# Chương 8

## Sự sống động và công bằng

Các thông số kỹ thuật mà chúng tôi đã viết cho đến nay cho biết hệ thống không được làm những gì. Đồng hồ không được tăng từ 11 lên 9; người nhận không được nhận tin nhắn nếu FIFO trống. Họ không yêu cầu hệ thống phải thực sự làm bất cứ điều gì. Đồng hồ không bao giờ cần tích tắc; người gửi không bao giờ cần gửi bất kỳ tin nhắn nào.
Thông số kỹ thuật của chúng tôi đã mô tả những gì được gọi là đặc tính an toàn. Nếu một thuộc tính an toàn bị vi phạm, nó sẽ bị vi phạm tại một số điểm cụ thể trong hành vi—bằng một bước làm tăng đồng hồ từ 11 lên 9 hoặc đọc sai giá trị từ bộ nhớ. Do đó, chúng ta có thể nói về một thuộc tính an toàn được thỏa mãn bởi một hành vi hữu hạn, có nghĩa là nó chưa bị vi phạm bởi bất kỳ bước nào cho đến nay.

Bây giờ chúng ta học cách xác định rằng điều gì đó sẽ xảy ra—đồng hồ vẫn tiếp tục tích tắc hoặc cuối cùng một giá trị được đọc từ bộ nhớ. Chúng tôi chỉ định các thuộc tính sống—những thuộc tính không thể bị vi phạm vào bất kỳ thời điểm cụ thể nào. Chỉ bằng cách kiểm tra toàn bộ hành vi vô hạn, chúng ta mới có thể biết rằng đồng hồ đã ngừng tích tắc hoặc một tin nhắn chưa bao giờ được gửi.

Chúng tôi thể hiện các đặc tính sống động như các công thức thời gian. Điều này có nghĩa là, để thêm các điều kiện tồn tại vào thông số kỹ thuật của bạn, bạn phải hiểu logic thời gian—logic của các công thức thời gian. Chương này bắt đầu, ở Phần 8.1, với cái nhìn sâu sắc hơn về ý nghĩa của công thức thời gian. Để hiểu một logic, bạn phải hiểu công thức thực sự của nó là gì. Phần 8.2 nói về các lặp thừa thời gian, các công thức thực sự của logic thời gian. Phần 8.4–8.7 mô tả cách sử dụng các công thức thời gian để xác định các thuộc tính sống động. Phần 8.8 hoàn thành nghiên cứu của chúng ta về logic thời gian bằng cách xem xét bộ định lượng thời gian . Cuối cùng, Phần 8.9 xem xét những gì chúng tôi đã làm và giải thích tại sao việc sử dụng logic thời gian một cách vô kỷ luật lại nguy hiểm.

Chương này là chương duy nhất có bằng chứng. Sẽ thật tuyệt nếu bạn tự học cách viết những bằng chứng tương tự, nhưng cũng không thành vấn đề nếu bạn không học.
Các bằng chứng ở đây vì việc nghiên cứu chúng có thể giúp bạn phát triển sự hiểu biết trực quan về các công thức thời gian mà bạn cần để viết các thông số kỹ thuật—

một sự hiểu biết làm cho tính đúng đắn của một phép lặp thừa thời gian đơn giản như F ≡ F trở nên hiển nhiên như tính đúng đắn của một định lý đơn giản về các số như n Nat : 2 n ≥ n.

Nhiều độc giả sẽ thấy rằng chương này đánh giá cao khả năng toán học của họ. Đừng lo lắng nếu bạn gặp khó khăn trong việc hiểu nó. Hãy coi chương này như một bài tập giúp bạn căng thẳng đầu óc và chuẩn bị cho việc thêm các đặc tính sống động vào các thông số kỹ thuật của mình. Và hãy nhớ rằng các đặc tính sống động có thể là phần ít quan trọng nhất trong đặc tả của bạn. Bạn có thể sẽ không mất nhiều nếu bạn bỏ qua chúng.

# 8.1 Công thức thời gian

Hãy nhớ lại rằng một trạng thái gán một giá trị cho mọi biến và một hành vi là một chuỗi vô hạn các trạng thái. Một công thức thời gian là đúng hay sai của một hành vi. Về mặt hình thức, một công thức thời gian F gán một giá trị Boolean mà chúng ta viết σ |= F cho một hành vi σ. Ta nói F đúng với σ, hoặc σ thỏa mãn F, nếu σ |= F bằng đúng. Để xác định ý nghĩa của công thức thời gian F, chúng ta phải giải thích cách xác định giá trị của σ |= F cho bất kỳ hành vi σ nào. Hiện tại, chúng tôi chỉ xem xét các công thức thời gian không chứa bộ định lượng tồn tại thời gian .

Thật dễ dàng để xác định ý nghĩa của sự kết hợp Boolean của các công thức thời gian theo ý nghĩa của các công thức đó. Công thức F G đúng với hành vi σ nếu cả F và G đều đúng với σ, và â€F đúng với σ iff F không đúng với σ. Những định nghĩa này được viết chính thức hơn như

σ |= (F G) = (σ |= F) (σ |= G)

σ |= F = и (σ |= F)

Đây là các định nghĩa về ý nghĩa của và của и như các toán tử trong các công thức thời gian. Ý nghĩa của các toán tử Boolean khác được xác định tương tự.
Chúng ta cũng có thể định nghĩa theo cách này các bộ định lượng logic vị từ thông thường và là các toán tử trên các công thức thời gian—ví dụ:

σ |= ( r : F) = r : (σ |= F)

Định lượng thông thường trên các tập hợp hằng số được định nghĩa theo cách tương tự. Ví dụ, nếu S là một biểu thức hằng thông thường—tức là biểu thức không chứa biến—thì

σ |= ( r S : F) = r S : (σ |= F)

Các bộ định lượng sẽ được thảo luận sâu hơn trong Phần 8.8 dưới đây.

Tất cả các công thức thời gian không định lượng mà chúng ta đã thấy đều là sự kết hợp Boolean của ba loại công thức đơn giản, có ý nghĩa sau:

- Vị từ trạng thái, được xem như một công thức thời gian, đúng với một hành vi nếu nó đúng ở trạng thái đầu tiên của hành vi.

Hàm trạng thái và vị ngữ trạng thái được xác định ở trang 25.

- Công thức P, trong đó P là vị từ trạng thái, đúng với một hành vi nếu P đúng trong mọi trạng thái của hành vi.
- Công thức [N ]v , trong đó N là một hành động và v là một hàm trạng thái, đúng với một hành vi nếu mỗi cặp bước kế tiếp trong hành vi đó là một bước [N ]v .

Vì vị từ trạng thái là một hành động không chứa biến số nguyên tố, nên chúng ta có thể kết hợp và khái quát hóa ba loại công thức thời gian này thành hai loại công thức A và A, trong đó A là một hành động. Trước tiên tôi sẽ giải thích ý nghĩa của hai loại công thức này, sau đó định nghĩa toán tử nói chung. Để làm điều này, tôi sẽ sử dụng ký hiệu rằng σi là trạng thái thứ nhất (i + 1) của hành vi σ, với mọi số tự nhiên i, do đó σ là hành vi σ0 σ1 σ2 · · · .

Chúng tôi diễn giải một hành động tùy ý A như một công thức thời gian bằng cách xác định σ |= A là đúng nếu hai trạng thái đầu tiên của σ là một bước A. Nghĩa là, chúng ta xác định σ | = A là đúng nếu σ0 σ1 là một bước A. Trong trường hợp đặc biệt khi A là một vị từ trạng thái, σ0 σ1 là một bước A nếu A đúng ở trạng thái σ0, do đó, định nghĩa này về σ |= A khái quát hóa cách giải thích của chúng ta về một vị từ trạng thái như một công thức thời gian.

Chúng ta đã thấy rằng [N ]v đúng với một hành vi nếu mỗi bước là một bước [N ]v . Điều này dẫn đến việc chúng ta định nghĩa σ |= A đúng nếu σn σn+1 là một bước A, với mọi số tự nhiên n.

Bây giờ chúng ta khái quát hóa từ định nghĩa σ |= A cho một hành động A đến định nghĩa σ |= F cho một công thức thời gian tùy ý F. Chúng ta đã định nghĩa σ |= A là đúng nếu σn σn +1 là một Một bước cho tất cả n. Điều này đúng nếu A, được hiểu như một công thức thời gian, đúng với hành vi có bước đầu tiên là σn σn+1, với mọi n. Hãy định nghĩa $\sigma_{+n}$ là hậu tố của σ thu được bằng cách xóa n trạng thái đầu tiên của nó:

$$\sigma_{+n} = \sigma_n \; \sigma_{n+1} \; \sigma_{n+2} \; \cdots$$

Khi đó σn σn+1 là bước đầu tiên của $\sigma_{+n}$, vậy σ |= A đúng nếu $\sigma_{+n}$ |= A đúng với mọi n. Nói cách khác

$$\sigma \models A \equiv \forall\, n \in Nat : \sigma_{+n} \models A$$

Tổng quát hiển nhiên là σ |= F

$$= \forall\, n \in Nat : \sigma_{+n} \models F$$

đối với mọi công thức thời gian F. Nói cách khác, σ thỏa mãn F nếu mọi hậu tố $\sigma_{+n}$ của σ đều thỏa mãn F. Điều này xác định ý nghĩa của toán tử thời gian .

Bây giờ chúng ta đã xác định ý nghĩa của bất kỳ công thức thời gian nào được xây dựng từ các hành động (bao gồm các vị từ trạng thái), toán tử Boolean và toán tử . Ví dụ:

$$\begin{aligned}
\sigma \models ((x = 1) \; (y > 0)) & \\
&\equiv \forall\, n \in Nat : \sigma_{+n} \models ((x = 1) \; (y > 0)) && \text{Theo ý nghĩa của} \\
&\equiv \forall\, n \in Nat : (\sigma_{+n} \models (x = 1)) \; (\sigma_{+n} \models (y > 0)) && \text{Theo ý nghĩa của} \\
&\equiv \forall\, n \in Nat : (\sigma_{+n} = (x = 1)) && \text{Theo ý nghĩa của} \\
&\qquad (\forall\, m \in Nat : (\sigma_{+n})_{+m} \models (y > 0))
\end{aligned}$$

Do đó, σ |= ((x = 1) (y > 0)) đúng if, với mọi n Nat, nếu x = 1 đúng ở trạng thái σn, thì y > 0 đúng ở mọi trạng thái σn +m với m ≥ 0.

Để hiểu các công thức thời gian một cách trực quan, hãy coi σn là trạng thái của vũ trụ tại thời điểm n trong quá trình hoạt động σ.[1] Đối với bất kỳ vị ngữ trạng thái P nào, biểu thức σ +n |= P khẳng định rằng P đúng tại thời điểm n. Do đó, ((x = 1) (y > 0)) khẳng định rằng, bất cứ khi nào x = 1 là đúng thì y > 0 sẽ đúng kể từ đó trở đi. Đối với một công thức thời gian tùy ý F, chúng tôi cũng giải thích σ +n |= F là khẳng định rằng F đúng tại thời điểm n. Công thức F sau đó khẳng định rằng F luôn đúng. Do đó chúng ta có thể đọc như mọi khi hoặc từ nay trở đi hoặc từ đó trở đi.

Chúng ta đã thấy trong Phần 2.2 rằng một đặc tả sẽ cho phép các bước lặp lại—những bước không thay đổi tất cả các biến xuất hiện trong công thức. Bước lắp bắp chỉ thể hiện sự thay đổi đối với một phần nào đó của hệ thống không được mô tả bằng công thức; việc thêm nó vào hành vi sẽ không ảnh hưởng đến tính đúng đắn của công thức. Chúng ta nói rằng công thức F là bất biến khi nói lắp[2] nếu việc thêm hoặc xóa một bước nói lắp đối với hành vi σ không ảnh hưởng đến việc σ có thỏa mãn F hay không. Một công thức hợp lý sẽ bất biến khi nói lắp. Không có ích gì khi viết các công thức không hợp lý, vì vậy TLA cho phép bạn chỉ viết các công thức tạm thời bất biến khi nói lắp.

Vị từ trạng thái (được xem như một công thức tạm thời) là bất biến trong tình trạng nói lắp, vì tính chân thực của nó chỉ phụ thuộc vào trạng thái đầu tiên của một hành vi và việc thêm một bước nói lắp không làm thay đổi trạng thái đầu tiên. Một hành động tùy ý không phải là bất biến khi nói lắp. Ví dụ: hành động [x = x + 1]x được thỏa mãn bởi hành vi σ trong đó x được giữ nguyên ở bước đầu tiên và tăng thêm 2 ở bước thứ hai; nó không được thỏa mãn bởi hành vi thu được bằng cách loại bỏ bước nói lắp ban đầu khỏi σ. Tuy nhiên, công thức [x = x + 1]x là bất biến khi nói lắp, vì nó được thỏa mãn bởi một hành vi nếu mỗi bước thay đổi x là một bước x = x + 1—một điều kiện không bị ảnh hưởng bởi việc thêm hoặc xóa các bước nói lắp .

Nói chung, công thức [A]v là bất biến khi nói lắp đối với mọi hành động A và hàm trạng thái v. Tuy nhiên, A không bất biến khi nói lắp đối với một hành động tùy ý A. Ví dụ: (x = x + 1) có thể bị làm sai bằng cách thêm một bước không thay đổi x . Vì vậy, mặc dù chúng ta đã gán ý nghĩa cho (x = x + 1), nhưng đó không phải là công thức TLA hợp pháp.

Tính bất biến trong tình trạng nói lắp được bảo toàn bởi và bởi các toán tử Boolean— nghĩa là, nếu F và G là bất biến trong tình trạng nói lắp, thì F, и F, F G, x S : F, v.v. các vị từ, công thức có dạng [N ]v và tất cả các công thức for-mul có thể thu được từ chúng bằng cách áp dụng và các toán tử Boolean là bất biến khi nói lắp.

[1] Đó là bởi vì chúng ta coi σn là trạng thái tại thời điểm n, và bởi vì chúng ta thường đo thời gian bắt đầu từ 0, tức là tôi đánh số trạng thái của một hành vi bắt đầu bằng 0 thay vì 1.

[2] Đây là một ý nghĩa hoàn toàn mới của từ bất biến; nó không liên quan gì đến khái niệm về sự bất biến đã được thảo luận rồi.

Bây giờ chúng ta xem xét năm lớp công thức đặc biệt quan trọng được xây dựng từ các công thức thời gian tùy ý F và G. Chúng tôi giới thiệu các toán tử mới để biểu diễn ba toán tử đầu tiên.

F được xác định bằng € €F. Nó khẳng định rằng F không phải lúc nào cũng sai, có nghĩa là F đúng ở một thời điểm nào đó:

σ |= F

≡ σ |= и и F Theo định nghĩa của .

≡ и (σ |= €F) ≡ Theo ý nghĩa của ¨. |=

и ( n Nat : σ ≡ $^{+n}$ €F) Theo ý nghĩa của .

и ( n Nat : и (σ |= $^{+n}$ |= F)) Theo ý nghĩa của €.

≡ n Nat : σ $^{+n}$ F Bởi vì и и tương đương với .

Chúng ta thường đọc như cuối cùng, lấy cuối cùng để bao gồm bây giờ.

F G được xác định bằng (F G). Cùng một kiểu tính toán mà chúng ta vừa đã làm với σ |= F cho

thấy σ |= (F G)

≡ n Nat : $^{+n}$ |= F) ( m Nat : (σ +(n+m) |= G))

(σ Công thức F G khẳng định rằng bất cứ khi nào F đúng thì G cuối cùng cũng đúng —nghĩa là G đúng khi đó hoặc tại một thời gian sau Chúng tôi đọc như dẫn đến.

Av được định nghĩa bằng и [и A]v , trong đó A là hàm hành động và trạng thái va. Nó khẳng định rằng không phải mọi bước đều là bước (€A) (v = v), vì vậy một số bước là bước и ((и A) (v = v)). Vì ¨(P Q) tương đương với (§P) (§Q), với mọi P và Q, hành động и ((и A) (v = v)) tương đương với A (v = v) . Do đó, Av khẳng định rằng một số bước là bước A (v = v)—tức là bước A thay đổi v. Chúng ta xác định hành động Av theo Av = A (v = v)

Tôi phát âm Av là góc A phụ v.

nên Av khẳng định rằng

cuối cùng bước Av xảy ra. Chúng tôi coi Av là công thức thu được bằng cách áp dụng toán tử cho Av , mặc dù về mặt kỹ thuật thì không phải vậy vì Av không phải là công thức tạm thời.

F khẳng định rằng tại mọi thời điểm F luôn đúng hoặc vào một thời điểm nào đó sau đó. Đối với thời điểm 0, điều này ngụ ý rằng F đúng tại một thời điểm nào đó n0 ≥ 0. Đối với thời gian n0 + 1, nó ngụ ý rằng F đúng tại một thời điểm nào đó n1 ≥ n0 + 1. Đối với thời điểm n1 + 1, nó ngụ ý rằng F đúng tại thời điểm n2 ≥ n1 + 1. Tiếp tục quá trình, ta thấy F đúng tại một chuỗi thời gian vô hạn n0, n1, n2, . . . . Vì vậy, F ngụ ý rằng F đúng ở vô số thời điểm. Ngược lại, nếu F đúng ở vô số thời điểm, thì tại mọi thời điểm, F phải đúng ở một thời điểm nào đó sau đó, vì vậy F là đúng. Do đó, F khẳng định rằng F thường đúng vô hạn. Đặc biệt, Av khẳng định có vô số bước Av xảy ra.

F khẳng định rằng cuối cùng (tại một thời điểm nào đó), F trở thành đúng và vẫn đúng sau đó. Nói cách khác, F khẳng định rằng F cuối cùng luôn đúng. Cụ thể, [N ]v khẳng định rằng, cuối cùng, mỗi bước đều là một bước [N ]v .

Các toán tử và có độ ưu tiên cao hơn (liên kết chặt chẽ hơn) so với các toán tử Boolean, vì vậy F G có nghĩa là ( F) ( G). Toán tử có độ ưu tiên thấp hơn và .

## 8.2 Đồng nghĩa thời gian

Định lý thời gian là một công thức thời gian được thỏa mãn bởi mọi hành vi. Nói cách khác, F là một định lý iff σ |= F bằng đúng với mọi hành vi σ. Ví dụ: mô-đun HourClock khẳng định rằng HC HCini là một định lý, trong đó HC và HCini là các công thức được xác định trong mô-đun. Định lý này thể hiện một tính chất của đồng hồ giờ.

Công thức HCini HCini cũng là một định lý. Tuy nhiên, nó không cho chúng ta biết gì về đồng hồ giờ vì nó đúng bất kể HCini được định nghĩa như thế nào. Ví dụ, thay x > 7 cho HCini mang lại định lý (x > 7) (x > 7).
Một công thức như HCini HCini đúng khi bất kỳ công thức nào được thay thế cho các định danh của nó được gọi là tautology. Để phân biệt chúng với các tautology của logic thông thường, các tautology chứa các toán tử thời gian đôi khi được gọi là tautology thời gian.

Hãy chứng minh rằng HCini HCini là một phản phục thời gian. Để tránh nhầm lẫn giữa định danh tùy ý HCini trong tautology này với công thức HCini được xác định trong mô-đun HourClock, hãy thay thế nó bằng F, do đó tautology trở thành F F. Có các tiên đề và quy tắc suy luận cho logic thời gian mà từ đó chúng ta có thể chứng minh bất kỳ tautology thời gian nào, như F F, không chứa định lượng.
Tuy nhiên, việc chứng minh chúng trực tiếp từ ý nghĩa của các toán tử thường dễ dàng và mang tính hướng dẫn hơn. Chúng ta chứng minh rằng F F là một phản phục bằng cách chứng minh rằng σ |= ( F F) bằng đúng, với mọi hành vi σ và mọi công thức F. Chứng minh rất đơn giản:

σ |= ( F F) ≡ (σ |= F) (σ |= F) ≡ (
n Nat : σ ≡ ( +n |= F) (σ |= F) Theo định nghĩa của .
n Nat : σ +n |= F) (σ +0 |= F) Theo định nghĩa của σ+0 .
≡ đúng Bằng logic vị từ.

Theo ý nghĩa của .

Phép lặp thừa thời gian F F khẳng định một sự thật hiển nhiên rằng, nếu F luôn đúng thì nó đúng ở thời điểm 0. Phép lặp lặp đơn giản như vậy sẽ hiển nhiên khi bạn đã quen với việc suy nghĩ theo các công thức thời gian. Dưới đây là ba phép lặp đơn giản hơn, cùng với bản dịch tiếng Anh của chúng.

¨F ≡ FF
không phải lúc nào cũng đúng nếu cuối cùng nó sai.

(F G) ≡ ( F) ( G)

F và G đều luôn đúng nếu F luôn đúng và G luôn đúng.

Một cách khác để nói điều này là phân phối trên .

(F G) ≡ ( F) ( G)

F hoặc G cuối cùng là đúng nếu F cuối cùng là đúng hoặc G cuối cùng là đúng.

Một cách khác để nói điều này là phân phối trên .

Trọng tâm của việc chứng minh mỗi phép đồng nghĩa này là phép lặp thừa của logic vị từ.
Ví dụ: bằng chứng cho thấy phân bố trên dựa vào thực tế là phân bố trên :

σ |= ( (F G) ≡ ( F) ( G)) ≡

(σ |= (F G)) ≡ (σ |= ( F) ( G)) ≡ ( σ |= Theo ý nghĩa của ≡.

(F G)) ≡ (σ |= F) (σ |= G) ≡ ( n Nat : Theo ý nghĩa của .

σ ( n Nat : σ ≡ +n |= (F G)) ≡ Theo định nghĩa của .

true Theo phép +n |= F) ( n Nat : σ +n |= G)

lặp logic vị ngữ ( x S : P Q) ≡ ( x S : P) ( x S : Q).

Toán tử không phân phối trên và cũng không phân phối trên . Ví dụ: ((n ≥
0) (n < 0)) không tương đương với ( (n ≥ 0) (n < 0)); công thức đầu tiên đúng với
bất kỳ hành vi nào trong đó n luôn là một số, nhưng công thức thứ hai là sai đối với hành
vi trong đó n nhận cả giá trị dương và âm. Tuy nhiên, hai công thức sau đây là tautology:

( F) ( G) (F G) (F G) ( F) ( G)

Một trong hai đồng nghĩa phản phục này có thể được bắt nguồn từ đồng nghĩa lặp lại kia bằng cách thay
thế ³/₄F cho F và ¼G cho G. Thực hiện sự thay thế này trong đồng nghĩa lặp thứ hai mang lại kết quả

true ≡ ((и F) (€G)) ( €F) ( и G) Bằng cách thay thế trong phép lặp thứ hai. ≡ и (F
G) ( €F) ( €G) ≡ и (F G) Bởi vì (P Â Q) ≡ и (P Q).
(€ F) (€ G) ≡ и (F G) и Bởi vì €H ≡ и H.
(( F) ( G)) ≡ ( F) ( G) (F Bởi vì (P Â Q) ≡ и (P Q).
G) Bởi vì (P Â Q) ≡ (Q P).

Cặp lặp thừa này minh họa một quy luật chung: từ bất kỳ lặp thừa thời gian nào, chúng ta
thu được một lặp thừa kép bằng cách thực hiện các phép thay thế.

và đảo ngược hướng của mọi hàm ý. (Bất kỳ ≡ hoặc € nào đều được giữ nguyên.)
Như trong ví dụ trên, tautology kép có thể được chứng minh từ bản gốc bằng cách thay thế
từng định danh bằng phủ định của nó và áp dụng các tautology (kép) и F ≡ и F và и F ≡
€F cùng với lý luận logic mệnh đề .

Một cặp lặp thừa kép quan trọng khác khẳng định rằng phân phối trên
và phân phối trên :

(8.1) (F G) ≡ ( F) ( G) (F G) ≡ ( F) ( G)

Điều đầu tiên khẳng định rằng F hoặc G đúng vô số lần nếu F đúng vô số lần
hoặc G thường đúng vô hạn. Sự thật của nó khá rõ ràng, nhưng hãy chứng minh điều đó.
Để suy luận về , việc giới thiệu ký hiệu ∞ sẽ giúp ích, có nghĩa là ở đó
tồn tại vô số. Cụ thể, ∞i Nat : P(i) có nghĩa là P(i) đúng
với vô số số tự nhiên i. Ở trang 91, chúng tôi đã chỉ ra rằng F khẳng định
rằng F thường đúng vô hạn. Sử dụng ∞, chúng ta có thể biểu thị điều này như

(8.2) (σ |= F) ≡ ( ∞i Nat : σ +tôi |= F)

Lý do tương tự chứng minh kết quả tổng quát hơn sau đây, trong đó P là bất kỳ
nhà điều hành:

(8.3) ( n Nat : m Nat : P(n + m)) ≡ ∞i Nat : P(i)

Đây là một tautology hữu ích khác liên quan đến ∞, trong đó P và Q là tùy ý
các toán tử và S là một tập tùy ý:

(8.4) ( ∞i S : P(i) Q(i)) ≡ ( ∞i S : P(i)) ( ∞i S : Q(i))

Sử dụng những kết quả này, giờ đây thật dễ dàng chứng minh rằng phân bố trên :

σ |= (F G)
≡ ∞i Nat : σ +tôi |= (F G) Bởi (8.2).
≡ ( ∞i Nat : σ +tôi |= F) ( ∞i Nat : σ +tôi |= G) Theo (8.4).
≡ (σ |= F) (σ |= G) Bởi (8.2).

Từ đó, chúng ta suy ra tautology kép, mà phân bố trên .

Trong bất kỳ phản ứng trùng lặp TLA nào, việc thay thế một công thức thời gian bằng một hành động sẽ mang lại một
trùng lặp—một công thức đúng cho mọi hành vi—ngay cả khi công thức đó không phải là một
công thức TLA hợp pháp. (Hãy nhớ rằng chúng ta đã định nghĩa ý nghĩa của nonTLA
các công thức như (x = x + 1).) Chúng ta có thể áp dụng các quy tắc logic để biến đổi chúng
các phản hồi không phải TLA thành các phản hồi TLA. Trong số các quy tắc này có những quy tắc sau
tương đương kép, rất dễ kiểm tra:

[A B]v ≡ [A]v [B]v A B v ≡ Av B v

(Bước thứ hai khẳng định rằng bước A B thay đổi v là bước A mà
thay đổi v hoặc bước B thay đổi v.)

Như một ví dụ về việc thay thế các hành động cho các công thức thời gian trong TLA
các bạn, hãy thay Av và B để có được v tautolo-cho F và G trong phép lặp thứ nhất của (8.1)

(8.5) (Av B v ) ≡ ( Av ) ( B v )

Đây không phải là tautology TLA, vì (Av B v ) không phải là công thức TLA.
Tuy nhiên, một quy tắc logic chung cho chúng ta biết rằng việc thay thế một công thức con bằng
một công thức tương đương sẽ mang lại một công thức tương đương. Thay A v với Av B v
B vào (8.5) ta có kết quả đồng nghĩa TLA sau:

$$A \quad B \quad {}_{v} \equiv ( \quad Av ) \quad ( \quad B v )$$

## 8.3 Quy tắc chứng minh tạm thời

Quy tắc chứng minh là quy tắc để suy ra các công thức đúng từ các công thức đúng khác. Vì
Ví dụ, Quy tắc Modus Ponens của logic mệnh đề cho chúng ta biết rằng, với bất kỳ
công thức F và G, nếu chúng ta đã chứng minh được F và F G thì chúng ta có thể suy ra
G. Vì các định luật logic mệnh đề cũng áp dụng cho logic thời gian, nên chúng ta
có thể áp dụng Quy tắc Modus Ponens khi suy luận về các công thức thời gian.
Logic thời gian cũng có một số quy tắc chứng minh của riêng nó. Một là

> Quy tắc tổng quát hóa Từ F chúng ta có thể suy ra F, với mọi công thức thời gian F.

Quy tắc này khẳng định rằng, nếu F đúng với mọi hành vi thì F cũng vậy. Để chứng minh điều đó,
chúng ta phải chỉ ra rằng, nếu σ |= F đúng với mọi hành vi σ, thì τ |= F đúng
cho mọi hành vi τ . Bằng chứng rất dễ dàng:

$$\tau \models F \equiv n \quad Nat : \tau \quad {}^{+n} \models F$$

Theo định nghĩa của .

$$\equiv n \quad Nat : \text{đúng}$$

Bằng giả sử rằng σ |= F bằng đúng với mọi σ.

$$\equiv \text{đúng}$$

Bằng logic vị từ.

Một quy tắc chứng minh tạm thời khác là

> Suy ra Quy tắc tổng quát hóa Từ F G ta suy ra F G,
> đối với mọi công thức thời gian F và G.

Quy tắc tổng quát hóa có thể được bắt nguồn từ Quy tắc tổng quát hóa ngụ ý
và tautology true = true bằng cách thay true cho F và F cho G.

Sự khác biệt giữa quy tắc chứng minh theo thời gian và quy tắc lặp lại theo thời gian có thể là
gây nhầm lẫn. Trong logic mệnh đề, mọi quy tắc chứng minh đều có một phép lặp tương ứng.
Quy tắc Modus Ponens khẳng định rằng chúng ta có thể suy ra G bằng cách chứng minh F và
F G, hàm ý đồng nghĩa F (F G) G. Nhưng trong logic thời gian, một chứng minh
quy tắc không nhất thiết phải hàm ý một sự lặp lại. Quy tắc tổng quát hóa, trong đó phát biểu rằng chúng ta
có thể suy ra F bằng cách chứng minh F, không hàm ý rằng F F là một phản phục. Các
quy tắc có nghĩa là, nếu σ |= F đúng với mọi σ, thì σ |= F đúng với mọi σ. Đó là
khác với khẳng định (sai) rằng F F là một phản phục, điều này sẽ
nghĩa là σ |= (F F) đúng với mọi σ. Ví dụ: σ |= (F F) bằng
sai nếu F là vị từ trạng thái đúng ở trạng thái đầu tiên của σ và sai
ở một trạng thái khác nào đó của σ. Quên đi sự khác biệt giữa quy tắc chứng minh và quy tắc
tautology là nguyên nhân phổ biến gây ra lỗi khi sử dụng logic thời gian.

## 8.4 Tính công bằng yếu

Thật dễ dàng để chỉ định các thuộc tính sống động bằng các toán tử thời gian và . Ví dụ, hãy xem xét thông số kỹ thuật đồng hồ giờ của mô-đun HourClock trong Hình 2.1 trên trang 20. Chúng ta có thể yêu cầu đồng hồ không bao giờ dừng bằng cách khẳng định rằng phải có vô số bước HCnxt. Cách rõ ràng để viết khẳng định này là HCnxt, nhưng đó không phải là công thức TLA hợp pháp vì HCnxt là một hành động, không phải là công thức tạm thời. Tuy nhiên, bước HCnxt sẽ nâng cao giá trị giờ của đồng hồ nên nó sẽ thay đổi giờ. Do đó, bước HCnxt cũng là bước HCnxt thay đổi hr— tức là bước HCnxthr . Do đó, chúng ta có thể viết thuộc tính sống động mà đồng hồ không bao giờ dừng là HCnxthr . Vì vậy, chúng ta có thể coi HC HCnxthr là thông số kỹ thuật của một chiếc đồng hồ không bao giờ dừng.

Trước khi tiếp tục, tôi phải thú nhận và sau đó dẫn bạn đi lạc đề ngắn gọn về các chỉ số dưới. Trước tiên, hãy để tôi thú nhận rằng lập luận mà tôi vừa đưa ra, rằng chúng ta có thể viết HCnxthr thay cho HCnxt, là rất cẩu thả (một thuật ngữ lịch sự để chỉ sai). Không phải mọi bước HCnxt đều thay đổi hr . Hãy xem xét một trạng thái trong đó hr có một giá trị nào đó không phải là số—có thể là giá trị ∞. Bước HCnxt bắt đầu ở trạng thái như vậy sẽ đặt giá trị mới của hr thành ∞ + 1. Chúng tôi không biết ∞ + 1 bằng bao nhiêu; nó có thể bằng hoặc không bằng ∞. Nếu đúng như vậy thì bước HCnxt không thay đổi hr, vì vậy nó không phải là bước HCnxthr. May mắn thay, những trạng thái trong đó giá trị của hr không phải là số thì không liên quan. Bởi vì chúng ta đang liên kết điều kiện sống với đặc tả an toàn HC nên chúng ta chỉ quan tâm đến các hành vi thỏa mãn HC . Trong tất cả các hành vi như vậy, hr luôn là một số và mỗi bước HCnxt là một bước HCnxthr. Do đó, HC HCnxthr tương đương với 3 công thức nonTLA HC HCnxt.

Khi viết các thuộc tính sống động, cú pháp của TLA thường buộc chúng ta phải viết Av thay vì A cho một số hành động A. Như trong trường hợp của HCnxt, đặc tả an toàn thường ngụ ý rằng bất kỳ bước A nào cũng sẽ thay đổi một số biến. Để tránh phải suy nghĩ về những biến A thực sự thay đổi, chúng ta thường lấy chỉ số dưới v là bộ của tất cả các biến, bộ này sẽ được thay đổi nếu có bất kỳ biến nào thay đổi. Nhưng nếu A cho phép các bước nói lắp thì sao? Thật ngớ ngẩn khi khẳng định rằng cuối cùng thì bước nói lắp cũng xảy ra, vì khẳng định như vậy không phải là bất biến trong tình trạng nói lắp. Vì vậy, nếu A cho phép các bước lặp lại, chúng tôi không muốn yêu cầu rằng bước A cuối cùng phải xảy ra mà là một bước A không lặp lại xảy ra—nghĩa là bước Av , trong đó v là bộ tất cả các biến của đặc tả. Cú pháp của TLA buộc chúng ta phải nói những gì chúng ta muốn nói.

Khi thảo luận về công thức, tôi thường bỏ qua dấu ngoặc nhọn và ký tự phụ. Ví dụ, tôi có thể mô tả HCnxthr như sự khẳng định rằng có vô số bước HCnxt, thay vì vô số Hnxthr, đó là điều nó thực sự khẳng định. Điều này kết thúc việc lạc,đề; bây giờ chúng ta quay lại việc xác định các điều kiện tồn tại.

---

3Mặc dù HC HCnxt không phải là công thức TLA nhưng ý nghĩa của nó đã được xác định, vì vậy chúng ta có thể xác định xem nó có tương đương với công thức TLA hay không.

Hãy sửa đổi thông số kỹ thuật của kênh mô-đun (Hình 3.2 trên trang 30) để yêu cầu mọi giá trị được gửi cuối cùng đều được nhận. Chúng tôi thực hiện điều này bằng cách kết hợp điều kiện sống động với Spec. Tương tự của điều kiện sống động của đồng hồ là Rcv chan, khẳng định rằng có vô số bước Rcv. Tuy nhiên, chỉ có thể nhận được giá trị đã gửi, do đó, điều kiện này cũng yêu cầu phải gửi vô số giá trị— một yêu cầu mà chúng tôi không muốn đưa ra. Chúng tôi muốn cho phép các hành vi trong đó không có giá trị nào được gửi đi, do đó không có giá trị nào được nhận. Chúng tôi chỉ yêu cầu bất kỳ giá trị nào được gửi cuối cùng sẽ được nhận.

Để đảm bảo rằng tất cả các giá trị cần nhận cuối cùng đều được nhận, chỉ cần yêu cầu giá trị tiếp theo cuối cùng cũng được nhận.
(Khi giá trị đó đã được nhận, giá trị sau nó sẽ trở thành giá trị tiếp theo được nhận, vì vậy cuối cùng nó phải được nhận, v.v.) Chính xác hơn, chúng ta chỉ cần yêu cầu nó luôn phải như vậy, nếu có một giá trị cần nhận thì cuối cùng giá trị tiếp theo sẽ được nhận. Giá trị tiếp theo được nhận bởi bước Rcv, do đó yêu cầu là4

(Có giá trị chưa nhận được Rcv chan)

Có một giá trị chưa được nhận nếu hành động Rcv được bật, nghĩa là có thể thực hiện bước Rcv. TLA+ xác định đã bật A làm vị từ đúng nếu hành động A được bật. Điều kiện sống động sau đó có thể được viết

(8.6) (đã bật Rcv chan Rcv chan)

Trong công thức được kích hoạt, việc chúng ta viết Rcv hay Rcv chan đều không thành vấn đề . Chúng ta thêm dấu ngoặc nhọn để hai hành động xuất hiện trong công thức

Trong bất kỳ hành vi nào đáp ứng tiêu chuẩn an toàn HC , giống nhau. luôn có thể thực hiện bước HCnxt thay đổi hr . Do đó, HCnxthr luôn được kích hoạt, vì vậy HCnxthr được kích hoạt là đúng trong suốt hành vi như vậy. Vì true HCnxthr tương đương với HCnxthr nên chúng ta có thể thay thế,điều kiện tồn tại HCnxthr cho đồng hồ giờ bằng

(đã bật HCnxthr HCnxthr )

Điều này gợi ý điều kiện tồn tại chung sau đây cho một hành động A:

(đã bật Av Av )

Điều kiện này khẳng định rằng, nếu A được bật thì bước A cuối cùng sẽ xảy ra—ngay cả khi A vẫn được bật chỉ trong một phần nano giây và không bao giờ được bật lại. Khó khăn thực tế rõ ràng khi thực hiện một điều kiện như vậy cho thấy rằng nó quá mạnh. Vì vậy, chúng tôi thay thế nó bằng công thức yếu hơn WFv (A), được xác định bằng

(8.7) ( bật Av Av )

4 (F G) bằng F G, vì vậy chúng ta có thể viết công thức này gọn hơn với . Tuy nhiên, sẽ thuận tiện hơn nếu giữ nó ở dạng (F G)

Công thức này khẳng định rằng, nếu A được kích hoạt vĩnh viễn thì bước A cuối cùng phải xảy ra. WF là viết tắt của Công bằng yếu, và điều kiện WFv (A) được gọi là công bằng yếu trên A. Chúng ta sẽ sớm thấy rằng các điều kiện sống động của chúng ta cho đồng hồ và kênh có thể được viết dưới dạng công thức WF. Nhưng trước tiên, hãy xem xét (8.7) và hai công thức sau đây, chúng tương đương với nó:

(8.8) (đã bật Av ) Av (8.9) (đã bật Av ) Av

Ba công thức này có thể được diễn đạt bằng tiếng Anh như sau:

(8.7) Luôn luôn xảy ra trường hợp nếu A được bật vĩnh viễn thì cuối cùng sẽ có bước A xảy ra.

(8.8) A thường xuyên bị vô hiệu hóa hoặc xảy ra vô số bước A.

(8.9) Nếu cuối cùng A được kích hoạt vĩnh viễn thì có vô số bước A xảy ra.

Sự tương đương của ba công thức này là không rõ ràng. Cố gắng suy luận sự tương đương của chúng từ các cách diễn đạt tiếng Anh thường dẫn đến nhầm lẫn. Cách tốt nhất để tránh nhầm lẫn là sử dụng toán học. Chúng ta chứng minh rằng ba công thức này tương đương bằng cách chứng minh rằng (8.7) tương đương với (8.8) và (8.8) tương đương với (8.9). Thay vì chứng minh rằng chúng tương đương với một hành vi cá nhân, chúng ta có thể sử dụng các phép lặp thừa mà chúng ta đã thấy để chứng minh trực tiếp sự tương đương của chúng.
Đây là bằng chứng cho thấy (8.7) tương đương với (8.8). Nghiên cứu nó sẽ giúp bạn học cách viết các điều kiện sống.

( đã bật Av Av )
≡ (и enabled Av Av ) Bởi vì (F G) ≡ (и F G). ≡ ( enabled Av Av ) Bởi vì и F ≡ €F. ≡ (đã bật Av Av ) Bởi vì F G ≡ (F G).
≡ (-enabled Av ) Av Bởi vì (F G) ≡ F G.

Sự tương đương của (8.8) và (8.9) được chứng minh như sau:

(-enabled Av ) Av
≡ и (đã bật Av ) Av Bởi vì и F ≡ и F ≡ и F. ≡ (đã bật Av ) Av Bởi vì (F G) ≡ (€F G).

Bây giờ chúng ta chỉ ra rằng các điều kiện sống động cho đồng hồ giờ và kênh có thể được viết dưới dạng điều kiện công bằng yếu.

Đầu tiên, hãy xem xét đồng hồ giờ. Trong mọi hành vi thỏa mãn HC , bước HCnxthr luôn được bật, vì vậy (đã bật HCnxthr ) bằng đúng. Do đó, HC suy ra rằng WFhr (HCnxt) bằng (8,9) tương đương với công thức HCnxthr , điều kiện sống động của chúng tôi cho đồng hồ giờ.

Bây giờ, hãy xem xét kênh. Tôi khẳng định rằng điều kiện sống động (8.6) có thể được thay thế bằng WFchan(Rcv). Chính xác hơn, Spec ngụ ý rằng hai công thức này tương đương nhau, do đó, việc kết hợp một trong hai công thức này với Spec sẽ mang lại các thông số kỹ thuật tương đương. Bằng chứng dựa trên quan sát rằng, trong bất kỳ hành vi nào đáp ứng Spec, khi Rcv được bật (vì một giá trị đã được gửi), nó chỉ có thể bị vô hiệu hóa bởi bước Rcv (nhận giá trị). Nói cách khác, luôn luôn xảy ra trường hợp nếu Rcv được bật thì nó sẽ được bật vĩnh viễn hoặc cuối cùng một bước Rcv sẽ xảy ra. Nói một cách chính thức, quan sát này khẳng định rằng Spec ngụ ý

(8.10) ( đã bật Rcv chan (đã bật Rcv chan) Rcv chan )

Chúng ta chứng tỏ rằng chúng ta có thể lấy WFchan(Rcv) làm điều kiện sống động bằng cách chỉ ra rằng (8.10) dẫn đến sự tương đương của (8.6) và WFchan(Rcv).

Bằng chứng chỉ bằng lý luận thuần tuý thời gian; chúng tôi không cần thông tin thực tế nào khác về đặc tả kênh. Để thu gọn và nhấn mạnh tính tổng quát trong lập luận của chúng ta, hãy thay thế Rcv chan bằng E và Rcv chan bằng A.
Sử dụng phiên bản (8.7) của định nghĩa WF, chúng ta phải chứng minh

(8.11) (E E A) ( (E A) ≡ ( E A))

Cho đến nay, tất cả các bằng chứng của chúng tôi đều được thực hiện bằng tính toán. Nghĩa là, chúng ta đã chứng minh rằng hai công thức là tương đương hoặc một công thức tương đương với giá trị đúng bằng cách chứng minh một chuỗi các giá trị tương đương. Đó là một cách hay để chứng minh những điều đơn giản, nhưng thường tốt hơn là giải một công thức phức tạp như (8.11) bằng cách chia chứng minh của nó thành từng phần. Chúng ta phải chứng minh rằng một công thức bao hàm sự tương đương của hai công thức khác. Sự tương đương của hai công thức có thể được chứng minh bằng cách chỉ ra rằng công thức này hàm ý công thức kia. Tổng quát hơn, để chứng minh P suy ra Q ≡ R, ta chứng minh P Q suy ra R và P R suy ra Q. Vì vậy, chúng ta chứng minh (8.11) bằng cách chứng minh hai công thức

(8.12) (E E A) (E A) ( E A) (8.13) (E

E A) ( E A) (E A)

Cả (8.12) và (8.13) đều có dạng F G H . Đầu tiên chúng ta chỉ ra rằng, với mọi công thức F, G và H, chúng ta có thể,suy ra F G H bằng cách chứng minh F G H . Chúng ta làm điều này bằng cách giả sử F G H và chứng minh F G H như sau:

1. (F G) H

   Chứng minh: Giả sử F G H và Quy tắc tổng quát hóa ngụ ý (trang 95), thay F G cho F và H cho G trong quy tắc.

2. F G H

   Chứng minh: Theo bước 1 và đồng thừa (F G) ≡ F G.

Điều này chứng tỏ rằng chúng ta có thể suy ra F G H bằng cách chứng minh F G H, với mọi F, G và H. Do đó chúng ta có thể chứng minh (8.12) và (8.13) bằng cách chứng minh

(8.14) (E E A) (E A) ( E A)

(8.15) (E E A) ( E A) (E A)

Việc chứng minh (8.14) rất dễ dàng. Trên thực tế, chúng ta thậm chí không cần liên từ đầu tiên; chúng tôi có thể chứng minh (E A) ( E A) như sau:

(E A)
≡ ( E E) (E A) Bởi vì E E là lặp thừa thời gian.
( E A) Bằng phép lặp đồng nghĩa (P Q) (Q R) (P R).

Chứng minh (8.15) chỉ sử dụng logic mệnh đề. Chúng ta suy ra (8.15) bằng cách thay E cho P, E cho Q và A cho R trong logic mệnh đề sau lặp lại:

(P Q R) (Q R) (P R)

Một chút suy nghĩ sẽ làm cho giá trị của phép lặp thừa này trở nên hiển nhiên. Nếu không, bạn có thể kiểm tra nó bằng cách xây dựng một bảng chân lý.

Chứng minh (8.14) và (8.15) này hoàn thành chứng minh mà ta có thể nhận được WFchan(Rcv) thay vì (8.7) làm điều kiện sống động của chúng tôi cho kênh.

## 8.5 Đặc tả bộ nhớ

### 8.5.1 Yêu cầu về tính sống động

Bây giờ chúng ta hãy tăng cường đặc tả bộ nhớ tuyến tính hóa của Phần 5.3 với yêu cầu về tính sống động mà mọi yêu cầu đều phải nhận được phản hồi. (Chúng tôi không yêu cầu phải đưa ra yêu cầu.) Yêu cầu về độ sống được kết hợp với đặc tả bộ nhớ trong, công thức ISpec của InternalMemory mô-đun (Hình 5.2 trên trang 52–53).

Chúng tôi muốn thể hiện yêu cầu về tính sống động ở mức độ công bằng yếu. ĐẾN làm điều này, chúng ta phải hiểu khi nào hành động được kích hoạt. Hành động Rsp(p) là chỉ được kích hoạt nếu hành động

(8.16) Trả lời(p, buf [p], memInt, memInt )

được kích hoạt. Hãy nhớ lại rằng toán tử Reply là một tham số không đổi, được khai báo trong mô-đun MemoryInterface (Hình 5.1 trên trang 48). Không biết thêm về toán tử này, chúng tôi không thể nói khi nào hành động (8.16) được bật.

Giả sử rằng hành động Trả lời luôn được bật. Nghĩa là, đối với bất kỳ giá trị pro-và bất tiếp tục p và trả lời r , kỳ giá trị cũ nào của memInt, đều có một giá trị mới

miNew của memInt sao cho Reply(p, r , miOld, miNew) là đúng. Để đơn giản,
chúng ta chỉ giả sử rằng điều này đúng với mọi p và r ,và thêm giả định sau
đến mô-đun MemoryInterface:

giả sử p, r , miOld : miNew : Trả lời(p, r , miOld, miNew)

Chúng ta cũng nên đưa ra giả định tương tự cho Gửi, nhưng ở đây chúng ta không cần nó.

Chúng ta sẽ ký hiệu các công thức tính công bằng yếu bằng bộ tất cả các biến,
vì vậy hãy đặt tên cho bộ dữ liệu đó:

vars = memInt, mem, ctl, buf

Khi bộ xử lý p đưa ra một yêu cầu, nó sẽ kích hoạt hành động Do(p), vẫn giữ nguyên
được bật cho đến khi bước Do(p) xảy ra. Điều kiện công bằng yếu WFvars (Do(p))
ngụ ý rằng bước Do(p) này cuối cùng phải xảy ra. Bước Do(p) cho phép
Hành động Rsp(p), vẫn được kích hoạt cho đến khi bước Rsp(p) xảy ra. Kẻ yếu
điều kiện công bằng WFvars (Rsp(p)) ngụ ý rằng bước Rsp(p) này tạo ra
phản ứng mong muốn cuối cùng phải xảy ra. Do đó, yêu cầu

(8.17) Các biến WF (Do(p)) Các biến WF (Rsp(p))

đảm bảo rằng mọi yêu cầu do bộ xử lý p đưa ra cuối cùng đều phải nhận được phản hồi.
Chúng ta muốn điều kiện này được giữ cho mọi bộ xử lý p, vì vậy chúng ta có thể lấy, làm
điều kiện sống động cho đặc tả bộ nhớ, công thức

(8.18) Sự sống động = p Proc : WFvar (Do(p)) WFvars (Rsp(p))

Thông số bộ nhớ trong khi đó là ISpec Liveness.

## 8.5.2 Một cách khác để viết nó

Tôi thấy một điều kiện công bằng đơn giản hơn việc kết hợp các điều kiện công bằng. Thấy sự
kết hợp của hai công thức công bằng yếu trong định nghĩa
của Liveness khiến tôi hỏi liệu nó có thể được thay thế bằng một điều kiện công bằng yếu trên Do(p)
Rsp(p) hay không. Việc thay thế như vậy không phải lúc nào cũng có thể thực hiện được; nói chung,
các công thức WFv (A) WFv (B) và WFv (A B) không tương đương. Tuy nhiên,
trong trường hợp này, chúng ta có thể thay thế hai điều kiện công bằng bằng một điều kiện. Nếu chúng ta định nghĩa

(8.19) Sự sống động2 = p Proc : WFvar (Do(p) Rsp(p))

thì ISpec Liveness2 tương đương với ISpec Liveness. Như chúng ta sẽ thấy, điều này
sự tương đương được giữ nguyên vì bất kỳ hành vi nào thỏa mãn ISpec đều thỏa mãn các điều sau
hai thuộc tính:

DR1. Bất cứ khi nào Do(p) được bật, Rsp(p) không bao giờ được bật trừ khi
bước Do(p) cuối cùng sẽ xảy ra.

DR2. Bất cứ khi nào Rsp(p) được bật, Do(p) không bao giờ có thể được bật trừ khi bước Rsp(p) cuối cùng xảy ra.

Các thuộc tính này được thỏa mãn vì yêu cầu tới p được đưa ra bởi bước Req(p), được thực hiện bởi bước Do(p) và được phản hồi bởi bước Rsp(p); và sau đó, yêu cầu tiếp theo tới p có thể được đưa ra theo bước Req(p). Mỗi bước trong số này chỉ có thể thực hiện được (hành động được kích hoạt) sau khi bước trước đó xảy ra.

Bây giờ chúng ta hãy chứng minh rằng DR1 và DR2 ngụ ý rằng sự kết hợp giữa tính công bằng yếu của Do(p) và của Rsp(p) tương đương với tính công bằng yếu của Do(p) Rsp(p). Để thu gọn và nhấn mạnh tính tổng quát của những gì chúng ta đang làm, hãy thay thế Do(p), Rsp(p) và vars bằng A, B và v tương ứng.

Đầu tiên, chúng ta phải trình bày lại DR1 và DR2 dưới dạng công thức thời gian. Hình thức cơ bản của DR1 và DR2 là

Bất cứ khi nào F đúng thì G không bao giờ có thể đúng trừ khi H cuối cùng cũng đúng.

Điều này được thể hiện theo logic thời gian là (F и G H ). (Khẳng định “P trừ khi Q” chỉ có nghĩa là P Q.) Thêm các chỉ số dưới phù hợp, do đó chúng ta có thể viết DR1 và DR2 theo logic thời gian như

DR1 = ( đã bật Av и đã bật B v Av ) v )

DR2 = ( đã bật B v и đã bật Av B

Mục tiêu của chúng tôi là chứng minh

(8.20) DR1 DR2 (WFv (A) WFv (B) ≡ WFv (A B) )

Điều này phức tạp nên chúng ta chia chứng minh thành từng phần. Như trong chứng minh (8.11) ở Mục 8.4 ở trên, chúng ta chứng minh sự tương đương bằng cách chứng minh hai hệ quả. Để chứng minh (8.20), ta chứng minh hai định lý

DR1 DR2 WFv (A) WFv (B) WFv (A B)

DR1 DR2 WFv (A B) WFv (A) WFv (B)

Chúng ta chứng minh chúng bằng cách chỉ ra rằng chúng đúng với một hành vi tùy ý σ. Nói cách khác, chúng tôi chứng minh

(8.21) (σ |= DR1 DR2 WFv (A) WFv (B)) (σ |= WFv (A B)) (8.22) (σ |= DR1 DR2 WFv (A B) ) (σ |= WFv (A) WFv (B))

Những công thức này có vẻ khó khăn. Bất cứ khi nào bạn gặp khó khăn trong việc chứng minh điều gì đó, hãy thử chứng minh bằng phản chứng; nó cung cấp cho bạn một giả thuyết bổ sung miễn phí - cụ thể là sự phủ định của điều bạn đang cố gắng chứng minh. Chứng minh bằng phản chứng đặc biệt hữu ích trong logic thời gian. Để chứng minh (8.21) và (8.22) bằng phản chứng, chúng ta cần tính (σ |= WFv (C )) cho một hành động C . Từ định nghĩa (8.7) của WF, ta dễ dàng có được

(8.23) (σ |= WFv (C )) ≡ n Nat : (σ +n |= đã bật C v ) (σ +n |= C v )

Điều này và tautology

и ( x S : P Q) ≡ ( x S : P €Q)

của logic vị ngữ mang lại

(8.24) и (σ |= WFv (C )) ≡
n Nat : (σ +n |= đã bật C v ) и (σ +n |= Cv )

Chúng ta cũng cần hai kết quả nữa, cả hai đều bắt nguồn từ phép lặp đồng nghĩa
A B ≡ Av B v . Kết hợp tautology này với tautology thời gian
(F G) ≡ F G mang lại

(8.25) A B v ≡ Av B v

Kết hợp tautology với quan sát rằng một hành động C D được kích hoạt
nếu hành động C hoặc hành động D được bật sẽ mang lại kết quả

(8.26) cho phép A B v ≡ đã bật Av đã bật B v

Bây giờ chúng ta có thể chứng minh (8.21) và (8.22). Để chứng minh (8.21), ta giả sử rằng σ thỏa mãn
DR1, DR2, WFv (A) và WFv (B), nhưng nó không thỏa mãn WFv (A B) và
chúng ta thu được một mâu thuẫn. Theo (8.24), giả sử σ không thỏa mãn
WFv (A B) có nghĩa là tồn tại một số số n sao cho

(8.27) σ +n |= đã bật A B v

(8.28) â(σ +n |= A B v )

Ta thu được mâu thuẫn từ (8.27) và (8.28) như sau:

1. €(σ +n |= Av ) и (σ +n |= B v )
   Chứng minh: Theo (8.28) và (8.25), sử dụng đồng nghĩa ¨(P Q) ≡ (€P §Q).
2. (a) (σ +n |= đã bật Av ) (σ +n |= -enabled B v )
   (b) (σ +n |= đã bật B v ) (σ +n |= и đã bật Av )
   Chứng minh: Theo định nghĩa DR1, giả sử σ |= DR1 ngụ ý
   (σ +n |= đã bật Av )
   (σ +n |= и đã bật B v ) (σ +n |= Av )
   và phần (a) suy ra từ 1. Chứng minh (b) tương tự.
3. (a) (σ +n |= đã bật Av ) (σ +n |= đã bật Av )
   (b) (σ +n |= đã bật B v ) (σ +n |= đã bật B v )
   Chứng minh: Phần (a) theo sau 2(a), (8.27), (8.26), và đồng nghĩa lặp lại theo thời gian
   (F G) €G F
   Chứng minh phần (b) tương tự.

4. (a) (σ +n |= đã bật Av ) (σ +n |= Av )
(b) (σ +n |= đã bật B v ) (σ +n |= B v )

Chứng minh: Giả sử σ |= WFv (A) và (8.23) suy ra

(σ +n |= enabled Av ) (σ +n |= Av )

Phần (a) tiếp nối phần này và 3(a). Chứng minh phần (b) tương tự.

5. (σ +n |= Av ) (σ +n |= B v )

Chứng minh: Vì F suy ra F nên với mọi F, (8.27) và (8.26) ngụ ý

(σ +n |= đã bật Av ) (σ +n |= đã bật B v )

Tiếp theo là bước 5 bằng logic mệnh đề từ bước 4.

Bước 1 và 5 cung cấp sự mâu thuẫn cần thiết.

Chúng ta có thể chứng minh (8.22) bằng cách giả sử rằng σ thỏa mãn DR1, DR2 và WFv (A B), rồi chứng minh nó thỏa mãn WFv (A) và WFv (B). Chúng tôi chỉ chứng minh rằng thỏa mãn WFv (A); chứng minh cho WFv (B) cũng tương tự. Bằng chứng là do mâu thuẫn; chúng ta giả sử rằng σ không thỏa mãn WFv (A) và dẫn đến mâu thuẫn. Bởi (8.24), giả định rằng σ không thỏa mãn WFv (A) có nghĩa là tồn tại một số số n sao cho

(8.29) σ +n |= đã bật Av

(8.30) - (σ +n |= Av )

Ta thu được điều mâu thuẫn như sau:

1. σ +n |= A B v

Chứng minh: Từ (8.29) và (8.26) ta suy ra σ +n |= đã bật A B v . Qua giả định σ |= WFv (A B) và (8.23), điều này suy ra σ +n |= A B | v .

2. σ +n = и cho phép B v

Chứng minh: Từ (8.29) ta suy ra σ +n |= đã bật Av tion ,mà theo giả định- σ |= DR1 và định nghĩa DR1 bao hàm

(σ +n |= и đã bật B v ) (σ +n |= Av )

Giả định (8.30) khi đó hàm ý σ +n |= и đã bật B v .

3. € (σ +n |= B v )

Chứng minh: Định nghĩa của Enable ngụ ý Enable B v . (MỘT v - B B bước v chỉ có thể xảy ra khi nó được bật.) Từ đây, thời gian đơn giản lý luận ngụ ý

(σ +n |= и đã bật B v ) и (σ +n |= B v )

(Một chứng minh hình thức sử dụng Quy tắc Tổng quát hóa Ngụ ý và phép lặp thừa €F ≡ и F.) Sau đó chúng ta suy ra и (σ +n |= B v ) từ 2.

4. € (σ +n |= A B v )

Chứng minh: Theo (8.30), 3, và (8.25), sử dụng phép đồng nghĩa §P ¨Q ≡ и (P Q).

Bước 1 và 4 cung cấp sự mâu thuẫn cần thiết. Điều này hoàn thành chứng minh của chúng ta về (8.22), điều này hoàn thành chứng minh của chúng ta về (8.20).

## 8.5.3 Khái quát hóa

Công thức (8.20) đưa ra quy tắc thay thế liên kết của tính công bằng yếu yêu cầu về hai hành động có tính công bằng yếu trong sự phân ly của chúng. Bây giờ chúng tôi khái quát nó từ hai hành động A và B thành n hành động A1, . . . , MỘT. Sự khái quát hóa của DR1 và DR2 là

DR(i, j) = ( đã bật Ai v и đã bật Aj v Ai v )

Nếu chúng ta thay thế A1 cho A và A2 cho B thì DR1 trở thành DR(1, 2) và DR2 trở thành DR(2, 1). Sự khái quát hóa của (8.20) là

(8.31) ( i, j 1 . . n : (i = j) DR(i, j))

(WFv (A1) . . . WFv (An) ≡ WFv (A1 . . . An))

Để quyết định xem bạn có thể thay thế sự kết hợp của các điều kiện công bằng yếu bằng một một cái duy nhất trong một đặc điểm kỹ thuật, bạn có thể sẽ thấy dễ dàng hơn khi sử dụng cái sau tuyên bố không chính thức của (8.31):

> Quy tắc kết hợp WF Nếu A1, . . . , An là những hành động sao cho, với bất kỳ phân biệt i và j, bất cứ khi nào Ai v được bật, Aj v không thể được bật trừ khi bước Ai v xảy ra, sau đó WFv (A1) . . . WFv (An) là tương đương với WFv (A1 . . . An).

Có lẽ cách tốt nhất để nghĩ về quy tắc này là như một sự khẳng định về một sự tùy ý hành vi cá nhân σ. Khi đó giả thuyết của nó là σ |= DR(i, j) đúng cho mọi phân biệt i và j; kết luận của nó là

σ |= (WFv (A1) . . . WFv (An) ≡ WFv (A1 . . . An))

Để thay thế WFv (A1) . . . WFv (An) bởi WFv (A1 . . . An) trong thông số kỹ thuật, bạn phải kiểm tra xem mọi hành vi có đáp ứng phần an toàn của thông số kỹ thuật không cũng thỏa mãn DR(i,j), với mọi i và j phân biệt.

Liên hợp và phân tách là những trường hợp định lượng đặc biệt:

F1 . . . $F_N$ ≡ i 1 . . n : Fi

F1 . . . $F_N$ ≡ i 1 . . n : Fi

Do đó, chúng ta có thể dễ dàng trình bày lại Quy tắc liên hợp WF như một điều kiện khi i S : WFv (Ai) và WFv ( i S : Ai) là tương đương với tập hữu hạn S. quy tắc kết quả thực sự hợp lệ cho mọi tập hợp S:

> Quy tắc định lượng WF Nếu, với mọi i S, Ai là những hành động sao cho, với mọi i và j khác biệt trong S, bất cứ khi nào Ai v được bật, Aj v không thể được kích hoạt trừ khi bước Ai v xảy ra, khi đó i S : WFv (Ai) là tương đương với WFv ( i S : Ai).

## 8.6 Sự công bằng mạnh mẽ

Chúng tôi xác định SFv (A), tính công bằng mạnh mẽ của hành động A, là một trong hai công thức tương đương sau:

(8.32) (đã bật Av ) Av (8.33) đã

bật Av Av

Về mặt trực giác, hai công thức này khẳng định

(8.32) Cuối cùng, A sẽ bị vô hiệu hóa vĩnh viễn hoặc có vô số bước A xảy ra.

(8.33) Nếu A được kích hoạt thường xuyên vô hạn thì có vô số bước A xảy ra.

Chứng minh (8.32) và (8.33) tương đương tương đương với chứng minh ở trang 98 chứng minh hai công thức (8.8) và (8.9) của WFv (A) là tương đương.

Định nghĩa (8.32) của SFv (A) thu được từ định nghĩa (8.8) của WFv (A) bằng cách thay thế (- đã bật Av ) bằng (- đã bật Av ). Vì F (thậm chí luôn luôn là F) mạnh hơn (ngụ ý) F (thường là F) đối với bất kỳ công thức F nào, nên độ công bằng mạnh sẽ mạnh hơn độ công bằng yếu. Chúng ta có thể diễn đạt sự công bằng yếu và mạnh như sau:

- Tính công bằng yếu của A khẳng định rằng bước A cuối cùng phải xảy ra nếu A được kích hoạt liên tục.
- Tính công bằng mạnh mẽ của A khẳng định rằng bước A cuối cùng phải xảy ra nếu A được kích hoạt liên tục.

Liên tục có nghĩa là không gián đoạn. Liên tục có nghĩa là lặp đi lặp lại, có thể bị gián đoạn.

Sự công bằng mạnh mẽ không nhất thiết phải mạnh hơn sự công bằng yếu ớt. Tính công bằng yếu và mạnh của một hành động A là tương đương nếu A thường bị vô hiệu hóa vô hạn ngụ ý rằng A cuối cùng sẽ bị vô hiệu hóa vĩnh viễn hoặc nếu không thì có vô số bước A xảy ra. Điều này được thể hiện một cách hình thức bằng phép lặp lặp

(WFv (A) ≡ SFv (A)) ≡

( (€ đã bật Av ) (€ đã bật Av ) Av )

Trong ví dụ về kênh, mức độ công bằng yếu và mạnh của Rcv là tương đương vì Spec ngụ ý rằng, khi được bật, Rcv chỉ có thể bị tắt bằng một bước Rcv. Do đó, nếu Rcv bị vô hiệu hóa thường xuyên thì cuối cùng nó sẽ bị vô hiệu hóa vĩnh viễn hoặc bị vô hiệu hóa thường xuyên bởi các bước Rcv.

Tương tự của Quy tắc kết hợp WF và Quy tắc định lượng WF (trang 105) ủng hộ sự công bằng mạnh mẽ—ví dụ:

> Quy tắc kết hợp SF Nếu A1, . . . , An là các hành động sao cho, đối với mọi i và j riêng biệt, bất cứ khi nào hành động Ai được kích hoạt, hành động Aj không thể được kích hoạt cho đến khi bước Ai xảy ra, sau đó SFv (A1) . . . SFv (An) tương đương với SFv (A1 . . . An).

Sự công bằng mạnh mẽ có thể khó thực hiện hơn sự công bằng yếu và đó là một yêu cầu ít phổ biến hơn. Chỉ nên sử dụng điều kiện công bằng mạnh mẽ trong đặc tả nếu cần thiết. Khi tính công bằng mạnh và tính công bằng yếu tương đương nhau thì tính chất công bằng phải được viết là tính công bằng yếu.

Đặc tính sống động có thể tinh tế. Việc thể hiện chúng bằng các công thức thời gian đặc biệt có thể dẫn đến sai sót. Chúng tôi sẽ chỉ định tính sống động là sự kết hợp của các thuộc tính công bằng yếu và/hoặc mạnh bất cứ khi nào có thể—và điều đó hầu như luôn luôn có thể xảy ra. Việc có một cách thể hiện sự sống động thống nhất sẽ làm cho các thông số kỹ thuật trở nên dễ hiểu hơn. Phần 8.9.2 dưới đây thảo luận về một lý do thậm chí còn thuyết phục hơn cho việc sử dụng sự công bằng để xác định tính sống động.

# 8.7 Bộ nhớ đệm ghi qua

Bây giờ chúng ta hãy thêm hoạt động vào bộ nhớ đệm ghi qua, được chỉ định trong Hình 5.5 trên trang 57–59. Chúng tôi muốn đặc tả của mình đảm bảo rằng mọi yêu cầu cuối cùng đều nhận được phản hồi mà không yêu cầu bất kỳ yêu cầu nào được đưa ra. Điều này đòi hỏi sự công bằng đối với tất cả các hành động tạo nên hành động ở trạng thái tiếp theo Tiếp theo ngoại trừ những hành động sau:

- Hành động Req(p), đưa ra yêu cầu.
- Hành động Evict(p, a), loại bỏ một địa chỉ khỏi bộ đệm.
- Một hành động MemQWr, nếu memQ chỉ chứa các yêu cầu ghi và không đầy đủ (có ít hơn phần tử QLen). Do phản hồi cho yêu cầu ghi có thể được đưa ra trước khi giá trị được ghi vào bộ nhớ, nên việc không thực hiện hành động MemQWr chỉ có thể ngăn phản hồi nếu nó ngăn chặn việc xếp hàng của một thao tác đọc trong memQ hoặc việc xếp hàng một thao tác (vì memQ là đầy).

Để đơn giản, hãy yêu cầu sự công bằng cho hành động MemQWr nữa; chúng tôi sẽ làm suy yếu yêu cầu này sau. Điều kiện sống của chúng ta khi đó phải khẳng định tính công bằng của hành động

```
MemQWr MemQRd Rsp(p) RdMiss(p) DoRd(p) DoWr (p)
```

cho tất cả p trong Proc. Bây giờ chúng ta phải quyết định nên khẳng định sự công bằng yếu hay mạnh đối với những hành động này. Tính công bằng yếu và mạnh tương đương với một hành động mà một khi đã được kích hoạt thì vẫn được kích hoạt cho đến khi nó được thực thi. Đây là trường hợp của tất cả các hành động này ngoại trừ DoRd(p), RdMiss(p) và DoWr (p).

Hành động DoRd(p) có thể bị vô hiệu hóa bằng bước Loại bỏ để loại bỏ dữ liệu được yêu cầu lại khỏi bộ nhớ đệm. Trong trường hợp này, tính công bằng của các hành động khác sẽ ngụ ý rằng dữ liệu cuối cùng sẽ được trả về bộ đệm, kích hoạt lại DoRd(p).

Dữ liệu không thể bị loại bỏ một lần nữa cho đến khi hành động DoRd(p) được thực thi và tính công bằng yếu sau đó đủ để đảm bảo rằng bước DoRd(p) cần thiết cuối cùng xảy ra.

Hành động RdMiss(p) và DoWr (p) sẽ thêm một yêu cầu vào hàng đợi memQ. Chúng bị vô hiệu hóa nếu hàng đợi đó đầy. RdMiss(p) hoặc DoWr (p) có thể được bật và sau đó bị vô hiệu hóa vì RdMiss(q) hoặc DoWr (q), đối với bộ xử lý q khác, sẽ thêm yêu cầu vào memQ. Do đó, chúng tôi cần sự công bằng mạnh mẽ cho các hành động RdMiss(p) và DoWr (p). Vì vậy, điều kiện công bằng chúng ta cần là

Tính công bằng yếu đối với Rsp(p), DoRd(p), MemQWr và MemQRd

Tính công bằng cao cho RdMiss(p) và DoWr (p).

Như trước đây, hãy định nghĩa vars là bộ của tất cả các biến.

vars = memInt, wmem, buf , ctl, cache, memQ

Chúng ta chỉ có thể viết điều kiện sống động như

(8.34) p Proc : WFvar (Rsp(p)) WFvars (DoRd(p))
SFvar (RdMiss(p)) SFvar (DoWr (p))
WFvar (MemQWr ) WFvar (MemQRd)

Tuy nhiên, tôi thích thay thế sự kết hợp của các điều kiện công bằng bằng một điều kiện công bằng duy nhất trên một phép tách, như chúng ta đã làm trong Phần 8.5 về đặc tả bộ nhớ. Quy tắc kết hợp WF và SF (trang 105 và 106) ngụ ý rằng điều kiện tồn tại (8.34) có thể được viết lại thành

(8.35) p Proc : WFvar (Rsp(p) DoRd(p))
SFvar (RdMiss(p) DoWr (p))
WFvar (MemQWr MemQRd)

Bây giờ chúng ta có thể cố gắng đơn giản hóa (8.35) bằng cách di chuyển bộ định lượng bên trong các công thức WF và SF. Đầu tiên, vì phân phối trên , nên chúng ta có thể viết lại liên hợp thứ nhất của (8.35) thành

(8.36) p Proc : WFvars (Rsp(p) DoRd(p))
p Proc : SFvars (RdMiss(p) DoWr (p))

Bây giờ chúng ta có thể thử áp dụng Quy tắc định lượng WF (trang 105) cho liên kết đầu tiên của (8.36) và Quy tắc định lượng SF tương ứng cho liên hợp thứ hai của nó. Tuy nhiên, quy tắc định lượng WF không áp dụng cho liên từ đầu tiên. Có thể kích hoạt đồng thời cả Rsp(p) DoRd(p) và Rsp(q) DoRd(q) cho hai bộ xử lý p và q khác nhau. Công thức

(8.37) WFvar ( p Proc : Rsp(p) DoRd(p))

được thỏa mãn bởi bất kỳ hành vi nào trong đó có vô số hành động Rsp(p) và DoRd(p) xảy ra đối với một số bộ xử lý p. Trong hành vi như vậy, Rsp(q) có thể được kích hoạt cho một số bộ xử lý q khác mà không bao giờ xảy ra bước Rsp(q), làm cho WFvars (Rsp(q) DoRd(q)) sai, ngụ ý rằng liên kết đầu tiên của (8.36) là sai. Do đó, (8.37) không tương đương với liên hợp thứ nhất của (8.36). Tương tự, quy tắc tương tự về tính công bằng mạnh mẽ không thể áp dụng cho liên hợp thứ hai của (8.36). Công thức (8.35) đơn giản nhất mà chúng ta có thể làm được.

Hãy quay lại quan sát rằng chúng ta không phải thực thi MemQWr nếu hàng đợi memQ chỉ chứa các yêu cầu ghi và không đầy. Nói cách khác, chúng ta phải thực thi MemQWr chỉ khi memQ đầy hoặc chứa yêu cầu đọc.
Hãy xác định

QCond = Len(memQ) = QLen i
1 . . Len(memQ): memQ[i][2].op = “Rd”

vì vậy cuối cùng chúng ta chỉ cần thực hiện một hành động MemQWr khi nó được bật và QCond là đúng, đó là trường hợp nếu hành động QCond MemQWr được bật.
Trong trường hợp này, bước MemQWr là bước QCond MemQWr. Do đó, chỉ cần yêu cầu tính công bằng yếu của hành động QCond MemQWr . Do đó chúng ta có thể thay thế liên hợp thứ hai của (8.35) bằng

WFvar ((QCond MemQWr ) MemQRd)

Chúng tôi sẽ làm điều này nếu chúng tôi muốn đặc tả mô tả điều kiện tồn tại yếu nhất thực hiện điều kiện tồn tại của đặc tả bộ nhớ. Tuy nhiên, nếu thông số kỹ thuật là mô tả của một thiết bị thực tế thì thiết bị đó có thể sẽ triển khai tính công bằng yếu trên tất cả các hành động MemQWr, vì vậy chúng tôi sẽ lấy (8.35) làm điều kiện về tính sống động.

# 8.8 Định lượng

Phần 8.1 mô tả ý nghĩa của việc định lượng thông thường các công thức thời gian. Ví dụ: ý nghĩa của công thức r : F, đối với bất kỳ công thức thời gian F nào, được xác định bởi

σ |= ( r : F) = r : (σ |= F)

trong đó σ là hành vi bất kỳ .

Ký hiệu r trong r : F thường được gọi là biến giới hạn. Nhưng chúng ta đang sử dụng thuật ngữ biến để có ý nghĩa khác—điều gì đó được khai báo bằng một câu lệnh biến trong mô-đun. “Biến” r bị ràng buộc thực chất là một hằng số

trong các công thức này—một giá trị giống nhau ở mọi trạng thái của hành vi.5 Vì
Ví dụ: công thức r : (x = r ) khẳng định rằng x có cùng giá trị trong mọi
trạng thái của một hành vi.

Định lượng giới hạn trên một tập hợp S được xác định bởi

σ |= ( r S : F) = ( r S : σ |= F)

σ |= ( r S : F) = ( r S : σ |= F)

Ký hiệu r được khai báo là hằng số trong công thức F. Biểu thức S nằm
nằm ngoài phạm vi khai báo của r nên ký hiệu r không thể xuất hiện trong S. Đó là
dễ dàng xác định ý nghĩa của các công thức này ngay cả khi S không phải là hằng số—vì
ví dụ, bằng cách cho r S : F bằng r : (r S) F. Tuy nhiên, đối với hằng số
S, tốt hơn nên viết r : (r S) F một cách rõ ràng.

Cũng dễ dàng xác định ý nghĩa của phép chọn như một toán tử thời gian. Chúng tôi
chỉ có thể đặt σ |= (chọn r : F) là một giá trị hằng số r tùy ý sao cho
σ |= F bằng đúng nếu r như vậy tồn tại. Tuy nhiên, toán tử chọn tạm thời
không cần thiết để viết thông số kỹ thuật, vì vậy hãy chọn r : F không phải là TLA+ hợp pháp
công thức nếu F là công thức thời gian.

Bây giờ chúng ta đến với bộ định lượng tồn tại theo thời gian . Trong công thức x : F,
ký hiệu x được khai báo là một biến trong F. Không giống như r : F, khẳng định
sự tồn tại của một giá trị r công thức x : F khẳng định sự tồn tại của một giá trị
cho x trong mỗi trạng thái của một hành vi. Ví dụ, nếu y là một biến thì
công thức x : (x y) khẳng định rằng y luôn có phần tử x nào đó nên y luôn bằng
một tập hợp khác rỗng. Tuy nhiên, phần tử x có thể khác nhau ở các trạng thái khác nhau,
vì vậy các giá trị của y ở các trạng thái khác nhau có thể khác nhau.

Chúng tôi đã sử dụng làm toán tử ẩn, coi x : F là F với
biến x ẩn. Định nghĩa chính xác của hơi phức tạp vì, như đã thảo luận trong Phần 8.1, công thức
x : F sẽ bất biến khi nói lắp.
Theo trực giác, x : F được thỏa mãn bởi một hành vi σ iff F được thỏa mãn bởi một hành vi τ
thu được từ σ bằng cách thêm và/hoặc xóa các bước nói lắp và thay đổi giá trị của x . Một định nghĩa
chính xác xuất hiện trong Phần 16.2.4 (trang 314).
Tuy nhiên, để viết thông số kỹ thuật, bạn có thể nghĩ đơn giản x : F là F với x
ẩn giấu.

TLA cũng có bộ định lượng phổ quát theo thời gian , được định nghĩa bởi

x : F = и x : F

Toán tử này hiếm khi được sử dụng. TLA+ không cho phép các phiên bản giới hạn của
toán tử và .

5Các nhà logic học sử dụng thuật ngữ biến linh hoạt cho biến TLA và thuật ngữ biến cứng nhắc
cho một ký hiệu như r đại diện cho một hằng số.

# 8.9 Kiểm tra logic thời gian

## 8.9.1 Đánh giá

Chúng ta hãy xem hình dạng của các thông số kỹ thuật mà chúng tôi đã viết cho đến nay. Chúng tôi bắt đầu với hình thức đơn giản

(8.38) Ban đầu [Tiếp theo]vars

trong đó Init là vị từ ban đầu, Tiếp theo là hành động trạng thái tiếp theo và vars là bộ tất cả các biến. Về nguyên tắc, loại đặc tả này khá đơn giản.
Sau đó, chúng tôi đã giới thiệu tính năng ẩn, sử dụng để liên kết các biến không xuất hiện trong đặc tả. Các biến bị ràng buộc đó, còn được gọi là biến ẩn hoặc biến nội bộ, chỉ dùng để giúp mô tả giá trị của các biến tự do (còn gọi là biến hiển thị) thay đổi như thế nào.

Việc ẩn các biến là đủ dễ dàng và nó rất tao nhã về mặt toán học và thỏa mãn về mặt triết học. Tuy nhiên, trong thực tế, nó không tạo ra nhiều khác biệt đối với thông số kỹ thuật. Một nhận xét cũng có thể cho người đọc biết rằng một biến nên được coi là nội bộ. Ẩn rõ ràng cho phép thực hiện có nghĩa là hàm ý.
Một đặc tả cấp thấp hơn mô tả việc triển khai có thể được cho là chỉ ám chỉ một đặc tả cấp cao hơn nếu các biến nội bộ của đặc tả cấp cao hơn, có giá trị không thực sự quan trọng, được ẩn rõ ràng. Mặt khác, việc triển khai có nghĩa là triển khai theo ánh xạ sàng lọc. (Xem Phần 5.8.) Tuy nhiên, như được giải thích ở Phần 10.8 bên dưới, việc thực hiện cũng thường liên quan đến việc sàng lọc các biến số hiển thị.

Để thể hiện sự sống động, đặc tả (8.38) được tăng cường về dạng

(8.39) Ban đầu [Tiếp theo]vars Sự sống động

trong đó Tính sống động là sự kết hợp của các công thức có dạng WFvars (A) và/hoặc SFvars (A), cho hành động A. (Tôi đang coi lượng hóa phổ quát là một dạng kết hợp.)

## 8.9.2 Đóng máy

Trong các đặc tả của biểu mẫu (8.39) mà chúng ta đã viết cho đến nay, các hành động A có đặc tính công bằng xuất hiện trong công thức Tính sống động có một điểm chung: chúng đều là các hành động phụ của hành động trạng thái tiếp theo Tiếp theo. Một hành động A là một hành động phụ của Next nếu mỗi bước A là một bước Tiếp theo. Tương tự, A là một phân nhóm của Next iff Liveness hàm ý Next[6] A. Trong hầu hết tất cả các thông số kỹ thuật có dạng (8.39), công thức

[6]Chúng ta cũng có thể sử dụng định nghĩa yếu hơn về hành động phụ sau đây: A là hành động con của for-mula (8.38) iff, với mọi trạng thái s của mọi hành vi thỏa mãn (8.38), nếu A được bật ở trạng thái s thì Tiếp theo A cũng được bật trong s.

phải là sự kết hợp của các công thức công bằng yếu và/hoặc mạnh cho các hành động phụ của Tiếp theo. Bây giờ tôi sẽ giải thích tại sao.

Khi xem xét đặc tả (8.39), chúng tôi mong đợi Init sẽ hạn chế trạng thái ban đầu, Next để hạn chế những bước có thể xảy ra và Liveness chỉ mô tả những gì cuối cùng phải xảy ra. Tuy nhiên, hãy xem xét công thức sau:

(8.40) (x = 0) [x = x + 1]x WFx ((x > 99) (x = x 1))

Hai liên kết đầu tiên của (8.40) khẳng định rằng x ban đầu bằng 0 và bất kỳ bước nào cũng tăng x lên 1 hoặc giữ nguyên. Do đó, chúng ngụ ý rằng nếu x vượt quá 99 thì nó mãi mãi lớn hơn 99. Thuộc tính công bằng yếu khẳng định rằng, nếu điều này xảy ra thì x cuối cùng phải giảm đi 1—mâu thuẫn với liên từ thứ hai. Do đó, (8.40) ngụ ý rằng x không bao giờ có thể vượt quá 99, nên nó tương đương với

(x = 0) [(x < 99) (x = x + 1)]x

Việc kết hợp tính chất công bằng yếu với hai liên kết đầu tiên của (8.40) sẽ cấm x = x + 1 bước khi x = 99.

Một đặc tả của dạng (8.39) được gọi là máy đóng nếu Liveness liên hợp không ràng buộc trạng thái ban đầu cũng như các bước có thể xảy ra. Một cách tổng quát hơn để diễn đạt điều này như sau. Giả sử một hành vi hữu hạn là một chuỗi hữu hạn các trạng thái.7 Chúng ta nói rằng một hành vi hữu hạn σ thỏa mãn một tính chất an toàn S nếu hành vi thu được bằng cách thêm vô số bước lặp lại vào cuối của σ thỏa mãn S. Nếu S là một tính chất an toàn, thì chúng ta định nghĩa cặp công thức S, L là máy đóng nếu mọi hành vi hữu hạn thỏa mãn S đều có thể được mở rộng thành một hành vi vô hạn thỏa mãn S L. Chúng ta gọi (8.39) máy đóng nếu cặp công thức Init [Tiếp theo ]vars , Liveness đã được đóng máy.

Chúng tôi hiếm khi muốn viết một đặc tả không được đóng bằng máy. Nếu chúng tôi viết một cái thì thường là do nhầm lẫn. Thông số kỹ thuật (8.39) được đảm bảo đóng máy nếu Liveness là sự kết hợp của các thuộc tính công bằng yếu và/hoặc mạnh cho các hành động phụ của Tiếp theo.[¨] Điều kiện này không đúng với thông số kỹ thuật (8.40), không phải là máy đóng, bởi vì (x > 99) (x = x 1) không phải là một hệ con của x = x + 1.

Yêu cầu về sự sống được thỏa mãn về mặt triết học. Đặc tả của biểu mẫu (8.38), chỉ xác định thuộc tính an toàn, cho phép các hành vi trong đó hệ thống không làm gì cả. Do đó, đặc điểm kỹ thuật được thỏa mãn bởi một hệ thống không làm gì cả. Việc thể hiện các yêu cầu về tính sống động bằng các đặc tính công bằng sẽ ít được thỏa mãn hơn. Những thuộc tính này rất tinh vi và rất dễ mắc sai lầm.

7 Do đó, một hành vi hữu hạn không phải là một hành vi mà là một chuỗi vô hạn các trạng thái. Toán học-các nhà ematician thường lạm dụng ngôn ngữ theo cách này.

8Chính xác hơn, đây là trường hợp của một kết hợp hữu hạn hoặc vô hạn đếm được của các thuộc tính có dạng WFv (A) và/hoặc SFv (A), trong đó mỗi Av là một phân lớp của Next. Kết quả này cũng đúng với định nghĩa yếu hơn về hành động phụ trong chú thích cuối trang ở trang trước.

Cần phải suy nghĩ một chút để xác định rằng điều kiện tồn tại cho bộ nhớ đệm ghi qua, công thức (8.35) trên trang 108, có nghĩa là mọi yêu cầu đều nhận được phản hồi.

Thật hấp dẫn khi thể hiện các đặc tính sống động một cách trực tiếp hơn mà không cần sử dụng các đặc tính công bằng. Ví dụ: thật dễ dàng để viết một công thức tạm thời xác nhận cho bộ đệm ghi mà mọi yêu cầu đều nhận được phản hồi. Khi bộ xử lý p đưa ra một yêu cầu, nó đặt ctl[p] thành “rdy”. Chúng ta chỉ cần khẳng định rằng, với mọi bộ xử lý p, bất cứ khi nào một trạng thái trong đó ctl[p] = “rdy” là đúng xảy ra thì cuối cùng sẽ có bước Rsp(p):

(8.41) p Proc : ((ctl[p] = “rdy”) Rsp(p)vars )

Mặc dù những công thức như vậy rất hấp dẫn nhưng chúng lại nguy hiểm. Rất dễ mắc lỗi và viết một đặc tả không được đóng bằng máy.

Ngoại trừ những trường hợp bất thường, bạn nên thể hiện tính sống động bằng các đặc tính công bằng cho các hành động phụ của hành động ở trạng thái tiếp theo. Đây là những thông số kỹ thuật đơn giản nhất và do đó dễ viết và dễ hiểu nhất. Hầu hết các thông số kỹ thuật của hệ thống, ngay cả khi rất chi tiết và phức tạp, đều có thể được viết theo cách đơn giản này. Các trường hợp ngoại lệ thường nằm trong lĩnh vực các thông số kỹ thuật cấp cao, tinh tế cố gắng rất chung chung. Một ví dụ về đặc tả như vậy xuất hiện trong Phần 11.2.

## 8.9.3 Khả năng đóng máy và khả năng

Việc đóng máy có thể được coi là một điều kiện có thể xảy ra. Ví dụ: việc đóng máy của cặp S, Av khẳng định rằng với mọi hành vi hữu hạn σ thỏa mãn S, có thể mở rộng σ thành một hành vi vô hạn thỏa mãn S trong đó xảy ra vô số hành động Av . Nếu chúng ta coi S là một đặc tả hệ thống, do đó một hành vi thỏa mãn S thể hiện khả năng thực thi của hệ thống, thì chúng ta có thể phát biểu lại việc đóng máy của S, Av như sau: trong bất kỳ quá trình thực thi hệ thống nào, luôn có thể có vô số hành động Av xảy ra.

Thông số kỹ thuật TLA thể hiện các đặc tính an toàn và sống động, không phải các đặc tính khả năng. Thuộc tính an toàn khẳng định rằng điều gì đó là không thể—ví dụ: hệ thống không thể thực hiện một bước không đáp ứng hành động ở trạng thái tiếp theo. Thuộc tính liveness khẳng định rằng điều gì đó cuối cùng phải xảy ra. Các yêu cầu của hệ thống đôi khi được nêu một cách không chính thức về những gì có thể thực hiện được. Hầu hết, khi được kiểm tra một cách nghiêm ngặt, những yêu cầu này có thể được thể hiện bằng các đặc tính sống động và/hoặc an toàn. (Trường hợp ngoại lệ đáng chú ý nhất là các thuộc tính thống kê, chẳng hạn như các xác nhận về xác suất xảy ra điều gì đó.)

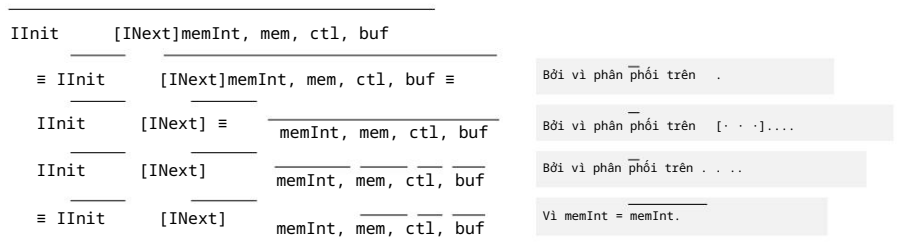
Chúng tôi không bao giờ quan tâm đến việc xác định rằng điều gì đó có thể xảy ra. Sẽ không bao giờ hữu ích nếu biết rằng hệ thống có thể đưa ra câu trả lời đúng. Chúng tôi không bao giờ phải chỉ định rằng người dùng có thể nhập “a”; chúng ta phải xác định điều gì sẽ xảy ra nếu anh ta làm vậy.

Việc đóng máy là thuộc tính của một cặp công thức, không phải của một hệ thống. Mặc dù một thuộc tính khả năng không bao giờ là một xác nhận hữu ích về một hệ thống, nhưng nó có thể là một khẳng định hữu ích về một đặc tả. Thông số kỹ thuật S của hệ thống có đầu vào bàn phím phải luôn cho phép người dùng nhập “a”. Vì vậy, mọi hành vi hữu hạn thỏa mãn S sẽ có thể mở rộng thành một hành vi vô hạn thỏa mãn S trong đó có vô số chữ “a” được gõ. Nếu hành động Av thể hiện việc gõ chữ “a”, thì việc nói rằng người dùng luôn có thể gõ vô số chữ “a” tương đương với việc nói rằng cặp S, Av phải được đóng bằng máy. Nếu S, Av chưa được đóng máy thì người dùng sẽ không thể gõ chữ “a”. Trừ khi hệ thống được phép khóa bàn phím, điều này có nghĩa là đã xảy ra lỗi với thông số kỹ thuật.

Loại tài sản khả năng này có thể được chứng minh. Ví dụ: để chứng minh rằng người dùng luôn có thể gõ vô số chữ “a”, chúng tôi chứng minh rằng việc kết hợp các điều kiện công bằng phù hợp cho các hành động đầu vào ngụ ý rằng người dùng phải gõ vô số chữ “a”. Tuy nhiên, việc chứng minh loại tính chất đơn giản này dường như không đáng nỗ lực. Khi viết một đặc tả, bạn nên đảm bảo rằng các khả năng được hệ thống thực cho phép cũng được đặc tả cho phép. Một khi bạn nhận thức được điều gì có thể thực hiện được, bạn thường sẽ gặp chút khó khăn trong việc đảm bảo rằng đặc tả kỹ thuật có thể thực hiện được điều đó. Bạn cũng nên đảm bảo rằng những gì hệ thống phải làm được ngụ ý bởi các điều kiện công bằng của đặc tả. Điều này có thể khó khăn hơn.

## 8.9.4 Ánh xạ sàng lọc và tính công bằng

Phần 5.8 (trang 62) mô tả cách chứng minh rằng bộ nhớ ghi thực hiện đặc tả bộ nhớ. Chúng ta phải chứng minh Spec ISpec, trong đó Spec là thông số kỹ thuật của bộ nhớ ghi, ISpec là thông số kỹ thuật bên trong của bộ nhớ (với các biến bên trong được hiển thị) và, đối với bất kỳ công thức F nào, chúng ta đặt F có nghĩa là F với các biểu thức omem, octl và obuf được thay thế cho các biến mem, ctl và buf. Chúng ta có thể viết lại hàm ý này thành (5.3) vì sự thay thế (chặn quá mức) phân bổ trên các toán tử như và , vì vậy chúng ta có




Việc thêm tính sống động vào các thông số kỹ thuật sẽ thêm liên kết vào các công thức Spec và ISpec. Giả sử chúng ta lấy công thức Liveness2, được định nghĩa trong (8.19) ở trang 101, là

thuộc tính sống động của ISpec. Khi đó ISpec có thêm thuật ngữ Liveness2, có thể đơn giản hóa như sau:

Sự sống động2
≡ p Proc : WFvars (Do(p) Rsp(p)) Theo định nghĩa của Liveness2.
≡ p Proc : WFvars (Do(p) Rsp(p)) Bởi vì phân phối trên .

Nhưng chúng tôi không thể tự động di chuyển bên trong WF vì sự thay thế nói chung không phân phối trên phần được kích hoạt và do đó nó không phân phối trên WF hoặc SF. Đối với các thông số kỹ thuật và ánh xạ sàng lọc xảy ra trong thực tế, bao gồm cả bản đồ này, chỉ cần thay thế từng WFv (A) bằng WFv (A) và từng SFv (A) bằng SFv (A) sẽ cho kết quả đứng. Tuy nhiên, bạn không cần phải phụ thuộc vào điều này. Thay vào đó, bạn có thể mở rộng định nghĩa của WF và SF để có được, ví dụ:

WFv (A) ≡ kích hoạt Av Av Theo định nghĩa của WF.
≡ kích hoạt Av Av Theo phân phối của .

Bạn có thể tính toán các vị từ được kích hoạt "bằng tay" và sau đó thực hiện phép thay thế. Khi tính toán các vị từ được kích hoạt, chỉ cần xem xét các trạng thái thỏa mãn phần an toàn của đặc tả là đủ, điều này thường có nghĩa là Av đã bật bằng A đã bật. Sau đó, bạn có thể tính toán các vị từ được bật bằng cách sử dụng các quy tắc sau:

1. đã bật (A B) ≡ (đã bật A) (đã bật B), cho mọi hành động A và B.

2. đã bật (P A) ≡ P (đã bật A), đối với mọi vị từ trạng thái P và hành động A

3. đã bật (A B) ≡ (đã bật A) (đã bật B), nếu A và B là các hành động sao cho cùng một biến không xuất hiện ở cả A và B.

4. đã bật (x = exp) ≡ đúng và đã bật (x exp) ≡ (exp = {}), cho bất kỳ biến x nào và hàm trạng thái exp.

Ví dụ:

đã bật (Do(p) Rsp(p))
≡ (ctl[p] = "rdy") (ctl[p] = "done") Theo quy tắc 1-4.
≡ (octl[p] = "rdy") (octl[p] = "done") Theo nghĩa của .

### 8.9.5 Sự không quan trọng của sự sống

Mặc dù quan trọng về mặt triết học, nhưng trong thực tế, tính chất sống của (8.39) không quan trọng bằng phần an toàn, Init [Next]vars . Mục đích cuối cùng của việc viết đặc tả là để tránh sai sót. Kinh nghiệm cho thấy rằng hầu hết lợi ích của việc viết và sử dụng đặc tả đều đến từ phần an toàn. Mặt khác, thuộc tính liveness thường khá dễ viết. Nó thường cấu thành ít hơn năm phần trăm của một đặc điểm kỹ thuật. Vì vậy, bạn cũng có thể viết phần sống động. Tuy nhiên, khi tìm kiếm lỗi, phần lớn nỗ lực của bạn nên dành cho việc kiểm tra phần an toàn.

### 8.9.6 Logic thời gian được coi là khó hiểu

Loại đặc tả chung nhất mà tôi đã thảo luận cho đến nay có dạng

(8.42) v 1, . . . , vn : Init [Next]vars Liveness

trong đó Liveness là sự kết hợp các thuộc tính công bằng của các hành động phụ của Next. Đây là một lớp công thức logic thời gian rất hạn chế. Logic thời gian khá biểu cảm và người ta có thể kết hợp các toán tử của nó theo đủ mọi cách để thể hiện nhiều thuộc tính khác nhau. Điều này gợi ý cách tiếp cận sau đây để viết một đặc tả: biểu thị từng thuộc tính mà hệ thống phải đáp ứng bằng một công thức thời gian, sau đó nối tất cả các công thức này. Ví dụ: công thức (8.41) ở trên biểu thị thuộc tính của bộ đệm ghi mà cuối cùng mọi yêu cầu đều nhận được phản hồi.

Cách tiếp cận này hấp dẫn về mặt triết học. Nó chỉ có một vấn đề: nó chỉ thực tế với những thông số kỹ thuật rất đơn giản nhất—và thậm chí đối với chúng, nó hiếm khi hoạt động tốt. Việc sử dụng logic thời gian một cách không kiểm soát sẽ tạo ra những công thức khó hiểu. Việc kết hợp một số công thức này sẽ tạo ra một đặc điểm kỹ thuật không thể hiểu được.

Dạng cơ bản của đặc tả TLA là (8.42). Hầu hết các thông số kỹ thuật nên có dạng này. Chúng ta cũng có thể sử dụng loại thông số kỹ thuật này làm nền tảng. Chương 9 và 10 mô tả các tình huống trong đó chúng ta viết đặc tả dưới dạng kết hợp của các công thức đó. Phần 10.7 giới thiệu một toán tử + tạm thời bổ sung và giải thích F và G lý do tại sao chúng ta có thể muốn viết một đặc tả F + G, toán tử trong đó có dạng (8.42). Nhưng những thông số kỹ thuật như vậy chỉ có tác dụng hạn chế trong thực tế. Hầu hết các kỹ sư chỉ cần biết cách viết các thông số kỹ thuật theo mẫu (8.42). Thật vậy, chúng có thể phù hợp khá tốt với các thông số kỹ thuật có dạng (8.38) chỉ thể hiện các đặc tính an toàn và không che giấu bất kỳ biến nào.