·计算机技术与应用·

同步数据流语言 pre 算子在 Coq 中的翻译验证

李春燕[1],赵长名[1*],杨　斐[2],马　权[2],侯荣彬[2]
(1. 成都信息工程大学计算机学院,四川 成都 610225;
2. 中国核动力研究设计院核反应堆系统设计技术重点实验室,四川 成都 610041)

摘　要: 对同步数据流语言的 pre 算子进行了详细的处理,除了将 pre 算子翻译至 fby 算子,还对 pre 算子在第一周期的值根据其输入参数类型的不同做了相应的初始化,解决了 pre 算子第一周期为空的问题。输入参数为整型和布尔型其第一周期初始化为 false,浮点型初始化为浮点零,数组和结构体类型根据其元素类型分别进行不同的初始化。pre 算子的翻译应用场景大多是核电安全级数字化控制系统(SDCS)。为了确保其编译的正确性及安全性,整个翻译过程都进行了严格的形式化验证,且都在辅助定理证明器 Coq 中完成。该翻译及验证方法已在 SDCS 中进行试用,而且能达到预期的翻译效果。

关键词: 同步数据流语言;可信编译器;形式化验证

中图分类号: TU352　　**文献标志码:** A　　**文章编号:** 1673–159X(2023)00 − 0001 − 10

doi:10.12198/j.issn.1673 − 159X.5024

Translation Verification of Synchronous Data Stream Language Pre Operator in Coq

LI Chunyan[1], ZHAO Changming[1*], YANG Fei[2], MA Quan[2], HOU Rongbin[2]
(1. *College of Computer, Chengdu University of Information Technology, Chengdu* 610225 *China*;
2. *Science and Technology on Reactor System Design Technology Laboratory, Nuclear Power Institute of China, Chengdu* 610041 *China*)

Abstract: The pre operator of the synchronous data stream language is processed in detail. In addition to translating the pre operator to the fby operator, the value of the pre operator in the first cycle is initialized according to the type of its input parameters. Solved the problem that the first cycle of the pre operator is empty. The input parameters are integers and booleans whose first cycle is initialized to false, and floating-point types are initialized to floating-point zero. Array and structure types are initialized differently according to their element types. The translation application scenarios of the pre operator are mostly nuclear power safety digital control systems (SDCS). In order to ensure the correctness and safety of its compilation, the

收稿日期:2023 − 08 − 11

基金项目:四川省重点研发计划项目“面向电力领域的规划评审知识库智能构建关键技术研究”(2021YFG0307);四川省科技计划资助(2019ZDZX0001)。

*** 通信作者:**赵长名(1984—)男,副教授,博士,主要研究方向为异构计算和虚拟化技术研究。
ORCID: 0000 − 0003 − 2878 − 4401　　E-mail: zcm84@cuit.edu.cn

引用格式:李春燕,赵长名,杨斐,等. 同步数据流语言 pre 算子在 Coq 中的翻译验证[J]. 西华大学学报(自然科学版),2023,42(X):1 − 10.

LI Chunyan, ZHAO Changming, YANG Fei, et al. Translation Verification of Synchronous Data Stream Language Pre Operator in Coq[J]. Journal of Xihua University(Natural Science Edition), 2023, 42(X): 1 − 10.

entire translation process has undergone strict formal verification, and all of them are in the auxiliary theorem prover Coq completed in. The translation and verification method has been tested in SDCS, and can achieve the expected translation effect.

Keywords: synchronous dataflow language; trusted compiler; formal verification

在核电领域,对同步数据流语言的应用十分频繁,但是由于同步数据流语言有时钟等特性,直接在同步数据流语言中对模型进行操作既不方便也不直观,很多时候需要将其转化为对应的 C 语言[1-2]进行后续操作;因此,将同步数据流语言转化为 C 语言在核电安全级数字化控制系统(SDCS)中使用就尤为重要。

清华大学 L2C 项目就是将同步数据流语言(Lustre 语言[3-4])编译生成 C 语言子集 Clight。针对同步数据流语言转化为 C 语言以及 L2C 项目,2014 年甘元科等[5]提出了同步数据流的可信排序,同年张玲波等[6]完成了同步数据流语言时态消去的可信翻译;2015 年刘洋等[7]提出了同步数据流语言中高阶运算的消去;2019 年杨萍等[8]介绍了同步数据流语言可信编译器的研究进展。L2C 项目大体上完成了同步数据流语言到 C 语言的转换,本文研究的内容也是基于 L2C 项目提出的。图 1 为 L2C 项目大体框架图,本文的研究内容是 Lustre* AST 和 LustreS 两个中间语句之间的 SimplLustreS 部分,完成对 pre 算子的初始化及翻译验证。

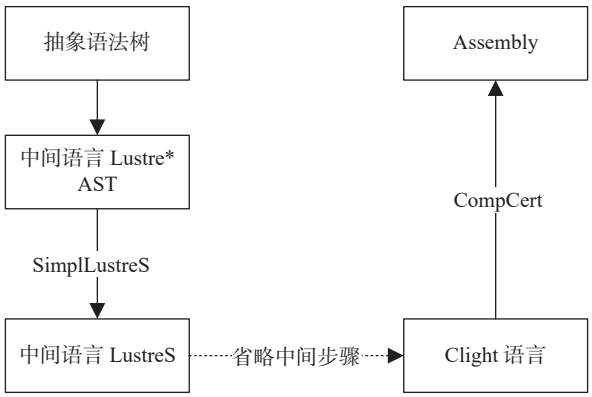


图 1 L2C 项目框架图

Fig. 1 L2C project frame

在工业生产中,同步数据流语言中 pre 算子的使用频率很高,然而在 C 语言中,并没有 pre 算子,因此对 pre 算子的消去工作就显得必不可少。为解决这问题,L2C 项目已经完成了对 fby 算子的消去工作,出于对整个项目的考虑,提出将 pre 算子转化成 fby 算子,从而完成其消去工作。此外,还需要解决 pre 算子在第一周期为空的问题。因此,本文在 L2C 项目的基础上,提出在为 pre 算子赋予初始值之后,将其翻译至 fby 算子的方法,并对整个过程进行了形式化验证[9-11],以确保其正确性。将 pre 算子翻译至 fby 算子,能够更好地将同步数据流语言(Lustre)转化为 C 语言,使 L2C 编译器更加适合在不同环境使用。这样既可以减少代码的复杂性,又能提高代码的可读性和可维护性,提高生产效率。

1 Lustre*语言的相关算子

Lustre*语言是一种流语言,主要应用领域有自动化控制、信号处理系统等[7],其对应的变量、等式等都是一个数据流。一个数据流由 2 部分组成:相同类型的数据值序列和逻辑时钟。在默认情况下,逻辑时钟值为 true,又被称为基本时钟[12]。一个流的值与其对应的时钟周期相关,例如变量的值序列为(1, 2, 3, …),那么在第 1 周期时,变量的值为 1,第 2 周期为 2,以此类推。

Lustre 语言中包含多种算子,不同的算子属于不同的种类。时钟相关算子用于操控时钟周期,如 when、pre 和 arrow(用符号“→”表示)等。状态存储算子主要应用于状态机中,用于操控状态机中的状态,如 current、init 和 restart 等。还有常见的逻辑运算符和比较运算符都属于 Lustre 中的算子类型,如 and、or、not 等。本文的工作主要与 pre 算子和 fby 算子相关。

pre 算子,又名 pre 操作符,是 Lustre*语言中使用频率较高的常见时态算子。pre 算子属于一元运算,它所产生的流比原来的流延迟一个激活时钟周期[13],相当于是其上一个周期的值。它也允许在循环 n 处引用循环 n-1 处流的值。假设表达式 A 的每个周期对应的值序列为(a_1, a_2, …, a_n, …),则表达式 B=pre(A) 的逻辑时钟与表达式 A 相同,当 n>1 时,B 的值为 A 的前一周期的值,当 n=1 时,B 的值或为空 (nil),或为默认值(如果设置了默认

值)。即 B=pre(A)的最后结果为 B=pre(A)=(nil, $a_1, a_2, \cdots, a_{n-1}$)。

fby 算子也是比较常见的一类时态算子,可以分为二元 fby 算子和三元 fby 算子(在项目中定义为 fbyn)。fby 算子是将 2 个表达式(或变量)进行运算得到一个表达式(或变量)。对于二元运算,输入的 2 个表达式(或变量)的逻辑时钟相同,经过 fby 运算后,第一周期的值为第一个表达式在第一周期的值,剩余周期的值为第二个表达式从第一周期开始的值。对于三元运算,要多输入一个常量 n 的值,代表前 n 个周期,其 2 个表达式的還辑时钟也相同,经过 fby 运算后,前 n 个周期的值为第一个表达式前 n 个周期的值,剩余周期的值为第 n+1 个表达式从第一周期开始的值。例如,表达式 A 和 B 每个周期对应的值序列为($a_1, a_2, \cdots, a_n, \cdots$)和($b_1, b_2, \cdots, b_n, \cdots$),那么二元 fby 为(fby A B)=($a_1, b_1, \cdots, b_n, \cdots$),三元 fby 为(fby($A$ n B))=($a_1, a_2, \cdots, a_n, b_1, b_2, \cdots$)。在本文研究中,只需定义初始周期的默认值,如果使用三元 fby 算子,n 的值为 1,性质与二元 fby 算子一样,因此,在翻译过程中直接使用二元 fby 算子,使翻译过程相对简单。表 1 示出在不同周期,经过不同操作后的值序列不同。

表 1　时钟变量及其操作值

Tab. 1　Clock variables and their operating values

Cycles	1	2	3	4	5	6	…
a	a_1	a_2	a_3	a_4	a_5	a_6	…
b	b_1	b_2	b_3	b_4	b_5	b_6	…
pre a	nil	a_1	a_2	a_3	a_4	a_5	…
fby a b	a_1	b_1	b_2	b_3	b_4	b_5	…

2　相关研究

8 在对 Lustre 同步语言程序的形式化验证技术中,比较先进的是用一阶逻辑公式对程序的行为进行建模,然后用基于可满足性模理论的求解技术对一阶逻辑公式的可满足性进行求解[14]。

形式化验证普遍用于源代码和编译器的开发过程。文献 [3] 提出了一种对定时同步数据流图进行形式化验证的方法,将同步数据流图自动转换成 Lustre 代码,将用户定义的属性转换成 Lustre 表达式,并在转换过程中进行形式化验证。一种简单命令式中间语言(Cminor)到 PowerPC 汇编代码的开发[15]、同步数据流语言编译器中 Lustre 到 Clight 翻译过程[16] 及支持浮点(FP)计算的编译器[17-18],都是采用形式化验证来保证其正确性。

针对高安全性应用开发环境(safety-critical application development environment, SCADE)的形式化验证组件 DesignVerifier不支持时态逻辑规范描述,林荣峰等[17] 提出了一种新的形式化验证框架,将 SCADE 中的模型转化为一种新型模型,通过时序性质规范进行模型检测。Leroy 等[15] 对 CompCert(Clight 到 PowerPC 汇编代码的编译器)的开发和形式化验证进行了说明,该编译器也使用的是 Coq 证明助手对编译器的编程及正确性证明。

现阶段,Lustre 转换为 C 语言的编译器中,对于算子的处理工作,只有 L2C 编译器已经完成了高阶算子的消去工作。高阶算子主要包括 map 和 fold 2 类算子,这 2 类算子都是以一个操作或函数调用为输入,作用在数组或参数列表上进行循环操作。基于高阶算子的特点,刘洋等 [7] 使用形式化验证在 Coq 中将其翻译成 C 语言中的 for 循环语句。

3　Lustre*相关语法语义

3.1　时态运算表达式定义

在 L2C 整个项目中,中间的翻译过程有多个层次,相邻 2 个层次之间的语法语义差异不大,差异部分取决于当前翻译阶段的主体目标[7]。一般中间层的翻译是从一层翻译至另一层。与其他中间翻译不同,本文的研究内容全部都基于 LustreS 层。本文将 pre 算子的翻译验证作为一个单独的翻译步骤,因为其证明过程并不简单,若将其融入其他中间层的翻译验证,其翻译与验证过程会与其他层当前翻译验证的主体目标相互交错。并且之后其他中间层的翻译在第一周期必须有值,而 pre 操作符在第 1 周期的值默认值为空,在进行操作时必会出错。

在 LustreS 层,一个程序包括类型定义块、常量定义块、节点定义块以及主节点名定义,其抽象语法如式(1)所示。

```
program: = { { decls} } + { { main_id} }
decls:: = type_block
        |const_block                    (1)
        |user_op_decl
main_id:: = ID
```

在 LustreS 中，节点类型分为函数、节点和外部函数。节点和函数的区别是，在节点(node)中会进行时态的相关操作并且含有时态操作时所需的信息，而在函数(function)中没有时态的相关操作。节点定义的抽象语法如式(2)所示，params 是节点的输入和输出参数列表，节点体或为空，或全部由等式列表(也称为语句)组成。

```
user_op_decl_1:: = op_kind ID
    params returns params opbody
op_kind:: = Node
                |External
opt_body:: = ;                                  (2)
                |[[local_block]]
                let{ { stmt;} } tel[[;]]
user_op_decl_2:: = function imported
    ID params returns params
```

节点中的等式列表在 LustreS 中由 stmt 表示，如式(3)。等式列表由左值和表达式组成，表达式中包含简单表达式、时态运算表达式、数组运算表达式等，如式(4)。时态运算表达式的抽象语法(只给出本文研究的 pre 和 fby 算子)如式(5)所示。

```
stmt:: = lhs = expr
lhs:: = ()                                      (3)
        |ID{ { ,ID} }
expr:: = simple_expr
        |tempo_expr
        |bool_expr
        |array_expr                             (4)
        |struct_expr
        |switch_expr
        |apply_expr
tempo_expr:: = pre simple_expr                  (5)
    |simple_expr fby simple_expr
```

3.2 操作语义

3.2.1 语义环境

Lustre*为同步数据流语言，含有时钟和时态算子，具有时钟特性，因此，其语义环境相较于其他串行语言难度大大提升。Lustre*的语义环境总体上由全局环境(ge)和局部环境(e)组成，全局环境的定义如式(6)所示。

$$\begin{aligned} &ge::=(F,P)\\ &F::(\mathrm{ID}\rightarrow f)\\ &P::=(\mathrm{ID},t)\rightarrow v \end{aligned} \tag{6}$$

全局环境中包含函数/节点映射(F)以及常量映射(P)。函数节点映射可以通过函数或节点的 ID 在全局环境中查找其具体定义。常量映射则通过常量的 ID 和类型(t)在全局环境中查找常量的值。

局部环境的定义比全局环境的定义复杂，其定义如式(7)所示。

$$\begin{aligned} &e::=(\mathrm{te},e*)\\ &\mathrm{te}::=\mathrm{le}*\\ &\mathrm{le}::=(\mathrm{ID},t)\rightarrow v \end{aligned} \tag{7}$$

局部环境由历史时态环境(te)和局部环境的子环境(e*)组成。历史时态环境包含节点环境中局部变量以及输入输出参数的历史值(每个周期的值)，由在当前周期的节点局部环境列表(le*)组成[7]。每个时钟周期的每个节点都会生成一个局部环境(le)[7]，并且能通过 ID 和类型在该局部环境中找到该变量值。对于每个节点的局部环境(e)是一个复杂的树状结构，其子环境又可以看作是新的局部环境。

图 2 为一个简单的节点局部环境的例子，a 节点的局部环境(a_e)由 a 节点的历史时态环境(a_{te})和 a 节点的子环境(a_{e*})组成，a 节点的子环境由 b 节点和 c 节点的局部环境组成，b 节点的局部环境由 b 节点的历史时态环境(b_{te})和其子环境(b_{e*})组成，c 节点局部环境的子环境由 b 节点的局部环境组成。c 节点的局部环境在 a 节点的局部环境中且调用了 b 节点的局部环境。

在 pre 算子翻译成 fby 算子的过程中，由于 pre 运算第 1 周期没有值，需要对其进行初始化生

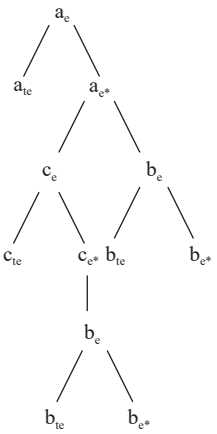


图 2 节点局部环境

Fig. 2 Node local environment

成全局初始化常量,因此,在翻译前后常量全局环境是不同的。初始化的参数属于全局常量,对于节点的局部环境是没有发生改变的。

3.2.2　语义定义

在 LustreS 中,pre 算子的初始化语义规则为:式(8)是 pre 算子的参数是数组或结构体类型的初始化语义规则;式(9)、式(10)分别是 32 位和 64 位的浮点型初始化语义规则;式(11)是整型及布尔型初始化语义规则。

$$\frac{\begin{array}{l}\text{is\_array(ty)} \Rightarrow \text{true}\\ \text{store\_zeros(ty)} \Rightarrow m\\ \text{has\_type}(m) \Rightarrow \text{ty}\end{array}}{\text{eval\_pre ty(Vmvl m)}} \quad (8)$$

$$\text{eval\_pre float32 float32.zero} \quad (9)$$

$$\text{eval\_pre float64 float64.zero} \quad (10)$$

$$\text{eval\_pre int bool bool.false} \quad (11)$$

对于 pre 算子,第一周期及之后周期的执行语义有所不同,在第一周期 pre 算子运算之后的值为对应的初始化值(无初始化值时为 nil),之后周期 pre 运算的值为前一周期对应的值。pre 算子第一周期的执行语义如式(12)所示。之后周期的执行语义的不同之处有 2 点:第一周期标志的值 flag 为 False;执行非初始语句的值。fby 算子在第一周期与之后周期的执行语义也不同,第一周期的值为前一变量(或表达式)第一周期的值,后续周期为后一变量(或表达式)在上一周期的值。fby 算子第一周期的执行语义如式(13)所示,之后周期的执行语义第一周期的标志 flag 为 False,之后周期的值为另一表达式的值。

$$\frac{\begin{array}{l}\text{empty, eh|flag} \Rightarrow \text{True}\\ \text{eval\_pre(typeof}(a1)) \Rightarrow v2\\ \text{typeof}(a1) \Rightarrow \text{typeof(lh)}\\ \text{caststypeof(lh)}v2 \Rightarrow v\\ \text{gc, te |locenv \_setlvar(lh,}v\text{, te, gc)} \Rightarrow \text{ te 1}\end{array}}{\begin{array}{c}\text{gc| te (eh, se), ta te 1, (eh, se), ta}\\ \xrightarrow{[\text{lh=pre}(a1)_{\text{id}}]^{\text{true}}_{\text{ck}}} \text{ te 1, (eh, se), ta}\end{array}} \quad (12)$$

$$\frac{\begin{array}{l}\text{empty, eh|flag} \Rightarrow \text{True}\\ \text{gc, te|}a2 \Rightarrow v2\\ \text{snd(lh)} \Rightarrow \text{typeof(lh)}\\ \text{caststypeof(lh)}v2 \Rightarrow v\\ \text{gc, te|locenv\_setlvar(lh,}v\text{, te, gc)} \Rightarrow \text{te1}\end{array}}{\begin{array}{c}\text{gc|te(eh, se), ta tel, (eh, se), ta}\\ \xrightarrow{[\text{lh=fby}(a2)_{\text{id}}a1]^{\text{true}}_{\text{ck}}} \text{ te 1, (eh, se), ta}\end{array}} \quad (13)$$

4　pre 算子翻译

在进行 pre 算子翻译之前首先要解决 pre 算子第一周期初始化值的问题。pre 算子初始化主要分 2 种情况。1)对于普通类型(整型、浮点型和布尔型等)的 pre 算子,初始化为对应类型的常量表达式:整型及布尔型初始化为 false(Sconst (Cbool false)ty);单精度和双精度浮点型分别初始化为单精度(Sconst (Csingle Float32.zero) ty)和双精度浮点零(Sconst (Cfloat.zero)ty)。2)对于数组和结构体类型的 pre 算子,初始化为全局数组或结构体常量声明(Svar constid ty)。初始化数组和结构体类型的 pre 算子之前需要获取节点的所有语句中数组或结构体类型 pre 语句(输入参数是数组或结构体的 pre 语句)的类型,根据获取的类型列表,生成全局常量 id 和类型列表。根据新生成的全局常量列表获取常量 ID 和其类型(find_type id ty tys),生成全局数组或结构体常量声明。初始化操作为:将 pre 算子初始化翻译至 fby 算子,fby 算子第一周期的值为初始化值,之后周期的值为 pre 算子输入参数在当前周期的上一周期的值。具体翻译为

```
|Spre lh id flag al ⇒
do a2 <- gen_pre_exp (typeof al) tys;
OK (Sfby lh id flag al a2)
```

pre 算子和 fby 算子都属于时态操作符,在翻译之后会为数组或结构体类型的 pre 语句添加时钟。翻译前后程序的类型块、节点定义和主节点没有发生变化,翻译后的常量块在翻译前常量块的基础上增加了初始化 pre 算子的全局常量列表。

5　pre 算子翻译证明

在 L2C 整个项目的翻译过程中,为确保整个程序的正确性,在源语言到目标语言的各个阶段都要用形式化的方法来证明其语义等价关系。

在证明语义等价的过程中,若存在明显不能证明的抽象定理时,先考虑语义定义上出错,若能确定语义定义的正确性,再考虑语法层的定义是否有漏洞,因此在证明过程中只需对语义进行证明即可。

证明语义是否等价的标准,就是在翻译前后程序的输入值相同的情况下,其输出结果也是一致的,这样在程序进行节点调用或节点进行嵌套调用

后结果是一致的,从而保证翻译过程是正确的。

5.1 pre 算子翻译的证明框架

在以上语义定义中可以看出翻译前后的语义对于第一周期和第二周期及其以后周期的处理有明显差别,因此,对翻译前后程序的证明需要分周期来讨论。第一周期需要对程序进行初始化,之后周期就是对算子进行循环的相同操作,对于整个证明框架大体上分为 2 个部分。对于语义的定义采用的是一种层次结构[7],因此,对程序的证明也是采用分层归纳的方法。程序由函数/节点组成,节点由语句组成,语句由表达式组成,因此证明过程从最底层开始逐层证明。证明框架如图 3 所示。

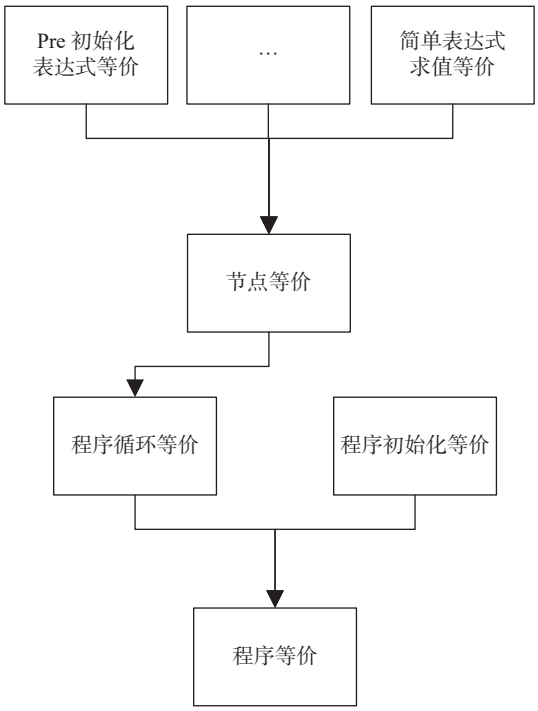


图 3 证明框架图

Fig. 3 Proof Frame

5.2 pre 算子翻译的语义等价性证明

5.2.1 pre 算子初始化执行等价

在开始证明之前,需要定义全局常量环境与类型列表匹配,包含 3 条性质:类型列表中的 ID 和类型都在类型列表中;对于确定类型的 pre 算子进行初始化对应的内存段为 m;在全局常量环境中 ID 对应的内存段也为 m 并且其类型大小与 m 的长度相同。

pre 算子的初始化值是根据 pre 算子输入参数的类型生成的,在生成初始值后根据其类型生成初始化表达式。程序中不重复的所有数组结构体类型 pre 的类型列表是在 pre 算子初始化之后新生成的,在程序的历史时态环境中并无对应值。翻译之后生成的全局常量环境和类型定义列表匹配,而且类型定义列表中所有类型的大小都大于 0,并且小于等于列表中的最大符号数。在翻译后生成的全局常量环境和历史时态环境中,新生成的初始化表达式的值就是 pre 算子初始化的值,具体语义等价的形式为

$$\frac{\begin{array}{l}\text{eval\_pre ty } v\\ \text{gen\_pre\_exp ty tys = OK } a\\ \text{ptree\_ids\_none(map fst tys) te}\\ \text{pre\_types\_match tys gc2}\\ \text{pre\_types sizeof tys}\end{array}}{\text{eval\_sexp gc2 te } a\ v} \tag{14}$$

证明时对生成的表达式的初始值根据定义的语义进行分类讨论,共分为 4 种情况:生成的初始值为数组或结构体、单精度和双精度的浮点型以及整型。对于单精度和双精度浮点型以及整型的初始值证明完全一样,将已知条件代入需要证明的结果中进行化简,再利用辅助证明工具 Coq 中自带的 constructor 策略找到当前环境下的归纳定义中可用于解决当前目标的构造子并进行使用,并对证明结果进行化简,从而保证其正确性。具体证明为

```
simpl in H.inv H.
constructor;simpl;auto.
```

对于初始值为数组或结构体类型的证明,相对于常数而言更为复杂,也分情况(结构体类型和数组类型)进行证明。初始值为数组类型时,应用 Compcert 公共定义文件中的 monadInv 宏对条件进行处理,值为 i 的数组类型在类型定义列表中查找得到的值为 x,类型为 $x0$。依次引入 2 个子目标(相当于新增的条件):类型 $x0$ 为数组类型,并将所有 $x0$ 替换为数组类型(assert($x0$=Tarray i ty z));值为 x 的数组在类型定义列表中(assert(A:In(x, Tarray i ty z)tys))。利用 constructor 策略和 auto 策略搜寻当前环境中能证明全局常量环境和 id 类型列表匹配的条件并证明。将目标表达式按地址求值进行讨论,按地址求表达式的值需要 3 个步骤证明。第 1 步,求历史环境中表达式的地址、偏移量以及变量类型,在当前环境下,存在对应的内存段和数组类型,利用已知条件可证明值 x 的内存段

为 *m*,类型为数组类型且数组类型的大小和 *m* 的长度相同以及读取数组的值操作正确。第 2 步,在历史环境中变量类型 *k* 的值为 *v*。第 3 步,变量类型 *k* 的值 *v* 与表达式的类型相同。步骤 2 和步骤 3 都利用当前环境中的条件进行逐步证明,为:

destruct H4 with *x* (Tarray *i* ty *z*) *m* as [??];auto.

constructor;auto.

apply eval_Rlvalue with *x* Int.zero Gid.

apply eval_Rlvalue with *x* Int.zero Gid;auto.

*constructor 2 with *m*;auto.

apply H3. apply in_map with (f:=fst) in *A*;auto.

*constructor 1.

exists *m*,(Tarray *i* ty *z*) .repeat split;auto.constructor;auto.

simpl typeof. rewrite H6. apply loadbytes_full; auto.

rewrite <- H6. apply H5 with *x*;auto.

rewrite Int.unsigned_zero. simpl. exists0. omega.

初始值为结构体类型的证明思路与数组类型大同小异,只是在假设初始值和类型的时候有所不同。

5.2.2　节点执行等价

节点执行等价的语义(分界线以上是证明条件,分界线以下是待证明目标)如下:

$$\frac{\begin{array}{l}\text{trans\_node (pre\_typeof\_prog prog1)}\\ \qquad\text{node1 = OK node2}\\ \text{locenv\_match gc1 gc2}\\ \text{pre types\_match}\\ \text{(pre\_type\_of\_prog prog1) gc2}\end{array}}{\begin{array}{c}\text{eval\_node prog2 gc2 } e\ e'\\ \text{node2 vargs vrets}\end{array}} \tag{15}$$

程序翻译之后执行节点得到的局部环境和输出值,在翻译前后全局常量环境匹配的情况(在翻译前环境有值,那么在翻译后的环境中有相同值)下,与翻译之前执行节点得到的局部环境输出值相同。证明节点执行等价就是分别证明组成节点的语句等价,在本文中主要证明在第一周期和之后周期的 pre 算子和 fby 算子的执行语义等价,主要通过证明翻译前后节点的局部环境相同来确定。pre 算子第一周期和之后周期的证明过程大同小异,以第一周期的证明为例,用 Compcert 公共定义文件中的 monaInv 宏对条件进行处理,对 pre 第一周期执行语义进行证明,其每条执行语义都能用当前环境中的条件进行证明。fby 算子第一周期与之后周期的证明过程也大同小异,从已知条件中推出新的条件覆盖原始条件,然后在当前环境中对其进行替换,对 fby 第一周期执行语义进行证明,其每条执行语义也能用当前环境中的条件进行证明。

5.2.3　程序执行等价

程序执行等价就是证明翻译前后程序的输入参数、输出参数以及输入输出数据流相等。具体的证明过程是依次证明程序初始化等价和程序循环执行等价。其等价语义为:

$$\frac{\begin{array}{l}\text{Lenv.initial\_state prog1 gc}\\ \qquad\text{init\_node main1 } e\\ \text{exec\_prog prog1 gc eval\_node}\\ \text{main1 } e1\ \text{maxn vass vrss}\end{array}}{\begin{array}{l}\text{exists main2 gc2, Lenv.initial\_state}\\ \text{prog2 gc2 init\_node main2 } e\\ \text{nd\_args(snd main1) =}\\ \qquad\text{nd\_args(snd main2)}\\ \text{nd\_rets(snd main1)}\\ \text{exec\_prog prog2 gc2 eval\_node}\\ \qquad\text{main2 } e1\ \text{maxn vass vrss}\end{array}} \tag{16}$$

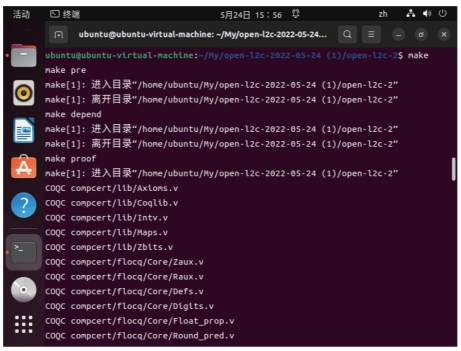
程序循环执行等价就是在翻译前后全局常量环境匹配以及翻译后的全局常量环境与翻译前的类型定义列表匹配的情况下,节点输入数据流和输出数据流相等,也就是证明翻译前后所有节点执行等价,在之前已经证明,只需将其要用到证明过程中即可。

程序初始化等价需要证明的结果有以下几条:在翻译后程序的主节点通过节点翻译函数对翻译前程序的主节点进行处理得到;翻译前后的全局常量环境匹配;翻译前的类型定义列表与翻译之后的全局常量环境相匹配。翻译后程序的主节点通过节点翻译函数得到,因此,证明其正确性只需证明节点翻译函数的正确性,节点函数的证明只需通过该环境中的已知条件和 Coq 函数库对函数的每条语句进行逐一验证。

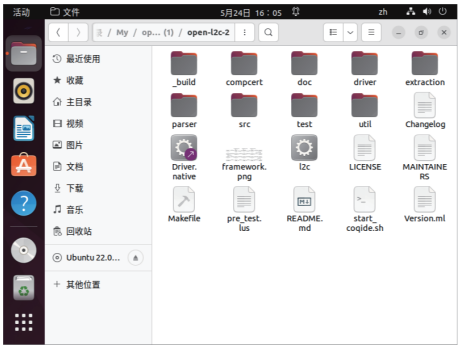
6　实验结果

本文是基于 L2C 项目进行的研究,因此,在使

用前需要先对源码进行编译,生成可执行的 l2c 文件,如图 4 所示。



(a) 项目编译



(b) 生成执行文件 12c

图 4 项目编译结果图

Fig. 4 Project compilation result graph

生成执行文件后再对需要处理的文件进行翻译,该实验的文件为 pre_test.lus。使用-save-temp 命令对文件进行处理后生成翻译的中间文件,其中后缀为 lust 的文件为 pre 算子翻译后的文件,如图 5 所示。

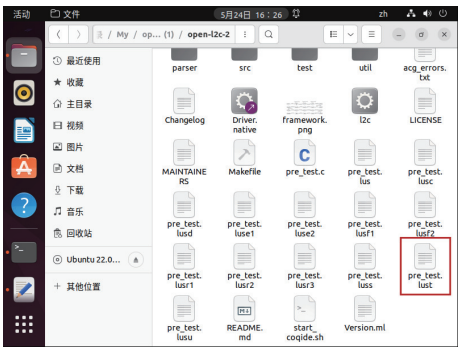
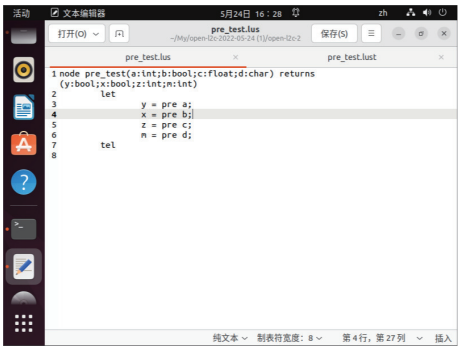


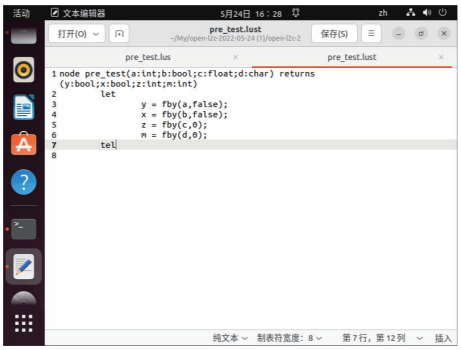
图 5 生成文件结果图

Fig. 5 Generate file result graph

基于 pre 算子和 fby 算子的特点,在整个 Lustre* 到 C 语言的翻译过程中,将 pre 算子翻译成 fby 算子,解决 pre 算子在第一周期值为空的问题。输入参数为除数组和结构体外的其他类型的翻译前后结果如图 6、表 2 所示。若输入参数为数组或结构体类型,根据数组或结构体类型中的数据类型,生成其对应的第一周期初始值,例如,输入为整型的数组 a[3]=[1,2,3],经过 pre 运算的初始化操作后,其对应的第一周期 pre 运算值为 false。



(a) 翻译前



(b) 翻译后

图 6 翻译结果图

Fig. 6 Translation result chart

表 2 基本类型翻译结果对比

Tab. 2 Comparison of basic types of translation results

比较项	a类型			
	整型	布尔型	浮点型	其他类型
翻译前	pre(a)	pre(a)	pre(a)	pre(a)
翻译后	fby(a, false)	fby(a, false)	fby(a, 0)	fby(a, 0)

7 总结

本文进行的翻译验证等过程都是在定理证明器 Coq 中实现的。在 Coq 中定义了翻译前后程序的语法及操作语义,在翻译完成之后从表达式开始以此对语句、节点、程序进行语义等价性证明,即在相同输入下翻译前后程序的输出结果相同,以此来保证翻译前后程序是可信的。本文所做的工作主要有 3 方面。

1)对 pre 算子第一周期根据类型的不同赋予对应的初始值。输入参数为整型和布尔型其第一周

期初始化为 false, 浮点型初始化为浮点零, 数组和结构体类型根据其元素类型分别进行不同的初始化。

2)将赋予初始值之后的 pre 算子翻译成 fby 算子。fby 算子与 pre 算子有一定的相似性, 将 pre 算子转化成 fby 算子, 项目后期就只需对 fby 算子进行处理, 从而完成对 pre 算子的消去。

3)对翻译前后的程序进行形式化验证以确保翻译程序的可信性。在核电等领域对程序的安全性有很高的要求, 对程序进行形式化验证后能在很大程度上确保程序的正确性。

在整个过程中, 相关语法语义的定义大约有 800 行, 翻译工作在 300 行左右, 程序的等价性证明约 1 100 行, 且本文的研究内容已在核电领域的系统中进行试用并取得预期效果。

参　考　文　献

[1] 章永明. C 语言下的计算机软件编程设计探讨[J]. 电子世界, 2021(17): 41 − 42.

ZHANG Y M. Discussion on computer software programming design in C language[J]. Electronics World, 2021(17): 41 − 42.

[2] 朱香卫, 张建. C 语言项目式教程 [M]. 北京: 北京理工大学出版社, 2021.

ZHU X W, ZHANG J. Project-based course of C language[M]. Beijing: Beijing Insititute of Technology Press, 2021.

[3] BOURKE T, JEANMAIRE P, PESIN B, et al. Verified Lustre Normalization with Node Subsampling[J]. ACM Transactions on Embedded Computing Systems (TECS), 2021, 20(5): 1 − 25.

[4] 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. New York: ACM, 1987: 178 –188.

[5] 甘元科, 张玲波, 石刚, 等. 同步数据流程序的可信排序[J]. 计算机应用与软件, 2014, 31(5): 1 − 5.

GAN Y K, ZHANG L B, SHI G, et al. Credible sorting for synchronous data-flow programs[J]. Computer Applications and Software, 2014, 31(5): 1 − 5.

[6] 张玲波, 甘元科, 石刚, 等. 同步数据流语言时态消去的可信翻译[J]. 计算机工程与设计, 2014, 35(1): 137 − 143.

ZHANG L B, GAN Y K, SHI G, et al. Certified translation for eliminating temporal feature of synchronous data-flow program[J]. Computer Engineering and Design, 2014, 35(1): 137 − 143.

[7] 刘洋, 甘元科, 王生原, 等. 同步数据流语言高阶运算消去的可信翻译[J]. 软件学报, 2015, 26(2): 332 − 347.

LIU Y, GAN Y K, WANG S Y, et al. Trustworthy translation for eliminating high-order operation of a synchronous dataflow language[J]. Journal of Software, 2015, 26(2): 332 − 347.

[8] 杨萍, 王生原. 同步数据流语言可信编译器的研究进展[J]. 计算机科学, 2019, 46(5): 21 − 28.

YANG P, WANG S Y. Survey on trustworthy compilers for synchronous data-flow languages[J]. Computer Science, 2019, 46(5): 21 − 28.

[9] 田聪, 邓玉欣, 姜宇. 形式化方法与应用专题前言[J]. 软件学报, 2021, 32(6): 1579 − 1580.

TIAN C, DENG Y X, JIANG Y. Preface to formal methods and applications[J]. Journal of Software, 2021, 32(6): 1579 − 1580.

[10] 刘畅, 蒋永平, 马春燕, 等. 基于 NuSMV 的 AADL 模型形式化验证技术[J]. 航空学报, 2022, 43(3): 443 − 458.

LIU C, JIANG Y P, MA C Y, et al. Formal verification technology for AADL models based on NuSMV[J]. Acta Aeronautica et Astronautica Sinica, 2022, 43(3): 443 − 458.

[11] 朱健, 胡凯, 张伯钧. 智能合约的形式化验证方法研究综述[J]. 电子学报, 2021, 49(4): 792 − 804.

ZHU J, HU K, ZHANG B J. Review on formal verification of smart contract[J]. Acta Electronica Sinica, 2021, 49(4): 792 − 804.

[12] 兰林, 马权, 侯荣彬, 等. Lustre 语言可信代码生成器研究进展[J]. 仪器仪表用户, 2020, 27(5): 68 − 72.

LAN L, MA Q, HOU R B, et al. Research and progress of lustre trusted code generator[J]. Instrumentation Customer, 2020, 27(5): 68 − 72.

[13] 康跃馨, 甘元科, 王生原. 同步数据流语言可信编译器 Vélus 与 L2C 的比较[J]. 软件学报, 2019, 30(7): 2003 − 2017.

KANG Y X, GAN Y K, WANG S Y. Comparison of two trustworthy compilers Vélus and L2C for synchronous languages[J]. Journal of Software, 2019, 30(7): 2003 −

2017.

[14] 孙毅, 陈哲, 冉丹, 等. 一种 SCADE 同步语言程序安全属性自动验证工具[J]. 小型微型计算机系统, 2022, 43(4): 858 − 864.

SUN Y, CHEN Z, RAN D, et al. Tool for automatically verify safety properties of SCADE programs[J]. Journal of Chinese Computer Systems, 2022, 43(4): 858 − 864.

[15] LEROY X. A formally verified compiler backend[J]. Journal of Automated Reasoning, 2009, 43(4): 363 − 446.[LinkOut].

[16] SHI G, ZHANG Y, SHANG S, ET al. A formally verified transformation to unify multiple nested clocks for a Lustre-like language[J]. Science China(Information Sciences), 2019, 62(1): 3.

[17] 林荣峰, 施健, 朱晏庆, 等. 基于 STP 方法的 SCADE 模型形式化验证框架[J]. 计算机工程, 2019, 45(10): 70 − 77.

LIN R F, SHI J, ZHU Y Q, et al. Formal verification framework of SCADE model based on STP method[J]. Computer Engineering, 2019, 45(10): 70 − 77.

[18] 张力. Lustre-的形式化操作语义及其性质研究[D]. 北京: 北京大学, 2012

ZHANG L. Research on formal operation semantics and properties of lustre-[D]. Beijing: Peking University, 2012.