面向对象Z的子类型继承和推理规则*

王云峰[1,2] 李必信[1] 郑国梁[1]

[1](南京大学计算机软件新技术国家重点实验室 南京 210093)

[2](解放军理工大学气象学院 南京 211101)

E-mail: wangyf@seg.nju.edu.cn

摘要 讨论了COOZ(complete object-oriented Z)中的继承关系,将继承分为增量继承和子类型化继承,并重点讨论了子类型化继承.定义了一种行为子类型化继承,证明了该方法的合理性,并据此定义了COOZ的规约继承及规约继承的推理规则.所讨论的子类型化方法与E.Cusack等人的方法相比,具有可构造性,并且比Liskov等人的方法具有更大的灵活性.

关键词 形式方法,面向对象,继承,子类型,形式规约.

中图法分类号 TP311

将形式方法与面向对象方法相结合已成为软件开发方法研究的一个重要方向.90年代初,将形式规约语言Z进行面向对象的扩充成为研究热点,先后产生了若干个Z的面向对象扩充版本[1].COOZ(complete object-oriented Z)是在分析以往Z的面向对象扩充的基础上,采用更为先进、合理的技术对Z进行面向对象的扩充.

对Z进行面向对象的扩充使得形式方法和面向对象方法相得益彰,如,OO方法中类及其关系的构造技术使Z适宜描述大型和复杂系统,同时Z本身的数学特性使我们可以对规约进行推理和计算,以保证规约的正确性.

继承是面向对象方法的最重要的概念,继承一般用于两个方面:行为继承和实现继承.行为继承即规约继承,是一种强子类型化继承[2].任意一处父类对象均可由子类对象代替.实现继承即增量继承,通过修改已有的类,派生出新类,体现了复用和代码共享.在实现子类型化时有多种方法,许多面向对象语言通过对实现继承增加约束条件来实现子类型化,约束条件一般通过方法的型构定义[3].这些约束条件使得在应用子类型多态时不致出现运行错误,但不能保证语义上的一致.

我们定义子类型化的目的在于:如果已知父类的属性和功能,即父类的规约,当对子类进行推理时,能直接利用已知的信息,增强对复杂系统功能进行推理的能力.其次,为了使子类型对象具有所有父类的属性和行为,保证子类型对象可以代替父类对象,使其成为一种精化手段.其三,保证COOZ规约验证的模块性[2].例如,方法m的验证是针对父类型对象规约的,当子类型对象代替父类型对象时,对象规约发生改变,此时,本应该针对新规约重新验证m,但这意味着对继承来的原有代码(或规约)均要重新验证.这是不必要的,可以定义一种模块化的验证方法,这一方法的基础就是规约继承.

我们在下面对COOZ的继承关系的讨论中,把继承分为派生和子类型化两种方式,分别给出形式定义,重点讨论具有较大灵活性的一种子类型化继承,并定义了规约继承的方法.同时,讨论了该方法的合理性.为了能利用规约中的子类型关系进行推理,给出了基于$\bar{\omega}$逻辑的COOZ继承的推理规则.

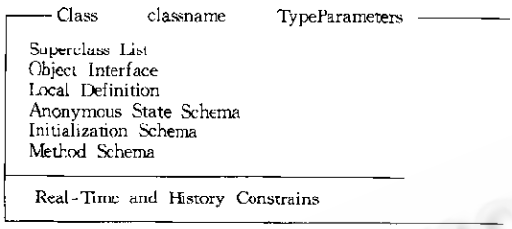
* 本文研究得到国家自然科学基金(No.69673006)和国家"九五"科技攻关项目基金(No.98-780-01-07-06)资助.作者王云峰,1964年生,博士生,讲师,主要研究领域为形式化方法,面向对象技术.李必信,1969年生,博士生,讲师,主要研究领域为程序理解,面向对象技术.郑国梁,1937年生,教授,博士生导师,主要研究领域为软件工程.

本文通讯联系人:王云峰,南京 210093,南京大学计算机软件新技术国家重点实验室

本文1998-12-22收到原稿,1999-04-12收到修改稿

1 COOZ 简介

COOZ[4]是一种面向对象的形式规约语言,其中的类由类模式(class schema)表示:



类模式由类名、类参数、父类列表、对象接口、局部定义、状态模式、初始化模式、方法模式以及表示对象实时历史约束的类不变式等组成,类模式的详细语法及语义见文献[4].

每个类模式都有 Anonymous State Schema(无名状态模式)描述类属性和类不变式,即描述该类对象的状态空间,在继承时,无名状态模式自动加以合并.

类模式可以有多个方法模式,类对象可接受的消息必须通过其方法模式加以说明,无名状态模式被自动引入该类对象的方法模式.在继承时,根据同名模式进行合并的原则,同名方法模式进行合并,这里,模式合并的含义是模式的合取[5].

2 COOZ 中的继承:派生和子类型化

我们把继承分为增量继承和子类型化继承两种.增量继承为一般意义上的派生继承,即在已有类的基础上构造新类,是重用现有规约的基础.子类型化继承是一个类替代另一个类的基础,可作为类的精化机制.下面,我们将分别讨论这两种继承,并且为了与规约精化的实现语言 C++* 相适应,我们引入一种将类作为类型、将子类作为子类型的子类型化继承方法.

2.1 增量继承——派生

增量继承是在现有的类定义上增加"方法和变量"构成新类的过程.增量继承不能保证派生类的对象也是父类的对象,即不能保证派生类是父类的子类型.我们简化并扩展 Cusack E[6]关于继承的定义.

设类 A 的状态模式为 S_A,X_A 为 S_A 的状态空间,O_j 为类 A 的 j 个方法.类 B 的状态模式为 S_B,X_B 为 S_B 的状态空间,P_j 为类 B 的 j 个方法.若 S_B 为 S_A 的实例(见文献[6]中的定义 2),则存在映射 $f:X_B\to X_A$,把 S_B 的实例映射到 S_A 的实例.根据类完整性定义,O_j 可看成是 X_A 之间的关系 $R(O_j)$,显然,$R(O_j)\subseteq X_A\times X_A$.同样可得:$R(P_j)\subseteq X_B\times X_B$.相应地,$f$ 可得映射 $f\times f:X_B\times X_B\to X_A\times X_A$.若相应类 A 的方法 O_j 和 P_j 在类中无显式定义,则有无派生类的定义如下.

定义 1. 若 B 的状态模式 S_B 的实例为类 A 的状态模式 S_A 的实例,即存在映射 $f:X_B\to X_A$,若类 B 的方法 P_j 由下式定义

$$R(P_j)^{f\times f}=R(O_j)\cap(X_B^t\times X_B^t),$$

则 B 为 A 的派生类,若 $R(P_j)^{f\times f}$ 为空,则 P_j 在 B 中无定义.

定义 1 是极为严格的继承定义,其定义的子类不能修改父类的方法.事实上,增量继承相当于宏定义,通过把类模式的继承语句全部展开为实际语句,可得到没有继承关系的规约.

2.2 子类型化继承

E. Cusack 定义的子类型化继承是一种非构造性方法,并且限制子类型修改父类型的方法.S. Drossoplou 等人提出的"ST&T"子类型化方法是一种极弱的子类型化方法,仅能满足子类型化的语法约束,可以保证子类型

* C++ 的类与类型合一,子类与子类型合一.

表达式代替父类型表达式时类型不出错，但不能保证子类型对象行为与父类型对象一致. 事实上，类型检查只能查出程序部分错误，类型正确不能保证行为正确.

为了达到前述子类型化的 3 个目的，需要定义一种“行为子类型化”概念，既满足子类型化的语法约束，又满足子类型化的语义约束. 保证子类型对象行为与父类型对象一致，即保证子类型对象代替父类型对象时，不会出现超出父类行为规约的行为(意外行为).

为了比较子类型和父类型，我们扩展了文献[6]中的映射. 对于存在子类型关系的集合 X,Y：$X\leqslant Y$(符号$\leqslant$表示子类型关系)，存在模拟函数 $f_{X\to Y}$：$X\leqslant Y$. 要判断子类型化继承，既要比较类型的状态模式，又要比较类型的方法模式. 比较类型的状态模式，就是比较类型状态空间中的不变式，我们用 $I_X(v_x)$ 表示类型 X 的不变式，其中 v_X 表示 X 的取值.

为了比较方法模式，我们用 $\vec{V},\vec{U}$ 分别表示类型 S,T 的方法 m 的输入变量类型，用 $\vec{V}_r,\vec{U}_r$ 分别表示类型 S，T 的方法 m 的输出变量类型.

用 $Pre_S^m(this,\vec{x})$ 表示 S 的方法 m 的前置条件，用 $Post_S^m(\Delta this,\Delta\vec{x},\vec{r}')$ 表示 S 的方法 m 的后置条件. 其中 ΔX 表示 $X\cap X'$，即操作前后的变量. 下面给出一种子类型化继承的定义.

定义 2(行为子类型化继承). 类型 S 为类型 T 的子类型，当且仅当下列条件满足：

• 不变式规则：对所有 S 的值 v_S

$$I_S(v_S)\Rightarrow I_T(f_{S\to T}(v_S)).$$

• 方法规则

对所有 S 的对象 $this:S$，输入变量 $\vec{y}:\vec{V}$.

(1) 前置条件规则

$$pre_T^m(f_{S\to T}(this),\vec{y})\Rightarrow pre_S^m(this,f_{\vec{V}\to\vec{U}}(\vec{y})).$$

(2) 后置条件规则

$$(pre_S^m(this,f_{\vec{V}\to\vec{U}}(\vec{y}))\Rightarrow post_S^m(\Delta this,f_{\vec{V}\to\vec{U}}(\vec{y}),f_{\vec{V}\to\vec{U}}(\vec{y}'),\vec{r}'))\Rightarrow$$
$$(pre_T^m(f_{S\to T}(this),\vec{y})\Rightarrow post_T^m(f_{S\to T}(this),f_{S\to T}(this'),\Delta\vec{y},f_{\vec{V}\to\vec{U}}(\vec{r}'))).$$

需要指出的是，式中 $f_{\vec{V}_r\to\vec{U}_r}:\vec{V}_r\leqslant\vec{U}_r$ 反映了子类型关系的逆变原则(contravariance)，而 $f_{\vec{V}\to\vec{U}}:\vec{V}\leqslant\vec{U}$ 则反映了子类型关系的协变原则(convariance).

2.3 强制规约继承

在使用 COOZ 建立程序规约时，为了保证在使用继承时自动保证子类型关系，我们根据上述子类型关系的定义，定义一种强制规约继承的方法. 虽然这样限制了继承的灵活性，但却具有在文章开始部分中谈到的 3 个好处，使用规约继承，强制父类型对象的方法在子类型对象中进行正确的操作. 而且，如下面所要讨论的，这一方法较其他子类型化方法更为灵活. 为表示方便，引入符号 $\uparrow I$ 表示继承的不变式，$\uparrow pre$，$\uparrow post$ 分别表示继承的前后置条件. 这些符号的定义在第 3.2 节中给出.

定义 3(强制规约继承). 设 S 为若干类型的子类型，则 S 的完整的规约为：

• 不变式 $I(v):Is(v)\wedge\uparrow I$.

• 前置条件 $pre^m(this,\vec{x})$ 为 $pre_S^m(this,\vec{x})\vee\uparrow pre^m$.

• 后置条件 $post^m(\Delta this,\Delta\vec{x},\vec{r}')$ 为 $(pre_S^m(this,\vec{x})\Rightarrow post_S^m(\Delta this,\Delta\vec{x},\vec{r}'))\wedge\uparrow post^m$.

值得注意的是，后置条件的定义比 Liskov[7]等人所定义的条件范围要大，允许了类型方法在父类型方法定义域之外仍然有效.

例如，左边两个操作模式分别表示父类型和子类型的方法 m，根据强制规约继承的定义，完整的子类型方法 m 的后置条件为

m	m
$!r:N$	$!r:N$
$!r=1$ if $x>0$	$!r=-2$ if $x<0$

$$((\text{if } x>0)\to !\ r=1)\wedge((\text{if } x<0)\Rightarrow !\ r=-2).$$

该后置条件并不蕴含父类型方法的后置条件，但在行为上，子类型对象可代替父类型对象. 为了便于比较，下面给出 Liskov 等人的行为子类型化继承的定义.

S 为 T 的子类型，需要满足下述条件：

(1) 不变式规则:$\forall s:S \cdot Is(s) \Rightarrow I_t(f(s))$,

(2)方法规则:$\forall s:S$.

前置条件规则: $Mt.pre[f(s)/s] \Rightarrow Ms.pre[s]$,

后置条件规则: $Ms.post[s'] \Rightarrow Mt.post[f(s)/s, f(s')/s']$.

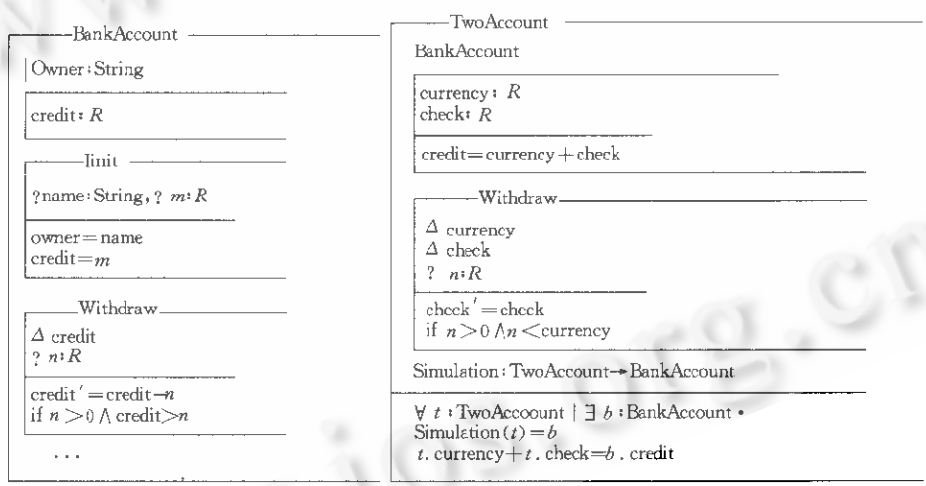
其中 s 为类 S 的状态变量,s' 表示后状态变量,f 表示模拟函数(见第 2.2 节中的定义),Mx 表示类 x 中的方法.

由此可见,强制规约继承的方法比 Liskov,E.Cusack 定义的行为子类型化继承的条件要弱.如,上例并不满足 Liskov 的条件,但在行为上,子类型对象可代替父类型对象.因此,强制规约继承的方法可在给程序规约设计者使用继承时提供更大的灵活性.另外,在对 m 的调用进行推理时,当作用于父类型对象时,子类型对象的 m 仍然是有效的.同时,子类型方法的后置条件蕴涵了父类型方法的规约,使父类型方法的规约在子类型对象中自动满足,为模块化推理提供了基础.

2.4 实 例

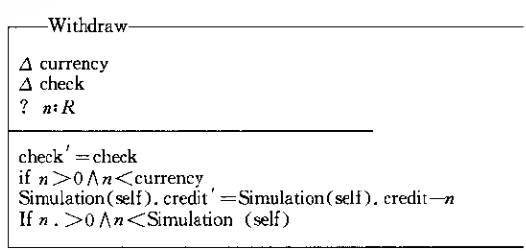
为了说明规约继承,下面给出一个简单的实例.类模式(schema)BankAccount 表示银行帐户,类模式 TwoAccount 为 BankAccount 的子类型,它将 BankAccoount 中的属性 credit 精化成 currency 和 check.由于 TwoAccount 具有更多的信息,为了保持父类型的规约,子类型应该提供建立其抽象值与父类型相应的抽象值关系的模式.这种抽象模式最初是用模拟函数(simulation function)表示,之后扩展为关系.为简单起见,实例中采用模拟函数.R,String 为已有类型,分别表示实数和字符串.

[R,String]



类模式(BankAccount) 类模式(TwoAccount)

其中 Simulation 代表从 TwoAccount 到 BankAccount 的模拟函数.根据规约继承,TwoAccount 的操作模式 Withdraw 完全展开的形式为



其中 self 表示 TwoAccount 对象.不难看出,Withdraw 反映了前面定义的规约继承.

3 COOZ 中继承关系的推理规则

在 COOZ 中，我们引入扩充的 Z 的逻辑 $\tilde{\omega}$[8]，建立相关的推理规则. 这里，我们先介绍 $\tilde{\omega}$，然后重点讨论 COOZ 中和继承相关的 $\tilde{\omega}$ 的扩充.

3.1 $\tilde{\omega}$ 逻辑

$\tilde{\omega}$ 是一种 Gentzen 式的相继式演算，公理和定理由相继式表示，然后运用推理规则推导出其余定理. 为了表示方便，引入元函数，这些元函数在本逻辑以外定义.

· 相继式

相继式的形式为：$d \mid \Psi \vdash \Phi$.

其中 d 为声明表，Ψ 为谓词集合，称为“前提”，Φ 也是谓词集合，称为“结果”. 在 d 的环境条件下，当 Ψ 的所有谓词均为真时，Φ 中至少有一个谓词为真，此时，形式 $d \mid \Psi \vdash \Phi$ 有效，事实上相当于 Ψ 的谓词合取，而 Φ 的谓词析取.

· 推理规则

相继式的推理规则采用以下形式：

$$\frac{\text{premisses}}{\text{conclusion}}(\text{name})[\text{proviso}].$$

其中 premisses 为规则的前提，由个相继式组成；conclusion 是单一的相继式，为规则的结论；proviso 是规则应用的环境中必须为真的谓词，为规则有效的条件. 如果 proviso 满足，且 premisses 有效，conclusion 有效，则称推理规则是合理的. 规则中的 name 用于标识该规则，称为规则名.

3.2 规约继承的推理规则

为讨论方便，先定义几个相关的元函数. 元函数 χ 返回子类继承的父类名的集合，如 S 继承 $T_1,\ldots,T_n$，则有 $\chi(S)=\{T_1,\ldots,T_n\}$. 元函数 Ω 返回类中的方法名，包括继承的方法.

下面以推理规则的形式给出 $\uparrow I$，$\uparrow pre$，$\uparrow post$ 的定义*.

· 继承的不变式 $\uparrow I$ 的定义

$$\frac{T_1 :: I_1(v_1),\ldots,T_n :: \Gamma I_n(v_n)}{S :: \Gamma \uparrow I = I_1(v_1) \wedge \ldots \wedge I_n(v_n)}(\uparrow I)[p],$$

其中

$$\begin{aligned} &p \hat{=} \chi(S)\{T_1,\ldots,T_n\},\\ &v_1 = f_{S\to T_1}(v_S),\\ &\quad\vdots\\ &v_n = f_{S\to T_n}(v_S), \end{aligned}$$

· 继承的操作(方法)的定义

(1) 继承的前置条件

$$\frac{T_1 :: \Gamma pre_1^m(this_1,\vec{x}_1)=p_1,\ldots,T_k :: \Gamma pre_k^m(this_k,\vec{x}_k)=p_k}{S :: \Gamma \uparrow pre^m = p_1 \vee \ldots \vee p_k}(\uparrow pre^m)[q_1],$$

其中

$$\begin{aligned} &q_1 \hat{=} \chi(S)=\{T_1,\ldots,T_n\},\\ &m \in \Omega(T_1) \cap \ldots \cap \Omega(T_k),\\ &m \notin \Omega(T_{k+1}) \cup \ldots \cup \Omega(T_n),\\ &\forall\, i,i \in \{i,\ldots,k\},\\ &this_i = f_{S\to T_i}(this),\\ &f_{\vec{v}_i\to\vec{v}}(\vec{x}_i)=\vec{x}. \end{aligned}$$

---

* 在推理规则表达式中，用 Γ 表示 $\vdash$.

(2) 继承的后置条件

$$\frac{T_1::\Gamma post_1=(p_1\Rightarrow post_1^m(\Delta this_1,\Delta\vec{x}_1,\vec{r}'_1)),\ldots,T_k::\Gamma post_k=(p_k\Rightarrow post_k^m(\Delta this_k,\Delta\vec{x}_k,\vec{r}'_k))}{S::\Gamma\uparrow post^m=post_1\wedge\ldots\wedge post_k}(\uparrow post^m)[q_2]$$

$$q_2\equiv q_1,$$

$$this'_i=f_{S\to T_i}(this'),$$

$$f_{\vec{v}_i\to\vec{v}}(\vec{x}'_i)=\vec{x}',$$

$$\vec{r}'_i=f_{\vec{v}_r\to\vec{v}_{ri}}(\vec{r}')(i=1,\ldots,k).$$

q_1 说明 m 是 $T_1,\ldots,T_k$ 共有的方法,m 不属于 $T_{k+1},\ldots,T_n$.

这样定义保证了 S 和 T_i 的规约继承的关系,这种定义可作为操作模式(schema)和状态模式合取($\wedge$)的语义基础.上述推理规则为 COOZ 规约中的继承关系的推理奠定了理论基础.

4 规约继承的合理性

为了讨论上述规约继承的合理性,引入以最弱前置条件表示程序语义的精化演算[9],用其中的 Frame 表示类中的方法:

$$[\![x:[pre,post]]\!]P\equiv\forall\ x\cdot pre\wedge post\Rightarrow P$$

其中 x 表示在操作中发生改变的变量,P 表示谓词,$\equiv$表示"定义为".

COOZ 中的子类型的方法定义为

$$x:[pre_t,post_t]\cup x:[pre_s,post_s]\equiv x:[pre_t\vee pre_s,(pre_t\Rightarrow post_t)\wedge(pre_s\Rightarrow post_s)],$$

其中下标 s,t 分别表示父类型和子类型.显然,上式即为规约继承的方法模式的定义.根据上述定义,用 $x:[pre,post]$表示子类型的完整的方法,不难证明:

$$[\![x:[pre,post]]\!]P\Rightarrow[\![x:[pre_s,post_s]]\!]P$$

即由规约继承所得到的子类型的方法满足行为子类型的性质,由此可以看出,本文所定义的规约继承是合理的.

值得指出的是,规约继承可作为类精化的手段.相关内容将另文讨论.规约继承作为一种行为子类型的方法,能否实现所有的行为子类型,即该方法是否完备,还需进一步研究.

5 结论和进一步的研究

本文讨论了 COOZ 中的继承关系,将继承分为增量继承和子类型化继承,重点讨论了子类型化继承.我们定义了一种行为子类型化继承,并据此定义了 COOZ 的规约继承及规约继承的推理规则.本文所讨论的子类型化方法与 E.Cusack 等人的方法相比,具有可构造性,并且比 Liskov 等人的方法具有更大的灵活性.文中讨论时省略了"约束条件",如何在继承中考虑 COOZ 的"实时和历史约束"值得研究.另外,根据文中的推理规则,给出推理策略,以便在规约精化和验证中加以应用,这项工作还需要进一步研究.

参考文献

1 Stepney S, Barden R, Cooper D. Object Orientation in Z. London: Springer-Verlag, 1992

2 Dhara K K, Leavens G T. Forcing behavioral subtyping through specification inheritance. In: Kemmerer R A ed. Proceedings of the ICSE-18. Washington, DC: IEEE Press, 1996. 258~267

3 Drossopolou S, Karathanos S, Yang Dan. Static typing for object oriented language. In: Goldsack S J, Kent S J H eds. Formal Method and Object Technology. London: Springer-Verlag, 1996. 262~286

4 Yuan Xiao-dong, Hu De-qiang, Xu Hao *et al*. COOZ: a complete object oriented extension to Z. ACM Software Engineering Notes, 1998,23(4):78~81

5 Spivey J M. The Z Notation: A Reference Manual. 2nd Edition, Series in Computer Science, London: Prentice-Hall, Inc., 1992

6 Cusack E. Inheritance in object oriented Z. In: America P ed. Proceedings of the ECOOP'91. Volume 512 of Lecture Notes

in Computer Science, New York: Springer-Verlag, 1991. 167～179

7 Liskov B, Wing J M. A behavioral notation of subtyping. ACM Transactions on Programming Languages and Systems, 1994,16(6):1811～1841

8 Smith G. Extending ϖ of object-Z. In: Bowen J P, Hinchey M G ed. ZUM'95: the Z formal specification notation. Proceedings of the 9th Annual Z User Meeting, Volume 967 of Lecture Notes in Computer Science. London: Springer-Verlag, 1995. 276～296

9 Morgan C C, Gardiner P H B. Data refinement by calculation. Acta Information, 1990,27(6):481～403

On Subtyping Inheritance and Inference Rules in Object-Oriented Z

WANG Yun-feng[1,2] LI Bi-xin[1] ZHENG Guo-liang[1]

[1](*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)

[2](*Meteorology College PLA University of Technology Nanjing University Nanjing* 211101)

Abstract The inheritance relation of COOZ is discussed. It is divided into increasing inheritance and subtyping inheritance. The latter is studied and a behavioral subtyping inheritance is defined, by which the specification inheritance and its inference rules are defined. The soundness of the method is proved. The offered method is constructive compared with that of E. Cusack and is more flexible than that of Liskov et al.

Key words Formal method, object oriented, inheritance, subtyping, formal specification