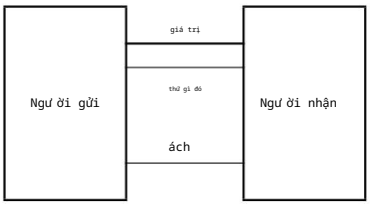
Chư ơ ng 3

Giao diện không đồng bộ

Bây giờ chúng tôi chỉ định giao diện để truyền dữ liệu giữa các thiết bị không đồng bộ. Ngư ời gửi và ngư ời nhận đư ợc kết nối như minh họa ở đây.



Dữ liệu đư ợc gửi trên val và dòng rdy và ack đư ợc sử dụng để đồng bộ hóa. Ngư ời gửi phải đợi một xác nhận (Ack) cho một mục dữ liệu trư ớc khi có thể gửi mục tiếp theo. Giao diện sử dụng giao thức bắt tay hai pha tiêu chuẩn, đư ợc mô tả bằng hành vi mẫu sau:

giá trị = 26
rdy = 0
ACK = 0
Gửi 37
giá trị = 37
thứ = 1
ACK = 0
Cảm ơ n
giá trị = 37
thứ = 1
ách = 1
Gửi 4
giá trị = 4
rdy = 0
ách = 1
Cảm ơ n
giá trị = 4
rdy = 0
ACK = 0
Gửi 19
giá trị = 19
thứ = 1
ACK = 0
Cảm ơ n
. . .

(Không quan trọng giá trị val có ở trạng thái ban đầu là bao nhiêu.)

Từ hành vi mẫu này, thật dễ dàng để thấy tập hợp tất cả các hành vi có thể có—sau khi chúng ta quyết định giá trị dữ liệu nào có thể đư ợc gửi. Tuy nhiên, trư ớc khi viết đặc tả TLA+ mô tả những hành vi này, hãy xem những gì tôi vừa làm.

Khi viết hành vi này, tôi đã quyết định rằng val và rdy sẽ thay đổi chỉ trong một bư ớc. Giá trị của các biến val và rdy biểu thị điện áp

trên một số bộ dây trong thiết bị vật lý. Điện áp trên các dây khác nhau không thay đổi chính xác tại cùng một thời điểm. Tôi quyết định bỏ qua khía cạnh này của hệ vật lý và giả vờ rằng các giá trị của val và rdy được biểu thị bằng các điện áp đó thay đổi tức thời. Điều này làm đơn giản hóa đặc tả nhưng phải trả giá bằng việc bỏ qua những gì có thể là chi tiết quan trọng của hệ thống. Trong quá trình triển khai giao thức thực tế, điện áp trên đường dây rdy không được thay đổi cho đến khi điện áp trên đường dây val ổn định; nhưng bạn sẽ không học được điều đó từ đặc điểm kỹ thuật của tôi. Nếu tôi muốn đặc tả truyền đạt yêu cầu này, tôi sẽ viết một hành vi trong đó giá trị của val và giá trị của rdy thay đổi theo các bước riêng biệt.

Một đặc tả là một sự trừu tượng. Nó mô tả một số khía cạnh của hệ thống và bỏ qua những khía cạnh khác. Chúng tôi muốn đặc tả càng đơn giản càng tốt, vì vậy chúng tôi muốn bỏ qua càng nhiều chi tiết càng tốt. Tuy nhiên, bất cứ khi nào chúng tôi bỏ qua một số khía cạnh của hệ thống khỏi đặc tả, chúng tôi sẽ thừa nhận một nguồn lỗi tiềm ẩn. Với thông số kỹ thuật của tôi, chúng tôi có thể xác minh tính đúng đắn của hệ thống sử dụng giao diện này và hệ thống vẫn có thể bị lỗi do người triển khai không biết rằng dòng val sẽ ổn định trước khi dòng rdy được thay đổi.

Phần khó nhất khi viết một đặc tả là chọn sự trừu tượng thích hợp.
Tôi có thể dạy bạn về TLA+, vì vậy việc thể hiện một cái nhìn trừu tượng về một hệ thống dưới dạng đặc tả TLA+ trở thành một nhiệm vụ đơn giản. Nhưng tôi không biết cách dạy bạn về sự trừu tượng. Một kỹ sư giỏi biết cách trừu tượng hóa bản chất của một hệ thống và loại bỏ những chi tiết không quan trọng khi xác định và thiết kế nó. Nghệ thuật trừu tượng chỉ được học thông qua kinh nghiệm.

Khi viết một đặc tả, trước tiên bạn phải chọn sự trừu tượng. Trong đặc tả TLA+, điều này có nghĩa là chọn các biến thể hiện trạng thái của hệ thống và mức độ chi tiết của các bước thay đổi giá trị của các biến đó. Dòng rdy và ack nên được biểu diễn dưới dạng các biến riêng biệt hay dưới dạng một biến duy nhất?
Giá trị và thứ tự nên thay đổi trong một bước, hai bước hay số bước tùy ý? Để giúp đưa ra những lựa chọn này, tôi khuyên bạn nên bắt đầu bằng cách viết vài bước đầu tiên của một hoặc hai hành vi mẫu, giống như tôi đã làm ở đầu phần này. Chương 7 sẽ nói nhiều hơn về những lựa chọn này.

3.1 Đặc điểm kỹ thuật đầu tiên

Hãy chỉ định giao diện không đồng bộ với mô-đun AsynchInterface. Thông số kỹ thuật sử dụng phép trừ các số tự nhiên, vì vậy mô-đun của chúng tôi mở rộng mô-đun Naturals để kết hợp định nghĩa của toán tử trừ "-".
Tiếp theo, chúng tôi quyết định giá trị có thể có của val sẽ là gì—nghĩa là giá trị dữ liệu nào có thể được gửi. Chúng ta có thể viết một đặc tả không đặt ra hạn chế nào đối với các giá trị dữ liệu. Thông số kỹ thuật có thể cho phép người gửi trước tiên gửi 37, sau đó gửi $\sqrt{15}$ và sau đó gửi Nat (toàn bộ tập hợp số tự nhiên).
Tuy nhiên, bất kỳ thiết bị thực nào cũng chỉ có thể gửi một tập hợp giá trị bị hạn chế. Chúng ta có thể chọn

một số tập hợp cụ thể—ví dụ: số 32 bit. Tuy nhiên, giao thức vẫn giống nhau bất kể nó
đư ợc sử dụng để gửi số 32 bit hay số 128 bit.
Vì vậy, chúng tôi thỏa hiệp giữa hai thái cực là cho phép gửi bất kỳ thứ gì và chỉ cho
phép gửi các số 32 bit bằng cách chỉ giả định rằng có một số Tập hợp giá trị dữ liệu có
thể đư ợc gửi. Hằng số Data là một tham số của đặc tả. Nó đư ợc tuyên bố bởi tuyên bố

```
dữ liệu không đổi
```

Ba biến của chúng tôi đư ợc khai báo bởi

```
các biến val, rdy, ack
```

Các từ khóa biến và biến là đồng nghĩa, cũng như hằng số và
hằng số.

Biến rdy có thể nhận bất kỳ giá trị nào—ví dụ: 1/2. Nghĩa là, tồn tại những trạng
thái gán giá trị 1/2 cho rdy. Khi thảo luận về đặc tả, chúng ta thư ờng nói rằng rdy
chỉ có thể giả sử các giá trị 0 và 1. Điều chúng tôi thực sự muốn nói là giá trị của rdy
bằng 0 hoặc 1 trong mọi trạng thái của bất kỳ hành vi nào thỏa mãn đặc tả. Như ng ngư ời
đọc thông số kỹ thuật không cần phải hiểu thông số kỹ thuật đầy đủ để tìm ra điều này.
Chúng ta có thể làm cho đặc tả dễ hiểu hơ n bằng cách cho ngư ời đọc biết những giá trị
nào mà các biến có thể giả định trong một hành vi thỏa mãn đặc tả. Chúng ta có thể làm
điều này bằng các nhận xét, như ng tôi thích sử dụng định nghĩa như thế này hơ n:

```
LoạiBất biến = (val Dữ liệu) (rdy {0, 1}) (ack {0, 1})
```

Tôi gọi tập hợp {0, 1} là loại rdy và tôi gọi TypeInvariant là loại bất biến.
Hãy xác định loại và một số thuật ngữ khác chính xác hơ n.

- Hàm trạng thái là một biểu thức thông thư ờng (không có số nguyên tố hoặc) có
  thể chứa các biến và hằng.
- Vị từ trạng thái là một hàm trạng thái có giá trị Boolean.
- Một Inv bất biến của một đặc tả Spec là một vị từ trạng thái sao cho
  Spec Inv là một định lý.
- Biến v có kiểu T trong đặc tả Spec iff v T là bất biến của
  Thông số kỹ thuật.

Chúng ta có thể làm cho định nghĩa của TypeInvariant dễ đọc hơ n bằng cách viết nó như
sau.

```
LoạiBất biến = val Dữ liệu
                  rdy {0, 1}
                     ack {0, 1}
```

Mỗi liên từ bắt đầu bằng một và phải nằm hoàn toàn bên phải của đó. (Liên từ có thể chiếm nhiều dòng). Chúng tôi sử dụng một ký hiệu tương tự cho các phân cách. Khi sử dụng ký hiệu danh sách dấu đầu dòng này, các hoặc phải xếp hàng chính xác (ngay cả trong đầu vào ascii). Bởi vì sự thụt đầu dòng là quan trọng nên chúng ta có thể loại bỏ dấu ngoặc đơn, làm cho ký hiệu này đặc biệt hữu ích khi các liên từ và phân cách được lồng vào nhau.

Công thức TypeInvariant sẽ không xuất hiện như một phần của đặc tả. Chúng tôi không cho rằng TypeInvariant là bất biến; đặc điểm kỹ thuật sẽ ngụ ý rằng nó là như vậy. Trong thực tế, tính bất biến của nó sẽ được khẳng định như một định lý.

Vị ngữ ban đầu là đơn giản. Ban đầu, val có thể bằng bất kỳ phần tử nào
Dữ liệu. Chúng ta có thể bắt đầu bằng rdy và ack cả 0 hoặc cả hai 1.

```
Ban đầu = val Dữ
          liệu rdy
          {0, 1} ack = rdy
```

Bây giờ là hành động ở trạng thái tiếp theo Tiếp theo. Một bước của giao thức sẽ gửi một giá trị hoặc nhận một giá trị. Chúng tôi xác định riêng hai hành động Gửi và Rcv mô tả việc gửi và nhận một giá trị. Bước tiếp theo (một hành động thỏa mãn Tiếp theo) là bước Gửi hoặc bước Rcv, do đó, đây là bước Gửi Rcv. Do đó, Tiếp theo được xác định bằng Gửi Rcv. Bây giờ hãy xác định Gửi và Rcv.

Chúng tôi nói rằng hành động Gửi được kích hoạt ở trạng thái mà từ đó có thể thực hiện bước Gửi. Từ hành vi mẫu ở trên, chúng ta thấy rằng Gửi được bật nếu rdy bằng ack. Thông thường, câu hỏi đầu tiên chúng tôi hỏi về một hành động là khi nào nó được kích hoạt? Vì vậy, định nghĩa của một hành động thường bắt đầu bằng điều kiện cho phép của hành động đó. Do đó, liên từ đầu tiên trong định nghĩa Gửi là rdy = ack.
Các liên từ tiếp theo cho chúng ta biết giá trị mới của các biến val, rdy và ack là gì. Giá trị mới val của val có thể là bất kỳ phần tử nào của Dữ liệu—tức là bất kỳ giá trị nào thỏa mãn val Data. Giá trị của rdy thay đổi từ 0 thành 1 hoặc từ 1 thành 0, do đó rdy bằng 1 rdy (vì 1 = 1 0 và 0 = 1 1). Giá trị của ack được giữ nguyên không thay đổi.

TLA+ định nghĩa v không thay đổi có nghĩa là biểu thức v có cùng giá trị ở trạng thái cũ và mới. Chính xác hơn, v không thay đổi bằng v = v, trong đó v là biểu thức thu được từ v bằng cách chuẩn hóa tất cả các biến của nó. Vì vậy, chúng tôi xác định Gửi bởi

```
Gửi = rdy = ack
          giá trị Dữ
          liệu rdy = 1
          rdy ack không thay đổi
```

(Tôi có thể viết ack = ack thay vì ack không thay đổi, nhưng tôi thích sử dụng cấu trúc không thay đổi trong thông số kỹ thuật hơn.)

Bước Rcv được kích hoạt nếu rdy khác với ack; nó bổ sung giá trị của ack và giữ nguyên val và rdy. Cả val và rdy đều không thay đổi nếu

```
──────────────────────── mô-đun AsynchInterface ────────────────────────
mở rộng Naturals
dữ liệu không đổi
các biến val, rdy, ack
LoạiBất biến  =   val   Dữ liệu
                  rdy   {0, 1}
                  ack   {0, 1}
──────────────────────────────────────────────────────────────────────
Ban đầu  =   val   Dữ
             liệu   rdy
             {0, 1}   ack = rdy
Gửi  =   rdy = ack
            giá trị   Dữ
             liệu   rdy = 1
             rdy   ack không thay đổi
Rcv  =   rdy = ack
            ack = 1   ack
             giá trị không đổi, rdy
Tiếp theo  = Gửi   Rcv
Thông số  = Ban đầu    [Tiếp theo]val,rdy,ack
──────────────────────────────────────────────────────────────────────
định lý Spec    TypeInvariant
──────────────────────────────────────────────────────────────────────
```

Hình 3.1: Đặc tả đầu tiên của chúng tôi về giao diện không đồng bộ.

cặp giá trị val, rdy được giữ nguyên. TLA+ sử dụng dấu ngoặc nhọn và để bao quanh các bộ dữ liệu có thứ tự, do đó Rcv khẳng định rằng val, rdy không thay đổi. (Dấu ngoặc nhọn được nhập trong ascii là << và >>.) Do đó, định nghĩa của Rcv là

```
Rcv  =   rdy = ack
            ack = 1   ack
             giá trị không đổi, rdy
```

Như trong ví dụ về đồng hồ của chúng tôi, thông số kỹ thuật hoàn chỉnh Spec sẽ cho phép các bước lặp lại—trong trường hợp này là các bước không thay đổi cả ba biến. Vì vậy, Spec cho phép các bước không thay đổi val, rdy, ack. Định nghĩa của nó là

```
Thông số  = Ban đầu    [Tiếp theo]val,rdy,ack
```

Mô-đun AsynchInterface cũng xác nhận tính bất biến của TypeInvariant. Nó xuất hiện đầy đủ trong Hình 3.1 trên trang này.

3.2 Thông số kỹ thuật khác

Mô-đun AsynchInterface là một mô tả hay về giao diện và giao thức bắt tay của nó. Tuy nhiên, nó không phù hợp lắm để giúp xác định các hệ thống sử dụng giao diện. Hãy viết lại đặc tả giao diện ở dạng thuận tiện hơn khi sử dụng như một phần của đặc tả lớn hơn.

Vấn đề đầu tiên với đặc tả ban đầu là nó sử dụng ba biến để mô tả một giao diện. Một hệ thống có thể sử dụng nhiều phiên bản khác nhau của giao diện. Để tránh sự phổ biến của các biến, chúng tôi thay thế ba biến val, rdy, ack bằng một biến duy nhất chan (viết tắt của kênh). Một nhà toán học sẽ làm điều này bằng cách cho giá trị của chan là một bộ ba có thứ tự—ví dụ: trạng thái [chan = 1/2, 0, 1] có thể thay thế trạng thái bằng val = 1/2, rdy = 0, và ack = 1. Nhưng các lập trình viên đã học được rằng việc sử dụng các bộ dữ liệu như thế này sẽ dẫn đến sai sót; rất dễ quên nếu dòng ack được biểu thị bằng thành phần thứ hai hoặc thứ ba. Do đó, TLA+ cung cấp các bản ghi bên cạnh các ký hiệu toán học thông thường hơn.

Hãy biểu thị trạng thái của kênh dưới dạng bản ghi với các trường val, rdy và ack. Nếu r là một bản ghi như vậy thì r .val là trường val của nó. Bất biến kiểu xác nhận rằng giá trị của chan là một phần tử của tập hợp tất cả các bản ghi r như vậy trong đó r .val là một phần tử của tập hợp Dữ liệu và r .rdy và r .ack là các phần tử của tập hợp {0, 1} . Bộ hồ sơ này được viết

[val : Dữ liệu, rdy : {0, 1}, ack : {0, 1}]

Các trường của bản ghi không được sắp xếp theo thứ tự, vì vậy việc chúng ta viết chúng theo thứ tự nào không quan trọng. Bộ hồ sơ tương tự này cũng có thể được viết là

[ack : {0, 1}, val : Dữ liệu, rdy : {0, 1}]

Ban đầu, chan có thể bằng bất kỳ phần tử nào của tập hợp này có trường ack và rdy bằng nhau, vì vậy vị từ ban đầu là sự kết hợp của bất biến kiểu và điều kiện chan.ack = chan.rdy.

Hệ thống sử dụng giao diện có thể thực hiện thao tác gửi một số giá trị dữ liệu d và thực hiện một số thay đổi khác phụ thuộc vào giá trị d.
Chúng tôi muốn trình bày một thao tác như vậy dưới dạng một hành động là sự kết hợp của hai hành động riêng biệt: một hành động mô tả việc gửi d và hành động kia mô tả các thay đổi khác. Do đó, thay vì xác định hành động Gửi để gửi một số giá trị dữ liệu không xác định, chúng ta xác định hành động Gửi(d) gửi giá trị dữ liệu d. Hành động ở trạng thái tiếp theo được đáp ứng bằng bước Gửi(d), đối với một số d trong Dữ liệu hoặc bước Rcv. (Giá trị mà bước Rcv nhận được bằng chan.val.) Nói rằng một bước là bước Gửi(d) đối với một số d trong Dữ liệu có nghĩa là tồn tại quảng cáo trong Dữ liệu sao cho bước đó thỏa mãn Gửi(d)—nói cách khác , bước đó là bước d Dữ liệu : Gửi (d). Vì vậy chúng tôi xác định

Tiếp theo = (d Dữ liệu : Gửi(d)) Rcv

Hành động Gửi(d) khẳng định rằng chan bằng bản ghi r sao cho

r .val = d r .rdy = 1 chan.rdy r .ack = chan.ack

Bản ghi này được viết bằng TLA+ dưới dạng

[val d, rdy 1 chan.rdy, ack chan.ack]

(Ký hiệu được nhập vào ascii dưới dạng |-> .) Vì các trường của bản ghi không được sắp xếp theo thứ tự nên bản ghi này cũng có thể được viết

[ack chan.ack, val d, rdy 1 chan.rdy]

Điều kiện kích hoạt của Send(d) là dòng rdy và ack bằng nhau, vì vậy chúng ta có thể định nghĩa

Gửi(d) =
chan.rdy = chan.ack
chan = [val d, rdy 1 chan.rdy, ack chan.ack]

Đây là một định nghĩa hoàn toàn đúng về Gửi(d). Tuy nhiên, tôi thích một cái hơi khác một chút. Chúng ta có thể mô tả giá trị của chan bằng cách nói rằng nó giống với giá trị của chan ngoại trừ trường val của nó bằng d và trường rdy của nó bằng 1 chan.rdy. Trong TLA+, chúng ta có thể viết giá trị này dưới dạng

[chan ngoại trừ !.val = d, !.rdy = 1 chan.rdy]

Hãy nghĩ về ! đại diện cho bản ghi mới mà biểu thức ngoại trừ hình thành bằng cách sửa đổi chan. Vì vậy, biểu thức có thể được đọc dưới dạng bản ghi ! giống như chan ngoại trừ !.val bằng d và !.rdy bằng 1 chan.rdy. Trong biểu thức !.rdy bằng, ký hiệu @ là viết tắt của chan.rdy, vì vậy chúng ta có thể viết biểu thức này ngoại trừ biểu thức là

[chan ngoại trừ !.val = d, !.rdy = 1 @]

Nói chung, với mọi bản ghi r , cách diễn đạt

[r ngoại trừ !.c1 = e1, . . . , !.cn = en]

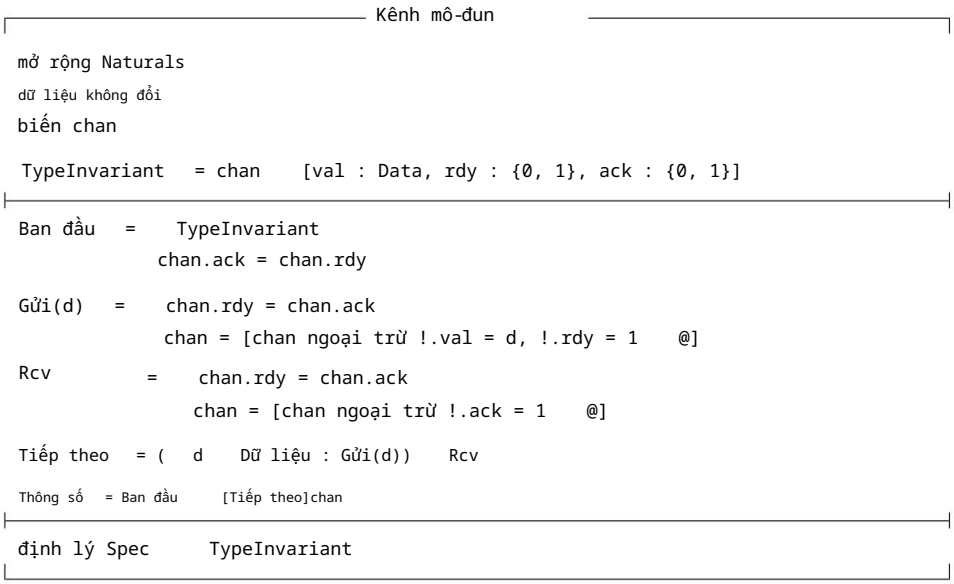
là bản ghi thu được từ r bằng cách thay thế r .ci bằng ei @ , cho mỗi i trong 1 . . N. MỘT trong biểu thức ei là viết tắt của r .ci . Sử dụng ký hiệu này, chúng ta xác

định Send(d) = chan.rdy = chan.ack
chan = [chan ngoại trừ !.val = d, !.rdy = 1 @]

Định nghĩa của Rcv rất đơn giản. Một giá trị có thể được nhận khi chan.rdy không bằng chan.ack và giá trị nhận được sẽ bổ sung cho chan.ack:

Rcv = chan.rdy = chan.ack chan
= [chan ngoại trừ !.ack = 1 @]

Thông số kỹ thuật đầy đủ xuất hiện trong Hình 3.2 ở trang tiếp theo.



Hình 3.2: Đặc tả thứ hai của chúng tôi về giao diện không đồng bộ.

3.3 Các loại: Lời nhắc

Như được định nghĩa trong Phần 3.1, một biến v có loại T trong đặc tả Spec iff v T là một bất biến của Spec. Vì vậy, hr có loại 1 . . 12 trong đặc điểm kỹ thuật HC của đồng hồ giờ. Khẳng định này không có nghĩa là biến hr chỉ có thể nhận các giá trị trong tập 1. . 12. Một trạng thái là sự gán giá trị tùy ý cho các biến, do đó tồn tại những trạng thái trong đó giá trị của hr là √ 2. Khẳng định này có nghĩa là, trong mọi hành vi thỏa mãn công thức HC, giá trị của hr là một phần tử của 1 . . 12.

Nếu bạn đã quen với việc gõ văn bản trong các ngôn ngữ lập trình, có vẻ lạ khi TLA+ cho phép một biến nhận bất kỳ giá trị nào. Tại sao không giới hạn các trạng thái của chúng ta ở những trạng thái trong đó các biến có giá trị đúng loại? Nói cách khác, tại sao không thêm hệ thống kiểu chính thức vào TLA+? Một câu trả lời đầy đủ sẽ đưa chúng ta đi quá xa. Câu hỏi này sẽ được giải quyết sâu hơn ở Phần 6.2. Hiện tại, hãy nhớ rằng TLA+ là ngôn ngữ chưa được định kiểu. Tính đúng đắn của loại chỉ là tên của một thuộc tính bất biến nhất định. Việc gán tên TypeInvariant cho một công thức sẽ không mang lại trạng thái đặc biệt nào.

3.4 Định nghĩa

Chúng ta hãy xem xét một định nghĩa có nghĩa là gì. Nếu Id là một mã định danh đơn giản như Init hoặc Spec, thì định nghĩa Id = exp xác định Id đồng nghĩa với biểu thức exp. Việc thay Id bằng exp hoặc ngược lại trong bất kỳ biểu thức nào cũng không làm thay đổi ý nghĩa của biểu thức đó. Việc thay thế này phải được thực hiện sau khi biểu thức được phân tích cú pháp, không phải trong "đầu vào thô". Ví dụ: định nghĩa x = a + b làm cho x c bằng (a +b) c chứ không phải a +b c, bằng a +(b c).

Định nghĩa của Gửi có dạng Id(p) = exp, trong đó Id và p là các định danh. Đối với bất kỳ biểu thức e nào, điều này xác định Id(e) là biểu thức thu được bằng cách thay thế e cho p trong exp. Ví dụ: định nghĩa Gửi trong mô-đun Kênh xác định Gửi(5) bằng

```
chan.rdy = chan.ack
chan = [chan ngoại trừ !.val = 5, !.rdy = 1 @]
```

Send(e) là một biểu thức, với mọi biểu thức e. Vì vậy, chúng ta có thể viết công thức Gửi(5) (chan.ack = 1). Bản thân mã định danh Gửi không phải là một biểu thức và Gửi (chan.ack = 1) không phải là một chuỗi được định dạng đúng ngữ pháp. Nó vô nghĩa về mặt cú pháp, như a + b + .

Chúng ta nói rằng Gửi là toán tử nhận một đối số duy nhất. Chúng ta định nghĩa các toán tử có nhiều hơn một đối số một cách rõ ràng, dạng tổng quát là

(3.1) Id(p1, . . , pn) = điểm kinh nghiệm

trong đó pi là các định danh riêng biệt và exp là một biểu thức. Chúng ta có thể coi các mã định danh được xác định như Init và Spec là các toán tử không có đối số, nhưng chúng ta thường sử dụng toán tử để chỉ một toán tử có một hoặc nhiều đối số.

Tôi sẽ sử dụng ký hiệu thuật ngữ để chỉ một mã định danh như Gửi hoặc ký hiệu toán tử như +. Mọi ký hiệu được sử dụng trong đặc tả phải là toán tử tích hợp của TLA+ (như) hoặc phải được khai báo hoặc xác định. Mọi khai báo hoặc định nghĩa ký hiệu đều có phạm vi trong đó ký hiệu có thể được sử dụng. Phạm vi của một khai báo biến hoặc hằng và của định nghĩa là một phần của mô-đun theo sau nó. Vì vậy, chúng ta có thể sử dụng Init trong bất kỳ biểu thức nào tuân theo định nghĩa của nó trong Module Channel. Câu lệnh mở rộng Naturals mở rộng phạm vi của các ký hiệu như + được xác định trong mô-đun Naturals sang mô-đun Kênh.

Định nghĩa toán tử (3.1) ngầm bao gồm việc khai báo các định danh p1, . . . , pn có phạm vi là biểu thức exp. Một biểu thức của hình thức

v S : điểm kinh nghiệm

có một khai báo v có phạm vi là biểu thức exp. Do đó, định danh v có ý nghĩa trong biểu thức exp (nhưng không có nghĩa trong biểu thức S).

Một biểu tượng không thể được khai báo hoặc xác định nếu nó đã có ý nghĩa. Cách diễn đạt

```
( v S : exp1) ( v T : exp2)
```

không sao cả, vì cả hai tuyên bố v đều không nằm trong phạm vi của tuyên bố kia.
Tương tự, hai khai báo ký hiệu d trong mô-đun Kênh (trong định nghĩa Gửi và trong biểu thức d trong định nghĩa Tiếp theo) có phạm vi khác nhau. Tuy nhiên, cách diễn đạt

```
( v S : (exp1 v T : exp2))
```

là bất hợp pháp vì việc khai báo v trong v thứ hai nằm trong phạm vi khai báo của nó trong v đầu tiên. Mặc dù toán học thông thường và các ngôn ngữ lập trình cho phép khai báo lại như vậy, nhưng TLA+ lại cấm chúng vì chúng có thể dẫn đến nhầm lẫn và sai sót.

3.5 Bình luận

Ngay cả các thông số kỹ thuật đơn giản như các thông số kỹ thuật trong mô-đun AsynchInterface và Channel cũng có thể khó hiểu chỉ bằng toán học. Đó là lý do tại sao tôi bắt đầu bằng phần giải thích trực quan về giao diện. Lời giải thích đó giúp bạn dễ hiểu hơn về công thức Spec trong mô-đun, đó là thông số kỹ thuật thực tế.
Mỗi thông số kỹ thuật phải được kèm theo một lời giải thích bằng văn xuôi không chính thức. Lời giải thích có thể có trong tài liệu đi kèm hoặc có thể được đưa vào dưới dạng nhận xét trong đặc tả.

Hình 3.3 trên trang tiếp theo cho thấy thông số kỹ thuật của đồng hồ giờ trong mô-đun HourClock có thể được giải thích bằng các nhận xét. Trong phiên bản sắp chữ, các nhận xét được phân biệt với chính thông số kỹ thuật bằng cách sử dụng phông chữ khác. Như được hiển thị trong hình, TLA+ cung cấp hai cách viết nhận xét trong phiên bản ascii. Một nhận xét có thể xuất hiện ở bất kỳ vị trí nào nằm giữa (* và *). Chú thích ở cuối dòng được đặt trước bởi *. Các chú thích có thể được lồng vào nhau, do đó bạn có thể chú thích một phần của đặc tả bằng cách đặt nó giữa (* và *), ngay cả khi phần đó chứa các chú thích.

Một nhận xét hầu như luôn xuất hiện trên một dòng hoặc ở cuối dòng.
Tôi đặt một nhận xét giữa HCnxt và = chỉ để chứng tỏ rằng điều đó có thể thực hiện được.

Để tiết kiệm không gian, tôi sẽ viết một vài nhận xét trong phần thông số kỹ thuật của ví dụ. Nhưng thông số kỹ thuật nên có rất nhiều ý kiến. Ngay cả khi có tài liệu kèm theo mô tả hệ thống, các nhận xét vẫn cần thiết để giúp người đọc hiểu đặc tả hình thức hóa mô tả đó như thế nào.

Nhận xét có thể giúp giải quyết vấn đề do cấu trúc logic của thông số kỹ thuật đặt ra. Một biểu tượng phải được khai báo hoặc xác định trước khi có thể sử dụng. Trong Kênh mô-đun, định nghĩa về Thông số kỹ thuật phải tuân theo định nghĩa của Tiếp theo, định nghĩa này phải tuân theo các định nghĩa về Gửi và Rcv. Nhưng nó thường dễ dàng nhất để

mô-đun GiờĐồng hồ

Mô-đun này chỉ định đồng hồ kỹ thuật số hiển thị giờ hiện tại. Nó bỏ qua thời gian thực, không xác định khi nào màn hình có thể thay đổi.

mở rộng Naturals

biến hr Biến hr đại diện cho màn hình.

HCini = hr (1 . . 12) Ban đầu, hr có thể có bất kỳ giá trị nào từ 1 đến 12.

HCnxt Đây là một nơ i kỳ lạ cho một bình luận. "

Giá trị của chu kỳ giờ từ 1 đến 12. hr =

if hr = 12 thì hr + 1 else 1

HC = HCini [HCnxt]giờ

Thông số kỹ thuật hoàn chỉnh. Nó cho phép đồng hồ dừng lại.

định lý HC HCini Tính đúng đắn của thông số kỹ thuật.

```
---------------------- MODULE Đồng hồ giờ ----------------------
   (*************************************************** *******)
   (* Mô-đun này chỉ định đồng hồ kỹ thuật số hiển thị *) (* giờ hiện tại. Nó bỏ qua
   thời gian thực, không phải *) (* chỉ định khi nào màn hình có thể thay đổi. *)
   (************ ***********************************************)

EXTENDS Naturals BIẾN
hr \* Biến hr đại diện cho màn hình.
HCini == hr \in (1 .. 12) \* Ban đầu, hr có thể có bất kỳ giá trị \* nào từ 1 đến
                                        12.
HCnxt (* Đây là một nơ i kỳ lạ để bình luận. *) ==
   (********************************** ****************)
   (* Giá trị của chu kỳ giờ từ 1 đến 12. *)
   (********************************** **************)
  hr' = IF hr # 12 THEN hr + 1 ELSE 1
HC == HCini /\ [][HCnxt]_hr (* Thông số
   kỹ thuật hoàn chỉnh. Nó cho phép đồng hồ dừng. *)
.....

THEOREM HC => []HCini \* Kiểu đúng của thông số kỹ thuật.
================================================== ============
```

Hình 3.3: Thông số đồng hồ giờ kèm theo chú thích.

hiểu mô tả từ trên xuống của một hệ thống. Đầu tiên chúng ta có thể muốn đọc các khai báo của Data và chan, sau đó là định nghĩa của Spec, sau đó là định nghĩa của Init và Next, sau đó là định nghĩa của Send và Rcv. Nói cách khác, chúng ta muốn đọc thông số kỹ thuật ít nhiều từ dưới lên trên.

Điều này đủ dễ thực hiện đối với một mô-đun ngắn như Kênh; thật bất tiện cho các thông số kỹ thuật dài hơn. Chúng ta có thể sử dụng các nhận xét để hướng dẫn người đọc thông qua một thông số kỹ thuật dài hơn. Ví dụ: chúng ta có thể đặt trước định nghĩa Gửi trong mô-đun Kênh bằng nhận xét

> Hành động Gửi và Rcv bên dưới là các phần tách biệt của hành động trạng thái tiếp theo Tiếp theo.

Cấu trúc mô-đun cũng cho phép chúng ta chọn thứ tự đọc thông số kỹ thuật. Ví dụ: chúng ta có thể viết lại đặc tả đồng hồ giờ bằng cách chia mô-đun HourClock thành ba mô-đun riêng biệt:

HCVar Một module khai báo biến hr .

HCActions Một mô-đun mở rộng các mô-đun Naturals và HCVar và de-
phạt HCini và HCnxt.

HCSpec Một mô-đun mở rộng các HCActions của mô-đun, xác định công thức và khẳng định định
HC , lý về tính đúng kiểu.

Mối quan hệ mở rộng hàm ý một thứ tự logic của các mô-đun: HCVar đứng trước HCActions, đứng trước HCSpec. Nhưng các mô-đun không nhất thiết phải được đọc theo thứ tự đó. Người đọc có thể được yêu cầu đọc HCVar trước, sau đó là HCSpec và cuối cùng là HCActions. Cấu trúc cá thể được giới thiệu bên dưới trong Chương 4 cung cấp một công cụ khác để mô-đun hóa các thông số kỹ thuật.

Việc chia tách một thông số kỹ thuật nhỏ như HourClock theo cách này sẽ thật lố bịch. Tuy nhiên, việc phân chia các mô-đun một cách hợp lý có thể giúp làm cho thông số kỹ thuật lớn dễ đọc hơn. Khi viết một đặc tả, bạn nên quyết định thứ tự đọc nó. Sau đó, bạn có thể thiết kế cấu trúc mô-đun để cho phép đọc nó theo thứ tự đó, khi mỗi mô-đun riêng lẻ được đọc từ đầu đến cuối. Cuối cùng, bạn nên đảm bảo rằng các nhận xét trong mỗi mô-đun sẽ có ý nghĩa khi các mô-đun khác nhau được đọc theo thứ tự thích hợp.