同步数据流语言可信编译器的研究进展

杨 萍[1] 王生原[2]

(北京语言大学信息科学学院 北京 100083)[1] (清华大学计算机科学与技术系 北京 100084)[2]

摘 要 同步数据流语言(如Lustre,Signal)近年来在航空、高铁、核电等安全关键领域得到了广泛应用,因此与这类语言相关的开发工具本身的安全性问题受到高度关注。同步数据流语言到串行命令式语言的可信编译器是此类工具的典型代表(如Scade)。构造可信编译器的途径可分为两大类:一类是传统的方法,例如通过大量测试和严格的过程管理等手段来实现;另一类是通过形式化方法,例如直接对编译器本身进行形式化证明,采用翻译确认的方法等。近年来,形式化方法作为构造和验证可信编译器的关键途径而得到广泛的重视,有望最大限度地解决"误编译"问题,因而成为新的研究热点。文章在介绍可信编译器的形式化构造和验证方法的基础上,特别聚焦于同步数据流语言可信编译器的相关研究工作,对其现状进行综述和分析。

关键词 同步数据流语言,经过验证的编译器,翻译确认,Lustre,Signal

中图法分类号 TP314 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.05.003

Survey on Trustworthy Compilers for Synchronous Data-flow Languages

YANG Ping[1] WANG Sheng-yuan[2]

(School of Information Science,Beijing Language and Culture University,Beijing 100083,China)[1]

(Department of Computer Science and Technology,Tsinghua University,Beijing 100084,China)[2]

Abstract Synchronous data-flow languages,such as Lustre and Signal,have been widely used in safety-critical industrial areas,such as airplane,high-speed railways,nuclear power plants and so on. Hence,the safety of development tools themselves for such languages has been paid highly attention on. The trustworthy compiler from a synchronous data-flow language to a sequential imperative language is typically one of such kinds of tools,such as Scade. There are two ways to implement a trustworthy compiler:the traditional method,for instance,plenty of testing and strict process management;the formal method,for example,formal verification for the compiler itself,and translation validation,etc. The formal method has been paid much attention in recent years,because it has been widely studied as the critical approach in the construction and verification of a trustworthy compiler,and it is expected to have the opportunity in solving the "miscompilation" problem to the utmost extent. After the introduction of formal methods to construct and verify a trustworthy compiler,the survey and analysis of the research work and the current status for the trustworthy compilers of synchronous data-flow language were specially focused on in this paper.

Keywords Synchronous data-flow languages,Certified compilers,Translation validation,Lustre,Signal

1 引言

计算机技术日益广泛地应用于航空航天、高速铁路、核电能源和医疗卫生等领域的安全关键系统(Safety-Critical System,SCS[1])中。SCS一旦失效,将给人类的生命财产、社会生产和生活环境带来巨大的破坏。在现代计算机技术的发展中,相比软件来说,硬件方面的安全性保障技术更加成熟,人为因素导致硬件故障的概率相对较小。虽然人们一直关注于软件安全性问题,并积累了大量研究成果和实践经验,但软件方面的安全保障目前仍旧是计算机系统安全性中的薄弱环节。如何为安全关键系统构造一个基础的安全软件环境是需要解决的首要问题,尤其是对操作系统、编译器等基础软件。本文将针对安全可信的编译器进行介绍。

所谓安全可信的编译器,就是要保证编译器做正确的事情,保证它能将符合源语言规范的程序正确翻译到目标语言程序。这里,"正确翻译"主要指功能方面,通常情况下是指源语言到目标语言的语义保持关系(通常被刻画为一种反向的行为模拟等价关系)。不能实现"正确翻译"的编译器,则一定存在某种程度的"误编译"错误。

作为用于产生代码的工具,编译器的实现和维护自然经

到稿日期:2018-05-11 返修日期:2018-09-02 本文受国家民用飞机重大专项项目基金(MJ-2015-D-066),核高基重大专项(2017ZX01030-301-003)资助。

杨 萍 女,硕士,副教授,主要研究方向为程序语言与编译器、智能语言与技术,E-mail:yangp@blcu.edu.cn;**王生原** 男,副教授,CCF高级会员,主要研究方向为程序语言与编译技术、程序验证,E-mail:wwssyy@tsinghua.edu.cn(通信作者)。

过了“精雕玉琢”,同时又“随时随地”经受着无数应用程序的“考验”,因此编译器的正确性问题容易被人们忽视。但了解实际情况的人都知道,编译器的“误编译”问题是司空见惯、习以为常之事。在许多领域中,由于“误编译”一般不会引发本质的问题,又是小概率事件,因此人们往往也忽视了“误编译”所带来的影响。然而,对于安全攸关系统而言,则必须考虑编译器引入的错误,否则在源程序级的高成本验证工作可能在目标程序级失效。实际上,在航空领域的 RTCA DO-178B/C 类标准中,编译器属于需要鉴定的工具类软件,需要按照机载软件的要求同等对待。

图 1(摘自 X. Leroy 在 POPL'2011 的特邀报告[2])描述了安全关键的嵌入式控制领域中典型的应用程序开发流程:从高级算法描述语言或建模语言所描述的安全级应用程序生成行为等价的 C 代码,后者接着被翻译成可执行的目标代码。图 1 中,Simulink 是面向建模语言 MatLab 的著名工具;Scade[3]是基于同步数据流语言 Lustre[4-5]的建模与开发工具。如图 1 所示,在对安全关键系统进行验证和确认(V&V)时,不能不考虑编译器可能引入的错误。另外,更值得关注的是:若代码生成器或编译器不可信,针对(高级算法描述语言或建模语言)源程序(模型)进行的有关安全性方面的各种努力(模型检查、正确性证明、静态分析及模拟仿真等)则会付之东流。因此,除了要保障应用程序本身的安全性之外,还需要有安全可信的编译器(或代码生成器)。

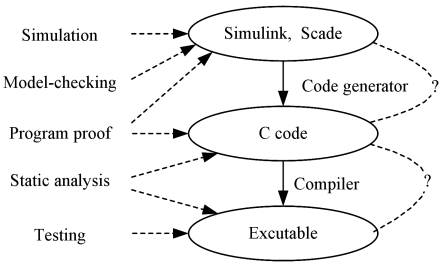

图 1 编译器(代码生成器)带来安全隐患的流程

Fig. 1 Process of compiler/Code generator inducing safety problem

本文将在介绍可信编译器的形式化构造和验证方法的基础上,特别聚焦于同步数据流语言可信编译器的相关研究工作及其发展现状。同步数据流语言(如 Lustre,Signal)近年来在安全关键领域得到了广泛应用,比如 Scade 工具将一种扩展的 Lustre 语言翻译成 C 语言。在实际应用中,通常会将诸如同步数据流语言之类的高级算法描述语言翻译成常规的串行命令式语言(通常为 C 语言),如图 1 的上半部分。同步数据流语言与串行命令式语言之间有很大的语义差距,正如 X. Leroy 所言,构造这样的可信编译器(或可信代码生成器)是一项很有价值且富有挑战性的工作[2]。

本文第 2 节先对同步数据流语言进行简要介绍;第 3 节分析并讨论当前实现可信编译器的主要途径;第 4 节侧重对同步数据流语言到串行命令式语言(通常为 C 语言)可信编译器的典型工作和发展情况进行综述;最后有针对性地进行小结。

2 同步数据流语言

传统的常规语言,如 C 语言,一直以来都是安全关键领域中使用最为普遍的开发语言。人们为安全有效地使用 C 语言下了很大功夫,积累了非常丰富的经验。尽管人们通过各种办法来使得基于 C 语言的系统开发更加安全高效,但毕竟 C 语言程序层次较低,不易使人们聚焦于问题本身,开发效率受到很大影响,并且也难于验证。因此,基于模型/模型驱动的开发逐步兴起并成为业界主流,由模型自动生成的代码(C 代码为主)已占据主导地位,比较著名的建模语言/工具有 Simulink 和 Scade 等。

有一类建模语言被称为同步语言[6],特别适合于实时控制系统的开发。所有同步语言均遵循同步假设(synchrony hypothesis),即每个周期内的计算(从输入值得到输出值)都是瞬间完成的;同步语言的语义被要求具有确定性。同步假设以及确定性可以极大地简化实时系统的验证。另外,同步语言通常有严格且可靠的数学定义,对于同步范型有友好的工程设计方法,因此它在实时嵌入式应用的建模、规范描述、验证以及实现等各个层面都已成为重要的技术选项[7-8]。

20 世纪 80 年代,人们就开始了同步范型的理论研究工作[9],随即又出现了许多基于同步假设的语言,并且这些语言都得到了广泛的应用。Esterel[10-11],Lustre[4-5]和 Signal[12-13]是最著名的几种有代表性的同步语言。其中,Esterel 是命令式语言;Lustre 和 Signal 是陈述式语言,具有数据流特征,常被称为同步数据流语言。Lustre 是函数风格的语言,而 Signal 是关系型的语言。除 Esterel,Lustre 和 Signal 外,学术界和工业界还有其他具有同步特征的语言:Lucid Synchrone[14]是对早期数据流语言 Lucid[15-16]的同步化,并采纳了 Lustre 的许多语言特征;Quartz[17]是一种基于 Esterel 的面向响应式系统的规范、验证以及实现的同步语言;Argos[18]是 Statecharts[19](一种著名的响应式系统的形式化图形语言)的同步化;Synccharts[20]是一种具有 Statecharts 风格和 Esterel 语义的图形语言;Grafcet[21]是一种基于有同步特征的 Petri 网的建模/仿真工具。

这些同步语言各自的特点有利于进行一些实质性的静态检查,甚至形式化分析和验证,从而有益于产生安全的代码。在基于同步语言的开发工具中,最著名的是 Scade,其代码生成器 KCG 将一种基于 Lustre 的建模语言翻译成 C 语言。KCG 应该是获得民用航空软件生产许可(DO-178B/C)的第一个商用代码生成器。目前,Scade 工具已渗透到我国航空航天、轨道交通及核电等安全关键领域。

虽然同步语言的发展已经相当成熟,并在安全关键的嵌入式实时系统中极具影响力,但一直以来,人们对安全关键系统的可信性要求日益提高,期待在各个环节上都达到最高程度的安全可信。同步语言实现工具(各阶段的编译器、代码生成器)本身的安全性,近年来已引起了学术界和工业界的极大关注和兴趣。

本文主要关注同步数据流语言的可信编译器,Lustre,Signal 以及 Lucid Synchrone 是此类语言的典型代表。上面提到的其余同步语言与本文话题关系不大,仅简单提及。此外,Lucid Synchrone 项目所积累的有关编译器验证的基础研究,均由同一课题组人员转化为对 Lustre 编译器的验证工作。事实上,关于同步数据流语言编译器的验证工作,从目前

学术界和工业界的进展来看,仅需关注与 Lustre 和 Signal 相关的工作即可。

同步数据流语言具有时钟同步、并发、流数据对象等特征。例如,Lustre 语言中,所有的数据都是流(stream),对应于无穷多个逻辑时钟周期,每个周期有一个取值。所有的运算都作用于流。每个逻辑时钟周期都执行同样的计算(即相同的代码),并假设在逻辑时钟周期内计算一定可以完成,即满足所谓的同步假设。各个周期内的计算由多个计算节点(node)组成,每个计算节点内部是一个等式的集合,每个等式都定义流上的运算,等式之间并发执行,是数据驱动的计算过程。时态运算和时钟运算是 Lustre 语言(以及其他同步数据流语言)中具有特色的流运算。基本的时态运算包括 pre,fby 和 arrow(→)等,其中 pre 和 fby 可用于访问历史信息;基本的时钟运算包括 when,current 和 merge 等算子,可用于改变时钟的快慢。

3　可信编译器的构造途径

如何才能保证编译器的安全和可信? 对于编译器来说,安全和可信的具体指标就是其正确性,要保证从源程序到目标程序的翻译过程正确,即保证源语言的行为特征能够在目标语言中被正确地体现,杜绝"误编译"。目前保证可信性的方法主要有两类:1)非形式化的方法(或传统的方法),如采用测试和过程管理;2)采用形式化的方法,主要包括直接对编译器本身进行验证和翻译确认,以及混合使用这些方法。

3.1　通过测试和过程管理实现编译器的可信性

为了保证编译器实现的正确性,普遍的做法是采用大量的测试用例对编译器进行测试。例如,GCC[22] 的 torture 测试集包含几千个 C 源程序用例,商用的 Plum Hall Standard Validation Suite for C[23] 有几万个用例,Lustre V6[24] 的开源软件包中含几百个源程序用例,等等。还有一些 Bug-hunting 工具,例如 Csmith[25],它可以产生更多或更独特的源程序用例,从而扩大覆盖范围,发现更多的编译器缺陷。

尽管如此,采用测试的手段仍是不完善的,如果测试用例的覆盖范围不够广泛,则可能会遗漏编译器中的错误。即便是通过测试发现了错误并且做了修改,也无法保证编译器自身的正确性。实际上,仅编译器自动测试工具 Csmith(截至 2011 年 2 月)以及 EMI/SPE(截至 2017 年 10 月)就发现了 GCC 和 LLVM 的近 1700 个编译错误。

Scade[3] 工具的代码生成器 KCG 经过了严格的 V&V 过程,符合民航电子系统的国际标准 DO-178B/C,并且在空客公司的多款客机如 A340 和 A380 中成功应用。然而,DO-178B/C 主要是以过程质量控制为主的标准,不足以说明 Scade 的编译器不存在"误编译"。KCG 并没有通过形式化的方法加以严格验证,虽然 DO-178C 中包含了一些非强制的形式化相关条款,但经调研,业界的用户仍会不时发现 Scade KCG 的某些翻译漏洞。

3.2　通过对翻译过程本身进行形式化验证实现可信编译器

为增加编译器的安全和可信程度,仅通过测试和严格的过程管理都是不够的。对编译器进行正确性验证,是解决问题的根本途径。最严格的验证手段莫过于采用形式化方法。工业界也早已意识到形式化软件开发的潜力,在一些安全攸关领域的安全级软件开发标准中也逐步新增了与形式化方法相关的目标或者相应的补充说明。CC(Common Criteria)安全评估标准[26] 中将可信性分为 7 个级别(EAL1 到 EAL7),可信性级别越高,其采用形式化规范和验证的程度就越高。航空无线电委员会(RTCA)近期也已推出民航电子系统的国际标准 DO-178C,其在 DO-178B 的基础上增加了对形式化规范和验证的要求[27],如 DO-178C 和 DO-333 等补充说明。

近年来,有关编译器形式化验证的研究工作取得了长足的进步,达到了实用化水平,为未来制定新的工业标准奠定了强有力的基础。CompCert[28] 编译器是经过形式化验证的可信编译器的杰出代表。该编译器将 C 的一个重要子集 Clight 翻译为 PowerPC 汇编代码(后来也支持 IA32 和 ARM 后端,目前已扩展至可支持 64 位处理器以及开源的 RISC V 体系结构),其编译过程划分为多个阶段,前端解析过程之后每个阶段的翻译正确性都借助辅助工具 Coq 进行了证明,且这些证明可由独立的证明检查器检查。这是迄今最强的形式化验证手段,达到了人们所能期望的最高可信程度[29]。由于其目标是对主体翻译过程本身进行验证(如图 2 所示),因此称此类可信编译器为经过验证的编译器。Yang 等[25] 关于 Csmith 的研究工作表明:CompCert 在正确性方面的表现明显优于常用的开源或商用 C 编译器。因 CompCert 编译器具有的杰出成就,其代表性论文[30] 的作者 Leroy 获得了 2016 年度的"十年前最有影响 POPL 论文奖"(Most Influential POPL Paper Award)。

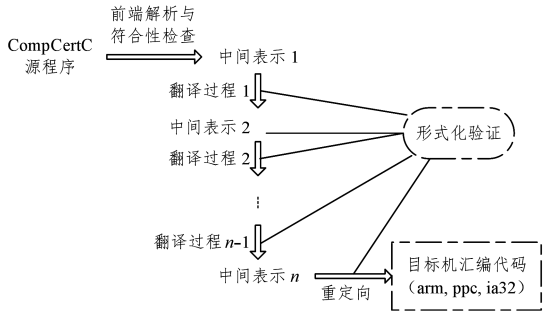

图 2　经过验证的编译器(以 CompCert 为例)

Fig. 2　Certified compiler(example:CompCert)

3.3　通过翻译确认技术实现可信编译器

尽管对编译器的翻译过程本身进行形式化验证是一种比较完美的保证可信性的办法,但是其难度高,工作量大,并且无法扩展(一旦编译器需要修改,那么证明就需要重做)。于是,Pnueli 等最早提出了翻译确认的方案[31]。翻译确认的方法不是直接验证翻译程序,而是用统一的语义框架为翻译过程的源和目标代码建模,两个模型之间定义一种特定的语义等价关系/模拟等价关系,设计一种可自动证明二者等价性的确认程序(返回成功与否,成功时或给出证明脚本,不成功时或给出反例)。根据不同的语义模型,确认程序可以通过自动求解或证明、符号计算、模型检查、静态分析等方式来实现模型间的等价性确认。

如图 3 所示的翻译确认过程中,在发现前后中间表示的语义不等价时,会使编译器停下来。若非主体翻译阶段,比如

某类中间代码的优化，则可以不必让编译器停下来，不做这一优化即可，如图 4 所示。

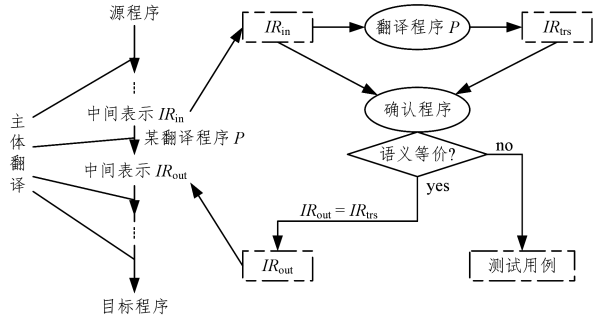

图 3 翻译确认(针对某个主体翻译阶段)

Fig. 3 Translation validation(for a major translation step)

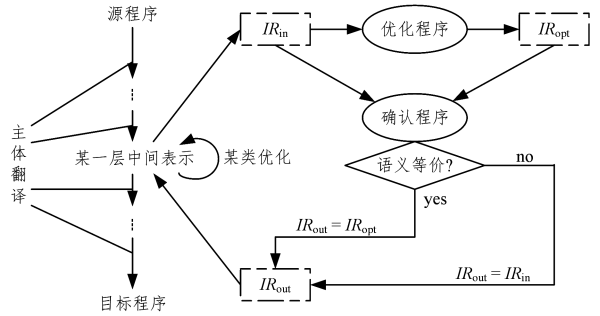

图 4 翻译确认(针对某个中间表示上的优化)

Fig. 4 Translation validation(for an intermediate optimization)

早期，Necula 和 Lee 提出了 PCC(proof carrying code)机制，并带动了关于确认式编译器(certifying compiler)的研究[32]。可以认为 Pnueli 的翻译确认是比 PCC 更通用的机制。PCC 机制主要被用来对编译结果进行安全性检查，而不是对编译器进行验证。

对于翻译确认来说，如果翻译前后的语义保持性的确认过程比较容易构造，或者说源语言与目标语言的语义模拟等价性关系比较容易定义，则证明也会相对容易，那么验证该等价关系是一个不错的选择[28]。因为该方法最大的优点就是可以不放弃现有的编译框架，其可扩展性较好。一些大型编译器也有类似的工作[33-35]。但是当源语言与目标语言差异较大(类似于将同步数据流语言翻译成串行语言)时，两种语言的语义等价性关系是很难定义的，其证明过程也会更加困难。

相比较而言，当源语言和目标语言的语义定义达到认可的程度时，对翻译过程进行验证是一种彻底的做法，原理上可以保证源程序的一般性质都可以保持到目标程序上。而翻译确认的做法往往只是关注部分性质的保持性(当然，也可以逐步逼近一般性质)，因而有可能会存在“虚假预警”(false alarms)。因此，直接验证翻译过程与翻译确认两种方案各有利弊。目前看来，针对一个完整的编译过程，根据各个翻译的特点，将这两种方案混合使用是一种十分有益的做法。比如，一些针对优化算法的翻译确认工作[36-38]可以很好地融合到 CompCert 编译器[28]中。

4 同步数据流语言的可信编译器

同步数据流语言是应用较为广泛的建模语言，比如著名工具 Scade 所使用的建模语言是在同步数据流语言 Lustre 基础上定义的。如图 1 所示，像 Scade 和 Simulink 这样的建模工具，往往是先将建模语言变换到领域语言(许多安全关键领域均采用传统 C 语言)，然后从 C 这样的领域语言编译到目标语言。

对于 C 语言的可信编译研究近年来已经取得了不错的进展，如前面所提到的 Leroy 等开发的 C 编译器 CompCert[28]，其原理图参见图 2。

近年来，针对图 1 中所示的从高级建模语言到 C 语言之间的可信编译研究已成为人们的聚焦点[2]。比如，Michael 和 Strichman 采用了翻译确认的方法对从 Simulink 到 C 的代码生成器 Real-Time Workshop(RTW)的编译过程进行了形式化验证[39]。Simulink 是 MATLAB 中的一种基于块状图设计环境的可视化仿真工具，可实现动态系统建模、仿真和分析，被广泛应用于线性系统、非线性系统、数字控制及数字信号处理的建模和仿真中。模型在 Simulink 中被认为是可执行的描述，借助于 RTW 代码生成器，这些模型可以生成 C 代码。

这一节主要介绍以 Lustre 和 Signal 为代表的同步数据流语言的可信编译器的研究现状。就目前的进展情况而言，Lustre 可信编译器的代表性研究主要集中于对翻译过程进行验证，即采用图 2 所示的方法；而 Signal 可信编译器的代表性研究则是以翻译确认的方法为主，即采用图 3 和图 4 所示的方法。4.1 节介绍 Lustre 可信编译器的代表性研究，4.2 节介绍 Signal 可信编译器的代表性研究，4.3 节介绍一些相关的基础性研究。

4.1 对翻译过程进行形式化验证的方法

对翻译过程进行形式化验证，通常需要对翻译前后语言的语法和语义以及翻译过程进行严格的形式化描述，然后对翻译前后的语义保持关系进行机器证明。这其中涉及到许多相关的基础性研究，本文为了更好地聚焦，不对这些方面的研究工作展开讨论，仅列举某些关系比较密切的内容，参见 4.3 节。

关于同步数据流语言编译器，这方面最早的工作或许是 Gimenez 等于 1999 年试图采用 Coq 针对 Scade V3 进行翻译过程的形式化验证[40]，但这项工作似乎没有坚持下来，因相关资料不详，这里不多述。

关于对翻译过程进行验证的同步数据流语言可信编译器的研究，在国际上较有代表性的长线项目主要是两个团队的工作：一个是法国 INRIA 的 Pouzet 团队，另一个是清华大学计算机系的 L2C 项目组。

受 CompCert 项目[28]的启发，Paulin 和 Pouzet 于 2006 年启动了一个有关同步数据流语言编译器验证的长线项目[41]，其源语言接近 Scade 的 Lustre 语言且具有 Lucid Synchrone[14]的特征。据了解，该项目初期的工作集中于前端，完成了类型检查和时钟演算相关过程的验证；后续又开展了因果性分析程序的验证工作[42]。后来，Biernacki 等[43-44]描述了该项目的一些相关工作细节，他们将 Lustre 语言的一个较早版本翻译至 Java 和 C 代码，采用了一种基于对象的中间语言。再后来，该团队的 Auger 在其博士论文[45]和技术论文[46]中对该项目进行了较完整的表述(特别是理论基础的论述方面)。从中可进一步了解到，Pouzet 团队的工作是针对一种小的但

具有全部 Lustre 语言关键特征的语言 MiniLS,并在此基础上实现了同步数据流语言可信编译器的原型系统[46]。最近,该团队的 Bourke 等又基于 Biernacki 和 Auger 等的工作,实现了从上述基于对象的中间语言 Obc 到 CompCert 的前端中间语言 Clight 的翻译与验证,从而得到一个核心 Lustre 子集的可信编译器 Vélus[47]。

清华大学计算机系的 L2C 项目组于 2010 年开始了一项 Lustre 语言核心子集到 C 语言的可信翻译器的研究(简称 L2C 项目)。该项工作的原理与 CompCert 项目一致,工作目标和上述 Pouzet 团队的项目相似。L2C 项目组创立时主要是面向国内核电领域的实际需求,先是实现了单一时钟情形下的一个完整翻译过程的形式化验证[48],后来又升级到一个可以支持多个嵌套时钟的版本[49]。L2C 项目所定义的源语言 Lustre * 综合参考了 Scade 工具的 Lustre 语言版本和 Lustre V6[50],能够体现同步数据流语言的主要特征。目前,该项目组正在开发和维护一个兼顾学术界和企业界的开源 L2C 版本[51]。

上述两个项目和 CompCert 项目一样,源语言、目标语言和各阶段中间语言的语法、语义、翻译过程的定义以及语义保持性证明都在交互式辅助定理证明工具 Coq[52-53]中实现。然而,二者在许多方面存在不同。首先是源语言的差异,L2C 编译器立项时是出于国内某安全关键领域的实际需求,因此企业版的源语言最终定格为 Scade Lustre 语言的一个核心子集,而开源版的源语言则是将其中的高阶迭代算子(9 个)替换为了 Lustre V6 中的高阶迭代算子(5 个);与 L2C 相比,Pouzet 团队的工作目前主要是面向学术研究,其源语言(包括最新版编译器 Vélus 的源语言)不支持许多数组和结构体上的操作,尚未支持任何高阶运算,并且对时态和时钟算子的支持也略少于前者。其次,L2C 项目可信编译器的目标语言 C 从一开始就借用了 CompCert 编译器中 Clight 的语法和语义定义[54],语义定义中直接采用了贯穿 CompCert 编译器几个阶段的底层存储模型;而 Vélus 是从中间语言 Obc 翻译到 Clight,语义定义中的存储模型有明显的跨度。另外一个明显的差异在于编译器结构,L2C 编译器在 Clight 之前的核心中间层超过 10 层(参见在建的 L2C 网站[51]首页中的图),而 Vélus 编译器在 Clight 之前的核心中间层仅有 2 层(SN-Lustre 和 Obc,参见文献[47]中图 1)。翻译过程分散为多个阶段,有利于简化各翻译阶段的正确性证明,且具有更好的可扩展性。对于实现工业级的可信编译器来说,可扩展性是非常重要的;但同时翻译过程也不应过多,否则会加重实现和维护的负担。

4.2　翻译确认的方法

1998 年,Pnueli[31]第一次提出了翻译确认的设计思想。Pnueli 所给示范例子的源语言就是 Signal 特征的多时钟同步数据流语言,其目标语言是 C。随后,Pnueli 在原有工作的基础上继续实现了对两种同步数据流语言到 C 的编译器的翻译确认[55](这两个编译器中包含了大约 100 个优化规则),证明了翻译确认方法也适用于优化编译器,同时实现了一个翻译确认器 CVT(code validation tool)[56],其可对编译器每一次执行所产生的代码进行检测,通过模型检验判定与源代码是否等价。这一技术用到编译器可追溯的信息,如状态变量是如何翻译的。

近年来,Ngo 和 Talpin 等基于翻译确认的思想,也开展了同步数据流语言 Signal 到 C 的编译器验证工作[57-61]。其主要工作包括:对源代码和目标代码使用统一的语义框架 PDS(polynomial dynamical systems)建模,给出一种源和目标之间的抽象时钟等价关系[59],通过对等价关系进行验证来保证编译器时钟语义的一致性,其证明采用 SMT 求解器[62]自动完成;同时,也基于一种由一阶逻辑公式定义的时钟模型,开展了保持时钟语义的翻译确认工作[57-58];利用同步依赖图(SDGs)对依赖关系的保持性进行确认[58];使用同步数据流求值图(SDVGs),对翻译前后的求值语义保持性进行确认[61];并基于这些技术,提出了一种大规模同步数据流语言编译器验证的扩展性良好的设计方案[60],该方案侧重于对时钟、数据依赖关系的保持性,以及变量的求值等价性等方面的翻译确认。

如 3.3 节所述,翻译确认方法不依赖于翻译的具体过程,相比于对翻译过程进行验证的方法更具有可扩展性。然而,这种方法也有自身的缺陷,比如一般情况下难免存在某种程度的误报(mis-alarm)率。总体来看,上述工作要达到工业级应用的要求,尚有较长的路要走。

4.3　一些相关的基础性研究

与同步数据流语言编译器验证相关的研究有很多,它们都采用了形式化的方法对同步数据流语言的语义进行定义,这对翻译过程中的形式化验证是很有帮助的。

Kahn[63]将一类异步确定并行程序(现称为 Kahn 网络)描述为基于流变换(stream transformation)的递归方程系统,并给出一种完全偏序集(CPO)语义。若将 Kahn 网络限制到同步程序,则可以认为其是 Lustre 或其他同步语言的基础。以 Kahn 网络为出发点,人们也便于对同步数据流范型进行扩展[64-65]。

无疑,对于同步数据流语言语义的形式化定义,Kahn 网络的 CPO 语义(指称语义)是重要的参考之一。Paulin-Mohring 基于 Coq 开发的 Kahn Networks in Coq 库[66-67]是这方面的一项重要工作,同时他们也开发了通用 CPO 库,其比 Kahn 为 Coq 开发的 CPO 库[68]更加完善。

由于是一脉相承,针对 Lucid[15-16]或 Lucid Synchrone[14]语义的一些研究[69-71],包括具体的操作或指称语义定义以及这些工作在 Coq 环境中的实现,在 Lustre 语言可信编译器的研发中都可以借鉴。

借鉴上述工作,可以给出同步数据流语言的指称语义,甚至使用 Coq 的描述语言给出精确的定义。然而,利用此类指称语义来指导编译器的实现以及完成翻译过程的验证有一定难度。

在 Lustre 语言的原始论文[4]中,作者给出了 Lustre 语言的基本操作语义规则,涵盖了时态算子的语义规则,也包含了时钟演算的规则。在 Caspi 和 Pouzet 的研究[64]中也采用类似方法来定义同步数据流语言的同步操作语义。这种操作语义的语义规则非常直观且容易理解,然而若是要对应到编译器的实现,还需要在面向实现方面进一步细化和改进。

还有许多与同步数据流语言相关的工作也值得关注。Cohen 等于 2005－2008 年间提出了针对同步数据流系统更加宽松的同步和时钟演算模型[65,72-73]。Delaval 等提出了空间类型系统，以支持分布式系统上同步数据流语言的设计[74]。

Schneider 等开展了类 Esterel 的命令式同步语言的翻译验证工作，解决了 Schizophrenia 问题[75]，辅助证明工具采用 HOL[76]。Schizophrenia 问题必须在因果分析(causality analysis)之前解决，因同步数据流语言不存在此类问题，故此处不赘述。

L2C 课题组基于一个小的 Lustre 语言子集实现了一个时钟归一化过程的原型，可以将多时钟流转化为单一的时钟流，并形式化验证了变换过程的语义保持性[77-78]。然而，这一方法仅适用于传统的 fby 算子(2 个参数)，在当前 L2C 项目[51,79-83]中未采用。

结束语 现阶段，与常规语言一样，同步数据流语言编译器(或代码生成器)形式化验证的方法可以大致分为两大类：一类是针对编译器本身的验证(整体或部分)，另一类是翻译确认的方法(验证目标代码和源代码之间的符合性，不验证编译器本身)。前者是一种较完美的做法，但工作量大，不容易扩展；后者的扩展性较好，但只适合于解决局部问题。目前已有一些基于辅助定理证明器针对编译器本身进行验证的工作，比如本文提到的 Pouzet 项目组以及 L2C 项目组，二者的源语言均基于 Lustre 语言。关于翻译确认，Pnueli 等首先提出这一方案时，所采用的示例就是基于同步数据流语言 Signal；目前进展较好的工作有 Ngo 和 Talpin 等所开展的同步数据流语言 Signal 到 C 的编译器验证工作。

同步数据流语言与串行命令式语言的语义之间有很大差距，因而对同步数据流语言编译器进行验证(特别是针对编译器本身进行验证)的难度和工作量较大，但实际中存在此类需求，比如，花大力气开发出的安全级产品一旦经过认证而投入使用后，允许再度更新的周期相当长。另外，也无需对编译器的所有环节进行验证，比如，目前很少有人特别重视 lexer 和 parser 部分的验证工作。还有，为降低难度，在某些扩展和维护工作量过大的翻译阶段，我们可以选择不对编译器本身进行验证，而选择采用翻译确认的方法作为补充，比如 CompCert 项目也有个别阶段采用了翻译确认方案。用这种混合的方法能更好地平衡编译器实现过程中的可信性、难度以及工作量之间的关系。

虽然经过形式化验证的编译器离商用尚有很长的路要走，但未来某些标准强制性地要求严格证明是极有可能的，工业界需要有一个逐渐适应的过程。对于可信编译器的开发，形式化验证的方法无疑代表了一种未来新技术变革的趋势。

现实中，可信编译还有另外一层含义，即尽可能保障被编译对象的可信。本文仅关注编译过程的可信，不涉及这方面内容。

致谢 清华大学 L2C 项目组的师生为本研究提供了许多有价值的素材，在此表示感谢。

参 考 文 献

[1] KNIGHT J C. Safety critical systems: challenges and directions [C]// Proceedings of the 24th International Conference on Software Engineering(ICSE 2002). 2002: 547-550.

[2] LEROY X. Verified squared: does critical software deserve verified tools? [J]. ACM SIGPLAN Notices, 2011, 46(1): 1-2.

[3] Scade-suitehome[EB/OL]. http://www.esterel-technologies.com/products/scade-suite.

[4] CASPI P, PILAUD D, HALBWACHS N, et al. Lustre: a declarative language for programming synchronous systems[C]// 14th ACM Symposium on Principles of Programming Languages (POPL'87). Munchen, 1987.

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

[6] HALBWACHS N. Synchronous programming of reactive systems, a tutorial and commented bibliography[M]// Tenth International Conference on Computer-Aided Verification, CAV'98. Berlin: Springer Verlag, 1998.

[7] HALBWACHS N. A Synchronous Language at Work: the Story of Lustre[C]// Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and models for Co-Design, MEMOCODE'05. IEEE Computer Society, Washington, DC, USA, 2005.

[8] BENVENISTE A, CASPI P, EDWARDS S A, et al. The Synchronous Languages Twelve Years Later[J]. Proceedings of the IEEE, 2003, 91(1): 64-83.

[9] MILNER R. Calculi for synchrony and asynchrony[J]. Theoretical Computer Science, 1983, 25(3): 267-310.

[10] BERRY G, GONTHIER G. The Esterel synchronous programming language: Design, semantics, implementation[J]. Science of Computer Programming, 1992, 19(2): 87-152.

[11] BERRY G, The foundations of ESTEREL[M]// proof, language and interaction: essays in honour of Robin Milner. Cambridge: MIT Press, 2000: 425-454.

[12] GUERNIC P L, GAUTIER T, BORGNE M L, et al. Programming real time applications with SIGNAL[J]. Proceedings of the IEEE, 1991, 79(9): 1321-1336.

[13] LE GUERNIC P, TALPIN J P, LE LANN J C. Polychrony for system design[J]. Journal for Circuits, Systems and Computers, 2002, 12(3): 261-304.

[14] LUCID SYNCHRONE home [EB/OL]. http://www.di.ens.fr/～pouzet/lucid-synchrone.

[15] WADGE W W, ASHCROFT E A. LUCID, the dataflow programming language[M]. Academic Press Professional, San Diego, CA, USA, 1985.

[16] ASHCROFT E A, WADGE W W. Lucid, a non procedural language with iteration[J]. Communications of the ACM, 1977, 20(7): 519-526.

[17] SCHNEIDER K. The synchronous programming language Quartz: Internal Report 375[R]. Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, 2009.

[18] MARANINCHI F. the ARGOS language: graphical representation of automata and description of reactive systems[C]// IEEE Workshop on Visual Languages. Kobe, Japan, 1991.

[19] HAREL D, STATECHARTS A. A visual formalism for complex systems[J]. Science of Computer Programming, 1987, 8(3):231-274.

[20] ANDRÉ C. Computing SyncCharts reactions[M]// Synchronous Languages, Applications, and Programming (SLAP'03), Electronic Notes in Theoretical Computer Science. Porto, Portugal, 2003:3-19.

[21] DAVID R. Grafcet: A Powerful Tool[J]. IEEE Transactions on Control Systems Technology, 1995, 3(3):253-268.

[22] GCC. the GNU Compiler Collection[EB/OL]. http://gcc. gnu. org.

[23] The Plum Hall Validation Suite for C™ [EB/OL]. http://www. plumhall. com/stec. html.

[24] Lustre V6[EB/OL]. http://www-verimag. imag. fr/Lustre-V6. html.

[25] YANG X J, CHEN Y, EIDE E, et al. Finding and understanding bugs in C compilers[C]// Proceedings of the 2011 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2011). 2011:283-294.

[26] http://www. commoncriteriaportal. org/cc.

[27] Summary of Difference Between DO-178B and DO-178C [EB/OL]. http://faaconsultants. com/html/do-178c. html.

[28] LEROY X. Formal verification of a realistic compiler[J]. Communications of the ACM, 2009, 52(7):107-115.

[29] MORRISETT G. Technical Perspective: A Compiler's Story [J]. Communications of the ACM, 2009, 52(7):106-106.

[30] LEROY X. Formal certification of a compiler back-end or: programming a compiler with a proof assistant[J]. Proceedings of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, 2006, 41(1):42-54.

[31] PNUELI A, SIEGEL M, SINGERMAN E. Translation Validation[C]// Proceedings of TACAS'98. 1998:151-166.

[32] NECULA G C, LEE P. The design and implementation of a certifying compiler[C]// Proc. Conf. Programming Language Design and Implementation. 1998:333-344.

[33] TRISTAN J B, PAUL G, GREG M. Forthcoming. Evaluating value-graph translation validation for LLVM[C]// Proceedings of the 32nd ACM SIGPLAN Conference on Programming and Language Design Implementation. 2011:295-305.

[34] KANADE A, SANYAL A, KHEDKER U. A PVS based framework for validating compiler optimizations[C]// 4th Software Engineering and Formal Methods. IEEE Computer Society, 2006:108-117.

[35] NECULA G C. Translation validation for an optimizing compiler [M]// Programming Language Design and Implementation 2000. ACM Press, 2000:83-95.

[36] TRISTAN J B, LEROY X. Verified validation of lazy code motion[C]// PLDI. 2009:316-326.

[37] TRISTAN J B, LEROY X. A simple, verified validator for software pipelining[C]// POPL. 2010:83-92.

[38] TRISTAN J B, LEROY X. Formal verification of translation validators: A case study on instruction scheduling optimizations [C]// 35th symposium Principles of Programming Languages. ACM Press, 2008:17-27.

[39] RYABTSEV M, STRICHMAN O. Translation validation: From simulink to c[C]// Proceedings of the 21st International Conference on Computer Aided Verification (CAV 2009). Berlin: Springer, 2009:696-701.

[40] GIMENEZ E, LEDINOT E. Certication de SCADE V3, Rapport final du project GENIE II, Verilog SA (Janvier 2000).

[41] PAULIN C, POUZET M. Certified compilation of scade/lustre [EB/OL]. http://www. lri. fr/paulin/lustreincoq. pdf, 2006.

[42] BERTAILS A, BIERNACKI D, PAULIN C, et al. A certified compiler for the synchronous language Lustre[EB/OL]. http://users. dimi. uniud. it/types07/slides/Bertails. pdf.

[43] BIERNACKI D, COLACO J L, HAMON G, et al. Clock-directed Modular Code Generation of Synchronous Data-flow Languages [C]// ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES). Tucson, Arizona, 2008.

[44] BIERNACKI D, COLACO J L, POUZET M. Clock-directed Modular Code Generation from Synchronous Block Diagrams [C]// Workshop on Automatic Program Generation for Embedded Systems (APGES 2007). Salzburg, Austria, 2007.

[45] AUGER C. Compilation Certifiée de SCADE/LUSTRE[D]. Université Paris Sud, 2013.

[46] AUGER C, COLAÇOB J L, HAMON G, et al. A Formalization and Proof of a Modular Lustre Compiler[EB/OL]. http://www. di. ens. fr/~pouzet/cours/mpri/cours4/scp12. pdf.

[47] BOURKE T, BRUN L, DAGAND P é, et al. A Formally Verified Compiler for Lustre[C]// Proceedings of the Programming Language Design and Implementation. 2017:586-601.

[48] SHI G, WANG S Y, DONG Y, et al. Construction for the Trustworthy Compiler of a Synchronous Data-Flow Language[J]. Journal of Software, 2014, 25(2):341-356. (in Chinese)
石刚,王生原,董渊. 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.

[49] SHANG S, GAN Y K, SHI G, et al. Key Translations of the Trustworthy Compiler L2C and Its Design and Implementation [J]. Journal of Software, 2017, 28(5):1233-1246. (in Chinese)
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.

[50] Lustre V6 reference manual[EB/OL]. http://www-verimag. imag. fr/DIST-TOOLS/SYNCHRONE/lustre-v6/doc/lv6-ref-man. pdf.

[51] Open L2C home[EB/OL]. http://soft. cs. tsinghua. edu. cn:8000/.

[52] The Coq Development Team. The Coq Proof Assistant Reference Manual Version V8. 3[EB/OL]. http://coq. inria. fr/.

[53] BERTOT Y, CASTÉRAN P. Interactive Theorem Proving and Program Development-Coq'Art: The Calculus of Inductive Constructions[M]// Texts in Theoretical Computer Science. Springer Verlag, 2004.

[54] BLAZY S, LEROY X. Mechanized semantics for the Clight subset of the C language[J]. Journal of Automated Reasoning, 2009, 43(3):263-288.

[55] PNUELI A, SHTRICHMAN O, SIEGEL M. Translation validation for synchronous languages[C]// Proceedings of ICALP'

1998. Berlin:Springer,1998:235-246.

[56] PNUELI A,SIEGEL M,SHTRICHMAN O. The code validation tool(CVT)- automatic verification of a compilation process [J]. Journal of Software Tools for Technology Transfer (STTT),1999,2(2):192-201.

[57] NGO V C,TALPI J P,GAUTIER T. Formal Translation validation for clock transformations in a synchronous compiler[C]// International Conference on Fundamental Approaches to Software Engineering(ETAPS/FASE'15). Springer,2015.

[58] NGO V C,TALPIN J P,GAUTIER T,et al. Formal verification of synchronous data-flow program transformations toward certified compilers[J]. Frontiers of Computer Science,2013,7(5):598-616.

[59] NGO V C,TALPI J P,GAUTIER T,et al. Formal Verification of Compiler Transformations on Polychronous Equations[M]// IFM 2012. Berlin:Springer,2012:113-127.

[60] NGO V C,TALPI J P,GAUTIER T,et al. Modular translation validation of a full-sized synchronous compiler using off-the-shelf verification tools (abstract)[C]// International Workshop on Software and Compilers for Embedded Systems (SCOPES'15). ACM,2015:109-112.

[61] NGO V C,TALPI J P,GAUTIER T. Translation Validation for Synchronous Data-flow Specification in the SIGNAL Compiler [C]// International Conference on Formal Techniques for Distributed Objects,Components and Systems (FORTE'15). IFIP,2015:66-80.

[62] DUTERTRE B,DE MOURA L. Yices sat-solver[EB/OL]. http://yices. csl. ri. com,2009.

[63] KAHN G. The semantics of a simple language for parallel programming[M]// Information Processing 74 Congress. North-Holland Publishing Compmany,Amsterdam,1974.

[64] CASPI P,POUZET M. Synchronous Kahn networks[C]// Proceedings of the first ACM SIGPLAN International Conference on Functional Programming (ICFP'96). Philadelphia,Pennsylvania,1996:226-238.

[65] COHEN A,DURANTON M,EISENBEIS M,et al. N-Sychronous Kahn Networks[C]// 33th ACM Symp. on Principles of Programming Languages (PoPL'06). Charleston,South Carolina,2006:180-193.

[66] PAULIN-MOHRING C. Kahn Networks in Coq[EB/OL]. http://www. lri. fr/~paulin/KahnNetworks/library. pdf.

[67] PAULIN-MOHRING C. A constructive denotational semantics for Kahn networks in Coq[M]// From Semantics to Computer Science. Essays in Honour of Gilles Kahn,2009:383-414.

[68] KAHN G. Elements of domain theory[EB/OL]. http://coq. inria. fr/pylons/contribs/view/DomainTheory/trunk.

[69] CASPI P,POUZET M. A co-iterative characterization of synchronous stream functions.[J]. Electronic Notes in Theoretical Computer Science,1998,11(4):1-21.

[70] BOULMÉ S,HAMON C. Certifying Synchrony for Free[C]// International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR). La Havana,Cuba,Springer Verlag,2001.

[71] Lucid-Synchrone in Coq [EB/OL]. http://www. di. ens. fr/~pouzet/lucid-synchrone/lucid_in_coq.

[72] COHEN A,DURANTON M,EISENBEIS C,et al. Synchronization of Periodic Clocks[C]// ACM Conf. on Embedded Software (EMSOFT'05). Jersey City,New York,2005:339-342.

[73] COHEN A,MANDEL L,PLATEAU F,et al. Abstraction of Clocks in Synchronous Data-flow Systems[C]// The Sixth ASIAN Symposium on Programming Languages and Systems (APLAS). 2008.

[74] DELAVAL G,GIRAULT A,POUZET M. A Type System for the Automatic Distribution of Higher-order Synchronous Dataflow Programs[C]// ACM International Conference on Languages,Compilers, and Tools for Embedded Systems (LCTES). Tucson,Arizona,2008.

[75] SCHNEIDER K,BRANDT J,SCHUELE T. A Verified Compiler for Synchronous Programs with Local Declarations[J]. Electronic Notes in Theoretical Computer Science,2006,153(4):71-97.

[76] HOL[EB/OL]. http://www. cl. cam. ac. uk/research/hvg/HOL.

[77] ZHANG Y. Source code to unify nested clocks in a lustre-like language [EB/OL]. https://github. com/yczhang89/unify-clock.

[78] SHI G,ZHANG Y C,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):012801:1-012801:3.

[79] GAN Y K,ZHANG L B,SHI G,et al. A Verified Sequentializer for Synchronous Data-Flow Programs[J]. Computer Applications and Software,2014,31(5):1-5. (in Chinese)
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.

[80] ZHANG L B,GAN Y K,SHI G,et al. A certified translation for eliminating temporal feature of a synchronize dataflow program [J]. Computer Engineering and Design,2014,35(1):137-143. (in Chinese)
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.

[81] LIU Y,YANG F,SHI G,et al. Brief Overview of Translation Validation Method in Trusted Compiler Construction[J]. Computer Science,2014,41(S1):334-338. (in Chinese)
刘洋,杨斐,石刚,等. 可信编译器构造的翻译确认方法简述[J]. 计算机科学,2014,41(S1):334-338.

[82] SHI G,GAN Y K,SHANG S,et al. A formally verified sequentializer for lustre-like concurrent synchronous data-flow programs[C]// Proceedings-2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). 2017:109-111.

[83] 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. (in Chinese)
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.