phiên bản trong đó biến chan của mô-đun Kênh được thay thế bằng biến in của mô-đun hiện tại của chúng tôi và phiên bản còn lại trong đó chan được thay thế bằng biến out. Trong cả hai trường hợp, Dữ liệu không đổi của mô-đun Kênh được thay thế bằng Thông báo. Chúng tôi có được trường hợp đầu tiên trong số này với câu lệnh

```
InChan = instance Channel with Data Message, chan in
```

Đối với mọi ký hiệu σ được xác định trong Kênh mô-đun, điều này xác định InChan !σ có cùng ý nghĩa trong mô-đun hiện tại như σ có trong Kênh mô-đun, ngoại trừ Thông báo được thay thế cho Dữ liệu và được thay thế cho chan. Ví dụ: câu lệnh này định nghĩa InChan !TypeInvariant bằng

```
trong [val : Tin nhắn, rdy : {0, 1}, ack : {0, 1}]
```

(Câu lệnh không xác định InChan !Data vì Dữ liệu được khai báo, không được xác định, trong Kênh mô-đun.) Chúng tôi giới thiệu phiên bản thứ hai của mô-đun Kênh với câu lệnh tương tự

```
OutChan = instance Channel with Data Message, chan out
```

Trạng thái ban đầu của các kênh vào và ra được chỉ định bởi InChan !Init và OutChan ! Init. Ban đầu, không có tin nhắn nào được gửi hoặc nhận, vì vậy q nên

[1] Tôi thích sử dụng danh từ số ít như Message hơn là danh từ số nhiều như Messages cho tên của một nhóm. Bằng cách đó, trong biểu thức m Thông báo có thể đọc được là a. Đây là quy ước tương tự mà hầu hết các lập trình viên sử dụng để đặt tên kiểu.

bằng dãy trống. Chuỗi trống là bộ 0 (chỉ có một,
và nó được viết), vì vậy chúng tôi xác định vị từ ban đầu là

```
Ban đầu  =   InChan !Ban đầu
             OutChan !Ban đầu
             q =
```

Tiếp theo chúng ta xác định kiểu bất biến. Các kiểu bất biến cho vào và ra đến từ
mô-đun Kênh và loại q là tập hợp các chuỗi thông báo hữu hạn.
Do đó, loại bất biến cho đặc tả FIFO là

```
LoạiBất biến  =   InChan !TypeInvariant
                      OutChan !TypeInvariant
                      q    Seq(Tin nhắn)
```

Bốn loại bước không nói lắp mà hành động ở trạng thái tiếp theo cho phép được mô tả
bằng bốn hành động:

SSend(msg) Người gửi gửi tin nhắn trên kênh trong.

BufRcv Bộ đệm nhận thông báo từ kênh trong và gắn nó vào đuôi của q.

BufGửi Bộ đệm xóa thông báo khỏi phần đầu của q và gửi
nó trên kênh ra.

Rrcv Người nhận nhận được tin nhắn từ kênh ra.

Các định nghĩa về những hành động này, cùng với phần còn lại của đặc tả, đều có trong
module InnerFIFO của Hình 4.1 ở trang tiếp theo. Lý do của tính từ
Nội bộ được giải thích trong Phần 4.3 dưới đây.

4.2 Đã kiểm tra việc khởi tạo

Câu lệnh instance hiếm khi được sử dụng ngoại trừ trong một thành ngữ dùng để ẩn các biến,
được mô tả trong Phần 4.3. Vì vậy, hầu hết độc giả có thể bỏ qua phần này và đi
trực tiếp đến trang 41.

4.2.1 Khởi tạo là sự thay thế

Xem xét định nghĩa Tiếp theo trong Kênh mô-đun (trang 30). Chúng ta có thể loại bỏ
mọi ký hiệu được xác định xuất hiện trong định nghĩa đó bằng cách sử dụng định nghĩa
của ký hiệu. Ví dụ: chúng ta có thể loại bỏ biểu thức Send(d) bằng cách khai triển
định nghĩa của Gửi. Chúng ta có thể lặp lại quá trình này. Ví dụ: “-” đó
xuất hiện trong biểu thức 1 @ (thu được bằng cách mở rộng định nghĩa của Gửi)

```
mô-đun InnerFIFO
mở rộng Naturals, Sequences
constant Biến thông
báo in, out, q InChan
  = instance Channel with Data    Message, chan    in OutChan   =
instance Channel with Data    Message, chan    out Init   =    InChan !
Init    OutChan !Init
         q =

TypeInvariant   =    InChan !TypeInvariant
                     OutChan !TypeInvariant    q
                        Seq(Message)

SSend(tin nhắn)   =    InChan !Send(msg)        Gửi tin nhắn trên kênh vào.
                      không thay đổi, q

BufRcv   =    InChan !Rcv                      Nhận tin nhắn từ kênh vào và nối nó
              q = Nối (q, in.val)                vào đuôi q.
                không thay đổi

BufSend   =    q =                             Chỉ được kích hoạt nếu q không trống.
               OutChan !Send(Head(q))    q     Gửi Head(q) trên kênh ra và xóa
               = Tail(q)                         nó khỏi q.
               không thay đổi trong

RRcv   =    OutChan !Rcv                       Nhận tin nhắn từ kênh ra.
            không thay đổi trong,

q Tiếp theo   =      msg    Tin nhắn:
           SSend(msg)
             BufRcv
             BufSend    RRcv

Thông số   = Ban đầu     [Tiếp theo]vào, ra, q
định lý Spec     TypeInvariant
```

Hình 4.1: Đặc tả của FIFO, với biến nội bộ q hiển thị.

có thể được loại bỏ bằng cách sử dụng định nghĩa “-” từ mô-đun Naturals.
Tiếp tục theo cách này, cuối cùng chúng ta thu được định nghĩa cho Tiếp theo chỉ dựa trên các toán tử tích hợp của TLA+ và các tham số Dữ liệu và chan của mô-đun Kênh. Chúng tôi coi đây là định nghĩa “thực” của Tiếp theo trong Kênh mô-đun. Tuyên bố

```
InChan   = instance Channel with Data    Message, chan    in
```

trong mô-đun InnerFIFO định nghĩa InChan !Next là công thức thu được từ định nghĩa thực tế này của Next bằng cách thay thế Message cho Data và in cho chan.
Điều này xác định InChan !Next chỉ xét về các toán tử tích hợp của TLA+ và các tham số Message và in của mô-đun InnerFIFO.

Bây giờ chúng ta hãy xem xét một câu lệnh thể hiện tùy ý

```
IM   = trường hợp M với p1    e1, . . .        , pn    vi
```

Đặt Σ là ký hiệu được xác định trong mô-đun M và đặt d là định nghĩa “thực” của nó. Câu lệnh thực thể định nghĩa IM !Σ để có định nghĩa thực sự của nó là biểu thức thu được từ d bằng cách thay thế tất cả các trường hợp của pi bằng biểu thức ei cho mỗi i. Định nghĩa của IM !Σ chỉ được chứa các tham số (hằng và biến được khai báo) của mô-đun hiện tại, không chứa các tham số của mô-đun M . Do đó, pi phải bao gồm tất cả các tham số của mô-đun M . Ei phải là các biểu thức có ý nghĩa trong mô-đun hiện tại .

4.2.2 Khởi tạo tham số

Đặc tả FIFO sử dụng hai phiên bản của mô-đun Kênh—một phiên bản được thay thế cho chan và phiên bản còn lại được thay thế cho chan. Thay vào đó, chúng ta có thể sử dụng một phiên bản được tham số hóa bằng cách đặt câu lệnh sau vào mô-đun InnerFIFO:

```
Chan(ch)   = instance Channel with Data    Message, chan    ch
```

Đối với bất kỳ ký hiệu Σ nào được xác định trong Kênh mô-đun và bất kỳ biểu thức exp nào, điều này xác định Chan(exp)!Σ bằng công thức Σ với Thông báo được thay thế cho Dữ liệu và exp được thay thế cho chan. Khi đó, hành động Rcv trên kênh vào có thể được ghi là Chan(in)!Rcv và hành động Gửi(tin nhắn) trên kênh ra có thể được ghi là Chan(out)!Send(msg).

Việc khởi tạo ở trên định nghĩa Chan !Send là một toán tử có hai đối số. Viết Chan(out)!Send(msg) thay vì Chan !Send(out, msg) chỉ là một đặc điểm riêng của cú pháp. Không có gì lạ hơn cú pháp của các toán tử trung tố, chúng ta viết a + b thay vì +(a, b).

Việc khởi tạo tham số hóa hầu như chỉ được sử dụng trong thành ngữ TLA+ để ẩn biến, được mô tả trong Phần 4.3. Bạn có thể sử dụng thành ngữ đó mà không hiểu nó, vì vậy bạn có thể không cần biết gì về việc khởi tạo tham số hóa.

4.2.3 Thay thế ngầm định

Việc sử dụng Thông báo làm tên cho tập hợp các giá trị được truyền trong đặc tả FIFO hơi lạ vì chúng ta vừa sử dụng tên Dữ liệu cho tập hợp tương tự trong đặc tả kênh không đồng bộ. Giả sử chúng ta đã sử dụng Dữ liệu thay cho Tin nhắn làm tham số không đổi của mô-đun InnerFIFO. Tuyên bố khởi tạo đầu tiên sau đó sẽ là

```
InChan  = instance Channel with Data  Data, chan  in
```

Dữ liệu thay thế Dữ liệu chỉ ra rằng tham số không đổi Dữ liệu của mô-đun được khởi tạo Kênh được thay thế bằng biểu thức Dữ liệu của mô-đun hiện tại. TLA+ cho phép chúng ta loại bỏ bất kỳ sự thay thế nào có dạng Σ Σ cho ký hiệu Σ. Vì vậy, câu lệnh trên có thể được viết là InChan = instance Channel

```
với chan  in
```

Chúng tôi biết có sự thay thế Dữ liệu Dữ liệu ngụ ý vì một câu lệnh phiên bản phải có sự thay thế cho mọi tham số của mô-đun được khởi tạo. Nếu tham số p nào đó không có sự thay thế rõ ràng thì sẽ có một sự thay thế ngầm p p. Điều này có nghĩa là câu lệnh thực thể phải nằm trong phạm vi khai báo hoặc định nghĩa ký hiệu p.

Việc khởi tạo một mô-đun bằng loại thay thế ngầm này là khá phổ biến. Thông thường, mọi tham số đều có một sự thay thế ngầm, trong trường hợp đó danh sách các sự thay thế rõ ràng trống. Sau đó, with được bỏ qua.

4.2.4 Khởi tạo mà không đổi tên

Cho đến nay, tất cả các phiên bản mà chúng tôi đã sử dụng đều liên quan đến việc đổi tên. Ví dụ: lần khởi tạo đầu tiên của mô-đun Kênh đổi tên biểu tượng được xác định Gửi thành InChan !Send. Kiểu đổi tên này là cần thiết nếu chúng ta đang sử dụng nhiều phiên bản của mô-đun hoặc một phiên bản được tham số hóa duy nhất. Hai phiên bản InChan !Init và OutChan !Init của Init trong mô-đun InnerFIFO là các for-mulas khác nhau nên chúng cần có tên khác nhau.

Đôi khi chúng ta chỉ cần một phiên bản duy nhất của một mô-đun. Ví dụ: giả sử chúng ta đang chỉ định một hệ thống chỉ có một kênh không đồng bộ duy nhất. Sau đó, chúng tôi chỉ cần một phiên bản của Kênh, do đó chúng tôi không phải đổi tên các ký hiệu được khởi tạo. Trong trường hợp đó, chúng ta có thể viết một cái gì đó như

```
instance Kênh có Dữ liệu  D, chan  x
```

Điều này khởi tạo Kênh mà không cần đổi tên mà có sự thay thế. Do đó, nó định nghĩa Rcv là công thức có cùng tên từ mô-đun Kênh, ngoại trừ D được thay thế cho Dữ liệu và x được thay thế cho chan. Phải xác định các biểu thức được thay thế cho các tham số của mô-đun đã khởi tạo. Vì vậy, câu lệnh instance này phải nằm trong phạm vi định nghĩa hoặc khai báo của D và x .

4.3 Ẩn hàng đợi

Mô-đun InnerFIFO của Hình 4.1 định nghĩa Spec là Init [Next]... , loại công thức mà chúng ta đã quen sử dụng như một đặc tả hệ thống. Tuy nhiên, công thức Spec mô tả giá trị của biến q cũng như của các biến vào và ra. Hình ảnh hệ thống FIFO tôi vẽ ở trang 35 chỉ hiển thị các kênh vào và ra; nó không hiển thị bất cứ điều gì bên trong các hộp. Đặc tả của FIFO chỉ mô tả các giá trị được gửi và nhận trên các kênh. Biến q, đại diện cho những gì đang diễn ra bên trong hộp có nhãn Bộ đệm, được sử dụng để chỉ định giá trị nào được gửi và nhận. Nó là một biến nội bộ và trong đặc tả cuối cùng, nó phải được ẩn đi.

Trong TLA, chúng tôi ẩn một biến bằng bộ định lượng tồn tại của logic thời gian. Công thức x : F đúng với một hành vi nếu tồn tại một số chuỗi giá trị—một chuỗi giá trị ở mỗi trạng thái của hành vi—có thể được gán cho biến x để làm cho công thức F trở thành đúng. (Ý nghĩa của được xác định chính xác hơn ở Mục 8.8.)

Cách rõ ràng để viết đặc tả FIFO trong đó q bị ẩn là sử dụng công thức q : Spec. Tuy nhiên, chúng tôi không thể đưa định nghĩa này vào mô-đun InnerFIFO vì q đã được khai báo ở đó và công thức q : . . . sẽ tuyên bố lại nó. Thay vào đó, chúng tôi sử dụng một mô-đun mới với sự khởi tạo tham số của mô-đun InnerFIFO (xem Phần 4.2.2 trên trang 39):

```
──────────────────────── mô-đun FIFO ────────────────────────

hằng số Thông báo
biến vào, ra

Bên trong (q)  = instance InnerFIFO

Spec  =  q : Bên trong (q)!Spec

──────────────────────────────────────────────────────────────
```

Lưu ý rằng câu lệnh instance là viết tắt của

```
Nội (q)  = thể hiện InnerFIFO
               với q  q, vào  vào, ra  ra, Tin nhắn  Tin nhắn
```

Tham số biến q của mô-đun InnerFIFO được khởi tạo bằng tham số q của định nghĩa Inner . Các tham số khác của mô-đun InnerFIFO được khởi tạo bằng các tham số của mô-đun FIFO.

Nếu điều này có vẻ khó hiểu, đừng lo lắng về nó. Chỉ cần học thành ngữ TLA+ để ẩn các biến được sử dụng ở đây và hài lòng với ý nghĩa trực quan của nó. Trên thực tế, đối với hầu hết các ứng dụng, không cần phải ẩn các biến trong đặc tả. Bạn chỉ có thể viết đặc tả bên trong và ghi chú trong phần nhận xét những biến nào sẽ được coi là hiển thị và biến nào là nội bộ (ẩn).

4.4 FIFO giới hạn

Chúng tôi đã chỉ định một FIFO không giới hạn—một bộ đệm có thể chứa số lượng tin nhắn không giới hạn. Bất kỳ hệ thống thực nào đều có lượng tài nguyên hữu hạn, do đó nó chỉ có thể chứa một số lượng giới hạn các tin nhắn truyền tải. Trong nhiều tình huống, chúng tôi muốn loại bỏ giới hạn về tài nguyên và mô tả một hệ thống dưới dạng FIFO không giới hạn. Trong các tình huống khác, chúng ta có thể quan tâm đến giới hạn đó. Sau đó, chúng tôi muốn tăng cường đặc điểm kỹ thuật của mình bằng cách đặt giới hạn N cho số lượng tin nhắn chưa xử lý.

Đặc tả của FIFO bị chặn khác với đặc tả của chúng tôi về FIFO không bị chặn chỉ ở hành động đó BufRcv không được bật trừ khi có ít hơn N thông báo trong bộ đệm—nghĩa là, trừ khi Len(q) nhỏ hơn N . Có thể dễ dàng viết một đặc tả hoàn chỉnh mới của FIFO bị chặn bằng cách sao chép mô-đun InnerFIFO và chỉ cần thêm liên kết Len(q) < N vào định nghĩa của BufRcv. Nhưng hãy sử dụng mô-đun InnerFIFO thay vì sao chép nó.

Hành động trạng thái tiếp theo BNext cho FIFO bị chặn giống như hành động trạng thái tiếp theo của FIFO Next ngoại trừ việc nó chỉ cho phép bước BufRcv nếu Len(q) nhỏ hơn N . Nói cách khác, BNext chỉ nên cho phép một bước nếu (i) đó là Bước tiếp theo và (ii) nếu đó là bước BufRcv thì Len(q) < N đúng ở trạng thái đầu tiên. Nói cách khác, BNext phải bằng

```
Tiếp theo    (BufRcv    (Len(q) < N ))
```

Mô-đun BoundedFIFO trong Hình 4.2 ở trang tiếp theo chứa thông số kỹ thuật.
Nó giới thiệu tham số không đổi mới N . Nó cũng chứa tuyên bố

```
giả sử (N    Nat)    (N > 0)
```

khẳng định rằng, trong mô-đun này, chúng ta giả định rằng N là một số tự nhiên dương. Giả định như vậy không ảnh hưởng đến bất kỳ định nghĩa nào được đưa ra trong mô-đun. Tuy nhiên, nó có thể được coi là một giả thuyết khi chứng minh bất kỳ định lý nào được khẳng định trong mô-đun. Nói cách khác, một mô-đun khẳng định rằng các giả định của nó bao hàm các định lý của nó. Bạn nên nêu loại giả định đơn giản này về các hằng số.

Một câu lệnh giả định chỉ nên được sử dụng để đưa ra các giả định về các hằng số. Công thức đang được giả định không được chứa bất kỳ biến nào. Có thể bạn sẽ muốn khẳng định các khai báo kiểu dưới dạng giả định—ví dụ: thêm vào mô-đun InnerFIFO giả định q Seq(Message). Tuy nhiên, điều đó sẽ sai vì nó khẳng định rằng, ở bất kỳ trạng thái nào, q là một chuỗi các thông điệp. Như chúng ta đã quan sát trong Phần 3.3, một trạng thái là một sự gán giá trị hoàn toàn tùy ý cho các biến, do đó có những trạng thái trong đó q có giá trị √ 17. Giả sử rằng trạng thái như vậy không tồn tại sẽ dẫn đến một mâu thuẫn logic.

Bạn có thể thắc mắc tại sao mô-đun BoundedFIFO giả định rằng N là số tự nhiên dương nhưng không cho rằng Thông báo là một tập hợp. Tương tự như vậy, tại sao chúng ta không

```
module BoundedFIFO

mở rộng Naturals, Chuỗi các biến
vào, ra hằng số
Thông báo, N giả sử (N
Nat)    (N > 0)

Nội (q)   = instance InnerFIFO BNext(q)

  =    Inner (q)!Next    Inner (q)!
                BufRcv    (Len(q) < N )

Spec   =    q : Bên trong (q)!Init     [BNext(q)]in,out,q
```

Hình 4.2: Đặc tả bộ đệm FIFO có độ dài N.

giả sử rằng tham số không đổi Dữ liệu trong thông số kỹ thuật giao diện không đồng bộ của chúng tôi là một tập hợp? Câu trả lời là, trong TLA+, mọi giá trị đều là một tập hợp.2 Một giá trị như số 3, mà chúng ta không coi là một tập hợp, chính thức là một tập hợp. Chúng tôi chỉ không biết các yếu tố của nó là gì. Công thức 2 3 là một công thức hoàn toàn hợp lý, nhưng TLA+ không chỉ rõ nó đúng hay sai. Vì vậy, chúng ta không cần phải giả định rằng Tin nhắn là một tập hợp vì chúng ta biết rằng nó là một tập hợp.

Mặc dù Tin nhắn tự động là một tập hợp nhưng nó không nhất thiết phải là một tập hợp hữu hạn. Ví dụ: Tin nhắn có thể được khởi tạo bằng tập hợp các số tự nhiên. Nếu bạn muốn giả sử rằng một tham số không đổi là một tập hợp hữu hạn thì bạn cần nêu rõ điều này như một giả định. (Bạn có thể thực hiện việc này với toán tử IsFiniteSet từ mô-đun FiniteSets, được mô tả trong Phần 6.1.) Tuy nhiên, hầu hết các thông số kỹ thuật đều có ý nghĩa hoàn hảo đối với các tập hợp thông báo hoặc bộ xử lý vô hạn, vì vậy không có lý do gì để giả định các tập hợp này là hữu hạn.

4.5 Những gì chúng tôi đang chỉ định

Tôi đã viết ở đầu chương này rằng chúng ta sẽ chỉ định bộ đệm FIFO. Công thức Spec của mô-đun FIFO thực sự chỉ định một tập hợp các hành vi, mỗi hành vi thể hiện một chuỗi các hoạt động gửi và nhận trên các kênh vào và ra. Các hoạt động gửi vào được thực hiện bởi người gửi và các hoạt động nhận ra được thực hiện bởi người nhận. Người gửi và người nhận không nằm trong bộ đệm FIFO; chúng hình thành nên môi trường của nó.

Đặc tả của chúng tôi mô tả một hệ thống bao gồm bộ đệm FIFO và môi trường của nó. Các hành vi thỏa mãn công thức Spec của module FIFO biểu diễn các lịch sử của vũ trụ trong đó cả hệ thống và môi trường của nó đều

2TLA+ dựa trên chủ nghĩa hình thức toán học được gọi là lý thuyết tập hợp Zermelo-Fr¨ankel, còn được gọi là ZF.

cư xử đúng đắn. Việc hiểu rõ một đặc tả thường hữu ích để chỉ ra rõ ràng bước nào là bước hệ thống và bước nào là bước môi trường. Chúng ta có thể làm điều này bằng cách xác định hành động ở trạng thái tiếp theo là

```
Tiếp theo = SysNext EnvNext
```

trong đó SysNext mô tả các bước hệ thống và EnvNext mô tả các bước môi trường. Đối với FIFO, chúng ta có

```
SysNext = BufRcv BufSend
EnvNext = ( tin nhắn Tin nhắn: SSend(tin nhắn)) RRcv
```

Mặc dù mang tính gợi ý nhưng cách xác định hành động ở trạng thái tiếp theo này không có ý nghĩa chính thức. Thông số Spec bằng Init [Next]... ; việc thay đổi cách chúng ta cấu trúc định nghĩa của Next không làm thay đổi ý nghĩa của nó. Nếu một hành vi không đáp ứng Spec thì không có gì cho chúng ta biết liệu hệ thống hoặc môi trường của nó có phải là nguyên nhân hay không.

Một công thức như Spec, mô tả hành vi đúng đắn của cả hệ thống và môi trường của nó, được gọi là đặc tả hệ thống khép kín hoặc hệ thống hoàn chỉnh.
Đặc tả hệ thống mở là đặc tả chỉ mô tả hành vi đúng của hệ thống. Một hành vi đáp ứng một đặc tả hệ thống mở nếu nó thể hiện một lịch sử trong đó hệ thống hoạt động chính xác hoặc nó không hoạt động đúng chỉ vì môi trường của nó đã làm sai điều gì đó. Phần 10.7 giải thích cách viết các đặc tả hệ thống mở.

Các thông số kỹ thuật của hệ thống mở thỏa mãn hơn về mặt triết học. Tuy nhiên, các thông số kỹ thuật của hệ thống đóng dễ viết hơn một chút và cơ sở toán học bên dưới chúng cũng đơn giản hơn. Vì vậy, chúng tôi hầu như luôn viết các thông số kỹ thuật của hệ thống đóng. Thông thường khá dễ dàng để biến đặc tả hệ thống đóng thành đặc tả hệ thống mở. Nhưng trên thực tế, hiếm khi có lý do gì để làm như vậy.