Machine Translated by Google

chư ơ ng 11

Ví dụ nâng cao

Sẽ rất tốt nếu cung cấp một loạt các ví dụ điển hình bao gồm hầu hết các vấn đề về đặc tả phát sinh trong thực tế. Tuy nhiên, không có thứ gọi là đặc điểm kỹ thuật điển hình. Mỗi thông số kỹ thuật thực tế dư ờng như đều đặt ra những vấn đề riêng của nó. Như ng chúng ta có thể phân chia tất cả các đặc tả thành hai lớp, tùy thuộc vào việc chúng có chứa khai báo biến hay không.

Một đặc tả không có biến sẽ xác định cấu trúc dữ liệu và các thao tác trên các cấu trúc đó. Ví dụ: mô-đun Trình tự xác định các hoạt động khác nhau trên trình tự. Khi chỉ định một hệ thống, bạn có thể cần một số loại cấu trúc dữ liệu khác với những loại đư ợc cung cấp bởi các mô-đun tiêu chuẩn như Chuỗi và Túi, đư ợc mô tả trong Chư ơ ng 18. Phần 11.1 đư a ra một số ví dụ về đặc tả cấu trúc dữ liệu.

Đặc tả hệ thống chứa các biến thể hiện trạng thái của hệ thống. Chúng ta có thể chia thêm các đặc tả hệ thống thành hai lớp—các đặc tả cấp cao mô tả ý nghĩa của việc hệ thống là chính xác và các đặc tả cấp thấp hơ n mô tả những gì hệ thống thực sự làm. Trong ví dụ về bộ nhớ của Chư ơ ng 5, đặc tả bộ nhớ tuyến tính hóa của Phần 5.3 là đặc tả mức độ chính xác cao, trong khi đặc tả bộ đệm ghi của Phần 5.6 mô tả cách hoạt động của một thuật toán cụ thể. Sự phân biệt này không chính xác; việc thông số kỹ thuật ở mức cao hay thấp là vấn đề tùy theo từng góc độ. Như ng nó có thể là một cách hữu ích để phân loại các thông số kỹ thuật của hệ thống.

Thông số kỹ thuật hệ thống cấp thấp hơ n có xu hư ớng tư ơ ng đối đơ n giản. Một khi mức độ trừu tư ợng đã đư ợc chọn, việc viết đặc tả thư ờng chỉ là vấn đề lấy đúng các chi tiết khi mô tả những gì hệ thống làm. Việc chỉ định độ chính xác ở mức cao có thể tinh tế hơ n nhiều. Phần 11.2 xem xét vấn đề đặc tả cấp cao—xác định chính thức bộ nhớ đa bộ xử lý.

Machine Translated by Google

11.1 Chỉ định cấu trúc dữ liệu

Hầu hết các cấu trúc dữ liệu cần thiết để viết thông số kỹ thuật đều đơn giản về mặt toán học và dễ xác định dưới dạng tập hợp, hàm và bản ghi.
Phần 11.1.2 mô tả đặc điểm kỹ thuật của một cấu trúc như vậy—một biểu đồ. Trong những trường hợp hiếm hoi, đặc tả sẽ yêu cầu các khái niệm toán học phức tạp.
Các ví dụ duy nhất tôi biết là các thông số kỹ thuật của hệ thống hybrid, được thảo luận trong Phần 9.5. Ở đó, chúng tôi sử dụng một mô-đun để mô tả nghiệm của các phương trình vi phân. Mô-đun đó được chỉ định trong Phần 11.1.3 bên dưới. Phần 11.1.4 xem xét vấn đề phức tạp trong việc xác định các toán tử để chỉ định ngữ pháp BNF. Mặc dù không phải là loại cấu trúc dữ liệu mà bạn có thể cần cho đặc tả hệ thống, nhưng việc chỉ định ngữ pháp BNF sẽ cung cấp một bài tập nhỏ hay về "mô phỏng toán học". Mô-đun được phát triển trong phần đó được sử dụng trong Chương 15 để xác định ngữ pháp của TLA+. Tuy nhiên, trước khi chỉ định cấu trúc dữ liệu, bạn nên biết cách tạo các định nghĩa cục bộ.

11.1.1 Định nghĩa cục bộ

Trong quá trình xác định một hệ thống, chúng tôi viết rất nhiều định nghĩa phụ trợ. Một đặc tả hệ thống có thể bao gồm một công thức Spec duy nhất, nhưng chúng tôi xác định hàng chục mã định danh khác theo đó chúng tôi xác định Spec. Những mã định danh khác này thường có tên khá phổ biến—ví dụ: mã định danh Tiếp theo được xác định trong nhiều thông số kỹ thuật. Các định nghĩa khác nhau của Next không xung đột với nhau bởi vì nếu một mô-đun định nghĩa Next được sử dụng như một phần của thông số kỹ thuật khác thì nó thường được khởi tạo bằng cách đổi tên. Ví dụ: module Channel được sử dụng trong module InnerFIFO ở trang 38 với câu lệnh

```
InChan  = Kênh phiên bản có . . .
```

Hành động Tiếp theo của mô-đun Kênh sau đó được khởi tạo dưới dạng InChan !Next, do đó định nghĩa của nó không xung đột với định nghĩa của Tiếp theo trong mô-đun InnerFIFO.

Một mô-đun xác định các hoạt động trên cấu trúc dữ liệu có thể sẽ được sử dụng trong câu lệnh mở rộng, không đổi tên. Mô-đun này có thể xác định một số toán tử phụ chỉ được sử dụng để xác định các toán tử mà chúng ta quan tâm. Ví dụ: chúng ta chỉ cần mô-đun DifferentialEquations để xác định toán tử duy nhất Tích hợp. Tuy nhiên, Tích hợp được định nghĩa theo các toán tử được xác định khác có tên như Nbhd và IsDeriv. Chúng tôi không muốn các định nghĩa này xung đột với các cách sử dụng khác của các mã định danh đó trong mô-đun mở rộng Phương trình vi phân. Vì vậy, chúng tôi muốn các định nghĩa của Nbhd và IsDeriv là cục bộ của mô-đun Phương trình vi phân.1

1Chúng ta có thể sử dụng cấu trúc let để đặt các định nghĩa phụ trợ này vào trong định nghĩa Tích hợp, nhưng thủ thuật đó sẽ không hiệu quả nếu mô-đun Phương trình vi sai xuất các toán tử khác ngoài Tích hợp được xác định theo Nbhd và IsDeriv.

Machine Translated by Google

TLA+ cung cấp một công cụ sửa đổi cục bộ để tạo các định nghĩa cục bộ cho một mô-đun. Nếu mô-đun M chứa định nghĩa

```
Foo cục bộ(x ) = . . .
```

thì Foo có thể được sử dụng bên trong mô-đun M giống như bất kỳ mã định danh được xác định thông thường nào. Tuy nhiên, mô-đun mở rộng hoặc khởi tạo M không có định nghĩa về Foo. Nghĩa là, câu lệnh mở rộng M trong mô-đun khác không định nghĩa Foo trong mô-đun đó. Tương tự, tuyên bố

```
N = trường hợp M
```

không định nghĩa N !Foo. Công cụ sửa đổi cục bộ cũng có thể được áp dụng cho việc khởi tạo. Tuyên bố

```
trình tự cá thể cục bộ
```

trong mô-đun M kết hợp vào M các định nghĩa từ mô-đun Trình tự.
Tuy nhiên, một mô-đun khác mở rộng hoặc khởi tạo M không có được những định nghĩa đó. Tương tự, một tuyên bố như

```
cục bộ P(x ) = thể hiện N
```

làm cho tất cả các định nghĩa được khởi tạo trở thành cục bộ của mô-đun hiện tại.

Công cụ sửa đổi cục bộ chỉ có thể được áp dụng cho các định nghĩa và câu lệnh thực thể. Nó không thể được áp dụng cho một khai báo hoặc cho một câu lệnh mở rộng, vì vậy bạn không thể viết một trong những điều sau đây:

```
hằng số cục bộ N

Trình tự mở rộng cục bộ
```

Đây không phải là tuyên bố pháp lý.

Nếu một mô-đun không có khai báo hằng hoặc biến và không có mô-đun con thì việc mở rộng và khởi tạo mô-đun đó là tương đương. Như vậy, hai phát biểu

```
mở rộng chuỗi chuỗi ví dụ
```

là tương đương.

Trong một mô-đun xác định các toán tử toán học tổng quát, tôi muốn đặt tất cả các định nghĩa cục bộ ngoại trừ những định nghĩa mà người dùng mô-đun mong đợi. Ví dụ: người dùng mong đợi mô-đun Trình tự xác định các toán tử trên trình tự, chẳng hạn như Nối thêm. Họ không mong đợi nó xác định các toán tử trên các số, chẳng hạn như +. Mô-đun Sequences sử dụng + và các toán tử khác được xác định trong mô-đun Naturals. Nhưng thay vì mở rộng Naturals, nó định nghĩa các toán tử đó bằng câu lệnh

```
phiên bản địa phương Naturals
```

Do đó, định nghĩa của các toán tử từ Naturals là cục bộ của Sequences.
Sau đó, một mô-đun mở rộng mô-đun Chuỗi có thể xác định + có nghĩa gì đó khác ngoài phép cộng các số.

Machine Translated by Google

11.1.2 Đồ thị

Biểu đồ là một ví dụ về loại cấu trúc dữ liệu đơn giản thường được sử dụng trong các thông số kỹ thuật. Bây giờ chúng ta hãy viết một mô-đun Đồ thị để sử dụng trong việc viết các thông số kỹ thuật của hệ thống.

Trước tiên, chúng ta phải quyết định cách biểu diễn biểu đồ theo các cấu trúc dữ liệu đã được xác định— cấu trúc dữ liệu TLA+ tích hợp sẵn như hàm hoặc cấu trúc dữ liệu được xác định trong các mô-đun hiện có. Quyết định của chúng tôi phụ thuộc vào loại biểu đồ mà chúng tôi muốn biểu diễn. Chúng ta quan tâm đến đồ thị có hướng hay đồ thị vô hướng? Đồ thị hữu hạn hay vô hạn? Đồ thị có hoặc không có vòng lặp tự (cạnh từ nút đến chính nó)? Nếu chúng ta đang chỉ định biểu đồ cho một thông số kỹ thuật cụ thể thì thông số kỹ thuật đó sẽ cho chúng ta biết cách trả lời những câu hỏi này. Trong trường hợp không có hướng dẫn như vậy, hãy xử lý các biểu đồ tùy ý. Cách ưa thích của tôi để biểu diễn cả đồ thị có hướng và vô hướng là chỉ định đồ thị có hướng tùy ý và xác định đồ thị vô hướng là đồ thị có hướng chứa một cạnh nếu nó chứa cạnh chỉ đối diện. Đồ thị có hướng có cách biểu diễn khá rõ ràng: đồ thị có hướng bao gồm một tập hợp các nút và một tập hợp các cạnh, trong đó một cạnh từ nút m đến nút n được biểu thị bằng cặp có thứ tự m, n .

Ngoài việc quyết định cách biểu diễn đồ thị, chúng ta phải quyết định cách cấu trúc mô-đun Graphs. Quyết định phụ thuộc vào cách chúng ta mong đợi mô-đun này sẽ được sử dụng như thế nào. Đối với thông số kỹ thuật sử dụng một biểu đồ, cách thuận tiện nhất là xác định các thao tác trên biểu đồ cụ thể đó. Vì vậy, chúng tôi muốn mô-đun Đồ thị có các tham số (không đổi) Nút và Cạnh đại diện cho tập hợp các nút và cạnh của một biểu đồ cụ thể. Một đặc tả có thể sử dụng một mô-đun như vậy với một câu lệnh

đồ thị mẫu có nút . . . , Cạnh . . .

ở đâu có ". . . "s là tập hợp các nút và cạnh của biểu đồ cụ thể xuất hiện trong đặc tả. Mặt khác, một đặc tả có thể sử dụng nhiều biểu đồ khác nhau. Ví dụ: nó có thể bao gồm một công thức xác nhận sự tồn tại của một đồ thị con, đáp ứng các thuộc tính nhất định của một số đồ thị G đã cho. Một đặc tả như vậy cần các toán tử lấy đồ thị làm đối số—ví dụ: toán tử Đồ thị con được xác định vì vậy Đồ thị con(G) là tập hợp tất cả các đồ thị con của đồ thị G. Trong trường hợp này, mô-đun Đồ thị sẽ không có tham số và các thông số kỹ thuật sẽ kết hợp nó với một câu lệnh mở rộng. Hãy viết loại mô-đun này.

Một toán tử như Subgraph lấy biểu đồ làm đối số, vì vậy chúng ta phải quyết định cách biểu diễn biểu đồ dưới dạng một giá trị. Đồ thị G bao gồm tập N nút và tập E cạnh. Một nhà toán học sẽ biểu diễn G theo thứ tự E . Tuy nhiên, G.node dễ thấy hơn G[1], vì cạnh E. , vậy chúng tôi biểu diễn cặp NG dưới dạng bản ghi với trường nút N và trường

Sau khi đưa ra những quyết định này, thật dễ dàng để xác định bất kỳ toán tử chuẩn nào trên biểu đồ. Chúng ta chỉ cần quyết định những gì chúng ta nên xác định. Dưới đây là một số toán tử thường hữu ích:

Machine Translated by Google

IsDirectedGraph(G)

Đúng nếu G là một đồ thị có hướng tùy ý—nghĩa là một bản ghi có trường nút N và trường cạnh E sao cho E là tập con của N × N . Toán tử này hữu ích vì một đặc tả có thể muốn khẳng định rằng thứ gì đó là đồ thị có hướng. (Để hiểu cách khẳng định G là bản ghi có các trường nút và cạnh, hãy xem định nghĩa về IsChannel trong Phần 10.3 trên trang 140.)

Đồ thị con được định hướng(G)

Tập hợp tất cả các đồ thị con của đồ thị có hướng G. Ngoài ra, chúng ta có thể định nghĩa IsDirectedSubgraph(H , G) là đúng nếu H là đồ thị con của G. Tuy nhiên, thật dễ dàng để biểu diễn IsDirectedSubgraph theo DirectedSubgraph:

IsDirectedSubgraph(H , G) ≡ H DirectedSubgraph(G)

Mặt khác, thật khó để diễn đạt DirectedSubgraph dưới dạng IsDirectedSubgraph:

DirectedSubgraph(G) = chọn
S : H : (H S) ≡ IsDirectedSubgraph(H , G)

Phần 6.1 giải thích tại sao chúng ta không thể xác định một tập hợp tất cả các đồ thị có hướng, vì vậy chúng ta phải xác định toán tử IsDirectedGraph.

IsUndirectedGraph(G)

Đồ thị con vô hướng(G)

Đây là tương tự như các toán tử cho đồ thị có hướng. Như đã đề cập ở trên, đồ thị vô hướng là đồ thị có hướng G sao cho với mọi cạnh m, n thuộc G.edge thì cạnh nghịch đảo n, m cũng thuộc G.edge. Lưu ý rằng DirectedSubgraph(G) chứa các đồ thị có hướng không phải là đồ thị vô hướng—ngoại trừ một số đồ thị “thoái hóa” G nhất định, chẳng hạn như đồ thị không có cạnh.

Đường dẫn(G)

Tập hợp tất cả các đường dẫn trong G, trong đó đường dẫn là chuỗi nút bất kỳ có thể thu được bằng cách đi theo các cạnh theo hướng mà chúng trỏ tới. Định nghĩa này rất hữu ích vì nhiều thuộc tính của đồ thị có thể được biểu diễn dưới dạng tập hợp các đường dẫn của nó. Thật thuận tiện khi coi chuỗi một phần tử n là một đường dẫn cho bất kỳ nút n nào.

AreConnectedIn(m, n, G)

Đúng nếu có một đường đi từ nút m đến nút n trong G. Tiện ích của toán tử này trở nên rõ ràng khi bạn thử xác định các thuộc tính đồ thị phổ biến khác nhau, như khả năng kết nối.

Có bất kỳ số thuộc tính và lớp đồ thị nào khác mà chúng ta có thể xác định. Hãy xác định hai điều này:

Machine Translated by Google

Được kết nối mạnh mẽ(G)

Đúng nếu G được kết nối mạnh, nghĩa là có một đường dẫn từ bất kỳ nút nào đến bất kỳ nút nào khác. Đối với đồ thị vô hướng, liên thông mạnh tương đương với định nghĩa thông thường về liên thông.

IsTreeWithRoot(G, r)

Đúng nếu G là cây có gốc r , trong đó chúng ta biểu diễn một cây dưới dạng biểu đồ có cạnh từ mỗi nút không phải gốc đến nút gốc của nó. Do đó, nút cha của nút không phải

gốc n bằng chọn m G.node : n, m G.edge

Mô-đun Đồ thị xuất hiện trên trang tiếp theo. Đến bây giờ, bạn đã có thể tự mình tìm ra ý nghĩa của tất cả các định nghĩa.

11.1.3 Giải phương trình vi phân

Phần 9.5 trên trang 132 mô tả cách chỉ định một hệ thống lai có trạng thái bao gồm một biến vật lý thỏa mãn phương trình vi phân thông thường. Thông số kỹ thuật sử dụng toán tử Tích hợp sao cho Tích hợp(D, t0, t1, x 0, . . . , x n 1) là giá trị tại thời điểm t1 của n-tuple

$$x, dx/dt, \ldots, d^{n-1}x/dt^{n-1}$$

trong đó x là nghiệm của phương trình vi phân

$$D[t, x, dx/dt, \ldots, d^n x/dt^n] = 0$$

có đạo hàm bậc 0 đến (n 1) tại thời điểm t0 là x 0, . . . , x n 1. Chúng tôi giả sử rằng có một giải pháp như vậy và nó là duy nhất. Xác định Tích hợp minh họa cách diễn đạt toán học phức tạp trong TLA+.

Chúng ta bắt đầu bằng việc định nghĩa một số ký hiệu toán học mà chúng ta sẽ sử dụng để định nghĩa đạo hàm. Như thường lệ, chúng ta thu được từ mô-đun Reals các định nghĩa về tập Real của các số thực và các toán tử số học thông thường. Giả sử PosReal là tập hợp tất cả các số thực dương:

$$PosReal = \{r \in Real : r > 0\}$$

và đặt OpenInterval(a, b) là khoảng mở từ a đến b (tập hợp các số lớn hơn a và nhỏ hơn b):

$$OpenInterval(a, b) = \{s \in Real : (a < s) \land (s < b)\}$$

(Các nhà toán học thường viết tập hợp này là (a, b).) Chúng ta cũng hãy định nghĩa Nbhd(r , e) là khoảng mở có chiều rộng 2e có tâm tại r :

$$Nbhd(r, e) = OpenInterval(r - e, r + e)$$

Machine Translated by Google

đồ thị mô-đun

Một mô-đun xác định các toán tử trên biểu đồ. Đồ thị có hướng được biểu diễn dưới dạng bản ghi có trường nút là tập hợp các nút và trường cạnh của nó là tập hợp các cạnh, trong đó một cạnh là một cặp nút có thứ tự.

phiên bản địa phương ng Naturals

trình tự cá thể cục bộ

IsDirectedGraph(G) = Đúng nếu G là đồ thị có hướng.

G = [nút G.node, cạnh G.edge]
G.edge (G.node × G.node)

DirectedSubgraph(G) = Tập hợp tất cả các đồ thị con (có hướng) của đồ thị có hướng.

{H [nút : tập con G.node, cạnh : tập con (G.node × G.node)] :
IsDirectedGraph(H) H .edge G.edge}

IsUndirectedGraph(G) = Đồ thị vô hướng là đồ thị có hướng trong đó mọi
IsDirectedGraph(G) cạnh có một hướng ngược lại.
e G.edge : e[2], e[1] G.edge

Đồ thị con vô hướng(G) = Tập hợp các đồ thị con (vô hướng) của đồ thị vô hướng.

{H DirectedSubgraph(G) : IsUndirectedGraph(H)}

Đường dẫn(G) = Tập hợp các đường dẫn trong G, trong đó một đường dẫn được biểu diễn dưới dạng một chuỗi các nút.

{p Seq(G.node) : p =
i 1 . . (Len(p) 1) : p[i], p[i + 1] G.edge}

AreConnectedIn(m, n, G) = Đúng nếu có đường đi từ m đến n trong đồ thị G.

p Path(G) : (p[1] = m) (p[Len(p)] = N)

IsStronglyConnected(G) = Đúng nếu đồ thị G liên thông mạnh. m,

n G.node : AreConnectedIn(m, n, G)

IsTreeWithRoot(G, r) = Đúng nếu G là cây có gốc r, trong đó các cạnh điểm
IsDirectedGraph(G) từ con tới cha mẹ.
e G.edge : e[1]
= r f G.edge : (e[1] = f [1]) (e
= f) n G.node : AreConnectedIn(n,r , G)

Hình 11.1: Mô-đun xác định toán tử trên đồ thị.

Machine Translated by Google

Để giải thích các định nghĩa, chúng ta cần một số ký hiệu cho đạo hàm của hàm số.
Khá khó để hiểu được ý nghĩa toán học của ký hiệu thông thường df /dt đối với đạo hàm của f . (T
chính xác là gì?) Vì vậy, hãy sử dụng đạo hàm đơn giản hơn về mặt toán học của hàm f dưới dạng ký
phải sử dụng ký hiệu TLA+ vì vi phân hữu hạn xuất hiện tự nhiên trong các định nghĩa của chúng ta.)
Hãy nhớ rằng f (0), đạo hàm cấp 0 của f , bằng f .

Bây giờ chúng ta có thể bắt đầu định nghĩa Tích hợp. Nếu a và b là số thì InitVals là n-bộ số và
D là hàm từ (n + 2)-bộ dữ liệu số thành số, khi đó Integrate(D, a, b, InitVals) = f (0) [b], . . . ,
f (n 1)[b]

trong đó f là hàm thỏa mãn hai điều kiện sau:

- D[r , f (0)[r], f (1)[r], . . . , f (n) [r]] = 0, với mọi r trong một khoảng mở nào đó chứa a và b.
- f (0)[a], . . . , f (n 1)[a] = Giá trị ban đầu

Chúng tôi muốn xác định Integrate(D, a, b, InitVals) theo hàm f này, mà chúng tôi có thể chỉ định
bằng cách sử dụng toán tử chọn. Cách dễ nhất là chọn không chỉ f , mà cả n đạo hàm đầu tiên của Như ng
nó. Vì vậy, chúng ta chọn hàm g sao cho g[i] = f (i) với i 0 . . N. Hàm g ánh xạ các số trong 0 . .
n vào các hàm. Chính xác hơn, g là một phần tử của

[0 . . n [OpenInterval(a e, b + e) Thực]]

đối với một số tích cực e. Hàm số trong tập hợp này thỏa mãn các điều kiện sau:

1. g[i] là i 2. đạo hàm của g[0], với mọi i 0 . . N.

D[r , g[0][r], . . . , g[n][r]] = 0, với mọi r trong OpenInterval(a e, b + e).

3. g[0][a], . . . , g[n 1][a] = Giá trị ban đầu

Bây giờ chúng ta phải diễn đạt những điều kiện này một cách hình thức.

Để biểu thị điều kiện đầu tiên, chúng ta sẽ định nghĩa IsDeriv sao cho IsDeriv(i, df , f) là
đúng nếu df là đạo hàm i của f . Chính xác hơn, điều này sẽ xảy ra nếu f là hàm có giá trị thực trên
một khoảng mở; chúng tôi không quan tâm IsDeriv(i, df , f) bằng bao nhiêu đối với các giá trị khác
của f . Khi đó điều kiện 1 là

tôi 1 . . n : IsDeriv(i, g[i], g[0])

Để thể hiện điều kiện thứ hai một cách chính thức, không có “. . . ”, chúng tôi lý luận như sau:

D[r , g[0][r], . . . , g[n][r]]
= D[r , g[0][r], . . . , g[n][r]] Xem trang 50.
= D[r g[0][r], . . . , g[n][r]] Các bộ dữ liệu là các
chuỗi = D[r [i 1 . . (n + 1) g[i 1][r]]] An (n +1)-tuple là hàm có miền 1 . . n +1.

Machine Translated by Google

Điều kiện thứ ba đơ n giản là

```
tôi   1 . . n : g[i   1][a] = InitVals[i]
```

Do đó chúng ta có thể viết công thức xác định g là

```
e   PosReal :   g   [0 . . n   [OpenInterval(a   e, b + e)   Thực] ]     i
                    1 . . n :   IsDeriv(i, g[i], g[0])   g[i
                                          1][a] = InitVals[i]     r
                    OpenInterval(a   e, b + e) : D[r   [i
                         1 . . (n + 1)   g[i   1][r ]] ] = 0
```

trong đó n là độ dài của InitVals. Giá trị của Integrate(D, a, b, InitVals) là bộ g[0][b], . . . , g[n 1][b], có thể viết chính thức là

```
[i   1 . . n   g[i   1][b]]
```

Để hoàn thành định nghĩa về Tích hợp, bây giờ chúng ta định nghĩa toán tử IsDeriv. Thật dễ dàng để định nghĩa đạo hàm i theo đạo hàm bậc nhất. Vì vậy, chúng tôi xác định IsFirstDeriv(df , f) là đúng nếu df là đạo hàm bậc nhất của f , giả sử rằng f là hàm có giá trị thực có miền xác định là khoảng mở. Định nghĩa của chúng ta thực sự đúng nếu tập xác định của f là một tập mở bất kỳ.2 Phép tính cơ bản cho chúng ta biết rằng df [r] là đạo hàm của f tại r iff

$$df\,[r\,] = \lim_{s \to r} \frac{f[s] \quad f[r]}{s \quad r}$$

Định nghĩa cổ điển “δ-” về giới hạn phát biểu rằng điều này đúng nếu, với mọi > 0, tồn tại một δ > 0 sao cho 0 < |s r | < δ ngụ ý

$$df[r] \quad \frac{f[s] \quad f[r]}{s \quad r} \quad <$$

Nói một cách chính thức, điều kiện này là

```
PosReal :
δ   PosReal :
   s   Nbhd(r , δ) \ {r} :   N bhd(df[r], f[s]  f[r] )
                                  s   r
```

Chúng tôi xác định IsFirstDeriv(df , f) là đúng nếu các miền của df và f bằng nhau và điều kiện này đúng với tất cả r trong miền của chúng.

Các định nghĩa về Tích hợp và tất cả các toán tử khác được giới thiệu ở trên xuất hiện trong mô-đun Phương trình vi phân của Hình 11.2 trên trang tiếp theo. Cấu trúc cục bộ được mô tả trong Phần 11.1.1 ở trên được sử dụng để biến tất cả các định nghĩa này thành cục bộ cho mô-đun, ngoại trừ định nghĩa Tích hợp.

2A tập S là mở nếu với mọi r S tồn tại một > 0 sao cho khoảng từ r đến r + nằm trong S.

Machine Translated by Google

mô-đun Phương trình vi phân

Mô-đun này xác định toán tử Tích hợp để xác định nghiệm của phương trình vi phân. Nếu a và b là số thực với a ≤ b; InitVals là một bộ n số thực; và D là hàm từ (n +1)-bộ dữ liệu thực đến số thực; thì đây là bộ giá trị n

$$f[b],\ \frac{df}{dt}[b],\ \ldots,\ \frac{d^{n-1}f}{dt^{n-1}}[b]$$

trong đó f là nghiệm của phương trình vi phân df d n f

$$D[t, f, \ldots] = \frac{d^n f}{dt\,dt\ldots dt}$$

như vậy mà

$$f[a],\ \frac{df}{dt}[a],\ \ldots,\ \frac{d^{n-1}f}{dt^{n-1}}[a] = InitVals\ dtn\ 1$$

phiên bản địa phương Reals

phiên bản cục bộ Chuỗi cục bộ

PosReal = {r ∈ Real : r > 0} local

OpenInterval(a, b) = {s ∈ Real : (a < s) ∧ (s < b)} local Nbhd(r ∈ Real, e ∈ PosReal) = OpenInterval(r − e, r + e) local IsFirstDeriv(df, f) = df ∈ [miền f → Real]

Câu lệnh phiên bản và các định nghĩa này là cục bộ, do đó, một mô-đun mở rộng mô-đun này chỉ nhận được định nghĩa Tích hợp.

Giả sử miền f là tập con mở của Tập thực, điều này đúng nếu f khả vi và df là đạo hàm bậc nhất của nó. Hãy nhớ lại rằng đạo hàm của f tại r là số df [r] thỏa mãn điều kiện sau: với mọi ε > 0 tồn tại a δ sao cho 0 < |s − r| < δ ⇒ |df [r] ∗ (s − r) − (f [s] − f [r])| < ε ∗ |s − r|.

∀r ∈ miền f : ∀ε ∈ PosReal : ∃δ ∈ PosReal : ∀s ∈ Nbhd(r, δ) \ {r} : |(f [s] − f [r])/(s − r) − df [r]| < ε

local IsDeriv(n, df, f) = Đúng iff f khả vi n lần và df là n của nó ⇒ ⇒ phát sinh.

đặt IsD[k ∈ 0 . . n, g ∈ [domain f → Real]] = if k = 0 then g = f else ∀gg ∈ [domain f → Real] : IsFirstDeriv(g, gg) ⇒ IsD[k − 1, gg]

IsD[k, g] = IsDeriv(k, g, f)

trong IsD[n, df]

Tích phân(D, a, b, InitVals) = để n == Len(InitVals) = ∃gg ∈ chọn g : e ∈ PosReal : g ∈ [0 . . n → [OpenInterval(a − e, b + e) → Thực]] ∀tôi ∈ 1 . . n : IsDeriv(i, g[i], g[0]) ∧ g[i − 1][a] = InitVals[i] ∧ r ∈ OpenInterval(a − e, b + e) : D[r ∘ [i ∈ 1 . . (n + 1) → g[i − 1][r]]] = 0

trong [i ∈ 1 . . n → gg[i − 1][b]]

Hình 11.2: Một mô-đun để xác định nghiệm của phương trình vi phân.

Machine Translated by Google

11.1.4 Ngữ pháp BNF

BNF, viết tắt của Backus-Naur Form, là một cách tiêu chuẩn để mô tả cú pháp của ngôn ngữ máy tính. Phần này phát triển mô-đun BNFGrammars, mô-đun này xác định các toán tử để viết ngữ pháp BNF. Ngữ pháp BNF không phải là loại cấu trúc dữ liệu phát sinh trong đặc tả hệ thống và TLA+ không đặc biệt phù hợp để chỉ định một ngữ pháp. Cú pháp của nó không cho phép chúng ta viết ngữ pháp BNF chính xác theo cách chúng ta muốn, như ng chúng ta có thể đến gần một cách hợp lý.
Hơ n nữa, tôi nghĩ thật thú vị khi sử dụng TLA+ để chỉ định cú pháp riêng của nó. Vì vậy, mô-đun BNFGrammars đư ợc sử dụng trong Chư ơ ng 15 để chỉ định một phần cú pháp của TLA+, cũng như trong Chư ơ ng 14 để chỉ định cú pháp của tệp cấu hình của trình kiểm tra mô hình TLC.

Hãy bắt đầu bằng cách xem lại ngữ pháp BNF. Hãy xem xét ngôn ngữ SE nhỏ của các biểu thức đơ n giản đư ợc mô tả bởi ngữ pháp BNF

```
expr ::= nhận dạng | kinh nghiệm và kinh nghiệm | ( điểm kinh nghiệm ) | hãy để
def trong expr def ::= ident == expr
```

trong đó op là một số lớp toán tử trung tố như + và ident là một số lớp định danh như abc và x . Ngôn ngữ SE chứa các biểu thức như

```
abc + (đặt x == y + abc trong x x )
```

Hãy biểu diễn biểu thức này dư ới dạng chuỗi

```
“abc”, “+”, “(”, “LET”, “x”, “==”, “y”,
      “+”, “abc”, “IN”, “x”, “ ”, “ x”, “)”
```

của dây. Các chuỗi như “abc” và “+” xuất hiện trong chuỗi này thư ờng đư ợc gọi là từ vị. Nói chung, một chuỗi các từ vị đư ợc gọi là một câu; và một tập hợp các câu đư ợc gọi là một ngôn ngữ. Vì vậy, chúng tôi muốn xác định ngôn ngữ SE bao gồm tập hợp tất cả các câu như vậy đư ợc mô tả bởi ngữ pháp BNF.3 Để biểu diễn ngữ pháp BNF trong TLA+, chúng tôi phải gán ý nghĩa toán học cho các ký hiệu không kết thúc như def , cho các ký hiệu đầu cuối như op , và hai sản phẩm của ngữ pháp. Phư ơ ng pháp mà tôi thấy đơ n giản nhất là để ý nghĩa của ký hiệu không kết thúc là ngôn ngữ mà nó tạo ra. Vì vậy, ý nghĩa của expr chính là ngôn ngữ SE. Tôi định nghĩa một ngữ pháp là một hàm G sao cho, đối với bất kỳ chuỗi “str” nào, giá trị của G[“str”] là ngôn ngữ đư ợc tạo bởi chuỗi không kết thúc str . Do đó, nếu G là ngữ pháp BNF ở trên thì G[“expr”] là ngôn ngữ SE hoàn chỉnh và G[“def”] là ngôn ngữ đư ợc xác định bởi quá trình tạo def , chứa các câu như

```
“y”, “==”, “qq”, “ ”, “wxyz”
```

Ngữ pháp 3BNF cũng đư ợc sử dụng để chỉ định cách phân tích cú pháp một biểu thức—ví dụ: a + b c đư ợc phân tích cú pháp thành a + (b c) thay vì (a + b) c. Bằng cách xem xét ngữ pháp để chỉ xác định một tập hợp các câu, chúng tôi cố tình không nắm bắt đư ợc cách sử dụng đó trong cách trình bày TLA+ của chúng tôi về ngữ pháp BNF.

Machine Translated by Google

Thay vì để miền của G chỉ gồm hai chuỗi “expr” và “def”, hóa ra sẽ thuận tiện hơn nếu đặt miền của nó là toàn bộ chuỗi các chuỗi và để G[s] là chuỗi trống. ngôn ngữ (tập trống) cho tất cả các chuỗi không phải là “expr” và “def”. Vì vậy, ngữ pháp là một hàm từ tập hợp tất cả các chuỗi đến tập hợp các chuỗi chuỗi. Do đó, chúng ta có thể định nghĩa tập ngữ pháp của tất cả các văn phạm bằng cách

```
Ngữ pháp   = [chuỗi    tập con Seq(chuỗi)]
```

Khi mô tả ý nghĩa toán học của các bản ghi, Phần 5.2 đã giải thích rằng r .ack là tên viết tắt của r [“ack”]. Đây là trường hợp ngay cả khi r không phải là bản ghi. Vì vậy, chúng ta có thể viết G.op thay vì G[“op”]. (Ngữ pháp không phải là một bản ghi vì miền của nó là tập hợp tất cả các chuỗi chứ không phải là một tập hợp hữu hạn các chuỗi.)

Một thiết bị đầu cuối như ident có thể xuất hiện ở bất kỳ đâu bên phải của “::=” mà một thiết bị đầu cuối như expr có thể xuất hiện, do đó, thiết bị đầu cuối cũng phải là một tập hợp các câu. Hãy biểu diễn một thiết bị đầu cuối dưới dạng một tập hợp các câu, mỗi câu là một chuỗi bao gồm một từ vị duy nhất. Giả sử mã thông báo là một câu bao gồm một từ vựng duy nhất, do đó, thiết bị đầu cuối là một tập hợp các mã thông báo. Ví dụ: mã nhận dạng thiết bị đầu cuối là một tập hợp chứa các mã thông báo như “abc”, “x” và “qq”. Bất kỳ thiết bị đầu cuối nào xuất hiện trong ngữ pháp BNF đều phải được biểu thị bằng một bộ mã thông báo, do đó == trong ngữ pháp cho SE là tập hợp {“==”}. Hãy xác định toán tử tok bằng cách

tok là viết tắt của mã thông báo.

```
tok(s)   = {s }
```

vì vậy chúng ta có thể viết bộ mã thông báo này dưới dạng tok(“==”).

Một sản phẩm thể hiện mối quan hệ giữa các giá trị của G.str đối với một số ngữ pháp G và một số chuỗi “str”. Ví dụ, việc sản xuất

```
def ::= ident == expr
```

khẳng định rằng một câu s nằm trong G.def nếu nó có dạng i e đối với “==” một số mã thông báo i trong ident và một số câu e trong G.expr . Trong toán học, một công thức về G phải đề cập đến G (có lẽ gián tiếp bằng cách sử dụng một ký hiệu được xác định theo G). Vì vậy, chúng ta có thể thử viết sản phẩm này bằng TLA+ dưới dạng

```
G.def ::= ident tok(“==”) G.expr
```

Trong biểu thức bên phải ::=, tính kề đang thể hiện một phép toán nào đó.
Giống như việc chúng ta phải thực hiện phép nhân rõ ràng bằng cách viết 2 x thay vì 2x, , chúng ta phải biểu diễn phép tính này bằng một toán tử tường minh. Hãy sử dụng &, để chúng ta có thể viết kết quả dưới dạng

```
(11.1) G.def ::= ident & tok(“==”) & G.expr
```

Điều này thể hiện mối quan hệ mong muốn giữa các tập G.def và G.expr của các câu nếu ::= được định nghĩa là đẳng thức và & được xác định sao cho L & M là

Machine Translated by Google

tập hợp tất cả các câu thu đư ợc bằng cách ghép một câu trong L với một câu trong M :

L & M = {s t : s L, t M }

Sản phẩm

expr ::= nhận dạng | kinh nghiệm và kinh nghiệm | (điểm kinh nghiệm) | hãy để def trong expr

tư ơ ng tự có thể đư ợc thể hiện như

Các quy tắc ư u tiên của TLA+ ngụ ý rằng a | b & c đư ợc hiểu là a | (b & c).

(11.2) G.expr ::= nhận dạng | G.expr & op & G.expr | tok(“(”) & G.exp & tok(“)”) | tok(“LET”) & G.def & tok(“IN”) & G.expr

Điều này thể hiện mối quan hệ mong muốn nếu | (có nghĩa là hoặc trong ngữ pháp BNF) đư ợc định nghĩa là tập hợp hợp ( ).

Chúng ta cũng có thể định nghĩa các toán tử sau đôi khi đư ợc sử dụng trong BNF ngữ pháp:

- Nil đư ợc định nghĩa sao cho Nil & S bằng S với bất kỳ tập hợp S nào của câu:

  Không = { }

- L + bằng L | L & L | L & L & L | . . . + = :

  đặt LL[n Nat] = L | nếu n = 0 thì L ngư ợc lại LL[n 1] | LL[n 1] & L

  trong hợp {LL[n] : n Nat}

  LL[n] = L . . . | L & . . . L. (n+1 bản)

  L + đư ợc gõ L^+ và tôi đư ợc gõ L^*.

- L bằng Nil | L | L & L | L & L & L | = . . . :

  L Không | L +

Ngữ pháp BNF cho SE bao gồm hai sản phẩm, đư ợc biểu thị bằng công thức TLA+ (11.1) và (11.2). Toàn bộ ngữ pháp là công thức duy nhất là sự kết hợp của hai công thức này. Chúng ta phải biến công thức này thành một định nghĩa toán học về ngữ pháp GSE, một hàm từ chuỗi đến ngôn ngữ.

Công thức là một khẳng định về một văn phạm G. Chúng ta định nghĩa GSE là văn phạm nhỏ nhất G thỏa mãn sự kết hợp của (11.1) và (11.2), trong đó ngữ pháp G1 nhỏ hơ n G2 có nghĩa là G1[s] G2[s] với mọi chuỗi s. Để biểu thị điều này trong TLA+, chúng ta định nghĩa một toán tử LeastGrammar sao cho LeastGrammar (P) là ngữ pháp nhỏ nhất G thỏa mãn P(G): LeastGrammar (P( )) =

chọn G Ngữ pháp:

P(G)

H Ngữ pháp : P(H) (s chuỗi : G[s] H [s])

Machine Translated by Google

Giả sử P(G) là sự kết hợp của (11.1) và (11.2), chúng ta có thể định nghĩa ngữ pháp GSE là Ngữ pháp nhỏ nhất (P). Sau đó chúng ta có thể định nghĩa ngôn ngữ SE bằng GSE.expr . Ngữ pháp nhỏ nhất G thỏa mãn công thức P phải có G[s] bằng ngôn ngữ trống cho bất kỳ chuỗi s nào không xuất hiện trong P. Do đó, GSE[s] bằng ngôn ngữ trống {} cho bất kỳ chuỗi s nào ngoài " expr" và "def".

Để hoàn thành thông số kỹ thuật của GSE, chúng tôi phải xác định bộ nhận dạng và mã thông báo. Chúng ta có thể định nghĩa tập hợp các toán tử bằng cách liệt kê chúng—ví dụ:

```
op    = tok("+") | tok("-") | tok(" * ") | tok("/")
```

Để diễn đạt điều này một cách ngắn gọn hơn một chút, hãy định nghĩa Tok(S) là tập hợp tất cả các mã thông báo được hình thành từ các phần tử trong tập S của từ vị:

```
Tok(S)   = {s : s    S}
```

Sau đó chúng ta có thể viết

```
op    = Tok( {"+", "  ", " * ", "/"} )
```

Hãy định nghĩa ident là tập hợp các mã thông báo có từ vựng là các từ được tạo hoàn toàn bằng các chữ cái viết thường, chẳng hạn như "abc", "qq" và "x". Để tìm hiểu cách thực hiện điều đó, trước tiên chúng ta phải hiểu chuỗi trong TLA+ thực sự là gì. Trong TLA+, chuỗi là một chuỗi ký tự. (Chúng tôi không quan tâm và ngữ nghĩa của Phần không chỉ định ký tự là gì.) Do đó, chúng tôi có thể áp dụng trình tự thông thường 16.1.10 307 cho nhiều toán tử hơn trên chúng. Ví dụ: Tail("abc") bằng "bc" và "abc" "de" "abcde". thành viên mà chúng tôi Các toán tử như & mà chúng tôi vừa xác định để biểu thị BNF đã được

TLA+ Xem trên trang về chuỗi. Re- bằng áp dụng lấy chuỗi và bộ dữ liệu để đồng bộ với các tập hợp câu, trong đó một câu là một chuỗi các từ vị. Những nhà khai thác nặc danh. cũng có thể được áp dụng cho các tập hợp thuộc bất kỳ loại trình tự nào —bao gồm cả các tập hợp chuỗi. Ví dụ: {"one", "two"} & {"s"} bằng {"ones", "twos"} và + là tập hợp bao gồm tất cả các chuỗi "ab", "abab", "abababab" , v.v. Vì vậy, là tập {"ab"} chúng ta có thể định nghĩa ident bằng Tok(Letter +), trong đó Letter hợp tất cả các từ vựng bao gồm một chữ cái viết thường:

```
Chữ   = {"a", "b", . . . , "z"}
```

Viết đầy đủ định nghĩa này (không có ".") thật tẻ nhạt. Chúng ta có thể làm điều này dễ dàng hơn một chút như sau. Đầu tiên chúng ta định nghĩa OneOf(s) là tập hợp tất cả các chuỗi một ký tự được tạo từ các ký tự của chuỗi s:

```
OneOf(s)   = {s[i] : i    miền s}
```

Sau đó chúng ta có thể định nghĩa

```
Chữ cái   = OneOf ("abcdefghijklmnopqrstuvwxyz")
```

Machine Translated by Google


```
GSE   =     thôi nào    = Tok( {“+”, “  ”, “  ”, “/”} )
                  nhận dạng   = Tok( OneOf (“abcdefghijklmnopqrstuvwxyz”)+ )
                  P(G)   =    G.expr ::=            nhận
                                                dạng | G.expr & op & G.expr
                                                | tok(“(”) & G.expr & tok(“)”) |
                                                tok(“LET”) & G.def & tok(“IN”) & G.expr
                                 G.def ::= ident & tok(“==”) & G.expr trong
            LeastGrammar (P)
```

Hình 11.3: Định nghĩa ngữ pháp GSE cho ngôn ngữ SE.

Định nghĩa đầy đủ về ngữ pháp GSE xuất hiện trong Hình 11.3 trên trang này.

Tất cả các toán tử chúng ta đã xác định ở đây để xác định ngữ pháp đều đư ợc nhóm lại vào mô-đun BNFGrammars, xuất hiện trong Hình 11.4 ở trang tiếp theo.

Sử dụng TLA+ để viết ngữ pháp BNF thông thư ờng hơ i ngớ ngẩn. Tuy nhiên, ngữ pháp BNF bình thư ờng không thuận tiện lắm cho việc mô tả cú pháp của một ngôn ngữ phức tạp như TLA+. Trên thực tế, họ không thể mô tả các quy tắc căn chỉnh cho danh sách các liên từ và các liên từ có dấu đầu dòng. Sử dụng TLA+ để chỉ định một ngôn ngữ như vậy không quá ngớ ngẩn. Trên thực tế, đặc tả TLA+ về cú pháp hoàn chỉnh của TLA+ đư ợc viết như một phần trong quá trình phát triển Bộ phân tích cú pháp, đư ợc mô tả trong Chư ơ ng 12. Mặc dù có giá trị khi viết trình phân tích cú pháp TLA+ , như ng đặc tả này không hữu ích lắm đối với ngư ời dùng thông thư ờng. của TLA+ nên nó không xuất hiện trong cuốn sách này.

11.2 Thông số kỹ thuật bộ nhớ khác

Phần 5.3 chỉ định bộ nhớ đa bộ xử lý. Thông số kỹ thuật đơ n giản đến mức phi thực tế vì ba lý do: bộ xử lý chỉ có thể có một yêu cầu chư a xử lý tại một thời điểm, điều kiện về tính đúng đắn cơ bản quá hạn chế và chỉ cung cấp các thao tác đọc và ghi đơ n giản. (Bộ nhớ thực cung cấp nhiều thao tác khác, chẳng hạn như ghi một phần từ và tìm nạp trư ớc bộ nhớ đệm.) Hiện tại, chúng tôi chỉ định một bộ nhớ cho phép nhiều yêu cầu chư a xử lý và có điều kiện thực tế, độ chính xác yếu hơ n. Để giữ cho đặc tả ngắn gọn, chúng ta vẫn chỉ xem xét các thao tác đơ n giản là đọc và ghi một từ trong bộ nhớ.

11.2.1 Giao diện

Điều đầu tiên chúng ta phải làm để xác định bộ nhớ là xác định giao diện. Giao diện chúng tôi chọn phụ thuộc vào mục đích của thông số kỹ thuật. Có nhiều lý do khác nhau khiến chúng ta phải chỉ định bộ nhớ đa bộ xử lý. Chúng ta có thể