第 7 卷第 6 期
1996 年 6 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 7, No. 6
Jun. 1996

XYZ 系统在电信领域中的应用

沈武威　　唐稚松

(中国科学院软件研究所　北京　100080)

摘要　最近几年,在软件工程界内部有一种趋势,这就是开发以面向具体领域的 CASE 环境. 作为这样的一种 CASE 工具,XYZ 系统是由一时序逻辑语言 XYZ/E 和一组基于该语言的工具集构成. 在 XYZ 系统中有很多的工具,它们被用来满足不同的需要. 众所周知,SDL(specification and description language)是电信领域中的一个国际标准语言,而且有关基于该语言的环境已在开发,但是有关该语言的验证工作,特别是利用有关时序逻辑语言进行验证的工作还不多. 作为一种尝试,本文将利用 XYZ 系统中的一个子系统 XYZ/VERI,对 SDL 所描述的有关电信领域中的例子进行验证.

关键词　时序逻辑,电信系统,SDL 语言,XYZ 系统,程序验证,CASE 工具,程序转换.

最近几年,在软件工程中有一个趋势,就是开发一些面向具体领域的 CASE 环境. 我们在几年前就因为如下的原因而预测到了这一情况:(1) 一般来讲,一个没有共同的形式语义的集成环境,总是受到缺乏可变性的困扰,因此不能满足不同的需求,特别是满足不同领域的用户的习惯思维方式的需要;(2) 对于一具体的应用领域,只有知识基础和可重用模块才是在现实中有用的.

一般来说,对于面向具体领域的环境,有 2 个因素使设计者必须考虑:(1) 为了适应在这些特殊领域的用户的特殊需要,许多与其领域相关的国际标准语言(例如 SDL,ESTELLE,LOTOS,VHDL 等)均被广泛地采用. 任何面向这些领域的环境必须要考虑使用这些标准语言,否则任何努力都是徒劳的;(2) 作为这些环境中的成员,即 CASE 工具,它们的主要部分都应基于同样的方法. 事实上许多工具可以在许多不同的领域中得到使用,或最多是可以做一些小的修改后使用,为了节省劳动力、时间和投资,这些公用的工具通过一次实现和重用便可以达到满足不同领域需求的目的.

基于上述考虑,整个 XYZ 系统分为 2 层:低层是由基于时序逻辑语言 XYZ/E 的一些普通 CASE 工具组成;在它的上层是一些面向具体领域的环境,作为它们的成员,许多 CASE 工具可由低层调用. 到目前为止,XYZ 面向具体领域的应用已经涉及到了如下领域:物理中的协议、电话系统、硬件描述和设计(VLSI)、进程控制、实时模拟和动画处理等.

---

* 本文研究得到国家自然科学基金和北京邮电大学程控交换机技术与通信网国家重点实验室资助.作者**沈武威**,1967 年生,助研,主要研究领域为语言转换与面向专用领域的软件工具与环境. **唐稚松**,1925 年生,研究员,中国科学院院士,博士导师,主要研究领域为计算机科学与软件工程.

本文通讯联系人:沈武威,北京 100080,中国科学院软件研究所

本文 1995－03－22 收到修改稿

XYZ 系统是由时序逻辑语言 XYZ/E 和一个工具集组成. 该工具集是基于 XYZ/E 语言及相应的 C 语言族而组成的,我们称之为 CC++(Concurrent C+C++). XYZ/E 具有很强的表达力,它不仅能描述在不同层次上的逻辑结构,而且还能表示一般语言的各个方面,例如指针的动态联接问题、CSP 中的通讯进程、并行及选择语句、中断及实时问题以及面向对象中的可重用模块和可视觉程序的特性等. 作为工具集,XYZ 系统包括如下 5 组工具:① 用于结构化设计的图形工具,诸如分布式系统、数据流图、状态转换图和 PAD 图;② 用于对形式描述逐步求精的工具;③ 用于验证和快速原型的工具;④ 用于语言转换的工具,诸如从源语言到 XYZ/E 的转换和从 XYZ/E 到其它语言及 XYZ/E 子语言的转换;⑤ 用于模块管理和系统集成的工具.

将这一方法应用到面向领域中去的一个基本特点就是充分利用了 XYZ 系统中的有关形式转换的特点. 这种转换是 2 个方向的. 因为 XYZ/E 的普通性,因此我们很容易地将 XYZ/E 转换成其它语言,对于其它语言转换到 XYZ/E,我们可以利用 XYZ 系统中的一个很强的工具,称为 XYZ/CCSS,它能形式化地将一个给定语言的程序转换成 XYZ/E 程序. 为了证明这一点,我们已经将诸如 ESTELLE,SDL,VHDL 等一些标准的国际通用语言转换到 XYZ/E 语言(许多其它语言像 LOTOS,ADA 等的转换,我们正在着手进行中). 设 P 是某一个语言中的一个程序. 在转换之后,便得到一个相应的程序 PE,记作 PE=XYZ/CCSS(P),我们可以利用 Plotkin-Liwei 的操作语义来验证 P 和 PE 是等价的. 现在我们作为一个应用,假设用户想知道 P 是否死锁,若要通过 P 的含义来解决这一问题是比较困难的. 众所周知,非死锁性质可以用时序逻辑很方便地表示,设它为 D,上面的问题可以表示如下:"PE⊢D". 因为 P 不是逻辑公式,这种推导不能在逻辑中讨论. 但是我们现在可以做这样的逻辑推导"PE⊢D". 在 XYZ 中有 2 种工具可以来验证这种推导的正确性:一种是验证工具,称为 XYZ/VERI,它是基于 Hoare 逻辑. 另一种我们称之为快速原型工具,即 XYZ/PROT,它是基于扩展的 PARLOG,可用来执行 PE 和 D,并且检测它们的一致性. 下面我们用电话系统中的标准语言 SDL 作为例子看看 XYZ 系统在具体领域中是如何应用的.

1 XYZ/E 简介

时序逻辑语言 XYZ/E 基于 Manna 和 Pnueli 提出的线性逻辑系统. 它既是声明式逻辑系统,又是命令式程序语言. 该逻辑语言的特点之一是,它能够以适当的形式表示一般高级语言的所有性质,同时又能在同一逻辑框架内将不同层次及不同范形的抽象描述表示出来. 为了满足不同的需要,XYZ/E 被分成不同的子语言,例如 XYZ/BE,XYZ/SE 等.

一个 XYZ/E 的程序具有如下形式:

Program::=Mainblock;Package;...;Package.

象其它高级语言一样,Mainblock 包括变量声明、进程或过程声明以及主算法等. Package 是被用来表示可重复使用的模块. Mainblock 可以表示如下:

```
Mainblock::=[Import-Export Declarations;
            System Function Library Declarations;
            Logic Variables Declarations;
            Shared Temporal Variables Declarations;
```

Procedures Declarations;

Processes Declarations;

Production Rules Units Declarations;

Body]

Body::=Unit

每一个单位(Unit)可以用一个子语言来表示,它们以一个符号,诸如%alg,%stm 和%rle开头,其后跟相应语言的控制结构序列,该序列是由";"连接而成的,在 XYZ/E 程序中";"表示逻辑与. 在 XYZ/BE 中条件元 conditional element 是单位(Unit)中的一种基本形式,它具有如下不同形式:

$$LB=y\wedge R\Rightarrow \$\bigcirc(v_1,\ldots,v_r)=(exp_1,\ldots,exp_r)\wedge \$\bigcirc LB=z$$

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

$$LB=y\Rightarrow(Q\wedge LB=y)\,\$U(R\wedge \$\bigcirc LB=z)$$

$$LB=y\wedge R\Rightarrow \$\bigcirc LB_1=\mathrm{START}_{P_1}\wedge\ldots\wedge \$\bigcirc LB_k=\mathrm{START}_{P_k};$$

WHERE ||[$P_1,\ldots,P_k$]

$$LB=y\wedge R\Rightarrow !!\ [C_1|>Q_1,\ldots,C_m|>Q_m]$$

这里 LB 或 $LB_i(i=1,\ldots,k)$表示系统控制变量,y 和 z 分别表示定义标号和转出标号;R,C_i,Q,Q_i 是一阶逻辑公式,R 和 C_i 表示条件,Q 和 Q_i 表示动作;$P_i(1,\ldots,k)$是一阶逻辑公式,它们表示进程实例或过程调用. 这里有 4 种时序逻辑符号:"$\bigcirc$"(Nexttime),"$\diamond$"(Eventually),"$\Box$"(Always),and"$U"(Until)(类似地,"$W"(Unless)),有关含义见文献[1].

从上面的表示中,我们可以得出 XYZ/BE 是基于状态转换这一模型的,然而用这样的形式书写程序和进行程序验证就显得很费力,因此,XYZ 系统又构造了一个新的子语言,即 XYZ/SE(Structured XYZ/E). 在这个子语言中,所有的句子都是结构化的,下面的"loop"语句就是类似高级程序设计语言中的"while"语句:

$*[LB=y\wedge P\Rightarrow(\$\bigcirc LB=w|\$\bigcirc LB=EXIT);$

$LB=w\{|R|\}\$\bigcirc LB=y]$

XYZ/SE 中的"case"语句可以表示如下:

$?[LB=y\wedge P_1\Rightarrow \$\bigcirc LB=z_1$

$|P_2\Rightarrow \$\bigcirc LB=z_2$

$\ldots$

$|\sim\Rightarrow \$\bigcirc LB=z_k;$

$LB=z_1\{|Q_1|\}\$\bigcirc LB=\mathrm{EXIT};$

$LB=z_2\{|Q_2|\}\$\bigcirc LB=\mathrm{EXIT};$

$\ldots;$

$LB=z_k\{|Q_k|\}\$\bigcirc LB=\mathrm{EXIT};]$

这里先来介绍 $X\{|Y|\}Z$ 的含义,设 Y 是具有如下形式的程序段:

$LB=u_1\wedge P_1\Rightarrow Q_1\wedge \$\bigcirc LB=l_1$

$\ldots$

$LB=l_n\wedge P_n\Rightarrow Q_n\wedge \$\bigcirc LB=v_1;$

这里 $LB=u_1$ 和 $\$\bigcirc LB=v_1$ 分别是 Y 的转入与转出标号,X 和 Z 分别具有 $LB=u$ 和 $\$\bigcirc LB=v$ 的形式. $X\{|Y|\}$的含义是用 $LB=u_1$ 替换 Y 中的转入标号 $LB=u$,并且 $\$\bigcirc LB$

$=v_1$ 将替换 Y 的转出标号 $\$\bigcirc LB=v$.

XYZ/SE 中最后一种"条件语句"具有如下 2 种形式:

$LB=y\wedge P\Rightarrow \$\bigcirc(Q_1\wedge LB=\mathrm{NEXT}\,|\,Q_2\wedge LB=\mathrm{NEXT})$

$LB=y\wedge P\Rightarrow \$\bigcirc(Q\wedge LB=\mathrm{NEXT})$

由于 XYZ/SE 的结构与 XYZ/BE 不同,因此需要将 XYZ/BE 转换成 XYZ/SE,该工具已经完成.因此,只要需要就可以将 XYZ/BE 程序转换成 XYZ/SE.下面有关验证的工作都是基于 XYZ/SE 的,第 3 节将给出有关证明规则.

2 SDL 到 XYZ/E 的转换及其实现

这里首先介绍用于从 SDL(specification and description language)到 XYZ/E 转换所用到的工具 XYZ/CCSS.[2]XYZ/CCSS 是被用来实现从一个语言到另一个语言转换的工具,同时它也可以用来实现一个语言的编译.然后本节再给出具体的转换规则.

2.1 XYZ/CCSS 简介

XYZ/CCSS 是 XYZ 系统中的一个子系统,由于它能够描述高级语言的形式语义,因此可被用作实现一个语言的编译,或从一个语言到另一个语言的转换.一般来讲,编译过程可以分成如下 2 个步骤:①是将源语言转换成中间形式,这里有关程序上下文相关部分已经被消除;②是将这种中间形式转换成 XYZ/E 程序.为了突出重点,下面只对第 2 步做一介绍.

第 2 步的转换是由 XYZ/DYNA 实现的,这里有一个源语言 XYZ/ML,用来书写有关转换规则,通过转换规则,XYZ/DYNA 可以生成 XYZ/E 程序,同时源程序的形式语义也就由 XYZ/E 表示出来了.

在 XYZ/ML 中包含了一些特殊的符号,解释如下:

(1) $[|X|]$

该记号表示 X 的语义,可递归地出现在转换规则中.

(2) LAB and $LAB_i(i=1,2,\ldots)$

如果在程序的相应位置中有一个标号,则任何该标号的出现都表示程序的相应位置,否则,XYZ/DYNA 将生成一个新的标号.在同一个语义函数中,所有的相同的元符号都将被赋成相同标号,但是不同语义函数中的相同元符号都被赋成不同的标号.

(3) $LB=y\{|X|\}\,\$\bigcirc LB=z$

此记法的解释如上所述,这里不再叙述.

下面给出一个例子.

wloop →WHILE *exp* DO *stat*

$$
\begin{aligned}
&[|wloop|]\Leftrightarrow\{LB=LAB\wedge[|exp|]\Rightarrow \$\bigcirc LB=LAB_1;\\
&\qquad LB=LAB_1\{[|stat|]\}\,\$\bigcirc LB=LAB;\\
&\qquad LB=LAB\wedge\sim[|exp|]\Rightarrow \$\bigcirc LB=\mathrm{NEXT};\}
\end{aligned}
$$

从上面的 while 语句中可以看出 XYZ/CCSS 在表达语言的形式语义及实现程序语言转换方面具有很大的优势,这里[|*wloop*|]和[|*exp*|]也可以用 YACC 中的[|$$|]和[|$2|]表示.

2.2 用 XYZ/CCSS 实现从 SDL 到 XYZ/E 的转换

介绍 XYZ/CCSS 之后,下面给出 SDL 到 XYZ/E 的转换实现. 这里 SDL 是 '88 年版.[3] 为了节省篇幅,这里只对有关重要规则作介绍,有关转换的内容可见文献[4].

sdlprogram→SYSTEM ID ';' *systembody* ENDSYSTEM ID ';'

[|*sdlprogram*|]⇔{%*PRO*[|*ID*|]==[][[|*systembody*|]]}

上述规则给出了一个有关 SDL 的总体结构转换. 下面给出有关系统内部的转换规则.

systembody→*signaldefinitions* ';' *channeldefinitions* ';' *blockdefinitions*

[|*systembody*|]⇔{[|*blockdefinitions*|]}

对于上述规则的转换,我们忽略了有关信号和通道的转换,这是因为在 SDL 中,通道的概念与 XYZ/E 中所讲的通道概念有很大的不同,SDL 中的通道表示了各个模块之间的一种具体的物理通道,而 XYZ/E 中它却表示了进程之间的一种通信方式. 另外,在 XYZ/E 中也没有有关信号的概念,但是对于 SDL 中的这些概念可以通过使用 XYZ/E 中的通道及有关进程参数等机制加以实现. 所以上述规则忽略了有关 SDL 中的通道与信号的转换,下面一条规则是有关模块层次上的转换.

blockdefinition→BLOCK *nameid*
　　processdefinitions
　　ENDBLOCK ID ';'

[|*blockdefinition*|]⇔{[|*processdefinitions*|]}

上述语义转换函数说明在模块层次上的转换,实际上就是它所包含的所有进程的转换,因为 SDL 只有在进程层次上才规定了实际的动作.

众所周知,SDL 在进程的层次上所规定的动作是基于扩展的有限状态自动机,一般说来进程将在某一个状态等待某一个条件的发生(SDL 称之为"激励",实际上是另一个进程向它发送信号),如果在某个时刻有等待的信号到达,则进程将完成一系列"迁移"动作,这些动作一般可以是赋值语句、输出语句及过程调用语句等. 下面就是有关 SDL 的进程层次上的转换.

processdefinition→PROCESS
　　processid
　　(,$UNSIGNED_{NO}$);
　　formalparameterdefinitions
　　processdeclarations
　　processbodydefinition
　　ENDPROCESS
　　ID;

[|*processdefinition*|]⇔{
　　%*PROS*[|$D|][|*processid*|]
　　([|$G|][|$H|];[|*formalparameterdefinitions*|])==[[
　　%*LOC*[[|*processdeclarations*|];]
　　%*ALG*[
　　[|*processbodydefinition*|];]]]}

在上述的规则转换中,我们使用了元符号 $D 来扩展进程名,因为如前所述,在转换中,XYZ/E 忽略了有关模块层次上的转换,而在 SDL 中允许不同模块层次中的进程取相同的名字,为了在 XYZ/E 中实现上述机制,我们采用了对进程名采取扩展的方法加以实现. 上

面的 $D 代表了进程所在的模块名.因此在 XYZ/E 中所有的进程名就不会相同了,另外,$G 与 $H 分别表示通道的输入与输出参数,下面给出的是有关 SDL 的状态转换部分的规则.

statetransition|−>STATE *Name*
trigger$_1$
transition$_1$
NEXTSTATE *Name*$_1$
...;
trigger$_n$
transition$_n$
NEXTSTATE *Name*$_n$
ENDSTATE *Name*

[|*statetransition*|]⇔{
$LB=[|Name|]\wedge[|trigger_1|]\Rightarrow[|transition_1|]\wedge \$\bigcirc LB=[|Name_1|]$;
$LB=[|Name|]\wedge[|trigger_2|]\Rightarrow[|transition_2|]\wedge \$\bigcirc LB=[|Name_2|]$;
...;
$LB=[|Name|]\wedge \sim([|trigger_1|]\wedge \ldots)\Rightarrow \$\bigcirc LB=[|Name|]$;

在上面的转换规则中,SDL 中的“激励”转换成 XYZ/E 的条件元(Conditional Element)中的条件,而将“迁移”转换成条件元的动作部分,上述规则的最后一条条件元的含义可表述如下:进程将在某一个状态等待某一个“激励”的发生,然后再完成相应的迁移动作,并转入新的状态,但是如果所有的“激励”都没有发生,则进程仍将在该状态中等待,这也就是上述转换的最后一条条件元语句的作用,至此就完成了 SDL 中的基本动作的转换.

assignment|−>TASK
ID ':' '=' *expression* ';'
[|*assignment*|]⇔{ $\bigcirc$[|$D|][|*ID*|]=[|*expression*|]}

这条规则是有关赋值语句转换的,它是 SDL 中的基本“迁移”动作.由于 XYZ/E 已被转换成可执行程序语言 $\mu System$,且在 Sun 4 工作站上实现,因此在完成从 SDL 到 XYZ/E 转换之后,SDL 的程序也就变得可执行了.以下将介绍有关 SDL 的验证部分.

3 利用 XYZ/E 来验证 SDL

XYZ/E 的一个重要的优势就在于它能在同一个框架内表示高层的描述,同时又能表示低层的行为.在完成 SDL 到 XYZ/E 的转换之后,我们就能通过 XYZ/E 来验证 SDL 的程序,有关顺序程序的验证部分已实现.[5]下面将给出的是一些重要的推理规则.

• Succession statement:

$\vdash \{R[e_1/x_1,\ldots,e_n/x_n]\}LB=l_1\Rightarrow \$\bigcirc LB=\mathrm{NEXT}\wedge \$\bigcirc x_1=e_1\wedge\cdots\wedge \$\bigcirc x_n=e_n\{R\}$

• Sequential composition:

$$\frac{\vdash \{R_1\}LB=l_1[|X|]\$\bigcirc LB=l_3\{R_2\}\quad \vdash \{R_2\}LB=l_3[|Y|]\$\bigcirc LB=l_2\{R_3\}}{\vdash \{R_1\}LB=l_1[|X;Y|]\$\bigcirc LB=l_2\{R_3\}}$$

• Loop−block:

$$\frac{\vdash \{INV\wedge P\}LB=w\{|R|\}\$\bigcirc LB=y\{INV\}}{\vdash \{INV\}LB=y\{|X|\}\$\bigcirc LB=\mathrm{EXIT}\{\sim P\wedge INV\}}$$

• Case—block:

$$\frac{\vdash \{PRE \wedge P_1\} LB = x_1 \{|Q_1|\}\ \$\bigcirc LB = \mathrm{EXIT}\{POST\} \quad \cdots \quad \vdash \{PRE \wedge P_k\} LB = x_k \{|Q_k|\}\ \$\bigcirc LB = \mathrm{EXIT}\{POST\}}{\vdash \{PRE\} LB = y \{|X|\}\ \$\bigcirc LB = \mathrm{EXIT}\{POST\}}$$

显然,上面的证明规则是用 Hoare 逻辑的形式出现的,有关这些规则的进一步的讨论可参见文献[6].

下面是一个用 SDL 书写的例子,该程序将所收到的电话费按升序排列.

```
NEWTYPE indexbyint Array(Integer,Integer)
ENDNEWTYPE indexbyint;
...;
PROCEDURE sort;
FPAR IN/OUT n Integer;
FPAR IN/OUT A indexbyint;
DCL i,j,m,t Integer;
START;
TASK i:=1;
a: DECISION i<n+1;
(true): TASK j:=i;
        TASK m:=i;
        b: DECISION j<n+1;
        (true): DECISION A(j)<A(m);
                (true): TASK m:=j;
                        TASK j:=j+1;
                        JOIN b;
                        (false): TASK j:=j+1;
                        JOIN b;
                        ENDDECISION;
                (false): TASK t:=A(i);
                         TASK A(i):=A(m);
                         TASK A(m):=t;
                         TASK i:=i+1;
                         JOIN a;
                ENDDECISION;
(false): STOP;
ENDDECISION;
ENDPROCEDURE sort;
```

下面是转换后的 XYZ/E 程序,这里为了节省篇幅,有关该程序的断言已经加入在程序之中.

```
{~(n<0)}
%PROC sort(%INP/n:INT;%IOP/A:ARRAY)==
%LOC[i,j,m,t:INT]
%STM [
LB=START⇒ $◯i=1∧ $◯LB=a;
  *[LB=a∧(i<n+1)⇒( $◯LB=12| $◯LB=RETURN)
    LB=12⇒ $◯j=i∧ $◯m=i∧ $◯LB=b;
     *[LB=b∧(j<n+1)⇒( $◯LB=15| $◯LB=19)
       ?[LB=15∧(A[j]<A[m])⇒ $◯LB=17|~⇒ $◯LB=18;
       LB=17⇒ $◯m=j∧ $◯LB=16;
       LB=18⇒ $◯LB=16;]
    LB=16⇒ $◯j=j+1∧ $◯LB=b;]{P1∧P2∧ Q}
    LB=19⇒ $◯t=A[i]∧A[i]=A[m]∧ $◯LB=110;
    LB=110⇒ $◯A[m]=t∧ $◯LB=111;
```

$LB=111\Rightarrow \$\bigcirc i=i+1\wedge \$\bigcirc LB=a;]\{P_1\wedge P_2\}$

]

$\{\forall\ p,q((0<p<q<(n+1)\rightarrow A[p]<A[q]))\}$

where

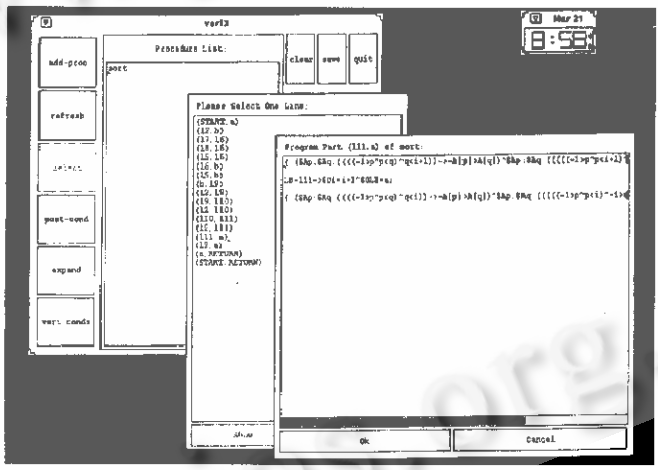
$P_1=\forall\ p,q((1\leqslant p<q<i)\rightarrow(A[p]\leqslant A[q]))$

$P_2=\forall\ p,q((1\leqslant p<i\leqslant q\leqslant n)\rightarrow(A[p]\leqslant A[q]))$

$Q=\forall\ q((1\leqslant q<j)\rightarrow(A[m]\leqslant A[q]))\wedge i\leqslant m<j$

这里程序的前条件是$\sim(n<0)$,它表示需要排序的电话费的个数n是大于零的,程序的后条件是$\{\forall\ p,q((0<p<q<(n+1)\rightarrow A[p]<A[q]))\}$,它表示经过程序的作用之后,所要的电话费是按升序排列的.根据现有的验证工具的要求,每个循环语句必须要有循环不变式.在该程序中有2个循环语句,因此也就需要提供2个所谓程序不变式,它们分别是$\{P_1\wedge P_2\wedge Q\}$和$\{P_1\wedge P_2\}$.

在调用验证工具以及上面所转换成的XYZ/E程序之后,系统将提供一些显示功能,它在Sun工作站中的显示如图中最左名为Veri2窗口所示.系统提供了一些可供用户选择显示程序前后断言的功能,如"select,post—cond,expand"等,这里在按下"select"按扭之后,一个新的窗口如图中间所示,用户可以根据自己的需要来显示相应的条件元的前后断言,这里我们选择语句(111,a),它的前后断言如图中右窗口所示.



然而有些规则的证明是由用户根据程序的性质来提供的,下面2条就是这样的.

- $P_1\wedge P_2\rightarrow\forall\ p,q((1\leqslant p<q<(i+1))\rightarrow(A[p]\leqslant A[q]))$
- $P_2\wedge(\forall\ q((i\leqslant q\leqslant n)\rightarrow(A[i]\leqslant A[q])))\rightarrow\forall\ p,q((1\leqslant p<(i+1)\leqslant n)\rightarrow(A[p]\leqslant[q]))$

对于这2条公式,根据前面的程序可以很容易地得到.现在讨论有关并发程序的验证,为了并行程序也具有前后断言的可分解性(Decomposibility),即有如下规则的成立:

$$\frac{\vdash\{P_1\}\ ProsInstId_1\{Q_1\}\quad\cdots\quad\vdash\{P_n\}\ ProsInstId_n\{Q_n\}}{\vdash\{P_1\wedge\cdots\wedge P_n\}\,||\,[ProsInstId1,\ldots,ProsInstIdk]\{Q_1\wedge\cdots\wedge Q_n\}}$$

我们需要将程序中的输入语句$LB=y\Rightarrow C?x\wedge\$\bigcirc LB=z;$

变换成如下形式:

$LB=y\Rightarrow C?x\wedge\$\bigcirc LB=\mathrm{NEXT};$

$LB=y'\wedge\sim pre(x)\Rightarrow\$\bigcirc LB=y;$

$LB=y' \wedge pre(x) \Rightarrow \$ \bigcirc LB=z;$

这里 $pre(x)$代表输入变量 x 所应满足的前置条件,这一变化保证了进程的后置断言的合理性,这里需要强调的一点是,当对输入变量 x 没有任何约束时,$pre(x)$就变成 True,这时前面所添加的2个条件元就可以被省略了. 有关用此方法对 SDL 进行验证的工作正在进行中.

4　与相关工作的比较

随着CCITT 在电信领域中的推广,其国际化标准语言 SDL 的工作不断深入,有关SDL 的程控软件环境也不断出现,并且已有数个工具商品化,如瑞典的 SDT 等,虽然在这些工具中也有有关验证的工作,例如 SDT 中的验证工具,但是这些工作都不是本文前面所讲述的验证概念(Verification),而是一种有效性证实或确认(Validation),它们都是通过测试或模拟执行来达到验证的目的,因此它们在处理一些大的系统时就会遇到一些所谓"状态爆炸"的问题,然而当使用逻辑语言表达上述问题时就不存在上述问题,本文正是基于此考虑,通过使用由时序逻辑语言 XYZ/E 作为基础的 XYZ 系统来完成有关 SDL 的验证工作,有关基于时序逻辑来完成这方面的工作,据我们了解现在还没有实际的工具出现. 我们将进一步扩充验证工具的能力,以期将 XYZ 系统在电信领域中得到更广泛的应用.

参考文献

1　Tang C S. A temporal logic language oriented toward software engineering—an introduction to XYZ system (I). Chinese Journal of Advanced Software Research, 1994,1(1):1～29.

2　孙淑玲等. XYZ/CCSS 技术报告与使用手册.

3　CCITT. Specification and description language(SDL) recommendations (Blue Book).

4　沈武威. SDL 到 XYZ/E 转换介绍. 技术报告编号,IS—CAS—XYZ—94—3.

5　Zhang Wenhui. Verification of XYZ/SE programs. Chinese Journal of Advanced Software Research. 1995,2(4): 364～373.

6　Xie Hongliang, Gong Jie, Tang C S. A structured temporal logic language: XYZ/SE. Journal of Computer Science and Technology, 1991,6(1):1～10.

APPLICATION OF XYZ SYSTEM IN TELECOMMUNICATION FIELD

Shen Wuwei　Tang Zhisong

(Institute of Software　The Chinese Academy of Sciences　Beijing　100080)

Abstract　In recent years there is a tendency in software engineering to develop the CASE environments oriented toward specific—domains. As such a CASE environment, XYZ system consists of a temporal logic language XYZ/E and a toolkit based on this language as well as its corresponding C family. There are many tools in XYZ system to meet

with the different needs. As well known, SDL(specification and description language) is a standard language in telecommunication field and the tools regarding to this language have been well developed. However the work about the verification of SDL has been little reported. As an attempt, the authors present how to apply XYZ/VERI, which is a subsystem used in program verification, to the telecommunication field in this paper.

Key words Temporal logic, telecommunication, SDL(specification and description language), XYZ, program verification, CASE tools, program transformation.