同步语言 Lustre 的编译前端的设计与实现

宋宇婷,孙小祥,冉 丹
(南京航空航天大学 计算机科学与技术学院,江苏 南京 211100)

摘 要:同步语言 Lustre 所描述的反应系统通常应用在航空航天、国防建设等领域,对系统的正确性和安全性都要求很高。如果系统在运行时出现了正确性问题,很可能会导致系统崩溃,产生非常严重的后果。系统中的任何一个词法错误或者语法错误都应该受到重视,而且应该被及时纠正。因此,对 Lustre 语言进行正确的编译是十分重要的。传统的 Lustre 语言的编译器都采用 OCaml 语言描述,无法保证所有人员都能够很容易地理解和使用,而且,需要耗费开发人员大量的时间和精力。基于上述问题,提出了一种新型的 Lustre 语言编译器。新型的 Lustre 语言编译器前端主要采用 C++语言进行描述,并对生成的抽象语法树的结构进行重新定义,简化了编译的过程。该编译前端会对一个经典的 Lustre 语言模型进行检测,通过对检测的结果进行分析,验证了该编译前端的可行性。

关键词:同步语言 Lustre;正确性;编译器前端;C++语言;抽象语法树
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2020)02-0033--04
doi:10.3969/j.issn.1673-629X.2020.02.007

Design and Implementation of Compiler Front-end of Synchronous Language Lustre

SONG Yu-ting, SUN Xiao-xiang, RAN Dan
(School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211100, China)

Abstract: The reaction system described by Lustre, a synchronous language, is usually applied in aerospace, national defense construction and other fields, which requires high accuracy and security. If the system runs incorrectly, it will probably lead to system crash, with very serious consequences. Any lexical or grammatical errors in the system should be paid attention to and corrected in time. Therefore, it is important to compile Lustre correctly. The traditional Lustre language compilers are described in OCaml language, which cannot be easily understood and used by all people. Moreover, it takes a lot of time and energy for developers. Based on the above problems, a new Lustre language compiler is proposed. The front-end of this Lustre language compiler is mainly described in C++ language, and the structure of the abstract syntax tree is redefined to simplify the compilation process. The compiler front-end detects a classical Lustre language model. The feasibility of the compiler front-end is illustrated by analyzing the results of the detection.

Key words: synchronous language Lustre; correctness; compiler front-end; C++ language; abstract syntax tree

0 引 言

随着计算机技术的飞速发展,人们的工作、学习和生活各个方面都越来越依赖于计算机技术以及计算机技术的产物,它们已经成为人们工作、学习和生活中不可分割的一部分。近年来,它们的安全性和正确性等方面也受到越来越多的关注。因为,一个小小的错误很可能造成很严重甚至是不可挽回的后果。为了保证系统设计的正确运行,需要一种可靠的编译器,它能够检测到词法语法等方面的错误,并将错误反馈出来,帮助及时修改系统程序,保证系统的正确运行。编译器的构建分为前端和后端两大部分,前端主要是负责分析输入的源代码,对源程序进行词法分析、语法分析和语义分析[1-2]。同时,前端还会将分析的有效信息保存起来,传递给后端。

传统的同步语言 Lustre 的编译器前端大都采用 OCaml 语言编写,OCaml 语言是一种函数式编程语言[3],适用度没有 C++语言[4-5]高,C++语言是开发人员和用户们最易于使用和易于理解的语言之一。新型的 Lustre 语言的编译前端采用 C++语言进行描述,可以准确地编译 Lustre 语言程序,保证 Lustre 语言所描

收稿日期:2019-04-10 **修回日期**:2019-08-12 **网络出版时间**:2019-11-07
基金项目:国家自然科学基金(U1533130)
作者简介:宋宇婷(1993-),女,硕士,研究方向为软件设计。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20191107.0908.012.html

绘的模型的正确运行。这对航空航天、国防建设、核电建设、金融监管等领域具有重要的意义。编译前端所产生的抽象语法树结构的实际应用也很广泛,既可以应用于编译器后端的开发,也可以应用于模型检测、软件验证等工具的开发。目前,国外比较成熟的是 Kind 和 Kind2[6],而国内对同步语言 Lustre 及其编译器的研究比较少,L2C 编译器是比较成熟的[7-8]。

1 系统分析

新型的 Lustre 语言的编译前端,提供了一种新型的检验 Lustre 语言所描述的模型是否正确的方法。如果 Lustre 语言所描述的模型是基本正确的,不存在任何词法和语法的错误,系统会正确地输出 Lustre 语言模型的抽象语法树;否则,系统会给出词法或者语法错误的提示。为了保证系统的正确性和可靠性,系统应该具备如下的性能需求。

(1)词法规则和语法规则的正确描述。系统在对 Lustre 语言模型进行词法分析和语法分析时,应该严格按照 Lustre 语言的说明文档进行解析;否则,很可能造成无法正确解析 Lustre 语言模型,给出错误结果,影响系统的正确运行。

(2)信息处理的准确性。当 Lustre 语言模型存在任何词法和语法错误时,系统能够及时快速地给出反馈信息,将模型的第几行存在词法错误或者语法错误的提示消息发送给用户,保证用户能够及时进行 Lustre 语言模型的更正。

2 系统的设计与实现

2.1 同步语言 Lustre

同步语言 Lustre 是一种常用的数据流语言,其基本的数据对象是流[5]。一个同步语言程序就是一个函数,具有零个或者更多的输入流以及一个或者更多的输出流,一个流就是一个变量值的序列。在同步语言 Lustre 程序中,任何变量和表达式都诠释了一个流,一个流是由给定类型的变量的无穷多序列值和一个时钟组成的[9-10]。流通常用(a,b,c,d,e)表示,括号内的 a,b,c,d,e 表示流在某个时刻点所对应的值。同步语言 Lustre 的时序操作符主要包括如下几种:

(1)pre(previous):求变量或者表达式的前一时刻的序列值。比如,整数型变量 X 的当前值为($x_1,x_2,\cdots,x_n,\cdots$),那么,pre(x)=(nil, $x_1,x_2,\cdots,x_{n-1},\cdots$)。

(2)->(followed by):定义了前一个表达式被后一个表达式赋值之后的初始值。比如, $X=(x_1,x_2,\cdots,x_n,\cdots)$ 和 $Y=(y_1,y_2,\cdots,y_n,\cdots)$ 是相同类型且具有相同时钟的两个表达式,那么 $X->Y$ 是与 X 和 Y 具有相同类型和相同时钟的表达式,且 $X->Y=(x_1,y_2,y_3,\cdots,y_n,\cdots)$。这意味着,除了时钟的第一时刻,$X->Y$ 总是等于 Y。在 Lustre 语言中,时序操作符-> 也可以采用关键字 fby 的形式进行表示,即:$X->Y$ 与 X fby Y 的含义是相同的。

(3)when:作为下一条语句执行的条件,与 C 语言中的 when 类似。如果 X 是一个表达式,B 是一个布尔表达式,而且它们有相同的时钟,那么 X when B 是一个表达式,且它的时钟由 B 决定。但是,要注意的是,X when B 表达式的值是在 B 为 true 时所对应的 B 的序列值。比如,B =(false, true, false, true, false, false, true),$X=(x_1,x_2,\cdots,x_7)$,那么,X when B =($,x_2,$ $,x_4,$ $,$ $,x_7$)。

(4)current:求变量或表达式的当前序列值。基于 when 时序操作符中的例子,假设 Y = current (X when B),那么,Y =(nil, x_2, x_2, x_4, x_4, x_4, x_7),其中 nil 表示空值。

2.2 Lustre 语言的词法分析设计

词法分析的主要任务是从左到右读入源程序的输入字符,然后根据构词规则识别单词符号。实际上就是将源程序翻译为词法单元,并将词法单元作为语法分析的输入[11]。通常情况下,词法单元分为如下的五大类。第一类是关键字词法单元,程序语言中的每个关键字都有一个词法单元,词法单元名就是关键字的大写。比如:Lustre 语言中的 VAR(var)、RETURNS (returns)等词法单元。第二类是运算符词法单元,每个运算符可以有一个独立的词法单元,也可以按照运算符的阶数或者含义进行分类。比如:Lustre 语言中的 FOLLOWBY(->)、PRE(pre)等。第三类是表示所有标识符的词法单元,比如程序中定义的变量名和数组名等。第四类是表示常量的词法单元,比如:INTEGER([1-9]+[0-9]* |0)。第五类是标点符号词法单元,而且,每个标点符号都有一个词法单元。比如:SEMI(;)、COMMA(,)等。除了关键字类的词法单元名,其余类别的词法单元名一般都是相应符号的英文名称的缩写。在 Lustre 语言中,逻辑运算符也是以关键字的形式表示的,所以,这类运算符词法单元要加入到关键字词法单元中。而且,这些词法单元都作为终结符号。Lustre 语言词法单元的部分定义如表 1 所示。

2.3 Lustre 语言的语法分析设计

语法分析的主要任务是将单词序列组合成各类语法短语,如“程序”、“语句”、“表达式”等等,实际上就是将词法单元翻译为抽象语法树。语法分析的目的是判断源程序在语法结构上是否符合正确的语法规则,而源程序的结构通常由上下文无关文法描述[11]。

Lustre 语言的编译前端的语法分析模块主要是通过查找可以与当前记号进行匹配的规则来进行操作。

主要的算法思想为:当语法分析模块读取符号时,每当它读取到的符号无法结束一条 Lustre 语法规则时,它会把这个符号压入一个内部堆栈,形成一个新的状态,这个新的状态能够反映出刚刚读取的字符。如果语法分析模块发现压入的所有字符正好可以组成语法规则的右部时,就将右部符号全部弹出堆栈,并将与语法规则右部对应的语法规则的左部符号压入堆栈,这个过程被称为“规约”。持续上面的进栈和规约过程,直到所有的 Lustre 语言的字符都已经扫描完毕。最终,所有的 Lustre 语言模型都会被规约为“Program”。其中,“Program”代表着所有 Lustre 语言程序的开始符号。

为了避免在词法分析和语法分析时出现二义性冲突,必须将关键字和特殊符号的定义与语法规则的相关定义进行优先处理。本次实验将关键字和特殊符号的定义放在标识符的定义之前进行判定和处理。

表 1　词法单元的定义

词法单元名	含义
RETURNS	关键字 returns
FOLLOWBY	时序运算符“->”
BINRELOP	所有二元比较操作符,包括:大于“>”、小于“<”、大于等于“>=”、小于等于“<=”和不等于“<>”
ID	所有由 a-z 或者 A-Z 作为首字母,并连接 0 个或者多个字母、下划线与数字所组成的字符串
INTEGER	所有正整数(包括数字 0)
SEMI	分号“;”

2.4　抽象语法树的结构设计

抽象语法树类 ASTNode[12-13] 定义了五个属性,一个用来记录该节点的父节点的变量 parent;一个用来记录节点的开始位置的变量 location;一个用来记录该节点在源程序中的位置 startPosition,startPosition 的初始值被设置为-1;一个 bool 类型的用来判断该元素是否是必要的变量 MANDATORY,此变量的初始值被设置为 true(表示当前的元素是必要的,不可缺少的);一个 bool 类型的判断操作的变量 OPTIONAL,这个变量的初始值为 false(表示当前的操作是不可以为空的)。如果一个变量带有 OPTIONAL 属性,就表示这个变量可以为空,否则这个变量的值是不可能为空的。抽象语法树类 ASTNode 还定义了几个方法,主要包括一个设置父节点的函数 setParent(),一个获取父节点的函数 getParent(),一个获取根节点的函数 getRoot(),一个删除当前节点的部分属性的函数 deleteNode()。

函数 getRoot()的算法思想主要是先调用 getParent()函数获得当前节点的父节点,然后判断这个父节点是否为空,如果父节点为空,就返回这个父节点,即:根节点;否则,就继续循环地判断这个父节点的父节点是否为空,直至满足前边的所有判断条件为止。函数 getRoot()的伪代码如图 1 所示。

```
Procedure getRoot( )
begin
定义临时变量 thisNode = this;
  While(true)
调用 getParent( ) 函数获得当前节点的父节点;
    If(父节点的值为空)
返回这个父节点;
将此父节点赋值给临时变量 thisNode;
end;{getRoot}
```

图 1　getRoot()的伪代码

类 ASTNode 中的函数 deleteNode()的算法思想是首先调用 getLocation()函数,获取到节点的开始位置,然后判断这个开始位置是否为空,如果为空,系统就什么都不做。再然后,判断这个开始位置是否是 ChildProperty 类别的,如果属于此类别,就先调用 getParent()函数获得当前节点的父节点,此父节点会调用 setStructuralProperty()函数设置结构属性。最后,判断这个开始位置是否是 ChildListProperty 类别的,如果属于此类别,就先调用 getParent()函数获得当前节点的父节点,此父节点会调用 getStructuralProperty()函数获取结构属性,再将获取到的结构属性赋值给孩子节点列表 childList。删除节点的属性的函数 deleteNode()的伪代码如图 2 所示。

```
Procedure deleteNode( )
Begin
调用 getLocation( ) 函数,获取到节点的开始位置;
  StructuralPropertyDescriptor *p=节点的开始位置;
  If(p= = NULL)
    return;
  If(p->isChildProperty( ))
调用 getParent( ) 函数获得当前节点的父节点;
此父节点会调用 setStructuralProperty( ) 函数设置结构属性;
  If(p->isChildListProperty( ))
调用 getParent( ) 函数获得当前节点的父节点;
此父节点会调用 getStructuralProperty( ) 函数获取结构属性;
    childList =获取到的结构属性;
End;{deleteNode}
```

图 2　deleteNode()的伪代码

3　实验结果展示

同步语言 Lustre 的编译前端系统的实验对象为图 3 所描绘的 Lustre 语言模型,Lustre 语言模型实例如图 3 所示。

图 3 描述的 Lustre 语言模型实例包含了四个 node 函数单元,第一个 node 函数单元名为 compute,主要包含了算术运算符加、减和乘;第二个 node 函数单元名为 E,主要包含了时序操作符->和 pre,以及逻辑运算

符与(and);第三个 node 函数单元名为 call_E,调用了第二个 node 函数单元,主要包含了逻辑运算符非(not);第四个 node 函数单元名为 Switch,主要包含了 if-then-else 表达式。在同步语言 Lustre 中,if-then-else 的使用比较特殊,它们不再代表常见语言 C/C++中的 if-then-else 语句,而是以表达式的形式展现,通常作为赋值表达式的右部。

```
node compute(x,y:int) returns (sum:int);
let
  sum=5*(x-y)+2;
tel
node E (Xin:bool) returns (Yout:bool);
let
  Yout=true->Xin and pre(Xin);
tel
node call_E (Xin:bool) returns (Yout:bool);
let
  Yout=E(not Xin);
tel
node Switch (s,res,init:bool) returns (1:bool);
let
  1=init->if s then true else if res then false else pre(1);
tel
```

图 3 Lustre 语言模型实例

```
computer@computer-PC /cygdrive/c/cygwin64/simple
$ ./pt.exe input2.c
print syntax tree:
Program (1)
  decls_temp (1)
    decls (1)
      user_op_decl (1)
        op_kind (1)
          NODE (1)

        ID :compute
        params (1)
          LP (1)
          var_decls (1)
            var_id (1)
              ID :x
            COMMA_var_id_temp (1)
              COMMA (1)
              var_id (1)
                ID :y
```

图 4 Lustre 语言模型的实验效果

图 4 描述了基于图 3 所描述的 Lustre 语言模型实例的部分实验效果。图 4 中的所有大写字母节点代表了程序中的终结符,都是抽象语法树的叶子节点;其余节点都是进行语法规则推导时需要按照语法规则说明定义的非终结符号。其中,每个孩子节点都会相对于它的父节点缩进两格输出,兄弟节点的输出则是处于同等水平的。比如,图 4 中的“Program(1)”表示 Lustre 语言程序的整体定义是在源程序的第一行开始的,它的孩子节点为“decl_temp(1)”。节点“decl_temp(1)”的孩子节点为“decls(1)”,表示 Lustre 语言程序的声明也是从源程序的第一行开始的。节点“ID:x”和节点“COMMA(1)”就是兄弟节点,表示标识符 x(node 函数单元 compute 中定义的一个变量)和关键字分号(;)是同级别的。其余节点的含义以及节点之间的关系都与这几个节点类似。

4 结束语

新型 Lustre 语言的编译前端的设计与实现准确地编译运行了 Lustre 语言程序,保证了 Lustre 语言程序不存在任何词法和语法的错误,为编译后端和模型检测等方面的研究提供了坚实的基础。经过检测,新型的 Lustre 语言的编译前端可以准确地检测所有 Lustre 语言程序。由于新型的 Lustre 语言的编译前端采用 C++语言进行设计,非常易于相关人员的理解和使用。未来,会有越来越多的计算机学者加入对同步语言 Lustre 进行编译和检测研究的行列,也会出现更加成熟的 Lustre 语言编译工具。

参考文献:

[1] AHO A,SETHI R,ULLMAN J. Compilers:principles, techniques,and tools[M]. Boston,MA,USA:Addison-Wesley Pub. Co. ,2002.

[2] 陈火旺. 程序设计语言编译原理[M]. 长沙:国防工业出版社,2000.

[3] YALLOP J. Practical generic programming in OCaml[C]// Proceedings of the 2007 workshop on workshop on ML 2007. Freiburg,Germany:ACM,2007:83-94.

[4] SHAYKHIAN G A. C++ programming language[J]. IEEE Software,2007,27(2):59-130.

[5] STROUSTRUP B. The C++ programming language[M]. Boston,MA,USA:Addison-Wesley Longman, 1997.

[6] CHAMPION A,MEBSOUT A,STICKSEL C,et al. The kind 2 model checker[M]. Toronto,Ontario,Canada:Computer Aided Verification,2016.

[7] 石 刚,王生原,董 渊,等. 同步数据流语言可信编译器的构造[J]. 软件学报,2014,25(2):341-356.

[8] 尚 书,甘元科,石 刚,等. 可信编译器 L2C 的核心翻译步骤及其设计与实现[J]. 软件学报,2017,28(5):1233-1246.

[9] HALBWACHS N,CASPI P,RAYMOND P,et al. The synchronous dataflow programming language LUSTRE[J]. Proceedings of the IEEE,1991,79(9):1305-1320.

[10] CASPI P,PILAUD D,HALBWACHS N,et al. LUSTRE:a declarative language for real-time programming[C]//Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on principles of programming languages. Munich,West Germany:ACM,1987:178-188.

[11] KORANNE S. Compiler construction[M]//Handbook of open source tools. [s. l.]:Springer,2011.

[12] 张 红. 基于 Clang 的 AST 提取结构体数据库插件的实现[J]. 电脑知识与技术,2017,13(6):19-21.

[13] NEAMTIU I,FOSTER J S,HICKS M. Understanding source code evolution using abstract syntax tree matching[J]. ACM SIGSOFT Software Engineering Notes,2005,30(4):1-5.