Tạp chí Tin học và Điều khiển học, T.21, S.3 (2005), 261-270

MÔ HÌNH ĐẠI SỐ QUAN HỆ CỦA HỆ THỐNG HƯỚNG ĐỐI TƯỢNG

NGUYỄN MẠNH ĐỨC[1], NGUYỄN VĂN VỴ[2], ĐẶNG VĂN ĐỨC[3]

[1] *Khoa Toán, Trường Đại học Sư phạm - Đại học Thái Nguyên*
[2] *Trường Đại học Công nghệ- Đại học Quốc gia Hà Nội*
[3] *Viện Công nghệ thông tin*

Abstract. This article presents semantics of object oriented system with classes, visibility, dynamic binding and recursive methods. The class declarations and commands are served as designs in terms of pre- and post conditions; the relations of components in system... This approach shows clearer relations of components in object oriented system, possibility of using tools and cheecking method to improve the refinement for further development of systems. The algebraical laws are expanded for design specification of object oriented programs.

Tóm tắt. Bài báo trình bày ngữ nghĩa của hệ thống hướng đối tượng với các lớp, tính trực quan, liên kết động và các phương thức đệ quy. Các khai báo lớp và các lệnh như là các thiết kế dựa trên tiền điều kiện, hậu điều kiện và các mối quan hệ của các thành phần trong hệ thống... Cách tiếp cận này sẽ cho thấy rõ ràng mối quan hệ của các thành phần trong hệ thống hướng đối tượng, và các khả năng sử dụng các công cụ và cách thức kiểm tra để cải tiến đặc tả hệ thống được phát triển sau này. Nhờ đó các luật đại số đã được phát triển áp dụng cho việc đặc tả chương trình hướng đối tượng.

1. GIỚI THIỆU

Thiết kế và phát triển hệ thống phần mềm với ngôn ngữ hướng đối tượng là rất phức tạp [1, 5]. Nhiều nhà nghiên cứu chỉ ra sự cần thiết phát triển công cụ hình thức hóa làm nền tảng cho việc phát triển phần mềm hướng đối tượng. Bài báo này sẽ tìm hiểu cách thức lập trình dựa trên lý thuyết của Hoare và He [2], dùng vào việc xây dựng một cách đúng đắn các chương trình hướng đối tượng. Lý thuyết lập trình được vận dụng để trình bày ngữ nghĩa của ngôn ngữ lập trình hướng đối tượng với các lớp, tính rõ ràng, liên kết động, các phương thức đệ quy và tính đệ quy. Để cho đơn giản, những gì liên quan đến các định nghĩa biến, tham chiếu tới các kiểu được bỏ qua ([2, 6]).

Ở đây, việc khai báo lớp và các lệnh xem như là các thiết kế. Với cách tiếp cận này, các lệnh biểu thị quan hệ nhị phân thông qua trạng thái của các biến, các đối tượng và các lớp. Các biến của các kiểu dữ liệu nguyên thủy vẫn nhận các giá trị tương ứng với kiểu của chúng. Trong khi đó, trạng thái các đối tượng được biểu diễn bằng bộ các giá trị của thuộc tính với tính đệ quy. Phương thức được coi như là thiết kế với tập các tham số, bộ lớp. Trạng thái bao gồm thông tin môi trường cần thiết để hỗ trợ cho việc kiểm tra tính xác định đúng(well definedness) của đặc tả. Trong trường hợp chung, chương trình được mô hình hóa bằng thiết kế có dạng WF $\Rightarrow$ D. Ở đây, WF thiết lập tính xác định đúng của chương trình

và thiết kế D đóng vai trò như đặc tả hành vi của nó. Hành vi này kết hợp kiểu động và cơ cấu kiểm tra với ngữ nghĩa rõ ràng hướng đặc tả truyền thống. Thông qua các thành phần để chỉ rõ tính chất của lớp và ý nghĩa các phương thức của chúng. Cũng như các lệnh, khai báo các lớp biểu thị quan hệ nhị phân thông qua các trạng thái. Mỗi khai báo cung cấp các thông tin ngữ cảnh về cấu trúc của lớp và quan hệ của chúng với những lớp khác.

Trong bài báo này, chúng tôi đưa ra một cách xây dựng đặc tả lôgic các chương trình hướng đối tượng như là sự mở rộng của các chuẩn lôgic trong [2]. Trước hết xét mô hình tính toán của hệ thống.

2. MÔ HÌNH TÍNH TOÁN

Một chương trình lệnh được xác định bằng cặp (α, P), trong đó:

• α biểu thị tập các biến đã biết trong chương trình.

• P là tập toán tử xác định quan hệ giữa các giá trị khởi tạo của các biến trong chương trình và các giá trị kết thúc của nó và có dạng $p(x) \vdash R(x, x')$, cụ thể hơn như sau:

$$p(x) \vdash R(x, x) \stackrel{\text{df}}{=\!=} ok \wedge p(x) \Rightarrow ok' \wedge R(x, x'),$$

trong đó, $p(x)$ được gọi là tiền điều kiện và phải có giá trị true trước khi chương trình bắt đầu. $R(x, x')$ gọi là hậu điều kiện nhận được sau khi chương trình kết thúc. x và x' biểu diễn giá trị khởi đầu và kết thúc của biến x trong chương trình. ok và ok' là các biến lôgic mô tả trạng thái hành vi ban đầu và cuối của chương trình: nếu chương trình được kích hoạt hợp thức ok là true, nếu việc thực hiện chương trình cuối cùng thành công ok' là true. Ngược lại chúng là false. Ký hiệu $\stackrel{\text{df}}{=\!=}$ được hiểu là "được định nghĩa".

Để hình thức hóa hành vi của chương trình hướng đối tượng, mô hình tính toán trong bài này có các đặc trưng như sau đây:

1. Một chương trình hướng đối tượng hoạt động không chỉ với các biến (chung và địa phương) mà còn cả các đối tượng. Để đảm bảo truy cập các biến, mô hình cần thêm tập các biến thuộc tính bổ sung vào tập các biến chung và riêng đã biết trong chương trình.

2. Do cấu trúc của lớp con, một đối tượng có thể nằm trong lớp con bất kỳ của một tổ chức được khai báo. Như ở kết quả, hành vi của phương thức đó sẽ phụ thuộc vào kiểu hiện tại của nó. Để hỗ trợ cơ cấu liên kết động của phương thức gọi, mô hình sẽ lưu giữ vết của kiểu động cho mỗi đối tượng. Khả năng này là phù hợp với các biểu thức và các lệnh trong phạm vi mà mỗi biến ứng với một bản ghi hợp thức.

3. Cũng như trong các ngôn ngữ lệnh, một trạng thái của biến là giá trị được tạo ra tại một thời điểm. Một đối tượng có thể chứa nhiều thuộc tính của nó, và giá trị các thuộc tính thường được biểu diễn bởi bộ hữu hạn là các mẩu tin của kiểu hiện tại của đối tượng và những giá trị của các thuộc tính đó (có tính đệ quy).

Vì chương trình hướng đối tượng được biểu diễn bởi quan hệ hai ngôi $(\alpha, P), \alpha$ là tập các biến mô tả môi trường trong đó chương trình được thi hành, α có thể được chia 3 phần như sau:

1. Phần thứ nhất cung cấp các thông tin ngữ cảnh lớp và các quan hệ của chúng:

• cname: Tập các lớp đã được khai báo.

• superclass: Hàm bộ phận mà ánh xạ từ lớp tới lớp cha của nó, tức là superclass$(M) = N$, với N là lớp cha của lớp M. N được gọi là lớp cha của M, nếu tồn tại họ hữu hạn

$\{N_i | 0 \leqslant i \leqslant n\}$ sao cho lớp $M = N_0$ và $N = N_n$ và superclass$(N_i) = N_{i+1}$ với mọi $0 \leqslant i < n$.

2. Phần thứ 2 mô tả chi tiết cấu trúc của mỗi lớp: với mỗi lớp $N \in$ cname, nó bao gồm:

• attribute(N): tập các thuộc tính (được khai báo hoặc kế thừa) của lớp N

$$\{\langle a_1 : U_1, c_1\rangle, ..., \langle a_m : U_m, c_m\rangle\}$$

trong đó U_i và c_i đóng vai trò là kiểu và giá trị khởi đầu của thuộc tính ai trong lớp N, và sẽ tham chiếu bởi các hàm type$(N.a_i)$ và initial$(N.a_i)$ (sẽ được thảo luận sau).

• meth(N): Tập các phương thức được khai báo hoặc kế thừa bởi N :

$$\{m_1 \to (\langle x_1 : T_{1,1}, y_1 : T_{1,2}, z_1 : T_{1,3}\rangle, D_1),$$
$$\cdots$$
$$m_k \to (\langle x_k : T_{k,1}, y_k : T_{k,2}, z_k : T_{k,3}\rangle, D_k)\}$$

trong đó, phương thức m_i có x_i, y_i và z_i tương ứng là giá trị, kết quả và giá trị tham số kết quả. Hành vi của phương thức m_i được biểu diễn bởi thiết kế D_i.

3. Phần thứ 3 nhận biết các biến mà chúng sử dụng được bởi chương trình:

• alphabet: Tập các biến chung đã biết trong chương trình: $\{x_1 : T_1, ..., x_n : T_n\}$. Ở đây T_i biểu diễn kiểu của biến x_i, có thể là một trong số kiểu dữ liệu đơn giản (Boolean: B, số nguyên: int, số thực: R, xâu: string), hoặc tên lớp và được tham chiếu bởi hàm type(x_i).

• locvar: Tập các biến địa phương trong phạm vi $\{v_1 : T_1, ..., v_m : T_m\}$.

• visibleattr: Tập các thuộc tính tường minh trong lớp hiện tại, tức là tất cả thuộc tính đã được khai báo thuộc miền private, protected và public (đã khai báo hoặc kế thừa) của lớp.

Để thuận lợi, chúng ta thừa nhận có 4 tập tên khác nhau: tên biến, tên lớp, tên thuộc tính và tên phương thức. Trạng thái mô hình liên kết các biến trong các tập alphabet và locvar với các giá trị hiện thời của chúng. Nếu biến là đối tượng thì giá trị của nó sẽ là một bộ giá trị của các thuộc tính với kiểu hiện tại: $\{myclass \to M\} \cup \{a \to value \mid a \in attribue(M)\}$.

3. BIỂU THỨC

Một biểu thức có thể có một trong các dạng như sau [2, 3]:

$$e ::= x \mid \text{null} \mid \text{new } N \mid e \text{ is } N \mid (N)e \mid e.a \mid (e; a : f) \mid f(e) \mid \text{self}.$$

Để xác định tính hợp lệ của biểu thức e, ta đưa ra xác nhận $D(e)$, nó là đúng trong trường hợp mà biểu thức e có thể được định giá thành công.

Một biến x được gọi là xác định đúng nếu nó đã biết trong chương trình.

$$D(x) \underline{\underline{\text{df}}}\ x \in (\text{alphabet} \cup \text{locvar}).$$

Ký hiệu null biểu diễn một đối tượng xác định đúng.

$$D(\text{null}) \underline{\underline{\text{df}}}\ \text{true}, \text{type}(\text{null}) \underline{\underline{\text{df}}}\ \text{NULL},$$

ở đây NULL là tên lớp đặc biệt. Ta quy ước rằng NULL $\leqslant N$ với mọi $N \in$ cname.

Biểu thức new$N$ là xác định đúng nếu lớp N đã được khai báo.

$$D(\text{new}N) \underline{\underline{\text{df}}}\ N \in \text{cname}.$$

Nó chỉ ra một đối tượng mới của lớp N đã được tạo ra.

$$\text{type}(\text{new}N) \underline{\underline{\text{df}}}\ N,$$

$$\text{new}N \underline{\underline{\text{df}}}\ \{\text{myclass} \to N\} \cup U\{a \in \text{ attribute } (N)) \mid a \to \text{initial}(N.a)\}.$$

Kiểu thử nghiệm (test) e is N là biểu thức (lôgic) xác định đúng khi N đã được khai báo và e là đối tượng xác định đúng

$$D(e \text{ is } N) \underline{\underline{\text{df}}} (N \in \text{cname}) \wedge D(e) \wedge (\text{type}(e) \in \text{cname}),$$

$$\text{type}(e \text{ is } N) \underline{\underline{\text{df}}} B$$

Giá trị của biểu thức được xác định rõ bằng giá trị của e là đối tượng của lớp N hay không.

$$(e \text{ is } N) \underline{\underline{\text{df}}} (e \neq \text{null}) \wedge (e(\text{myclass}) \leqslant N).$$

Kiểu tính gộp (cast) $(N)e$ bằng e nếu nó là đối tượng khác null trong lớp con của N

$$D((N)e) \underline{\underline{\text{df}}} D(e \text{ is } N) \wedge (e \neq \text{null}) \wedge (e(\text{myclass}) \leqslant N), (N)e \underline{\underline{\text{df}}} e.$$

Kiểu của $(N)e$ là lớp N : $\text{type}((N)e) \underline{\underline{\text{df}}} N$.

Thuộc tính lựa chọn $e.a$ là xác định đúng khi e là đối tượng khác null, và là thuộc tính hiển nhiên từ lớp hiện tại

$$D(e.a) \underline{\underline{\text{df}}} D(e) \wedge (\text{type}(e) \in \text{cname}) \wedge (e \neq \text{null}) \wedge (e(\text{myclass}).a) \in \text{visibleattr},$$

$$e.a \underline{\underline{\text{df}}} e(a), \text{type}(e.a) \underline{\underline{\text{df}}} \text{type}(e(\text{myclass}).a).$$

Thuộc tính cập nhật $(e; a : f)$ là xác định đúng với điều kiện là $e.a$ là hiển hiện và kiểu của f là kiểu con của $e.a$

$$D(e; a : f) \underline{\underline{\text{df}}} D(e.a) \wedge D(f) \wedge (\text{type}(f) \leqslant \text{type}(e(\text{myclass}).a)).$$

Kiểu của $(e; a : f)$ là lớp của e, và giá trị của nó có thể thu nhận được từ giá trị của e bằng cách thay đổi giá trị của thuộc tính a cho giá trị của f

$$\text{type}(e; a : f) \underline{\underline{\text{df}}} \text{type}(e), \ (e; a : f) \underline{\underline{\text{df}}} e \oplus \{a \to f\}.$$

Ví dụ sau xác định đúng của biểu thức cài đặt:

$$D(e \text{ and } f) \underline{\underline{\text{df}}} D(e) \wedge (\text{type}(e) = B) \wedge D(f) \wedge (\text{type}(f) = B).$$

Biến self có thể chỉ được tham khảo trong định nghĩa của các phương thức, và được coi như là biến địa phương trong mô hình của chúng tôi: $D(\text{self}) \underline{\underline{\text{df}}} \text{self} \in \text{locvar}$.

4. CÁC LỆNH

Phần này xem xét các lệnh hỗ trợ việc xây dựng chương trình hướng đối tượng tiêu biểu [3].

$$c ::= |\text{skip} \mid \text{chaos} \mid c \lhd b \rhd c \mid c \Pi c \mid c; c \mid b * c \mid \text{var} x : T \mid \text{end} x \mid le := e \mid o.m(e).$$

Ở đây b là biểu thức lôgic, c là lệnh, e là một biểu thức, le là có thể xuất hiện ở vế trái của phép gán và có dạng $le ::= x|le.a$ với x là biến đơn còn a là thuộc tính của đối tượng.

skip (bỏ qua): skip không làm gì, và cuối cùng là thành công: skip $\underline{\underline{\text{df}}} \emptyset : (\text{true} \vdash \text{true})$.

chaos (không xác định): chaos hành vi của nó không xác định trước được:

$$\text{chaos} \underline{\underline{\text{df}}} \emptyset : (\text{false} \vdash \text{true}).$$

conditional (điều kiện): Cho P và Q là các thiết kế. $P \lhd b \rhd Q$ chương trình thực hiện P nếu giá trị b là true, ngược lại thực hiện Q.

$$P \lhd b \rhd Q \underline{\underline{\text{df}}} (D(b) \wedge \text{type}(b) = B) \to (P \wedge b \vee Q \wedge \neg b).$$

Cho $\{Pi \mid 1 \leqslant i \leqslant n\}$ là một họ của thiết kế. Sự luân phiên if $\{(b_i \to P_i) \mid 1 \leqslant i \leqslant n\} fi$, với việc chọn P_i để thực hiện nếu b_i là true. Khi mọi b_i là false thì không xác định (chaos).

non-determinism (không tiền định): Cho P và Q là các thiết kế. Ký hiệu $P \Pi Q$ biểu diễn cho chương trình được thực hiện P hoặc Q, nhưng sẽ không được chọn trước: $P \Pi Q \underline{\underline{\text{df}}} PVQ$.

composition (cấu thành): Cho P và Q là các thiết kế, thành phần cấu trúc của chúng: $P; Q$, mô tả chương trình có thể được thực hiện P trước, khi P kết thúc thì Q sẽ bắt đầu, trạng thái kết thúc của P là phù hợp cho trạng thái bắt đầu của Q.

$$P(s, s'); Q(s, s') \underline{\underline{\text{df}}} \exists m \bullet P(s, m) \wedge Q(m, s').$$

iteration (lặp): Cho P là một thiết kế, b là một điều kiện, ký hiệu $b * P$: nghĩa là P được thực hiện chừng nào b còn là true trước mỗi lần lặp: $b * P \underline{\underline{\text{df}}} \mu X \bullet (P; X) \lhd b \rhd \text{skip}$.

assignment (gán): Lệnh gán có dạng $le := e$, ở đây le là một trong các dạng sau đây: biến x của chương trình, thuộc tính $le.a$ của đối tượng le, đối tượng với kiểu khuôn mẫu $(N)le$.

Khai báo biến: Một cách hình thức, việc khai báo và kết thúc khai báo biến v được xác định bởi:

var $v : T \underline{\underline{\text{df}}} v \notin (\text{alphabet} \cup \text{locvar}) \Rightarrow \text{locvar} : (\text{true} \vdash \text{locvar}' = \text{locvar} \cup \{v : T\}$,

end $v \underline{\underline{\text{df}}} v \in \text{locvar} \Rightarrow \text{locvar} : (\text{true} \vdash \text{locvar}' = \{v\} \trianglelefteq \text{locvar})$,

ở đây $\{v\} \trianglelefteq \text{locvar}$ biểu diễn tập locvar sau khi đã loại bỏ biến v. Điều đó giải thích rằng: những ràng buộc trong tập locvar không cho phép định nghĩa lại biến trong phạm vi của nó.

Method call (phương thức gọi): Cho v, r và vr là danh sách các biểu thức. Chương trình $O.m(v, r, vr)$ gán các tham biến hiện tại v và vr các giá trị chính thức và giá trị tham biến kết quả trong phương thức m của đối tượng O và sau đó thực hiện các lệnh trong phần thân của phương thức m. Sau khi nó kết thúc, các giá trị kết quả và giá trị tham số kết quả của m được trả lại cho các tham biến hiện tại là v và vr.

$O.m(v, r, vr) \underline{\underline{\text{df}}} (D(O) \wedge \text{type}(O) \in \text{cname} \wedge m \in \text{meth}(O(\text{myclass}))) \Rightarrow$

$$\text{if}\{(O(\text{myclass}) = N \to \begin{pmatrix} \text{var self} : N, x, T1, y : T2, z : T3; \\ \text{self}, x, z := O, v, r, vr; \\ N.m \\ O, v, r, vr := \text{self}, y, z; \\ \text{end self}, x, y, z; \end{pmatrix}$$

$$N \leqslant \text{type}(O) \wedge m \in \text{meth}(N)\} fi.$$

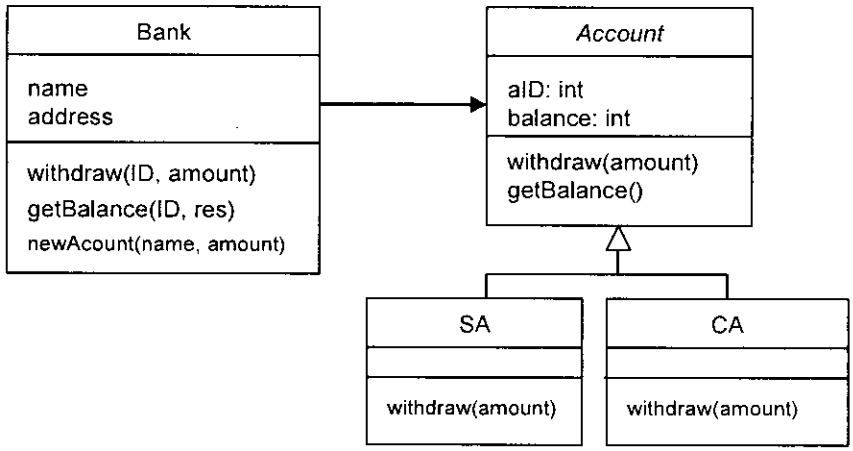
Ở đây:

• x, y và z là giá trị, kết quả và giá trị tham biến kết quả trong phương thức m của lớp $O(\text{myclass}); T1, T2$ và $T3$ là các kiểu của chúng; $N.m$ đóng vai trò thiết kế liên kết với phương thức m của lớp N.

• self giá trị tham số kết quả tường minh được sử dụng quy cho đối tượng có phương thức hoạt động hiện tại ràng buộc với đối tượng O.

5. KHAI BÁO LỚP

5.1. Khai báo lớp



Hình 1. Hệ thống Bank

Trong phần này sẽ xem xét chi tiết việc khai báo lớp, tính xác định đúng của chúng [2]. Một chương trình hướng đối tượng có thể được chỉ ra bởi dạng cdecls•P. Nó bắt đầu bằng phần khai báo một số lớp tiếp sau là khối lệnh P biểu diễn phương thức main của chương trình. Việc khai báo lớp cdecls có thể theo ngôn ngữ tựa Java như sau:

[private] class N [extends M]
  pri : $t_1 : T_1, ..., t_i : T_i$;
  pro: $u_1 : U_1, ..., u_j : U_j$;
  pub: $v_1 : V_1, ..., v_k : V_k$;
  meth: $m_1(\underline{x}_{11} : \underline{T}_{11}, \underline{y}_{12} : \underline{T}_{12}, \underline{z}_{13} : \underline{T}_{13})\{P_1\}$;
    ...
    $m_\ell(\underline{x}_1 : \underline{T}_{\ell 1}, \underline{y}_{\ell 2} : \underline{T}_{\ell 2}, \underline{z}_{\ell 3} : T_{\ell 3})\{P_\ell\}$;
  end N

Ở đây, N là tên lớp được khai báo và M là lớp cha của nó. Nếu không có private thì lớp N được khai báo ngầm định kiểu public. Phần pri định nghĩa các các thuộc tính riêng (private) của lớp N. Phần pro định nghĩa các thuộc tính được bảo vệ (protected) của N. Phần pub định nghĩa các thuộc tính chung (public) cho lớp N. Phần meth khai báo các phương thức của lớp N, trong đó $m_1, m_2, ..., m_\ell$ là các phương thức, ở đây $(x_{i1} : T_{i1}), (y_{i2} : T_{i2}), (z_{i3} : T_{i3})$ và p_i biểu diễn giá trị, kết quả, giá trị tham biến kết quả và phần thân của phương thức m_i.

Ví dụ: Xét hệ thống Bank đơn giản được minh họa bởi biểu đồ lớp UML [7] trong hình vẽ 1. Account là lớp ảo có hai lớp con là Current Account (CA) và Saving Account (SA).

Việc khai báo lớp Account được chỉ ra bởi declAccount như sau:

class Account
  pro: aID: int, balance: int;
  meth:
    getBalance(0, b: int, 0) { b := balance };
    withdraw(x : int, 0, 0) { balance $\geqslant x \vdash$ balance' = balance - x};

end Account

Việc khai báo declCA của CA như sau:

class CA extends Account

meth: withdraw(x: int, 0, 0) { balance := balance - x};

end CA

Việc khai báo declSA của SA như sau:

class SA extends Account

meth: withdraw(x: int, 0, 0) { skip $\triangleleft$ (balance $< x$)$\triangleright$ (balance := balance -x)};

end SA

Việc khai báo declBank của Bank như sau:

class Bank

pri: name: string, address: string, ac: Account, y: Account, A: set(Account)

meth: getBalance(aID: int, b: int, 0) {skip $\triangleleft$ ($\exists!ac \in A$) $\triangleright$ $ac.aID = aID \vdash$
if { $y.aID = aID \rightarrow y$. getBalance($b$)$|y \in A\}fi$};

withdraw(aID: int, x: int, 0) {skip $\triangleleft$ ($\exists!ac \in A$) $\triangleright$ $ac.aID = aID \vdash$
if $\{y.aID = aIDy$.withdraw $(x)|y \in A\}fi$};

newAccount(a: int b: boolean, 0) {

var x : Account;

$(x :=$ new $CA(a);) \triangleleft b \triangleright (x :=$ new $SA(a);)$

$A := A \cup \{x\}$};

end Bank

Việc khai báo lớp cdecl gọi là xác định đúng, được chỉ ra bởi WD(cdecl) khi thỏa mãn các điều kiện sau đây [2, 3]: a) N và M là khác biệt. b) Tên các thuộc tính là khác nhau. c) Những giá trị khởi đầu của các thuộc tính trong chúng phù hợp với các kiểu dữ liệu tương ứng. d) Tên của các phương thức là khác nhau. e) Các tham số của mọi phương thức là khác nhau.

Sự khai báo lớp cdecl để bổ sung thông tin cấu trúc của lớp N vào trạng thái tiếp theo và vai trò của nó được thu nhận bởi thiết kế.

cdecl $\underline{\text{df}}$ $WD(cdecl) \vdash$

$$\begin{pmatrix} \text{alphabet}' = \text{alphabet} \wedge \text{cname}' = \{N\} \wedge \text{superclass}' = \{N \mapsto M\} \wedge \\ \text{pri}' = \{N \mapsto \{\langle t_1 : T_1, c_1\rangle, ..., (t_i : T_i, c_i)\}\} \wedge \\ \text{pro}' = \{N \mapsto \{\langle u_1 : U_1, d_1\rangle, ..., \langle u_j : U_j, d_j\rangle\}\} \wedge \\ \text{pub}' = \{N \mapsto \{\langle v_1 : V_1, e_1\rangle, ..., \langle v_k : V_k, e_k\rangle\}\} \wedge \\ \text{meth}' = \{N \mapsto \{(m_1 \mapsto (\langle \underline{x}_{11} : T_{11}, \underline{y}_{12} : T_{12}, \underline{z}_{13} : T_{13}\rangle, p_1), ..., \\ (m_\ell \mapsto (\langle \underline{x}_{\ell 1} : T_{\ell 1}, \underline{y}_{\ell 2} : T_{\ell 2}, \underline{z}_{\ell 3} : T_{\ell 3}\rangle, p_\ell)\}\} \end{pmatrix}$$

Ở đây các biến lôgic pri, pro, pub đã được giới thiệu ở các thuộc tính đã khai báo của lớp N, từ các tập giá trị của attribute(N) thu nhận được hoàn toàn quan hệ với lớp cha trong số các lớp đã khai báo. Hành vi động của các phương thức có thể không được hình thức hóa trước khi quan hệ phụ thuộc giữa các lớp theo lý thuyết. Như kết quả, biến lôgic meth(N) liên kết mỗi phương thức m_i tới thân p_i của nó đúng hơn là ý nghĩa của nó, điều đó sẽ được tính toán ở cuối phần khai báo.

Tiếp tục ví dụ trên: Dễ ràng thấy rằng cả declAccount và declCA đều là xác định đúng đắn. Ngữ nghĩa của declAccount được chỉ ra bởi thiết kế sau:

declAccount = true $\vdash$

$$\begin{pmatrix} \text{cname}' = \{\text{Account}\} \wedge \text{pro}' = \{\text{Account} \mapsto \{\langle aID : \text{int}\rangle, \langle \text{balance} : \text{int}\rangle\}\} \wedge \\ \text{meth}' = \{\text{Account} \mapsto \{\text{getBalance} \mapsto (\langle 0, b : \text{int}, 0\rangle, b := \text{balance}), \\ \text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, \text{balance} \geqslant x \vdash \text{balance} - x)\}\} \end{pmatrix}$$

Ngữ nghĩa của declCA là như sau:

$$declCA = true \vdash \begin{pmatrix} \text{cname}' = \{CA\} \wedge \text{superclass}' = \{CA \mapsto \text{Account}\} \wedge \\ \text{meth}' = \{\text{Account} \mapsto \\ \{\text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, \text{balance} := \text{balance} - x)\}\} \end{pmatrix}$$

Ngữ nghĩa của declSA là như sau:

$$\begin{pmatrix} \text{cname}' = \{SA\} \wedge \text{superclass}' = \{SA \mapsto \text{Account}\} \wedge \\ \text{declSA} = \text{true} \vdash \text{meth}' = \{Account \mapsto \\ \{\text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, \text{skip} \triangleleft (\text{balance} < x) \triangleright balance := balance - x\};\}\} \end{pmatrix}$$

5.2. Thành phần cấu trúc của lớp

Sau đây ta xem xét thành phần cấu trúc của lớp khai báo cũng như quan hệ của các thành phần đó với nhau. Phần khai báo cdecls của chương trình là sự cấu thành của một số lớp:

$$\text{cdecls} \equiv \text{cdecl}_1; \text{cdecl}_2; ...; \text{cdecl}_k$$

Thành phần cấu trúc lớp khai báo đơn giản là thêm vào nội dung cho môi trường hiện tại α đã phát sinh bởi các thành phần của lớp khai báo đã cung cấp mà không phải định nghĩa lại lớp trong phạm vi của nó. Nó có thể được xác định bởi kết hợp song song như sau [2]: (cdecl1; cdecl2) $\stackrel{\text{df}}{=}$ cdecl1$\|_M$ cdecl2. Ở đây $X(m, m')\|_M Y(m, m')$ được cho bởi thành công thức:

$$(X(m, m'_1) \wedge Y(m, m'_2)); M(m_1, m_2, m')$$

và thiết kế M kết hợp với các đại lượng ra $m1$ và $m2$ của X và Y tới đại lượng ra m của cấu trúc song song được xác định bởi:

$$(\text{cname1} \cap \text{cname2} = \emptyset) \vdash \begin{pmatrix} \text{alphabet}' = \text{alphabet1} \cup \text{alphabet2} \wedge \\ \text{cname}' = \text{cname1} \cup \text{cname2} \wedge \\ \text{superclass}' = \text{superclass1} \cap \text{superclass2} \wedge \\ \text{pri}' = \text{pri}_1 \cup \text{pri}_2 \wedge \\ \text{pro}' = \text{pro}_1 \cup \text{pro}_2 \wedge \\ \text{pub}' = \text{pub}_1 \cup \text{pub}_2 \wedge \\ \text{meth}' = \text{meth1} \cup \text{meth2} \end{pmatrix}$$

Ví dụ: Tiếp tục ví dụ ở trên, ta thấy rằng không có việc khai báo lại lớp trong phần khai báo của hệ thống Bank. Trước hết, ta tính declAccount và declCA:

declAccount; declCA = true $\vdash$

$$\begin{pmatrix} \text{cname}' = \{\text{Account}, CA\} \wedge \text{superclass}' = \{CA \mapsto \text{Account}\} \wedge \\ \text{pro}' = \{\text{Account}\{\langle aID : \text{int}\rangle, \langle \text{balance} : \text{int}\rangle\}\} \wedge \\ \text{meth}' = \{\text{Account}\{\text{getBalance}(\langle 0, b : \text{int}, 0\rangle, b := \text{balance})\}, \\ \text{withdraw} \mapsto (\langle x : int, 0, 0\rangle, b \geq x \vdash \text{balance}' := \text{balance} - x)\}, \\ CA \mapsto \{\text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, \text{balance} := \text{balance} - x)\}\} \end{pmatrix}$$

Việc tính khai báo khai báo declSA của SA hoàn toàn tương tự như trên, từ đó ta sẽ tính declAccount; declCA; declSA như sau:

declAccount; declCA; declSA = true $\vdash$

$$\begin{pmatrix} \text{cname}' = \{\text{Account}, CA, SA\} \wedge \text{superclass}' = \{CA \mapsto \text{Account}, SA \mapsto \text{Account}\} \wedge \\ \text{pro}' = \{\text{Account}\{\langle aID : \text{int}\rangle, \langle \text{balance} : \text{int}\rangle\}\} \wedge \\ \text{meth}' = \{\text{Account} \mapsto \{\text{getBalance} \mapsto (\langle 0, b : \text{int}, 0\rangle, b := \text{balance}), \\ \text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, b \geq x \vdash \text{balance}' := \text{balance} - x)\}, \\ CA \mapsto \{\text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, \text{balance} := \text{balance} - x)\} \\ SA \mapsto \{\text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, \text{skip} \triangleleft (\text{balance} < x) \triangleright \text{balance} := \text{balance} - x)\}\} \end{pmatrix}$$

Tiếp tục ta tính được declAccount; declCA; declSA; declsBank như sau:

declAccount; declCA; declSA; declsBank = true $\vdash$

$$\begin{pmatrix} \text{cname}' = \{\text{Account}, CA, SA\text{Bank}\} \wedge \text{superclass}' = \{CA \mapsto \text{Account}, SA \mapsto \text{Account}\} \wedge \\ \text{pri}' = \{\text{Bank} \mapsto \langle \text{cname} : \text{string}, \text{address} : \text{string}, A : \text{set}(\text{Account})\rangle\} \wedge \\ \text{pro}' = \{\text{Account} \mapsto \{\langle aID : \text{int}\rangle, \langle \text{balance} : \text{int}\rangle\}\} \wedge \\ \text{meth}' = \{\text{Account} \mapsto \{\text{getBalance} \mapsto (\langle 0, b : \text{int}, 0\rangle, b := \text{balance}), \\ \text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, bx \vdash \text{balance}' := \text{balance} - x)\}, \\ CA \mapsto \{\text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, \text{balance} := \text{balance} - x)\} \\ SA \mapsto \{\text{withdraw} \mapsto (\langle x : \text{int}, 0, 0\rangle, \text{skip} \triangleleft (\text{balance}\langle x\rangle \triangleright \text{balance} := \text{balance} - x)\}, \\ \text{Bank} \mapsto \{\text{getBalance} \mapsto (\langle aID : \text{int}, b : \text{int}, 0\rangle\{\text{skip} \triangleleft (\exists! ac \in A) \triangleright ac.aID = aID \vdash \\ if\{y.aID = aID \rightarrow y.\text{getBalance}(b) | y \in A\} fi)\}, \\ \text{withdraw} \mapsto \{(\langle aID : \text{int}, x : \text{int}, 0\rangle, \{\text{skip} \triangleleft (\exists! ac \in A) \triangleright ac.aID = aID \vdash \\ if\{y.aID = aID \rightarrow y.\text{withdraw}(x) | y \in A\} fi)\}, \\ \text{newAccount} \mapsto \{(\langle a : \text{int}, b : \text{Boolean}, 0\rangle, \\ \text{var } x : \text{Account}; \\ x := \text{new } CA(a) \triangleleft b \triangleright x := \text{new } SA(a); A := A \cup \{x\} \\ \text{end}x)\};\} \end{pmatrix}$$

Từ mô hình tương tự như trên, He Jifeng và các cộng sự đã có những cải tiến tính toán cho cho hệ thống đối tượng rCOS của họ [4].

6. MỘT SỐ KẾT QUẢ

Trên đây, chúng tôi đã trình bày ngữ nghĩa các quan hệ đại số cho đặc tả hệ thống hướng đối tượng, và đã xem xét toán tử chương trình trong ngôn ngữ hướng đối tượng là một cách mô tả chính xác giống như bản sao trong ngôn ngữ lệnh. Do đó hầu hết các luật đại số đã được phát triển của ngôn ngữ lênh dễ dàng được áp dụng trong việc thiết kế các

chương trình hướng đối tượng. Ví dụ như điều kiện, tính không tiền định và trình tự trong hệ thống là các đối tượng giống như các luật cơ bản được thiết kế cho các ngôn ngữ lệnh.

7. NHẬN XÉT VÀ KẾT LUẬN

Mô hình quan hệ đại số được phát triển dựa vào một ngôn ngữ hướng đối tượng. Việc xử lý kết hợp kiểu động và kiểm tra trạng thái với ngữ nghĩa giống như hướng đặc tả truyền thống. Các toán tử chương trình được xử lý chính xác như là bản sao trong ngôn ngữ ràng buộc. Do đó, hầu hết các luật đại số được phát triển cho ngôn ngữ lệnh vẫn có thể áp dụng được cho đặc tả thiết kế chương trình hướng đối tượng. Các cơ sở tính toán cho hệ thống hướng đối tượng trong bài này cũng có thể được sử dụng để hình thức hóa liên kết các mô hình ca sử dụng (use case) và mô hình lớp trong UML. Nó cũng có thể sử dụng như là cơ sở cho việc cải tiến tính toán cho tiến trình phát triển của chương trình hướng đối tượng.

Các đặc tả trong ngôn ngữ này dễ dàng được viết và hiểu tương tự như ngôn ngữ Java hoặc C^{++}, và ý nghĩa là hoàn toàn dựa trên cách tiếp cận trạng thái đã được thiết lập đúng.

TÀI LIỆU THAM KHẢO

[1] Booch, G., Rumbaugh, J. and Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[2] C.A.R. Hoare and He Jifeng, *Unifing Theories of Programming*, Prentice Hall, 1998.

[3] He Jifeng, Zhiming Liu and Li Xiaohan, A Relational Model for Object-Oriented Programming, "Technical report UNU/IIST No.231, UNU/IIST: International Institute for software technology", the United Nations University, Macau, May 2001.

[4] He Jifeng, Li Xiaohan and Zhiming Liu, rCOS: Refinement Caclculus for Object Systems, Technical report UNU/IIST No.322, UNU/IIST: International Institute for software technology, the United Nations University, Macau, May 2005.

[5] Ivar Jacopson, Gray Booch and James Rumbaugh, *The Unified Software Development Process*, Addision-Wesley, 2000.

[6] P. America, F.de Boer, Reasoning about dynamically evolving process structures, *Formal Aspect of Computing* (6) (1994) 269–316.

[7] Rational Rose Corp., Rational Rose 2002, http://www.rational.com/uml/.

Nhận bài ngày 29 - 7 - 2005