Verilog 代数语义研究*

李勇坚[1+], 何积丰[2], 孙永强[3]

[1](中国科学院 软件研究所,北京 100080)

[2](澳门联合国大学 国际软件技术研究所,澳门)

[3](上海交通大学 计算机科学与工程系,上海 200030)

Study on the Algebraic Semantics of Verilog

LI Yong-Jian[1+], HE Ji-Feng[2], SUN Yong-Qiang[3]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](International Institute for Software Technology, United Nations University, Macau, China)

[3](Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200030, China)

+ Corresponding author: E-mail: lyj238@ios.ac.cn

http://www.ios.ac.cn; http://www.iist.unu.edu

Received 2001-10-10; Accepted 2002-05-17

Li YJ, He JF, Sun YQ. Study on the algebraic semantics of Verilog. *Journal of Software*, 2003,14(3):317~327.

Abstract: In this paper, the algebraic semantics of Verilog is explored, which is a collection of laws associated with Verilog constructs. These laws provide a precise framework for describing and defining the semantics of Verilog. The special features of the semantics of Verilog are shown. All the laws presented above are sound with respect to the operational semantics, i.e., if the two processes are the two sides of a law, then they are bisimilar. At last, the completeness of the algebraic laws with respect to a subset of Verilog and the operational semantics, i.e., are explored, if such programs are bisimilar, then they are algebraically equivalent. For the proof of completeness, this method will be the discovery of a normal form program for any such programs. Each such program will have an equivalent normal form program (through transformation by the algebraic laws), but two normal form programs will be bisimilar in the operational semantics if and only if they are syntactically equivalent in a simple way. These results are of theoretical significance, for the theories of process algebra are concentrated on the channel-communication concurrent languages. But there is little work on the shared-variable concurrent languages, and a general and effective treatment to the research of such kind of complex concurrent languages is proposed in this paper.

Key words: Verilog; algebraic semantics; soundness; completeness; normal form reduction; event; event triggering

摘 要: 给出了Verilog的代数语义.这是一个等式公理体系,它将Verilog语义特征通过代数规则简洁而准确地

* Supported by the DTfRTS (Design Techniques for Real-Time Hybrid Systems) Project of the International Institute for Software Technology, United Nations University (澳门联合国大学国际软件技术研究所"实时混成系统的研究技术"研究计划)

第一作者简介: 李勇坚(1974－),男,山东青岛人,博士,助理研究员,主要研究领域为程序语义,并行理论.

表达出来;并且这个代数语义相对于已经所作的操作语义模型来讲是可靠的,即所有的这些代数规则左右两边的进程在操作语义的观察模型下都是互模拟的.研究了此代数语义的相对完备性,即参照前面的操作语义模型,相对于扩展 Verilog 语言的一个子集而言,此代数语义是完备的.即所有符合这样语法的程序,如果它们是互模拟等价的,那么它们同样可以在所提出的代数系统中被推导相等.在完备性证明过程中,采用范式方法,即构造一种语法上特殊的程序,任何属于上述子集中的一个程序通过该代数规则都能够被转化为范式程序,而且范式程序在操作语义模型下是互模拟的当且仅当它们是语法相同的.上述结果具有重要的理论意义,因为现有的进程代数理论主要是针对管道通信并行语言而展开的,而对于像 Verilog 这种以共享变量通信为基础的复杂并行语言研究还是比较少的,对此类复杂的基于共享变量的并行语言的进程代数理论研究提出了一种通用、有效的方法.

关键词: Verilog;代数语义;可靠性;完备性;范式归约;事件;事件触发

中图法分类号: TP301 **文献标识码**: A

代数语义是研究并发程序语义的一种重要方法,它的基本思想是:将抽象语法作为多基类代数的型构定义,将程序作为代数表达式,将程序复合构造子作为函数符号,利用程序表达式之间一系列等式或不等式规则给出程序语言的语义.这种代数描述方式同样能为定义与描述语言语义提供一个精确的理论框架,并且有助于语义形式定义文本的规范化与模块化,降低描述的复杂度.代数语义研究中的两个重要问题就是该代数公理体系的可靠性与完备性,即相对于语言的操作语义模型而言:(1) 可靠性指的是等式规则左右两边的进程在具体语义模型下也应该是“等价”的;(2) 完备性指的是所有在具体语义模型中“等价”的进程都可以通过代数语义的规则推导出来.在文献[1]中,我们研究了 Verilog 的操作语义,并在此基础上提出了 Verilog 进程互模拟的概念,证明了互模拟关系的同余性,从而成功地解决了 Verilog 代数语义的可靠性问题.本文的工作是以上工作的继续,它主要包括两方面的内容:(1) 将 Verilog 的代数语义的所有规则(特别是那些 Verilog 语言独有的一些代数规则)找出来,这些规则的探询反映了我们对语言语义精确理解的程度.文献[1]的工作成功地解决了代数规则的可靠性问题(即验证所提出的规则是可靠的),但是没有解决代数的完备性问题(即到底有多少代数规则);(2) 在代数框架下把我们所需要研究的各种程序等价方式表示出来,因为在代数系统中的等号指的是等式左右两边的程序在任意的场合下都等价;而在实际的研究中,我们可能更关心两个程序在一定的环境下等价.这个环境的含义指的是某些变量的初始化条件;如何将两个程序在一定的环境下的相对等价性在我们的代数理论下表示出来也是本文重点讨论的问题.其中本文所要研究的相对等价问题是状态等价性和状态集等价性.实际上,这里所谓的相对等价性在 Verilog 代数语义的完备性研究中也是必不可少的.

本文第 1 节介绍代数语义所研究的 Verilog 程序的语法,注意,此语法与文献[1]的语法有所区别,关键的不同在于对并行操作子使用的限制,之所以要做这样的限制,我们将随后介绍.第 2 节介绍与各类程序构造子相关的所有代数规则.第 3 节参照文献[1]中的操作语义模型对代数语义的可靠性与完备性结果作了深入的分析.第 4 节对本文的工作做了一个小结,特别是强调了 Verilog 的一些本质语义特征以及相应的代数研究方法,并对一些相关的工作进行了比较分析.

1 展平式 Verilog HDL 并行程序的语法

在研究代数语义时,我们将仅仅研究文献[1]所提出的 Verilog 语言的一个子集,这个子集的最大特征就是展平式并行特性.在这个子集中,并行操作子仅仅能被用于最外层.其语法如下:

(1) 进程(顶层模块)

$$P ::= \text{begin } S \text{ end} \mid P \| P$$

(2) 线程

$$S ::= v = e \mid g \mid \text{skip} \mid \text{chaos} \mid \text{stop} \mid S; S \mid \text{if } b \text{ } S \text{ else } S \mid \text{while } b \text{ } S$$

其中 g 为用于事件调度的卫式控制语句,主要有:延时卫式#n(n 为大于 0 的整数)和事件卫式@(η).

$$g ::= @(\eta) \mid \#n \mid \uparrow v \mid \downarrow v \mid @(\eta_1 \vee \eta_2 \ldots \vee \eta_n)$$

从上述语法可以看出,我们在本文所要研究的 Verilog 程序主要分为两个层次:(1) 第 1 个层次我们称为线程,它包括赋值进程等原子语句以及顺序、分支、循环等一般顺序语言的成分,这些成分主要是用于行为级描述.(2) 第 2 个层次被称为进程,它包括两种进程:简单进程,仅仅由一个线程组成的进程,它的语法形式是 begin S end;并行进程,它包括两个或两个以上的线程,并行操作子主要用于描述硬件模块线路互连的结构性说明成分.线程与进程的差别就在于进程的行为肯定是完整的,而线程则不能保证如此.从操作语义的角度来讲[1],这里所谓行为的完整性指的是该程序的所有可能做的行为是否由一系列原子输出动作、事件触发动作、延迟动作这 3 类可观察的行为完整地组成.一个线程程序,如一个赋值语句,它的行为就不是完整的,因为它无法构成一个原子输出动作.在讨论我们的代数语义的完备性时,只讨论行为完整的进程程序,只有做了这样的限制之后,才能得到一个相对的完备性结果;否则,由于讨论的程序对象过于复杂,代数语义的完备性研究很难进行下去.正是由于这个原因,在人们进行基于共享变量的并发程序的代数性质研究时,将并发程序自然地限制为展平式并发程序是一个共同的简化方法,这一点是不同于基于管道通信的并发程序的代数性质研究方法的[2~4].另外一类重要的程序就是卫式选择程序,它们的引入是为了通过顺序与不确定性来表达 Verilog 进程的并行特征,任何一个 Verilog 进程程序都能够被相应的规则展开成卫式选择程序.不过需要注意,我们这里的卫式选择构造子并不像 CSP 等理论,它不是直接作为源程序的一个构造子.我们在此引入它纯粹是为了在代数推理中研究并行操作子的性质,绝不是为了增强语言的表达能力.根据上述 Verilog 程序的分类,我们所要研究的 Verilog 代数规则也将分为 3 部分:

- 有关线程程序的规则,这些规则除了几条以外,大多与通常的顺序语言的代数规则非常类似.
- 有关卫式选择程序的代数规则,这些规则主要描述了 Verilog 进程的可观察行为.
- 有关简单进程以及并行进程的展开规则,这些规则将进程程序转化为卫式选择程序.

另外,我们将本文及以后所用的各种常用符号记法大致介绍一下.P,Q 用于表示 Verilog 程序,G 用于表示卫式选择程序或者卫式选择项表,g,h 用于表示卫式,x,y 表示变量表,e,f 表示表达式表,x_i,y_i 表示变量,e_i,f_i 表示表达式,I,J 表示有限的下标集,$|x|$表示表 x 的长度,b 表示关于变量表 x 的一个布尔合取表达式.记号 $P{=}Q$ 表示程序 P 与 Q 的语法完全相同,即它们为同一字符串;而 $P=_{alg}Q$ 表示两者在代数系统中是等价的,即 $P=_{alg}Q$ 或者为代数系统的基本规则,或者能够通过代数推理(即等式替换与等式代入)能够推导出来的;而 $P\approx Q$ 表示两者在我们的操作语义的观察模型中是观察等价的(互模拟)的.为节省篇幅,本文仅介绍体现 Verilog 特有语义特征的若干规则,其他详细内容可参考文献[5].

2 代数规则

2.1 顺序线程的规则

在 Verilog 语言中,skip 并不是顺序组合子的么元,例如,在我们的代数系统中 skip;@(↓v)与@(↓v)是不等价的.下面,我们首先来定义所谓的事件敏感语句.

定义 1(事件敏感). 进程 P 如果满足下列条件之一,那么将被称为事件敏感的:(1) 它就是事件卫式;(2) 一个卫式选择语句,其中包含一个选择项,该选择项的卫式是事件卫式;(3) 它是一个并发语句,其中一个线程为事件敏感的.

如果 skip 语句放在事件不敏感程序面,那么它是不起任何作用的.

[(seq-3)*] skip;$P=_{alg}P$,其中 P 是事件不敏感的非终止程序. 〈weak-skip-left-zero〉

如果将事件卫式与延迟卫式放在 stop 前,那么它们将不起任何作用.

[(seq-5)*] g;stop$=_{alg}$stop,其中 g 是一延迟或事件卫式. 〈guard-seq-right-zero〉

一个长时间延迟卫式可以拆分为两个较短时间延迟的顺序组合.

[(seq-6)*] #(m+n)$=_{alg}$#m;#n 〈delay-sequence〉

[(IF-3)*] IF (TRUE P)$=_{alg}$skip P〈IF-TRUE-unit〉

注意,在通常的顺序语言中有 IF (TRUE P)$=_{alg}P$,但在我们的理论中,这并不成立.

2.2 卫式选择程序的规则

卫式选择语句表达了一个进程与环境是如何交互的情况.一个卫式选择进程能够提供如下 3 种选择:要么向环境输出事件,要么从环境接受事件,或者与环境一起让时间进行推移.注意,为了以后方便起见,我们在很多时候采用了中缀写法:$G_1[]G_2[]..G_n$,其中 G_i 代表一个选择项;而且在很多时候,我们将$[](g\rightarrow P)$或者$[](g)$直接写为$(g\rightarrow P)$,或者 g.但是正如我们在文献[1]中对卫式选择进程类型的讨论时所说的那样,在语法或者语义上,一个或若干个选择项 G_i 并不是作为一个单独的程序来看待的.我们采用以上写法主要是为了方便起见.另外,我们用大写字母 G,H 来代表一个选择项表,$G[]H$ 表示这样一个卫式选择程序,它的所有选择项由选择项表 G,H 共同提供,即 $G[]H=[](G_1,G_2,...,G_m,H_1,H_2,...,H_n)$,其中 m,n 分别为表 G,H 的长度.由于我们对选择项的处理方法是并不把它当作程序处理,所以这给我们给出某些卫式选择程序(特别是有关选择项替换的)的规则书写带来一定的麻烦.因为若 G,H 分别代表一个选择项表的话,我们不能直接写诸如 $G=_{alg}H$,而要写成 $G[]G'=_{alg}H[]G'$的形式,因为选择项或者选择项表都不是程序.即在列出这些规则时,我们要列出选择项,或者选择项表互相替换的上下文.为了书写方便,我们定义:

$$G=_{alg}{}'H=_{df}G[]G'=_{alg}H[]G',$$

其中 G,H 为选择项表,而 G'为使上述等式左右两边都有意义的任一选择项表.

若一个输出卫式选择项的卫式子进程为事件敏感的进程,那么我们就要考虑该布尔赋值卫式产生的事件是否会触发随后的事件敏感进程.即我们应该有反映这种触发性质的代数规则,但是首当其冲的问题是如何在我们的代数系统下形式化地(或者是通过语法)判别一个输出动作产生的信号是否能真正触发另外一个事件卫式.根据 Verilog 源语言的语法,事件卫式的最基本单元关于一个变量 x_i:$@(\uparrow x_i)$,$@(\downarrow x_i)$,$@(\sim x_i)$.但是,若我们要考虑自触发问题,事件卫式的表示粒度就显得太粗了,它不能真正地反映对应的触发事件所包含的所有变量的变化信息,因为可能触发它的事件信号不只一个.如对两个变量 x_1,x_2 而言,事件 x_1 上升,x_2 不变,与事件 x_1,x_2 都上升是不同的,但它们都能触发事件卫式$@(\uparrow x_1)$.所以问题的本质在于,无论是我们的一般布尔赋值卫式还是事件卫式,它们还没有足够的能力来表达相对一个程序所有变量最细微的变化.为了以后能够讨论触发问题,我们首先引入以下定义:

定义 2(x-基本事件卫式). 若 x 是一个变量表并且我们用$|x|$来表示该表的长度,那么一个 x-基本事件卫式的形式为:与复合事件卫式 $@(\wedge_{1\le i\le|x|}f(x_i))$,其中 $f(x_i)$的形式为$\uparrow x_i,\downarrow x_i$,或者 $x_i\wedge\neg(\sim)x_i,\neg(x_i\wedge\neg(\sim)x_i$,并且至少有一个变量将发生变化,即至少有一个 $f(x_i)$具有以下形式:$f(x_i)=\downarrow x_i$ or $f(x_i)=\uparrow x_i$.

一个 x-基本事件卫式是关于一个变量表 x 的,并且它所对应的触发事件正好完全反映了变量表 x 中每一个变量 x_i 的变化情况.显而易见,若 g 为源语言中定义的一个事件卫式,并且表 x 包含 g 中所有的变量,那么 g 等于若干 x-基本事件卫式的析取,即 $g=@(\vee_{i\in I}g_i)$,其中对于所有的 $i\in I,g_i$ 是一个 x-基本事件卫式.对应于 x-基本事件卫式,我们同时还要定义产生基本事件的基本布尔赋值卫式.

定义 3(x-基本布尔赋值卫式). 若 x 是一个变量表且 e 为一个与变量表 x 具有相同长度的表达式表,并且对所有的 $1\le i\le|e|,e_i$=TRUE 或者 e_i=FALSE.一个 x-基本布尔赋值卫式具有如下形式:$@(b\&x=e)$,其中 b 为变量表 x 的一个布尔合取表达式,即 $b=@(\wedge_{1\le i\le|x|}f(x_i))$,其中 $f(x_i)$的形式为 x_i(肯定出现),或者$\neg x_i$(否定出现).

一个 x-基本布尔赋值卫式$@(b\&x=e)$恰好代表了一个反映变量表 x 最细微变化的事件产生动作,该事件包含了变量表 x 所有变量的变化情况,并且该事件为从状态 b 到状态 $BExp[e/x]$的变化,其中 $BExp[e/x]$为另外一个关于变量表 x 的布尔合取表达式,它的形式为

$$BExp[e/x]=\vee_{1\le i\le|x|}f(x_i),\text{其中}f(x_i)=\begin{cases}x_i,e_i=\text{TRUE}\\ \neg x_i,e_i=\text{FALSE}\end{cases}.$$

下面我们就可以定义 x-基本布尔赋值卫式与 x-基本事件卫式的触发关系了.

定义 4(x-基本布尔赋值卫式与 x-基本事件卫式的触发关系). 对于一个 x-基本事件卫式 η 与一个 x-基本布尔赋值卫式$@(b\&x=e)$,$@(b\&x=e)\vdash\eta$当且仅当对所有的 $1\le i\le|x|,x_i$ 在 b 中的出现为肯定的(即仅仅是变量本身),并且 e_i=FALSE,x_i 在 η 中的出现是$\downarrow x_i$,或者 x_i 在 b 中的出现为否定的(即为$\neg x_i$ 并且 e_i=TRUE,x_i 在 η 中的出现是$\uparrow x_i$,或者 x_i 在 b 中的出现为肯定的,并且 e_i=TRUE,x_i 在 η 中的出现为 $x_i\ \&\neg(\sim)x_i$,或者 x_i 在 b 中的出现为否定的,并且

e_i=FALSE,x_i 在 η中的出现是$\neg x_i$ &$\neg(\sim)x_i$.

下面,我们定义那些不能产生事件信号的 x-基本布尔赋值卫式,以后称为空输出,所谓空输出指的是那些没有导致任何全局变量产生变化的 x-基本布尔赋值卫式.

定义 5(空输出). 对于一个 x-基本布尔赋值卫式@(b&x=e),[b ⊨x=e]当且仅当对所有的 $1\le i\le|x|$,x_i 在 b 中的出现是肯定的,并且 e_i=TRUE,或者 x_i 在 b 中的出现是否定的,并且 e_i=FALSE.

下面的定义给出了一个 x-基本布尔赋值卫式、x-基本事件卫式的前件与后件的定义.

定义 6. 若一个 x-基本布尔赋值卫式@(b&x=e)与一个 x-基本事件卫式满足@(b&x=e) ⊢η,定义:

$$pre(\eta)=_{df} b, post(\eta)=_{df} BExp[e/x]$$

$$pre(@(b\,\&\,x=e))=_{df} b, post(@(b\,\&\,x=e))=_{df} BExp[e/x]$$

$$pre(\#1)=_{df} post(\#1)=_{df} \text{FALSE}$$

以上关于 x-基本布尔赋值卫式与 x-基本事件卫式的引进是非常重要的,这些定义使得我们在代数理论中能形式化地将最基本的事件变化以及触发关系表述出来,这样才能使我们以后能够清晰而严格地讨论有关事件的触发问题.由于我们以后讨论的对象总是针对这样一些卫式选择程序,它们所包含的选择项的卫式是关于某个变量表 x 的 x-基本布尔赋值卫式与 x-基本事件卫式,所以我们首先给出这样的程序的定义.

定义 7. 若一个卫式选择程序(一个卫式选择项表)G 满足下列条件,G=[]$_{i\in I}$ $gseq(g_i,G_i)$,我们则称其为一个 x-卫式选择程序(x-卫式选择项表).其中 g_i 为一个 x-基本布尔赋值卫式或者 x-基本事件卫式或者一个单位延迟范式,并且我们定义:@(b&x=e) ⊢$G=_{df}\exists i\in I$,@(b&x=e) ⊢g_i.

上述定义实际上严格地给出了一个 x-基本布尔赋值卫式所能触发的卫式选择程序的形式,下面我们就能够给出有关自我触发的代数规则了.与一般并发理论不同,Verilog 程序的一个事件输出动作不仅能够影响环境,同时也能影响其自身.一个事件卫式选择程序可能被其前面所执行的事件输出动作所解放.

[(guard-7)*] @(b&x=e)→$G=_{alg}'$@(b&x=e)→(g→G),其中@(b&x=e)是一个 x-基本布尔赋值卫式,g 是一个 x-基本事件卫式,@(b&x=e) ⊢g,并且 G 是事件无关的非终止程序或者 G 是一个包含事件选择项的 x-卫式选择程序,并且@(b&x=e)γ G ⟨self-trigger-1⟩

若一个卫式选择程序的一个 x-基本赋值卫式能够解放其卫式子进程的某一个 x-基本事件卫式选择项,那么该卫式子进程的其余选择项可以被删除掉,因为在输出卫式产生事件后必然要选择其能够触发的那个分支执行下去.

[(guard-8)*] @(b&x=e)→(G[]$gseq$(g,P))=$_{alg}'$@(b&x=e)→$gseq$(g,P),其中@(b&x=e)是一个 x-基本布尔赋值卫式,并且 g 是一个 x-基本事件卫式,@(b&x=e) ⊢g. ⟨self-trigger-2⟩

如前所述,在实际的研究中,我们关心的是两个程序在一定环境条件下是相对等价的.考虑相对等价性问题的出发点在于,我们有时要在忽略一些影响条件或者增加一些影响条件来考虑两个程序之间的等价性.但是在我们的代数理论中,我们只有一个最基本的"=$_{alg}$"用以表示两个程序在一定场合下是等价的,如何将两个程序在一定环境下的相对等价性在我们的代数理论下表示出来是本文重点要讨论的问题.我们用于解决这个问题的基本思想是看两者在一定的上下文能否完全相互替换来表示,这个上下文条件反映的就是我们所谓的环境条件,即我们通过"$C[P]=_{alg}C[Q]$"来表示 P 与 Q 的相对等价性,其中 C[]是给定的上下文.下面我们讨论的是两种相对等价性.

2.3 给定初始数据状态下的程序等价性

因为 Verilog 是基于共享变量的并发特性,所以共享(全局)变量的初始状态,对两个程序的等价性的比较是有很大影响的.有些程序并不是在任意的数据状态下其行为均相等,而仅仅是在一些给定的初始数据状态下才相等.为了在我们的代数理论中表示两个程序在某些给定的初始数据状态下的行为相同,我们引入以下记号:

$$G=^{b}_{alg} H=_{df}@(b\&x=e)\to G=_{alg}'(@(b\&x=e)\to H),\text{其中 } b\ \models[x=e].$$

给定一个初始数据状态 b,G 的一个选择项 $gseq(g_i,G_i)$在状态 b 下可能是无效的,即它可能永远不会输出信号或者被触发,这是因为 g_i 能够作用的前件正好是另外一个事件选择项卫式 g_j 的后件,而且 g_j 在状态 b 是能够

立即被触发的.即 $pre(g_i)=post(g_j)\wedge pre(g_j)=b$.下面的规则说明了如何将一个卫式选择程序的某些无效选择项删除掉.

[(guard-14)*] $G=^{b}_{alg}G'$,其中 $G''=[]_{i\in I-I'}gseq(g_i,G_i)$,$I''\subseteq\{i|\exists j\in I\ pre(g_i)=post(g_j)\wedge pre(g_j)=b\wedge g_j$ 是一个 x-基本事件卫式$\}$ ⟨void-guard-elimini⟩

一个卫式选择项的 g_i 对于状态 b 无效,那么该选择项的卫式子进程 G_i 的形式不影响该程序语义.

[(guard-15)*] $G=^{b}_{alg}G'$,其中 $G'=[]_{i\in I-I'}gseq(g_i,G_i)[][]_{i\in I'}gseq(g_i,\text{chaos})$,其中 $I'\subseteq\{i|\exists j\in I\ pre(g_i)=post(g_j)\wedge pre(g_j)=b\wedge g_j$ 是一个 x-基本事件卫式$\}$ ⟨void-guarded-process⟩

定义 8(事件首部完全). 若 G 为一个 x-事件卫式选择程序(或者 x-事件选择项表),并且 $G=[]_{i\in I}gseq(g_i,G_i)$,如果它满足下列条件,那么 G 是事件首部完全的,该条件是:(1) 存在一个 x 的子表 y 满足 $\vee_{i\in I}g_i=\vee_{1\le j\le|y|}f(y_j)$;(2) 对所有的 $1\le j\le|y|$,$f(y_j)=@(\uparrow y_j)$或者 $f(y_j)=@(\downarrow y_j)$或者 $f(y_j)=@(\sim y_j)$.

并且我们定义:$head(G)=_{df}y$;$headg(G,y_j)=_{df}f(y_j)$,其中 $1\le j\le|y|$.$Part(G,x')=_{df}[]_{i\in I''}gseq(g_i,G_i)$,其中 x' 是另外一个变量表,满足 $x'\subseteq y,I'=\{i|i\in I,g_i\Rightarrow\vee_{1\le k\le|x|}headg(G,x'_k)\}$.

定义 9(相对于某一个状态的立即能触发的事件首部). 若一个首部完全的 x-事件卫式选择程序(或者 x-事件选择项表)$G=[]_{i\in I}gseq(g_i,G_i)$,并且 b 是一个关于变量 x 的布尔合取表达式,如果 G 满足下列条件,那么我们称 G 相对于状态 b 有一个立即能触发的事件首部,该条件是存在一个变量表 y,满足:(1) $head(G)=y$;(2) 对所有的 $1\le j\le|y|$,$b\Rightarrow f(y_j)$,其中 $f(y_j)=@(\uparrow y_j)$或者 $f(y_j)=@(\downarrow y_j)$,其中 $b\Rightarrow f(y_j)=_{df}\exists b',\langle b,b'\rangle\models f(y_j)$.

下面的规则告诉我们,如何将一个事件首部对于状态 b 不能立即触发的卫式选择程序转化成等价的一个针对状态 b 能够立即触发的卫式选择程序.

[(guard-16)*] 若 G 为一个 x-事件选择项表,且 $G=[]_{i\in I}gseq(g_i,G_i)$,$head(G)=y$,并且 G 对于状态 b 而言不是立即能触发的;假如 G'满足下列条件,那么 $G[]H=^{b}_{alg}G'[]H$(其中 H 是一个不包含事件卫式选择项的选择项表),该条件是:(1) G' 为一个 x-事件选择项表,(2) G''对于状态 b 是能够立即触发的,$G'=[]_{j\in J'}gseq(h_j,G_{f(j)})$ $[][]_{j\in J''}(h_j,G[]H)$ $[][]_{j\in J''}(h_j,\text{chaos})$,$head(G')=y$,$f$ 是从集合 J' 到 I 的一个函数,并且满足 $g_{f(j)}=h_j$,$J'=\{j|h_j$ 对于状态 b 是有效的并且 $h_j\Rightarrow G\}$,$J''=\{j|h_j$ 对于状态 b 是有效的并且 $h_j\not\Rightarrow G\}$,$J'''==\{j|h_j$ 对于状态 b 是无效的$\}$ ⟨insignificiant-to-significiant⟩

2.4 在状态集下程序之间语义的等价性

在很多时候,我们还要考虑,在一个状态集中的所有状态下,两个程序的语义是否等价.在对这个问题进行讨论之前,我们不妨先看两个程序:$G'[]gseq(\#1,G)$,$G'[]gseq(\#1,H)$,G'对应于一个事件卫式选择表,而且 G'对于某个状态 b 而言具有立即能够触发的完全首部,那么若给定初始状态 b,单位延迟动作后所能到达的数据状态实际上是受到限制的,即单位延迟动作如果能够得到执行,那么必定是在事件选择项部分还不能触发的前提下得以执行的.所以要比较 $G'[]gseq(\#1,G)$与 $G'[]gseq(\#1,H)$在状态 b 下的语义是否等价,实际上就是要比较 G 和 H 在延迟动作执行后的所有状态下是否会等价.这些状态构成一个特定的状态集.这就是我们要考虑的在一个状态集中的所有状态下两个程序等价的语义背景.

定义 10. 若 b 为一变量表 x 的布尔合取表达式,那么对于 $0<i\le|x|$,我们定义:

$$sproj(b_i,x_i)=\begin{cases}x_i, & b\Rightarrow x_i\\ \neg x_i, & \text{否则}\end{cases}.$$

定义 11. 若变量表 y 是变量表 x 的一个子表,而且 c 为变量表 y 的一个布尔合取表达式,那么我们定义一个状态集:$T_c=_{df}\{b|b$ 是变量表 x 的一个合取表达式,并且 $b\Rightarrow c\}$;若 b 是变量表 x 的一个合取表达式,G 在状态 b 下能被立即触发,并且 $G=[]_{i\in I}gseq(g_i,G_i)$,那么我们定义:

$$pretrigsta(G,b)=_{df}T_c,\text{其中 }head(G)=y,c=\wedge_{0<i\le|y|}sproj(b,y_i);$$

$$postrigsta(G,b)=_{df}T_{\text{TRUE}}\neg pretrigsta(G,b).$$

于是我们就能给出程序在某个状态集的所有状态下,两个程序语义等价性的定义:

$$G=^{T}_{alg}H=_{df}G'[]gseq(\#1,G)=^{b}_{alg}G'[]gseq(\#1,H).$$

其中 G'是在状态 b 下的一个首部完全的立即能触发的 x-事件卫式选择程序,并且 $pretrigsta(G',b)=T_c$.

定义 12. 给定一个状态集 T_c,其中 c 为变量表 y 的一个布尔合取表达式,而变量表 y 是变量表 x 的一个子表,一个 x-事件卫式选择程序(x-事件选择项表)G,若 G 满足下列条件:

(1) $head(G)=x$;

(2) $headg(G,x_i)=\begin{cases}@(\square x_i), x_i \notin y\\ @(f(x_l)), x_l \in y\end{cases}$, 其中 $f(y_i)=\begin{cases}\uparrow y_l, sproj(b,y_i)=\neg y_i\\ \downarrow y_l, sproj(b,y_i)=y_i\end{cases}$,

那么我们称 G 对于状态集 T_c 是全触发的.

若一个 x-卫式选择程序 G 的事件选择项部分构成的表对于状态集 T_c 是全触发的,那么我们也称 G 对于状态集 T_c 是能够全触发的.对于一个给定的状态集 T_c,一个卫式选择程序的某些输出选择项或者事件卫式选择项是无效的,因为它们产生作用或者被触发的前提条件永远无法得到满足;下面的规则说明如何将一个全触发的 x-事件卫式选择程序的无效选择项删除掉.

[(guard-17)*] 若 G 为相对于状态集 T_c 的能全触发的卫式选择程序,那么 $G=_{alg}^{T_c} G'$,其中 $G'=[]_{i\in I-I'} gseq(g_i,G_i), I'\supseteq \{i|pre(g_i)\in T_c \vee pre(g_i)=\text{false}\}$ ⟨set-void- guard-elimini⟩

若卫式选择项 g_i 对于状态 T_c 是无效的,那么该选择项的卫式子进程 G_i 不影响该程序的语义.

[(guard-18)*] $G=_{alg}^{T_c} G'$,其中 $G'=[]_{i\in I'} gseq(g_i,G_i)[][]_{i\in I-I''} gseq(g_i,\text{chaos}), I''' \supseteq \{i|pre(g_i)\in T_c \vee pre(g_i)=\text{false}\}$ ⟨set-void- guarded-process⟩

而下面的规则能够将事件首部对于某状态集非全触发的卫式选择程序转化为对于该状态集全触发的卫式选择程序.

[(guard-19)*] 若 G 为一个 x-事件卫式选择项表,并且 $G=[]_{i\in I} gseq(g_i,G_i)$,那么 $G[]H=_{alg}^{T_c} G'[]H$,其中:(1) H 为一个非事件卫式选择项表;(2) G' 为一个相对于状态 T_c 能够全触发的 x-事件卫式选择项表,$G'=[]_{j\in J'} gseq(h_j,G_{f(j)})$ $[][]_{j\in J''}(h_j,G[]H)[][]_{j\in J'''}(h_j,\text{chaos})$,$head(G')=y$,$f$ 是 J' 到 I 的一个函数,满足 $g_{f(j)}=h_j$.$J'=\{j|h_j$ 对于状态集 T_c 是有效的,并且 $h_j \Rightarrow G\}$,$J''=\{j|h_j$ 对于状态集 T_c 是有效的,并且 $h_j \not\Rightarrow G\}$,$J'''=\{j|h_j$ 对于状态集 T_c 是无效的$\}$ ⟨set-to-alltrig⟩

2.5 进程程序的展开规则

进程是我们在代数语义中讨论的重点,我们的范式归约的目标正是将进程归约为相应的卫式选择范式.进程行为是独立而完整的,即有限进程的行为(从其开始动作直至终止)总可以拆分为先后发生的若干原子动作.这与线程是不同的,线程的语义动作可能就不能构成一个原子动作.以下是简单进程的展开定理.

[(begin-1)] begin chaos end$=_{alg}$ chaos ⟨begin-chaos-end⟩

[(begin-2)] begin stop end$=_{alg}$ stop ⟨begin-stop-end⟩

[(begin-3)] begin g end$=_{alg}$ g ⟨begin-g-end⟩

[(begin-4)] begin $(g \to P)$ end$=_{alg}$ $g\to$begin P end ⟨begin-g-P-end⟩

[(begin-5)] begin $x=e$ end$=_{alg}(@(\text{TRUE}\&x=e))$ ⟨begin-assignment-end⟩

[(begin-6)] begin $x=e;G$ end$=_{alg}(@(\text{TRUE}\&x=e)\to$begin G end) ⟨begin-assignment-G-end⟩

[(begin-7)] begin $\text{IF}_{i\in I}$ b_i P_i end$=_{alg}[]_{i\in I}(gseq(b_i\&x=e'_i,P_i)$,其中

$$e'_i=\begin{cases}x_i, P_i=\text{chaos}\\ e_i, P_i=seq(x=e_i,G)\end{cases},\quad P'_i=\begin{cases}\text{chaos}, P_i=\text{chaos}\\ \text{begin } G \text{ end}, P_i=(x=e_i;G)\\ \varepsilon, P_i=(x=e_i)\end{cases}.$$ ⟨begin-IF-end⟩

下面我们来讨论与并行操作子有关的代数规则.并行操作子满足交换律与结合律.

[(par-1)] $P\|Q=_{alg}Q\|P$ ⟨par-sym⟩

[(par-2)] $(P\|Q)\|R=_{alg}P\|(Q\|R)$ ⟨par-assoc⟩

如果并行操作子的参数为 stop 与进程 P,那么该并行程序的行为等价于与进程 P 与 stop 的顺序组合.

[(par-3)] $P\|\text{stop}=_{alg}seq(P,\text{stop})$ ⟨par-stop⟩

下面的规则是所谓的并行程序的扩展规则,利用它可以将一个并行程序展开为一个卫式选择程序.

[(par-4)] 若 $G=[]_{i\in I}(gseq(g_i,G_i)),H=[]_{j\in J}(gseq(h_j,H_j))$,其中 G,H 分别为 x-卫式选择程序,那么 $G[]H=[]_{k\in K}(r_k,R_k)$,其中元组$\langle r_k,R_k\rangle$可能的形式有以下几种:

(1) $R_k=G_i||H,r_k=g_i=@(b\&x=e)$,或者 $r_k=g_i=@(\eta)$,并且对所有的 $j\in J,@(\eta)\neq h_j$,或者 $r_k=g_i=\#1$,并且对所有的 $j\in J,h_j=@(\eta)$,这里,$@(\eta)$表示一个 x 基本事件卫式.

(2) $R_k=G||H_j$,其中 $r_k=h_j=@(b\&x=e)$,或者 $r_k=h_j=@(\eta)$,并且对所有的 $i\in I,@(\eta)\neq g_i$,或者 $r_k=h_j=\#1$,并且对所有的 $i\in I,g_i=@(\eta)$,这里,$@(\eta)$表示一个 x 基本事件卫式.

(3) $R_k=G_i||H_j$,其中 $r_k=g_i=h_j=@(\eta)$,或者 $r_k=g_i=h_j=\#1$. 〈general expansion laws〉

3 可靠性与完备性

如前所述,我们的代数语义工作的可靠性与完备性是相对于文献[1]的操作语义模型而言的.在该操作语义模型中,我们用互模拟关系来表示程序之间的等价性,所以要验证代数语义的可靠性,实际上就是要证明 $P=_{alg}Q$ 即意味着 $P\approx Q$,即验证代数语义中代数规则左右两边的进程在我们的操作语义模型中是观察等价的,正如以下定理所述:

定理 1. 若 P,Q 分别是 Verilog 程序,并且 $P=_{alg}Q$,那么 $P\approx Q$.

证明:只要验证:对于本文所列的各种基本的代数规则 $P=_{alg}Q,P\approx Q$ 即可,其中若干典型的代数规则的可靠性验证证明可参考文献[5].

相对于可靠性,完备性的证明则要复杂得多.在研究完备性时,我们把程序对象限制在本文展平式源语言语法的有限进程程序之内;只要两个符合这样语法的有限进程程序在操作语义模型中是互模拟的,我们的代数语义就应该有足够的规则能够推导出它们是等价的,即 $P\approx Q$ 就意味着 $P=_{alg}Q$.而又因为在这里 P,Q 都是进程程序,它们是阻挡程序,根据阻挡程序的性质,我们有 $P\approx Q$ 与 $P\approx^{\mathrm{TRUE}}Q$ 是等价的,所以要证 $P\approx^{\mathrm{TRUE}}Q$ 也就意味着 $P=_{alg}Q$,即:

定理 2. 若 P,Q 分别是符合第 1 节的语法 Verilog 的有限进程程序,并且 $P\approx Q$,那么 $P=_{alg}Q$.

限于篇幅,我们不可能在此详细地列出证明步骤,感兴趣的读者可以参见文献[5]中的详细证明过程,它包括了文献[5]的大部分内容,在此我们只能把证明的关键之处讨论一下:在证明完备性命题时,采用了范式方法,即构造一种语法上特殊的程序,任何一般的展平式源语言语法的有限进程程序通过我们的代数规则能够被转化为范式程序,而且语法不同的范式程序在操作语义模型下是不互模拟的.进程范式有两种:一种是状态范式,另一种是状态集范式.一个进程状态(状态集)范式代表了此进程在相应的状态(或者状态集)的典型行为,这种典型性正是通过范式的语法体现出来的,范式构造应该有其很直观的操作背景.范式构造的直观操作背景是:范式用语法的层次体现其可观察行为步骤的先后,并且每层语法都非常“恰当”地规定了程序语义的所谓特征步,这些特征步是:(1) 死锁;(2) 终止;(3) 能够产生一个事件,并且继续下一步的行为;(4) 接受外界环境产生的事件然后被触发(程序本身发生变化),并且继续下一步的行为;(5) 经过单位时间延迟后被触发(程序本身发生变化),并且继续下一步的行为.语法的最外层体现了其语义上所能执行的第 1 步,而语法的内层正好体现了随后所能执行的步骤,并且每层语法都非常“恰当”地规定了程序语义的所谓特征步,从而保证这些语法限制能够将一个范式程序从相应的互模拟等价类中选出来,并使得不同的语法范式程序在相应的状态是不互模拟的或者在相应的状态集下是不互模拟的;范式程序又是递归定义的,范式程序的任何一个卫式子进程还是某种范式,并且该卫式子进程的范式种类是由该选择项的卫式的形式所决定的.

关于范式的归约,首先我们将进程程序转化成相应的 x-full 卫式选择程序.其中,转化过程所用的代数定理由第 2.1 节有关线程程序的规则、第 2.5 节有关简单进程以及并行进程的展开定理所保证.其次,我们继续将步骤 1 的规约结果归约为最后的范式形式,并且本步骤归约所用的代数规则主要是第 2.2 节~第 2.4 节的有关卫式选择程序的代数规则.在步骤 2 的范式归约过程中,我们对卫式选择程序采用了组合式归约方法(compositional reduction),即给定卫式程序 $G=[](gseq(g_1,G_1),gseq(g_2,G_2),...,gseq(g_n,G_n))$,先求出各子程序 $G_1,G_2,\ldots,G_n$ 的相应规范形式 $NG_1,NG_2,...,NG_n$,其中 NG_i 的形式完全由其对应的卫式 g_i 的形式所决定:它们的叶结点总是终止、发散,或者死锁程序;这 3 种程序就是最基本的状态或者状态集范式程序;然后对每一个非叶结点的子程序 G,我们在

保证它的每一个卫式子进程已经被归约为相应的状态或者状态集范式之后,才应用相应的范式归约规则将 G 本身归约为相应的状态范式以及状态集范式.

完备性工作的重要性在于:只有通过完备性研究,我们才能找到足够多的规则来完全地区分什么样的进程是等价的,而什么样的进程是不等价的;如果不能找到足够多的规则,说明我们的代数语义对于操作语义模型而言是不完备的.实际上,本文很多反映 Verilog 特征的代数规则,如自我触发、状态等价、状态集等价等都是在研究完备性时提出来的.另外,范式归约过程不仅为我们证明程序等价性提供了一个有限判定过程,而且能够帮助我们理解各种规则的作用,理顺它们之间的关系.另外,范式归约过程实质上也是程序转化的一个过程(将一般程序转化为语法受限的范式程序),这个过程本身也体现了代数归约方法的一种极为重要的应用方式.实质上,硬件综合、软件编译以及软硬件划分采用的全是范式归约方法.从这个角度来讲,只有在对完备性问题进行了深入的研究之后,我们才能对互模拟的代数性质以及对程序之间的转化有一个非常深入的理解,这也正是代数语义公理化研究方法的基本思想.

4 结 论

本文提出了一个代数公理体系作为定义与描述 Verilog 语义性质的框架,实质上,这种代数语义方法已经成为研究并发程序语言语义的极其重要的方法.这种代数方法所研究的几个基本问题是:(1) 找出定义与描述语言语义性质的代数公理体系,并对该公理体系的形式化演算进行探讨;(2) 为所提出的代数体系找出相应的操作语义或者指称语义模型,这样不仅为所提出的代数体系提供了一个支撑模型(使得研究代数的完备性与可靠性成为可能),而且也将操作(或者指称)语义与代数语义两大语义理论联系起来.其中有一些对我们的工作影响比较大的理论.Hoare,He 等人在文献[6]中提出的用于一个描述顺序语言语义的各种代数规则,该语言包含了非确定性以及递归计算等特征.Roscoe 等人在文献[7]中为 OCCAM 语言提出了相应的代数语义.但是他们在研究代数语义的可靠性与完备性时,都是参照语言的指称语义模型.因为他们所处理的语言相对比较简单,但即使 OCCAM 采用的也是基于管道通信的机制,而变量的使用是严格限制在局部变量上的,而且也没有涉及时间延迟以及零延迟计算等混合特征,所以它们的关系指称语义还不算太复杂,以指称语义为参照在验证代数语义的一致性与完备性时还是比较容易的.在研究 Verilog 的代数语义时,我们最先也想以指称语义为模型来参照,并且也定义了 Verilog 指称语义,但在研究代数语义的一致性时就遇到了很大的困难,其主要原因在文献[1]中已经阐述.从我们已有的工作来看,对于 Verilog 这样非常复杂的语言,在研究其代数性质时,不妨采用以操作语义为参照模型的技术路线,因为互模拟技术是我们在操作语义模型下证明程序等价的基本技术,而利用互模拟技术来证明程序等价的本质实际上很自然地采用了归纳法,从而降低了证明难度.而若利用指称语义模型来证明就不具有这个优点,以文献[2]为例,他们是在一个扩展 DC 语义的框架下给出了 Verilog 的指称语义,一个 Verilog 程序的语义用一个扩展 DC 公式来刻画,两个进程的等价性证明就必须通过它们所对应的 DC 公式的等价性来证明,这种证明已经复杂到使得证明几乎进行不下去了的程度.这是因为 Verilog 是非常复杂的并行语言,一个 Verilog 进程与环境交互过程中的所有观察现象都是非常复杂的,而我们又要把这些所有的观察现象全部用一个 DC 公式描述出来,这样的一个 DC 公式自然是极其复杂的,而且这样的公式之间的等价性证明也是很困难的.所以最后我们是以操作语义模型为参照来研究 Verilog 的代数语义,从这一点上来说,我们的研究方法类似于 CCS 的代数语义研究方法,但是与 CCS 不同的是,我们在定义互模拟概念时,考虑到了发散现象,这实际上是把发散(失败)程序与正常程序的等价性区分开来考虑,我们认为,一个程序的发散就意味着它与外界环境可能永远无法交互;而且对于外界环境而言,所有的发散程序都是等价的,这是因为外界环境永远无法与发散程序交互,所以它也并不能知道该发散程序是在无穷地做着 $x=a$ 的动作,还是在无穷地做着 $y=b$ 的动作.将发散现象考虑进来,使得我们的互模拟概念更加符合实际的程序等价概念.

正如在本文前面所说的那样,由于 Verilog 本身语义的复杂性,大多数语义工作集中在操作语义或者指称语义上面,而有关 Verilog 的代数语义工作几乎还没有开始.应该说,本文的工作是对 Verilog 的代数语义的初步尝试,我们认为以下几点是比较重要的:

(1) 如果一个变量表 x 包含一个程序 P 的所有变量,那么实际上与 x 相关的布尔合取表达式 b 就代表了相

对于该程序而言最细的一个数据状态,这个最细指的是 b 描述了程序所有变量的状态信息;而相应的 x-基本布尔赋值卫式以及 x-基本事件卫式对应的事件则正好对应着最细的状态变化,即它细致地描述了程序所有变量的变化.状态、事件的细致描述手段使得在我们的代数理论中有了描述事件产生、触发的能力,从而保证了以后代数推理的可能,这正是 Verilog 语义基于事件调度及传播的核心语义.

(2) 引入布尔赋值卫式彻底解决了事件产生(信号输出)动作在我们的代数理论中的表示问题.布尔赋值卫式概念中有两点是非常重要的:一是它的原子性,二是它体现了从布尔前件 b 到布尔后件 $BExp[e/x]$的一个事件.在文献[2]中,Zhu 等人已经意识到,为了以后的代数语义工作能够进行下去,需要引入表示事件产生的原子动作,他们使用了赋值卫式的概念.赋值卫式能够表示输出动作的原子性,但却不能体现事件是状态变化的本质.状态变化就意味着从一个原有的状态到新的状态的变化.所以他们所引进的赋值卫式并不能彻底描述事件产生原子动作的本质.

(3) 因为 Verilog 是基于共享变量的并发通信机制,进程的行为不仅与自身的程序有关,而且也依赖于共享(全局)变量的初始状态,即共享(全局)变量的初始状态对两个程序的等价性的比较是有很大影响的.在我们的代数理论中,有些程序并不是在任意的数据状态下其行为均相等,而仅仅是在一些给定的数据状态下才相等.正如在第 2.3 节描述的那样,在一个给定的数据状态下,一个卫式选择程序的事件输出执行的前提条件或事件卫式选择项的触发条件可能是永远无法得到满足的.这实际上从另外一个角度反映了事件是状态变化的本质,正因为如此,我们分别用两个小节专门研究了给定初始数据状态下的程序等价性与给定状态集下的程序等价性.研究这两种等价性的直观背景就是在一个卫式执行完毕后,它所能到达的数据状态或者是一个给定的数据状态(一个 x-基本事件卫式或者一个 x-基本布尔赋值卫式被触发后就只可能达到一个数据状态)或者是一个状态集(一个延迟卫式被触发后就可能达到一个数据状态集).在我们的代数系统中,分别用 $G=_{alg}H, G=^{b}_{alg}H, G=^{T_c}_{alg}H$ 来表达 G 与 H 在所有的数据状态下,在一个的数据状态 b 下,以及在一个给定的状态集 T_c 下语义相等;从代数演算的角度来看,不同的等价方式实际上也限制了 G 与 H 能够互相替换的上下文:若 $G=_{alg}H$,那么 G 与 H 就能在任意上下文的情况下进行互相替换;而 $G=^{b}_{alg}H$ 则说明,G 与 H 在 $G'[](g\rightarrow[])$的上下文下能够互相替换(其中 g 为一个 x-事件基本卫式或者一个空输出卫式,并且其后件为 b,而[]表示 G 或者 H 出现的地方);而 $G=^{T_c}_{alg}H$ 则说明,G 与 H 在$(G'[](g\rightarrow(G_0[](\#1\rightarrow[])$的上下文下能够互相替换(其中 g 为一个 x-基本卫式,并且其后件为 b,而 $pretrigsta(G_0,b)=T_c$).

(4) 另外一个很重要的概念就是程序行为的原子性,之所以强调原子行为,主要在于当我们分析进程与环境的交互时,必须以原子行为为基本单位来分析这些交互行为.所以不像文献[1]中的操作语义,我们在文献[1]中已经很明确地把源语言的研究对象分为进程与线程,只有进程程序的行为才肯定是完整的,从这个意义上说,我们才将范式归约的对象定在进程程序上.这也是我们从本文开始,把源语言的语法定为展平式语法的原因.从我们以后研究完备性的过程来看,如果不把研究对象确定在源语言的有限进程上,完备性是很难取得的.另外,对于那些只有一个线程程序构成的进程,我们使用了一个 begin end 对来保证简单进程行为的完整性.现有的并行理论,如 CCS,CSP 等,其环境与进程之间的交互是通过管道通信来进行的,通信的原子性是很自然的[9,10].

本文与文献[1]的工作是互相补充的,共同展示了一种 Verilog 语言操作语义与代数语义,并将两者联系在一起的研究方法.虽然我们侧重的是理论上的结果,但是其中的一些结果仍然可以为以代数求精为基础的形式化开发方法提供更多的代数规则,具体的一些工作可以参考文献[11,12].

References:

[1] Li YJ, He JF, Sun YQ. Study on the operational semantics of Verilog. Journal of Software, 2002,13(10):2021~2030 (in Chinese with English Abstract).

[2] Zhu HB, He JF. A DC-based semantics for Verilog. In: Feng YL, Notkin D, Gaudel, M-C, eds. Proceedings of the 16th IFIP World Computer Congress 2000: Theory and Practice (ICS2000). Beijing: Publishing House of Electronics Industry, 2000. 421~432.

[3] Hoare CAR, He JF. Unifying Theories of Programming. Prentice Hall, 1998.

[4] Xu QW. A theory of state-based parallel programming [Ph.D. Thesis]. Oxford University Computing Laboratory, 1992.

[5] Li YJ. A study on the formal semantics of Verilog [Ph.D. Thesis]. Shanghai: Shanghai Jiaotong University, 2001 (in Chinese with English Abstract).

[6] Hoare CAR, Hayes IJ, He JF. Laws of programming. Communications of the ACM, 1987,30(8):672~686.
[7] Roscoe AW, Hoare CAR. Laws of Occam programming. Theoretical Computer Science, 1988,60:177~229.
[8] Sampaio A. An Algebraic Approach to Compiler Designer. World Scientific, 1997.
[9] Miler R. Communication and Concurrency. Prentice Hall, 1989.
[10] Hoare CAR. Communicating Sequential Processes. Prentice Hall, 1985.
[11] Iyoda J, He JF. Towards an algebraic synthesis of Verilog. Technical Report 218, Macau: UNU/IIST, 2001.
[12] Iyoda J, He JF. A prolog prototype for the synthesis of Verilog. Technical Report 237, Macau: UNU/IIST, 2001.

附中文参考文献:
[1] 李勇坚,何积丰,孙永强.Verilog 操作语义研究.软件学报,2002,13(10):2021~2030.
[5] 李勇坚.Verilog 语言形式化语义研究[博士学位论文].上海:上海交通大学,2001.

敬 告 作 者

《软件学报》创刊以来,蒙国内外学术界厚爱,收到许多高质量的稿件,其中不少在发表后读者反映良好,认为本刊保持了较高的学术水平.但也有一些稿件因不符合本刊的要求而未能通过审稿.为了帮助广大作者尽快地把他们的优秀研究成果发表在我刊上,特此列举一些审稿过程中经常遇到的问题,请作者投稿时尽量予以避免,以利大作的发表.

1. 读书偶有所得,即匆忙成文,未曾注意该领域或该研究课题国内外近年来的发展情况,不引用和不比较最近文献中的同类结果,有的甚至完全不列参考文献.

2. 做了一个软件系统,详尽描述该系统的各个方面,如像工作报告,但采用的基本上是成熟技术,未与国内外同类系统比较,没有指出该系统在技术上哪几点比别人先进,为什么先进.一般来说,技术上没有创新的软件系统是没有发表价值的.

3. 提出一个新的算法,认为该算法优越,但既未从数学上证明比现有的其他算法好(例如降低复杂性),也没有用实验数据来进行对比,难以令人信服.

4. 提出一个大型软件系统的总体设想,但很粗糙,而且还没有(哪怕是部分的)实现,很难证明该设想是现实的、可行的、先进的.

5. 介绍一个现有的软件开发方法,或一个现有软件产品的结构(非作者本人开发,往往是引进的,或公司产品),甚至某一软件的使用方法.本刊不登载高级科普文章,不支持在论文中引进广告色彩.

6. 提出对软件开发或软件产业的某种观点,泛泛而论,技术含量少.本刊目前暂不开办软件论坛,只发表学术文章,但也欢迎材料丰富,反映现代软件理论或技术发展,并含有作者精辟见解的某一领域的综述文章.

7. 介绍作者做的把软件技术应用于某个领域的工作,但其中软件技术含量太少,甚至微不足道,大部分内容是其他专业领域的技术细节,这类文章宜改投其他专业刊物.

8. 其主要内容已经在其他正式学术刊物上或在正式出版物中发表过的文章,一稿多投的文章,经退稿后未作本质修改换名重投的文章.

本刊热情欢迎国内外科技界对《软件学报》踊跃投稿.为了和大家一起办好本刊,特提出以上各点敬告作者.并且欢迎广大作者和读者对本刊的各个方面,尤其是对论文的质量多多提出批评建议.