XYZ/E 面向对象程序语义概述*

郭 亮+, 唐稚松

(中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

An Overview Towards the Semantics of XYZ/E Object-Oriented Programs

GUO Liang+, TANG Zhi-Song

(Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: 86-10-62556910, E-mail: gls@ios.ac.cn

http://www.ios.ac.cn

Received 2001-11-15; Accepted 2001-12-27

Guo L, Tang ZS. An overview towards the semantics of XYZ/E object-oriented programs. *Journal of Software*, 2003,14(3):356~361.

Abstract: The element acted as object in XYZ/E object-oriented program is agent, a module consists of a data package and a process. In this paper, the semantics of XYZ/E object-oriented programs are defined, including their language elements, under the framework of temporal logic. And several theories for proving the semantics consistency between these elements are also provided.

Key words: temporal logical language XYZ/E; object-oriented program; agent; semantic consistency; refinement

摘 要: XYZ/E 面向对象程序中表示对象概念的语言成分是代理机构:一种由一个数据包块和与之匹配的进程所组成的模块.在时序逻辑框架下给出了面向对象程序及其包含的各种语言成分的语义,并提供了几个用于证明这些语言成分之间的语义一致性的定理.

关键词: 时序逻辑语言 XYZ/E;面向对象程序;代理机构;语义一致性;精化

中图法分类号: TP311 **文献标识码**: A

面向对象程序设计语言(object-oriented programming language)的研究始于 20 世纪 60 年代的 Simula,这类语言均包含一种特定的语言成分:围绕一种数据结构及其规定的一组运算组成的模块(module).这种模块之间可根据其数据结构间的包含关系,规定其相互间运算(或属性)的继承性.面向对象程序即由这样一些模块(对象)组成,模块中定义的数据结构或运算只在其定义模块中考虑实现细节,而对其他引用这些运算(或属性)的模块进行隐藏,从而只对外界开放必要信息,以实现模块的封装.这样的模块可被方便地重复引用,从而加强其可重用性.将形式化方法应用于面向对象技术,定义对象的形式语义模型[1]并讨论对象之间的求精关系[2],是当前面向对象技术研究领域的一个重要发展方向.

XYZ/E 语言[3]引入代理机构(agent)表示对象概念.代理机构由一个数据包块(package)和与之匹配的进程

* Supported by the National Natural Science Foundation of China under Grant No.60073020 (国家自然科学基金); the National High Technology Development 863 Program of China under Grant No.863-306-ZT02-04-01 (国家 863 高科技发展计划)

第一作者简介: 郭亮(1976－),男,江西吉安人,博士,主要研究领域为软件工程.

(process)组成,是一种具有封装性和继承性特点的模块,XYZ/E 面向对象程序即由这样一些模块组成.本文在线性时序逻辑框架下给出面向对象程序及其包含的各种语言成分(包括条件元、单元、过程、进程和代理机构)的语义,并提供一些定理用于证明它们的语义一致性.

1 条件元和单元

XYZ/E 语言最基本的语言成分是条件元(conditional element).如形式(1)所示条件元直接定义了程序相邻状态之间的转换关系;形式(2)所示条件元表示程序抽象规范,其中符号@可以是下一时刻算子$O 和最终时刻算子◊;两种形式的条件元都是一个时序逻辑式,其语义即为此时序逻辑式的语义模型.

$LB=y\wedge R\Rightarrow \$O(v_1,v_2,\ldots,v_n)=(e_1,e_2,\ldots,e_n)\wedge \$OLB=z$, (1)

$LB=y\wedge R\Rightarrow @(Q\wedge LB=z)$. (2)

XYZ/E 中表示算法的语言成分为如形式(3)所示的单元(unit).其中 $A_1,A_2,...,A_n$ 是条件元,符号“;”等同于逻辑联结词合取.单元也是一个时序逻辑式,其语义即对应于此时序逻辑式的语义模型.

$\Box[A_1;A_2;\ldots;A_n]$. (3)

2 过程和过程调用

串型程序中具有完整功能及独立结构的程序段称为过程(procedure),其形式如(4)所示,其中,Parameter DeclPart 部分声明过程的参数变量,LocalVarPart 部分声明过程的局部变量,Probody 部分为表示过程具体算法的单元,WherePart 部分为在以时序逻辑公式形式刻画的过程运行时必须满足的静态约束.

Procedure::=ProcName[ParameterDeclPart]==[[LocalVarDeclPart]Probody][WherePart]. (4)

过程中可引用的变量分为 3 种:局部变量$\{i_1,\ldots,i_{ni}\}$、参数变量$\{p_1,\ldots,p_{np}\}$和过程声明所在包块(程序)中定义的全局变量$\{g_1,\ldots,g_{ng}\}$.我们定义 $Int_{Porc}=\{i_1,\ldots,i_{ni}\}$为过程 *proc* 内部变量集,定义 $Ext_{proc}=\{p_1,\ldots,p_{np},g_1,\ldots,g_{ng},LB\}$.过程语义对应于如(5)所示的时序逻辑式的语义模型,在此时序逻辑公式中,每个约束变元 *u* 都存在一个辖域,本文为方便起见,省略不写.

$LF_{Proc}=_{def}\exists_{i1,\ldots,in}$:Probody∧WherePart. (5)

过程实现的功能只有在被程序或其他进程(过程)调用时才能体现,过程调用形式如(6)所示,是如(7)所示条件元组的缩写.

$LB=m_j\wedge P\Rightarrow Proc(apar|par)$ /**apar* 和 *par* 分别对应于过程的实参和形参*/ (6)

$LB=m_j\Rightarrow \$O(finps)=(ainps)\wedge \$OPUSH(RTS,m_{j+1})\wedge \$OLB=START\_P$;

$LB=m_{j+1}\Rightarrow \$O(aoutps)=(foutps)\wedge \$OLB=NEXT$. (7)

/**finps*,*foutps*,*ainps* 和 *aoutps* 分别对应于过程的输入形参、输出形参、输入实参和输出实参;*RTS* 是与调用过程的程序相对应的标号返回栈,$\$OPUSH(RTS,m_{j+1})$是指将执行过程后的转返标号存入到 *RTS* 中;每个过程转出标号都为 RETURN,是 *POP*(*RTS*)的缩写,即将 *RTS* 顶上所存标号转返为当前控制标号*/.

过程调用采用实参取代声明中的形参,而形参对过程调用所在单元并不可见,算作单元的辅助变量,因此,(6)所示过程调用的语义即为如(8)所示时序逻辑式的语义模型.

$\exists_{finps,foutps}$: $LF_{Proc}\wedge L$ /**L* 为(7)所示条件元组对应的时序逻辑式*/ (8)

3 进程和通信

在并行程序中引用的具有完整功能及独立结构的程序段称为进程,其语法与过程类似,但进程除了完成特定运算之外,还需要与其他进程动态通信.进程间通信存在两种实现方式:基于共享变量通信和基于通道并通过通信命令通信.显然,采用通道及通信命令的进程通信方式可以被看做是一种采用特殊结构的共享缓冲区变量实现的通信方式.缓冲区的长度可以为任意正整数 *N* 或无穷大.

由于本文主要讨论面向对象程序的语义,每个对象是一个由进程和包块组成的代理机构,代理机构之间如

果存在共享变量,且对这些共享变量没有特殊的约束机制,则单个代理机构将无法约束其他代理机构何时以及如何修改此共享变量值,这将破坏代理机构的封装性,导致其语义难以讨论,因此本文对此种通用的基于共享变量通信的方式不予考虑.

在基于通道通信方式中,对每个通道而言,存在一对发送进程和接收进程.按发送、接收进程的等待方式分为同步和异步两种通信机制,而同步通信机制又可通过采用握手方式的异步通信机制来实现,因此本文假定进程之间只能异步通信.此时,通道可看做是一个存储特定数据类型数据的先进先出缓冲区,若缓冲区未满,则发送进程可向缓冲区尾部写入数据;若缓冲区非空,则接收进程可从此缓冲区头部读取数据.

进程通过通信命令实现通信.通信命令包含输入命令和输出命令,语法分别如(9)、(10)所示,实现接收进程通过通道从发送进程接收信息和发送进程通过通道向接收进程发送信息.其中,*chn* 是进程 P_1(接收进程)和进程 P_2(发送进程)之间为实现点到点通信而建立的通道;x_1,x_2 分别是进程 P_1,P_2 中定义的具有与 *chn* 通道传递的信息相同数据类型的变量.如果将通道看做是缓冲区,则输入(输出)命令则可解释为向(从)缓冲区写入(读出)数据,那么通信命令的语义可通过将其转换为形式为(1)的条件元而得到.

$$LB_{P1}=y_1\wedge R_1\Rightarrow \$O chn?x_1\wedge \$O LB_{P1}=w_1, \tag{9}$$

$$LB_{P2}=y_2\wedge R_2\Rightarrow \$O chn!x_2\wedge \$O LB_{P2}=w_2. \tag{10}$$

类似于过程,一个进程 *pros* 可引用的变量集同样划分为内部变量集 $Int_{pros}=\{i_1,\ldots,i_{ni}\}$ 和外部变量集 Ext_{pros}.其中,进程与其他进程通信所使用的通道只能在其参数部分定义.设进程 *pros* 定义的输入通道为 $chin_1,\ldots,chin_m$,输出通道为 $chout_1,\ldots,chout_n$,则进程语义对应于如(11)所示逻辑式 LF_{pros} 的语义模型.

$$LF_{pros}=_{def}\exists_{i1,i2,\ldots,in}:\text{Probody}\wedge\text{WherePart}\wedge$$
$$\Box(\wedge_{i=1,\ldots,m}:(\$O chin_i=chin_i\vee\exists_s:\$O chin_i!s)\wedge\Lambda_{j=1,\ldots,n}:(\$O chout_j=chout_j\vee\exists_t:\$O chout_j?t)). \tag{11}$$

4 并行语句

进程必须成组出现才能体现其并行性.XYZ/E 语言中采用并行语句表示一组进程组成一个并发程序,形式如(12)所示.其中,$ProsInsNmi_j(j=1,...,k)$ 是程序中定义的某个进程(代理机构)通过形如 $\$O prosInsNm=ProsName$ 的实例化语句产生的实例,$pari_j$ 为此进程实例化时传入的实参.进程中声明的通道只有在并行语句中才能够由语法决定此进程实例与哪个进程的实例通过此通道通信.

$$LB=y_i\wedge R\Rightarrow\|[ProsInsNmi_1(Pari_1);\ldots;ProsInstNmi_k(Pari_k)]. \tag{12}$$

进程的转出标号与过程不同,是 STOP 而非 RETURN,但表示的基本含义相同.(12)所示并行语句是如(13)所示一组条件元的缩写.

$$LB=y_i\wedge R\Rightarrow \$O(finps_1)=(ainps_1)\wedge \$O PUSH(RTS_1,y_{i+1})\wedge \$O LB_1=START_1\wedge\ldots\wedge$$
$$\$O(finps_k)=(ainps_k)\wedge \$O PUSH(RTS_k,y_{i+k})\wedge \$O LB_k=START_k$$
$$LB_1=y_{i+1}\Rightarrow \$O(aoutps_1)=(foutps_1)\wedge \$O LB_1=STOP_1;\ldots$$
$$LB_k=y_{i+k}\Rightarrow \$O(aoutps_k)=(foutps_k)\wedge \$O LB_k=STOP_k;$$
$$LB=y\wedge(LB_1=STOP_1\wedge\ldots\wedge LB_k=STOP_k)\Rightarrow \$O LB=STOP. \tag{13}$$

类似于过程调用,并行语句中也采用实参取代进程声明的形参,而各进程实例中的控制标号和引入的通道实参对并行语句所在单元不可见,算作本单元辅助变量.此外,在并行语句执行之前,所有通道对应的缓冲区必须为空.假定 $chn_1,\ldots,chn_m$ 为并行语句中出现的所有通道名称,则(16)所示并行语句的语义对应如(14)所示时序逻辑式的语义模型,其中谓词 *EmptyChannel*(*chn*)表示通道 *chn* 对应缓冲区为空,具体定义这里从略.

$$\exists_{finps_1,\ldots,finps_k,foutps_1,\ldots,foutps_k,LB_1,\ldots,LB_k,chn_1,\ldots,chn_m}:\wedge_{i=1,\ldots,m}:EmptyChannel(chn_i)\wedge$$
$$L\wedge\wedge_{i=1,\ldots,k}:LF_{pros_i}(LB_i/LB)\quad /*L\text{ 为(13)所示条件元组对应的时序逻辑式}*/. \tag{14}$$

5 包块和代理机构

包块是由一组运算(过程)围绕一个数据结构封装而成的模块,对外部可见的只是包块中定义的数据及这些

数据上定义的运算,而运算实现的细节则对外部隐藏,各包块之间只有通过各自可见部分来交换信息.包块与外界的联系从本质而言是静态的,通过编译时将外部信息移入(import)和将自身信息移出(export)的机制实现,因此在可重用性和可靠性方面有较大的优越性.

代理机构由一个数据包块和一个与之匹配的进程组成,数据包块部分实现模块的封装与继承两方面的特性,表示模块静态语义;进程部分表示模块与外界通信过程所实现的动态语义.编译时数据包块中定义的变量和过程被转换为进程中的局部变量和局部过程,因此可在进程中被引用和调用.代理机构的语法如(15)所示,其语义即为代理机构中进程的语义.

Agent::=[*AgentName*==][Package,Process]. (15)

通常,代理机构中进程的动态行为相似,各代理机构的交互通过双方间的互相调用来实现.这种情况下,进程在代理机构可以不出现,从而简化为如(16)所示的形式,这与常见的类(class)形式非常类似.

Agent::=[*AgentName*==][Package]. (16)

代理机构在面向对象程序中通过实例化语句生成具体实例,实例间通过并行语句合作完成特定任务.

6 面向对象程序

XYZ/E 中面向对象程序可以在串型或并发的环境下实现.在并发环境下,还可区分是否为分布式环境.本文主要讨论非分布式并发环境下的面向对象程序,得到的所有结论同样适合于在分布式环境和各进程及代理机构之间不包含共享变量的串型程序.在非分布式并发环境下,面向对象程序形式如(17)所示,进程声明部分的每个进程可看做是一个对应包块内容为空的代理机构.

OOProgram::=%*OOPROG ProgramName*==[][[AgentDeclPart;][ProsDeclPart;]Probody][WherePart]. (17)

面向对象程序的局部变量分为外部变量(可观察变量)和内部变量(不可观察变量)两种.外部变量是对程序外可观察的变量,程序的语义由其可观察变量的计算集合决定;而内部变量是程序为完成特定功能而引入的辅助变量,对外界隐藏.程序的语义由其程序体决定,对程序中声明的代理机构(进程)实例化及并行执行都是通过程序体中的并行语句实现的.设程序 *program* 的内部变量集 $Int_{program}=\{i_1,\ldots,i_n\}$,外部变量集 $Ext_{program}=\{e_1,\ldots,e_m\}$,则(17)所示面向对象程序 program 语义对应如(18)所示时序逻辑式的语义模型,此逻辑式的自由变元集即为程序 program 的外部变量集 $Ext_{program}$.

$LF_{program}=_{def}\exists_{i1,\ldots,in}$:Probody∧WherePart. (18)

上面我们给出了条件元、单元、过程、进程、代理机构和面向对象程序的语法和语义,并且对于这几种语言成分都定义了内、外部变量集.相应地,对于每个条件元 *ce*,也可为其定义内、外部变量集 Int_{ce} 和 Ext_{ce},分别等于空集和条件元所在程序体(过程体、进程体、代理机构进程体)的所有变量的集合.

7 面向对象程序各语言成分求精关系的定义

前面我们给出了面向对象程序及其引用的各种语言成分的语义.本节主要讨论这些语言成分之间的求精关系的定义,并给出一些用于证明它们之间语义一致性的定理.

定义 1. XYZ/E 两个面向对象程序(条件元、过程、进程、代理机构)R_1,R_2 之间是可求精(refinable)的,当且仅当元素 R_1,R_2 之间具有相同的外部变量集,即 $Ext_{R_1}=Ext_{R_2}$.

定义 2. 对于 XYZ/E 面向对象程序中两个元素 R_1,R_2,我们说 R_2 是 R_1 的求精,当且仅当它们之间可求精且 LF_{R_2} 的语义模型是 LF_{R_1} 的语义模型的子集,也即 $LF_{R_2}\rightarrow LF_{R_1}$ 为真.这种求精关系是可传递的.

公式(18)中∃量词后面的内部变量是时序变量,超出一阶时序逻辑的范畴.为了证明两个程序 P_1,P_2 之间的求精关系成立,文献[4]中提出了一种解决方法,设 $Int_{P_1}=\{u_1,\ldots,u_n\}$ 且 $Int_{P_2}=\{v_1,\ldots,v_m\}$,可先找出一个从 Int_{P_2} 到 Int_{P_1} 的求精映射组(refinement mappings):$f=\{f_1,\ldots,f_n\}$,使得对于任意 $i=1,\ldots,n$ 有 $u_i=f_i(v_1,\ldots,v_m)$,然后证明如公式(19)的正确性,关于求精映射组的寻找,请参见文献[5].

$(\text{Probody}_{P_2}\wedge\text{WherePart}_{P_2})\rightarrow(\text{Probody}_{P_1}\wedge\text{WherePart}_{P_1})[f_1/u_1,\ldots,f_n/u_n]$. (19)

在给出面向对象程序及元素的求精关系形式化定义后,我们可以得到如下几个证明 XYZ/E 面向对象程序在不同粒度上进行求精后程序之间的语义一致性的定理.其中,定理 1、定理 2 分别对应于程序在条件元和过程粒度上的求精,而定理 3 和定理 4 对应于程序在代理机构粒度上的求精.

定理 1. 对于 XYZ/E 程序(过程、进程、代理机构)R_1,设 R_2 由将 R_1 的程序体(过程体、进程体、代理机构进程体)中某个条件元(组)ce_1 替换为条件元(组)ce_2 得到,则若 ce_2 是 ce_1 的求精,那么 R_2 是 R_1 的求精.

由于单元内的条件元之间是逻辑交的关系,因此定理 1 的正确性显而易见.

定理 2. 对于 XYZ/E 程序(过程、进程、代理机构)R_1,以及在 R_1 程序(R_1 过程所在包块或程序、R_1 进程所对应包块或程序、R_1 代理机构包含的包块)中声明的过程 $Proc_1$,设 R_2 由将 R_1 中过程 $Proc_1$ 的声明替换为另一个过程 $Proc_2$,$Proc_2$ 和 $Proc_1$ 声明的参数变量相同,且将 R_1 程序体中某个过程调用语句 $LB=y\wedge P\Rightarrow Proc_1(apar|par)$ 替换为由过程调用语句 $LB=y\wedge P\Rightarrow Proc_2(apar|par)$得到,则若过程 $Proc_2$ 是 $Proc_1$ 的求精,那么 R_2 是 R_1 的求精.

过程调用语句与程序体中其他条件元仍为逻辑交的关系,则由过程调用语义对应形式(5)所示时序逻辑式的语义模型即可直接推导出定理 2 成立,具体证明这里从略.

定理 3. 对于多个代理机构(进程)$A_1,\ldots,A_n$ 和 $A'_1,\ldots,A'_n$,如果它们之间满足关系:A'_1 是 A_1 的求精,…,A'_n 是 A_n 的求精,则对于任意实参 $ainps_1,aoutps_1,\ldots,ainps_n,aoutps_n$,有如形式(20)所示的并行语句是形式(21)所示的并行语句的求精.

$$LB=y\wedge R\Rightarrow|||[A_1(ainfps_1,aoutfps_1);\ldots;A_n(ainfps_n,aoutfps_n)], \tag{20}$$

$$LB=y\wedge R\Rightarrow|||[A'_1(ainfps_1,aoutfps_1);\ldots;A'_n(ainfps_n,aoutfps_n)]. \tag{21}$$

由(18)给出的并行语句和其引用的各进程之间的语义关系,可直接推导出定理 3 成立,具体证明这里从略.定理 3 的直接推论就是,如果代理机构 A'保持代理机构 A 的语义,则将 XYZ/E 面向对象程序 P 中代理机构 A 替换为 A'后得到的新程序 P'将保持程序 P 的语义.

前面给出的 3 个定理考虑了将程序中某元素替换成与其具有相同外部变量的精化元素.而对于代理机构,其包块中定义的共享变量将作为包块中声明的过程的外部变量和进程的局部变量.系统开发时由于效率或其他原因,可能需要对此共享变量数据结构进行调整,将相应地改变过程的实现细节.下面的定理 4 即针对这种情况,考虑如何证明代理机构之间的语义一致性,这种具有不同数据结构的代理机构的语义一致性是一种数据求精[6]的概念.此定理中选择了具有形式(16)所示的简化代理机构,是因为这种代理机构结构简单,功能直观,可以将对代理机构间的语义一致性的讨论转化为对两代理机构间各个同名过程的语义关系的讨论.

定理 4. 设代理机构 A,A'具有(16)所示的简化形式,各自包块声明的变量集合为$\{u_1,\ldots,u_n\}$和$\{v_1,\ldots,v_m\}$,过程为$\{Proc_1,\ldots,Proc_k\}$和$\{Proc_1',\ldots,Proc_k'\}$,且对任意 i=1,...,k,过程 $Proc_i$ 和 $Proc_i'$的过程名 $ProcName_i$ 及参数相同.设 I 是在 A'中共享变量上定义的逻辑式,且 A'中任意过程被调用时谓词 I 都满足,也即逻辑式$\Lambda_{i=1,\ldots,k}:(LB=START_{Proc_i'}\rightarrow I)$是代理机构 A'的不变量,则如果存在映射组$f=\{f_1,\ldots,f_n\},f_i:\oplus_{k=1,\ldots,m}:Domain(v_k)|\rightarrow Domain(u_i)$,使得对于任意 i=1,...,k,逻辑式 $I\wedge LF_{Proc_i'}\rightarrow f\bullet LF_{Proc_i}$ 为真.其中,对于逻辑式 L,$f\bullet L=_{def}L(f_1/u_1,\ldots,f_n/u_n)$.那么代理机构 A'是代理机构 A 的求精.

证明:对于(16)所示简化形式的代理机构,设其包块声明了过程$\{Proc_1,\ldots,Proc_k\}$,则其可看做是隐含了一个如(22)所示的进程.

%*PROS ProsName*(%*CHN*/*chin*(*env*,*):*ProcCallType*;%*CHN*/*chout*(*,*env*):*ProcResultType*)==[][

/**ProcCallType* 是一个包含过程调用名称分量 *name* 和过程调用输入实参分量 *ainps* 的记录数据结构;*ProcResultType* 是一个包含过程名称是否错误分量 *isError* 和过程调用输出实参分量 *aoutps* 的记录数据结构.分量 *ainps*,*aoutps* 都可看做是 *Int* 数据类型变量,这是因为一个过程无论定义了多少输入、输出实参以及实参的数据类型,所有实际的输入、输出实参值都可以通过某种编/解码手段分别与一个整数对应*/

%*Var*[*inpValue*:*ProcCallType*; *outpValue*:*ProcResultType*]

%*Alg*[

LB=START⇒$O*outpValue.isError*=$*F*∧$O*LB=LB_ReceiveCall*;

LB=LB_ReceiveCall⇒$O*chin*?*inpValue*∧$O*LB=LB_ProcessCall*;

$LB=LB\_ProcessCall\wedge(inpValue.name=ProcName_1)\Rightarrow Proc_1(inpValue.finps,outpValue.foutps)$;

…

$LB=LB\_ProcessCall\wedge(inpValue.name=ProcName_k)\Rightarrow Proc_k(inpValue.finps,outpValue.foutps)$;

$LB=LB\_ProcessCall\wedge(inpValue.name\neq ProcName_1\wedge\ldots\wedge inpValue.name\neq ProcName_k)\Rightarrow$

$O outpValue.isError=T\wedge O LB=LB\_ReturnResult$;

$LB=LB\_ReturnResult\Rightarrow O chout!outpValue\wedge O LB=LB\_ReceiveCall]]$. (22)

将(22)中各个过程调用语句展开,由$\Lambda_{i=1,\ldots,k}:(LB=START_{Proc_i'}\rightarrow I)$是代理机构 A'的不变量即知,对于任意 $i=1,\ldots,k$,有 $LB=LB\_ProcessCall\rightarrow I$ 也是代理机构 A'的不变量.因此,由代理机构的语义可知,要证明 A'是 A 的求精,只需证明对任意 $i=1,\ldots,k$,都有如(23)所示的逻辑式成立.

$(LB=LB\_ProcessCall\rightarrow Proc_i'(inpValue.finps,outpValue.foupts)\wedge LB=LB\_ProcessCall\rightarrow I)$

$\rightarrow f\bullet(LB=LB\_ProcessCall\rightarrow Proc_i(inpValue.finps,outpValue.foutps))$. (23)

由过程调用语句的语义如(5)所示可知,要证明式(23)成立,只需证明 $I\wedge LF_{Proc_i'}\rightarrow f\bullet LF_{Proc_i}$,而这是前提条件,因此定理 4 正确. □

在定理 4 中,我们为证明两个具有不同数据结构的代理机构的求精关系,引入了一个表示在过程调用时刻都会成立的不变量 I,I 可看做是对代理机构状态的静态约束.而为了证明求精前后代理机构之间的语义一致性,则必须首先证明求精后的代理机构只有在满足这种约束的条件下,才允许调用其声明的过程.

8 结束语

XYZ/E 面向对象程序中引入代理机构表示对象的概念,代理机构由一个表示对象静态语义的数据包块和一个表示动态语义的进程组成.本文在时序逻辑框架下定义了 XYZ/E 面向对象程序及其包含的各种语言成分的语义,并提供了一些定理分别用于证明 XYZ/E 面向对象程序在各个不同粒度上(包括对象级、过程级和条件元级)进行求精后的程序间的语义一致性.

由于 XYZ/E 既是一种规范语言,又是一种设计语言,因此在设计 XYZ/E 面向对象程序时,可以在统一的时序逻辑框架下,从抽取出的程序抽象规范出发,采用逐步求精的方式,经过多步不同粒度上的求精,最终得到程序的可执行代码.而得到的可执行程序满足其抽象规范的正确性,则是建立在证明各个求精步骤的语义一致性的基础上的.

References:

[1] Duke D, Duke R. Towards a semantics for object-Z. Lecture Notes in Computer Science, 1990,428:242~262.

[2] Derric J, Boiten E. Refinement of objects and operations in object-Z. In: Scott FS, Carolyn LT, eds. Formal Methods for Open Object-Based Distributed Systems IV. Kluwer: Academic Publishers, 2000. 257~277.

[3] Tang Zhi-Song. Temporal Logical Programming and Software Engineering. Beijing: Science Press, 1999 (in Chinese).

[4] Lamport L. The temporal logic of actions. ACM Transactions on Programming Languages and Systems, 1994,16(3):872~923.

[5] Abadi M, Lamport L. The existence of refinement mapping. Theoritical Computer Science, 1991,83(2):253~284.

[6] Morgan CC. Auxiliary variables in data refinement. Information Processing Letters, 1988,29(6):293~296.

附中文参考文献:

[3] 唐稚松.时序逻辑程序设计与软件工程.北京:科学出版社,1999.