同步语言多线程代码生成的语义保持证明方法

袁胜浩[1)] 杨志斌[1),2)] 张博林[1)] 周勇[1)] 薛垒[3)] BODELEIX Jean-Paul[4)] FILALI Mamoun[4)]

[1)](南京航空航天大学 计算机科学与技术学院，南京 211106)

[2)](高安全系统的软件开发与验证技术工信部重点实验室，南京 211106)

[3)](上海航天电子技术研究所，上海 201109)

[4)](IRIT-University of Toulouse, Toulouse France 31062)

摘 要 同步语言具有确定性并行和精确时间语义等特性，因此被广泛用于设计和验证安全关键软件。随着安全关键领域应用多核处理器逐渐成为趋势，同步语言的多线程代码生成及其语义保持证明研究成为研究热点。目前，已有同步语言代码生成方法还较少考虑多线程代码生成的语义保持证明。因此，本文提出一种同步语言 SIGNAL 多线程代码生成的语义保持证明方法：首先形式化定义编译过程中源、目标、中间语言的结构化操作语义；其次形式化定义多线程代码生成过程；最后基于互模拟等价思想证明编译前后的语义一致性。

关键词 同步语言；安全关键软件；多任务代码生成；语义保持证明； Coq

中图法分类号 TP311

A Semantic Preservation Proving Method of Multi-threaded Code Generation for Synchronous Language

YUAN Sheng-Hao[1)] YANG Zhi-Bin [1)2)] ZHANG Bo-Lin[1)] ZHOU Yong[1)] XUE Lei[3)] BODEVEIX Jean-Paul[4)] FILALI Mamoun[4)]

[1)](School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106)

[2)](Key Laboratory of Safety-critical Software, Ministry of Industry and Information Technology, Nanjing 211106)

[3)](Shanghai Aerospace Electronic Technology Institute, Shanghai 201109)

[4)](IRIT-University of Toulouse, Toulouse 31062, France)

Abstract Safety-critical software is widely used in the fields of avionics, space systems, and nuclear power plants: Malfunctions of safety-critical software can lead to accidents that can potentially put people, environment, property, and mission in serious risks such as environmental catastrophes and loss of lives. Synchronous Languages are adopted for the design and the verification of safety-critical software, due to their characteristic features, for instance the description of deterministic concurrency behaviors and precise timing semantics. Recently, safety-critical domains have evolved to use high computation performance provided by multicore

本课题得到国家自然科学基金(61502231)、航空科学基金(201919052002)、GF基础科研重点项目(JCKY2016203B011)、中央高校基本科研业务费专项资金资助(NP2017205)、南京航空航天大学研究生创新基地（实验室）开放基金(kfjj20181603)资助。袁胜浩(**通信作者**)，硕士研究生，CCF学生会员（NO.55228G），主要研究领域为形式化方法、定理证明。E-mail：shyuan@nuaa.edu.cn。杨志斌（**通信作者**），博士，副教授，CCF会员（NO.08632M），主要研究领域为安全关键软件、形式化验证。E-mail：yangzhibin168@163.com 。 张博林，硕士研究生，CCF学生会员（NO.88946G），主要研究领域为模型验证、软件工程。E-mail：m18756801996@163.com 。 周勇，男，博士，副教授，CCF会员（NO.54088M），研究方向为软件工程、形式化方法等。E-mail：zhouyong@nuaa.edu.cn。 薛垒，高级工程师，研究方向为嵌入式软件验证、嵌入式软件系统设计。E-mail：jzgunking@163.com。Bodeveix Jean-Paul，教授，研究方向为实时系统、形式方法。E-mail： bodeveix@irit.fr。 Filali Mamoun，高级研究员，研究方向为实时系统、形式方法。E-mail： filali@irit.fr。

platforms to implement more complex functionality. Therefore the multi-threaded code generation approach for synchronous languages and related semantic preservation proof have been research hotspots. However, the existing studies on synchronous languages mainly concern the semantic preservation of sequential code generation from synchronous languages. Such as the verified synchronous language Lustre compiler Vélus supports sequential C code generation, the verified Lustre compiler L2C translates Lustre models to Clight programs which can be considered as the input of the CompCert compiler and the translation validation approach is used to verify the transformation steps from the synchronous language SIGNAL to sequential C code. They pay a little attention to the semantic preservation of multi-threaded code generation. As multi-core processors gradually become widely used in safety-critical systems, it is necessary to consider the semantic preservation of multi-threaded code generation approach for synchronous languages. Therefore, this paper presents a verified multi-threaded code generation method for the synchronous language SIGNAL. Firstly, a common SIGNAL subset which is compatible with two existing SIGNAL compilers Polychrony and MiniSIGNAL, is selected as the start point to consider the proof of semantic preservation of the multi-threaded code generation process and its formal syntax and structural operational semantics (based on transition systems) is proposed; Secondly, the clock calculus procedure is formally defined to generate clockedSIGNAL programs from source SIGNAL models, the procedure consists of the formal definition of the extraction function of clock relations and construction function of clause sets, the formal description of solving clock equations system and the definition of the clock dependency graph, the operational semantics of clockedSIGNAL structure is also constructed according to the transition systems of source SIGNAL models and the result of the clock calculus procedure; Thirdly, an abstract C-like subset structure is considered as the target language and its formal definition as well as the operational semantics are proposed, the transformation step from clockedSIGNAL programs to target C-like subset code contains two important functions: the computation function and the memory update function. The semantics consistency of the transformation step (named clockedSIGNAL2C) includes the proof of two directions: from clockedSIGNAL to C-like and from C-like to clockedSIGNAL which correspond to two lemmas. Considering the proof processes of two lemmas are similar, this paper mainly introduces the first lemma, i.e. clockedSIGNAL2C satisfies the unidirectional bisimulation equivalent relation. Finally, a theorem about the semantics consistency of the transformation step is proved to guarantee the semantic consistency between the SIGNAL program after clock calculus and the target C program after translation based on the bisimulation equivalent relation.

Key words synchronous language; safety-critical software; multi-threaded code generation; semantic preservation; Coq

1 引言

安全关键软件(Safety-Critical Software)[1]是指应用于航空、航天、交通、能源等领域的安全关键系统中，且其运行情况可能引起系统处于危险状态，从而导致财产损失、环境破坏或者人员伤害的一类软件。近年来，模型驱动(Model-Driven)尤其是采用形式化模型驱动的安全关键软件设计与开发方法逐渐受到重视，并被工业界认为是切实可行的重要手段。其中，同步语言是一种被广泛用于设计和验证安全关键软件的形式化语言。例如，空客使用 SCADE[2](即同步语言 LUSTRE[3]的工业界版本)对 A380 的飞控子系统进行建模和代码生成。

目前主流的同步语言有 ESTEREL[4]、LUSTRE、SIGNAL[5]、QUARTZ[6]等。其中 ESTEREL、LUSTRE 以及 QUARTZ 采用纯同步模式(Perfect Synchrony)，即存在一个全局时钟(单时钟)，而 SIGNAL 采用多态同步模式(Polychrony)，即多时钟，能够更自然和方便地表达分布式系统。随着多核处理器的不断发展，嵌入式系统设计人员更多关注多态同步模式。

现有 SIGNAL 编译器 Polychrony[7-8]主要支持串行代码生成和仿真分析，较少考虑在多核处理器

上的多任务代码生成。在文献[9][10][11]中，我们提出了一个面向多核平台的多线程代码生成工具MiniSIGNAL，同时支持生成串行、多线程程序。

诸多安全关键领域标准(如 DO-330)规定安全关键软件的开发工具本身必须经过验证，但目前Polychrony 和 MiniSIGNAL 的多线程代码生成过程中较少考虑语义一致性（即语义保持）的形式化证明。在前期研究[9-11]的基础上，本文进一步提出一种同步语言 SIGNAL 多线程代码生成的语义保持证明方法。该方法主要包括：

1) 形式化定义编译过程中多种语言的结构化操作语义；
2) 形式化定义多线程代码生成各个阶段的翻译函数；
3) 基于互模拟等价思想证明编译前后的语义一致性。

本文第 2 节介绍 MiniSIGNAL 代码生成工具；第 3 节给出 SIGNAL 多线程代码生成核心编译过程的语义保持证明方法；第 4 节介绍相关工作；第 5 节是本文的总结和展望。

2 MiniSIGNAL 简介

本节首先介绍 MiniSIGNAL 的模块化编译过程，然后介绍 SIGNAL 多线程代码生成核心步骤的语义保持证明框架。

2.1 MiniSIGNAL模块化编译过程

MiniSIGNAL 是一个同步语言 SIGNAL 的多任务代码生成工具，整个工具使用函数式编程语言 OCaml 实现。本文简要给出 MiniSIGNAL 多线程代码生成的基本步骤(见图 1)。详细见文献[10]。

1) 词法语法分析：支持对用户输入的 uSIGNAL 程序进行词法语法分析；
2) 程序标准化：将 uSIGNAL 程序中扩展结构转换为基本结构，以及将复合结构转换为基本结构。标准化后的程序只包括基本结构(称为 kSIGNAL 程序)；
3) S-CGA 生成：基于转换规则将 kSIGNAL 程序中基本结构所表达的功能/时钟关系转换为 S-CGA 中对应的卫式动作(也称为守卫条件)；
4) 时钟演算：从 S-CGA 程序中提取时钟关系等式，并进行消解。基于 BDD/SMT 计算时钟等价类并优化 S-CGA 程序；
5) 依赖分析：分析卫式动作间的读写依赖关系，生成数据依赖图 CDDG；
6) 任务划分：基于划分算法对依赖图进行划分，生成的划分结果 partition 表明了 S-CGA 程序中的并行信息；
7) 虚拟多线程代码生成：基于划分结果，将 S-CGA 程序转换为虚拟多线程代码 vmt；
8) 多线程代码生成：根据虚拟多线程代码进一步生成多线程 Ada/C/Java 程序。

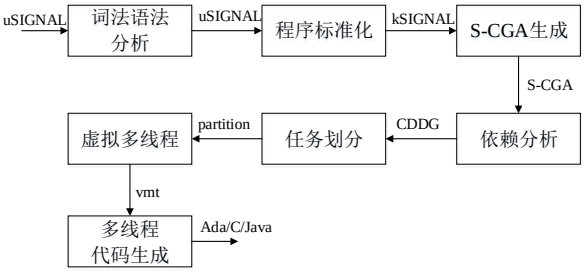

图 1 MiniSIGNAL 多线程代码生成模块化编译步骤

其中，中间语言 S-CGA 和虚拟多线程结构等的形式语法语义分别基于定理证明工具 Coq 进行形式化表示。而程序标准化到 S-CGA 的转换过程中的语义一致性也在定理证明器 Coq 中证明[12]。

和 Polychrony 的编译步骤相比，MiniSIGNAL 中新增 S-CGA 生成和虚拟多线程结构：前者用于支持 MiniSIGNAL 集成其它同步语言；后者用于生成平台无关的多线程代码。同时，在任务划分阶段，MiniSIGNAL 支持引入多种划分算法例如基于拓扑排序划分、基于流水线方式[10]和基于 RPDMA 算法[13]等计算并行执行信息。除代码生成外，MiniSIGNAL 还将进一步支持仿真实验(如 Simulink 程序)。和形式化验证(如 UPPAAL 模型)等。

2.2 SIGNAL多线程代码生成语义保持证明框架

由于 Polychrony 和 MiniSIGNAL 的编译步骤有所差异，因此本文在语义保持证明过程中，选择了两者的公共子集(如图 2 所示)，即时钟演算和(多线程)代码生成。本文选择的源语言为 SIGNAL 语言核心子集 kSIGNAL，该子集包括 SIGNAL 语言中的基本结构、组合、局部声明等。经时钟演算后生成的程序记为 clockedSIGNAL 程序。本文选择的目标语言为 C 语言子集。

SIGNAL 多线程代码生成语义保持证明框架包括如下步骤：首先定义代码生成过程中间语言的操作语义(基于标记变迁系统)，具体包括 kSIGNAL、clockedSIGNAL 和目标 C 语言子集；其次形式化定义 kSIGNAL 到目标程序的翻译过程，包括时钟演

算和 clockedSIGNAL2C 翻译函数；最后基于互模拟等价关系证明翻译函数 clockedSIGNAL2C 保持 clockedSIGNAL 和目标 C 子集的语义一致性。

图 2 SIGNAL 多线程代码生成语义保持证明

需要注意的是，选择 kSIGNAL 的主要原因在于该子集包括了 SIGNAL 语言的基本表达能力，便于后续验证编译器工作的开展，但我们计划逐步增加该子集的规模，直到覆盖完整的 SIGNAL 语言。而选择C子集作为目标语言的主要原因在于计划将生成的多线程 C 程序进一步经过可信编译器 Compositional CompCert[14]编译生成机器指令。此外，本文将依赖分析、任务划分和代码生成阶段抽象为 clockedSIGNAL2C 翻译函数。

3 多线程代码生成语义保持证明方法

本节首先给出 kSIGNAL 的形式语法和形式语义；其次给出时钟演算的形式化描述并构造 clockedSIGNAL 形式语义；然后形式化定义 clockedSIGNAL2C 编译过程和 C 子集的形式语义；最后基于互模拟等价思想证明 clockedSINGAL2C 过程的语义一致性。

3.1 kSIGNAL

3.1.1 kSIGNAL形式语法

SIGNAL 是一种声明式数据流同步语言。SIGNAL 语言中定义了一种带类型的无限长的值序列，称为信号(signal)。在给定逻辑时刻下，信号可以是”存在”状态并携带对应的数值，或者是”缺失”状态，记为⊥。信号 x 的逻辑时钟($\hat{x}$)为信号存在状态对应逻辑时刻组成的集合。在SIGNAL中，两个信号同步当且仅当其逻辑时钟相同。

SIGNAL 的核心子集 kSIGNAL 包括四类基本数据流结构、组合和局部声明，其抽象语法定义为：

$$\begin{aligned} P ::= & \; x := f(x_1,...,x_n) \\ & | \; x := x_1 \; \$ \; init \; c \\ & | \; x := x_1 \; when \; x_2 \\ & | \; x := x_1 \; default \; x_2 \\ & | \; P \,|\, P' \\ & | \; P / x \end{aligned}$$

在语义保持证明过程中，局部声明 P/x 可以作为一类特殊的输出进行处理，因此后续语义保持证明中将不考虑局部声明。

3.1.2 kSIGNAL形式语义

SIGNAL 语言具有多种形式语义，如踪迹语义和标签模型语义[15]等。本文遵循 Plotkin 关于结构化操作语义的原则[16]，将 kSIGNAL 的操作语义定义为一个标签变迁系统$\langle S, A, \rightarrow \rangle$，其中状态集 S 中状态为进程 P 所有可能状态，标签 A 对应进程 P 和外界环境的交互，$\rightarrow : S \times A \times S$ 为变迁规则。约定变迁规则的记号为：

$$\frac{C}{S \triangleright P \xrightarrow{\alpha} S' \triangleright P}$$

其中 $S \triangleright P$ 和 $S' \triangleright P$ 分别表示进程 P 的两个状态；$\alpha \in \{Receive(\vec{I}, \overrightarrow{value}), Send(\vec{O}), \tau\}$ 表示进程 P 的动作；C 是与 $S \triangleright P$、$S' \triangleright P$ 和 α 有关的前置条件。该记号表示在前置条件满足时，进程 P 可以在状态 S 下完成动作 α 并变迁到 S' 状态。

首先，进程 P 的状态定义如下。

定义 1 **状态** 进程 P 的状态定义为逻辑时钟、信号变量、端口和基本进程执行标记的取值组成：

$$\begin{aligned} & S : T + P_{Var} + \{port, \; flag_{P_{Set}}\} \rightarrow \\ & \omega_0 + Value_{\perp}^{\omega_0} + Boolean \end{aligned}$$

其中，T 表示 P 当前的逻辑时钟取值，信号变量集合 P_{Var} 表示在不同时钟下取值可能不同(包括⊥)，$port$ 和 $flag_{P_{Set}}$ 分别表示端口状态和进程 P 中基本数据流等式的执行状态。P 的初始状态为 $S_0 = \{T : 0\} \cup \{v_i[T] : \perp \,|\, v_i \in P_{Var}\} \cup \{port : True\} \cup \{flag_i : False \,|\, i \in P_{Set}\}$。

其次，定义谓词前提。

定义 2 **前提 preC** 在给定逻辑时刻 T 下，基本数据流等式的前提 $preC$ 归纳定义为：

$$(1) \; preC(x := f(x_1,\ldots,x_n)) = \left(\bigwedge_{1 \le i \le n} ((x_i[T] \neq \perp))\right) \wedge (\neg port) \wedge flag_{x := f(x_1,\ldots,x_n)}$$

$(2)\ preC^1(x := x_1\$\ init\ c)\ \square$
$(x_1[T]\neq\perp)\wedge(\underset{i<T}{\wedge}(x_1[i]=\perp))\wedge(\neg port)\wedge flag_{x:=x_1\$\ init\ c}$

$(3)\ preC^2(x := x_1\$\ init\ c)\ \square$
$(x_1[T]\neq\perp)\wedge(\underset{i<T}{\vee}(x_1[i]\neq\perp))\wedge(\neg port)\wedge flag_{x:=x_1\$\ init\ c}$

$(4)\ preC(x := x_1\ when\ b)\ \square$
$(x_1[T]\neq\perp)\wedge(b=True))\wedge(\neg port)\wedge flag_{x:=x_1\ when\ b}$

$(5)\ preC^1(x := x_1\ default\ x_2)\ \square$
$(x_1[T]\neq\perp)\wedge(\neg port)\wedge flag_{x:=x_1\ default\ x_2}$

$(6)\ preC^2(x := x_1\ default\ x_2)\ \square$
$(x_1[T]=\perp)\wedge(x_2[T]\neq\perp)\wedge(\neg port)\wedge flag_{x:=x_1\ default\ x_2}$

最后，依次定义四类基本结构、组合操作和与环境交互的变迁规则。其中外部不可见动作 τ 被忽略。

基本结构

设 P 为瞬时函数 $x := f(x_1, x_2, \ldots, x_n)$，则 P 的语义规则 R1.1 定义为：

$$\frac{S\ `\ preC(P)}{S \triangleright P \to S[f(x_1[T],\ldots,x_n[T])/x[T], False/\ flag_P] \triangleright P}$$

设 P 为延迟操作 $x := x_1\ \$\ init\ c$，则 P 的语义规则 R1.2 和 R1.3 分别定义为：

$$\frac{S\ `\ preC^1(P)}{S \triangleright P \to S[c/x[T], False/\ flag_P] \triangleright P}$$

$$\frac{S\ `\ preC^2(P)}{S \triangleright P \to S[x_1[i]/x[T], False/\ flag_P] \triangleright P}$$

其中 $i = max\{i < T \mid x_1[i] \neq \perp\}$。

设 P 为条件采样 $x := x_1\ when\ b$，则 P 的语义规则 R1.4 定义为：

$$\frac{S\ `\ preC(P)}{S \triangleright P \to S[x_1[T]/x[T], False/\ flag_P] \triangleright P}$$

设 P 为确定性并发 $x := x_1\ default\ x_2$，则 P 的语义规则 R1.5 和 R1.6 分别定义为：

$$\frac{S\ `\ preC^1(P)}{S \triangleright P \to S[x_1[T]/x[T], False/\ flag_P] \triangleright P}$$

$$\frac{S\ `\ preC^2(P)}{S \triangleright P \to S[x_2[T]/x[T], False/\ flag_P] \triangleright P}$$

组合操作

设 P 为组合操作 $P_1 \mid P_2$，则 P 的语义规则 R1.7 和 R1.8 分别定义为：

$$\frac{S \triangleright P_1 \to S' \triangleright P_1}{S \triangleright (P_1 \mid P_2) \to S' \triangleright (P_1 \mid P_2)}$$

$$\frac{S \triangleright P_2 \to S' \triangleright P_2}{S \triangleright (P_1 \mid P_2) \to S' \triangleright (P_1 \mid P_2)}$$

环境交互

设 P 的输出和输出变量集合分别为 $\vec{I}$ 和 $\vec{O}$，进程 P 和外部环境交互的语义规则 R1.9 和 R1.10 分别定义为：

$$\frac{S\ `\ port}{S \triangleright P \xrightarrow{Receive(\vec{I},\vec{value})} S[\overrightarrow{value}/\vec{I}, \overrightarrow{True}/\ \overrightarrow{flag_P}, False/\ port] \triangleright P}$$

$$\frac{S\ `\ (\underset{p\in P_{Set}}{\wedge}(flag_p \to (\neg preC(p))))\wedge(\neg port)}{S \triangleright P \xrightarrow{Send(\vec{O})} S[True/\ port, Succ(T)/T] \triangleright P}$$

以 R1.1 规则为例解释对应变迁规则的具体含义：瞬时函数的语义规则表明当等式右侧所有变量在状态 S 中都有值，且该瞬时函数未被执行，即对应 flag 标志位为 $True$，则将 x 的值赋为运算符 f 作用下的取值，且置对应 flag 为 $False$。

3.2 clockedSIGNAL

3.2.1 时钟演算

clockedSIGNAL 程序是 kSIGNAL 经过时钟演算后生成的，因此本节在构造 clockedSIGNAL 操作语义之前，需要形式化定义时钟演算。时钟演算的形式化描述具体包括：

1) 形式化定义时钟关系抽取函数和子句集构造函数；
2) 形式化表示求解时钟等式系统算法；
3) 基于时钟等式系统 Eq 定义时钟依赖图；

kSIGNAL 的基本结构同时包括功能关系和时钟关系，时钟关系抽取函数 $\phi(P)$ 用于计算出 kSIGNAL 模型中的时钟关系。$\phi(P)$ 可根据进程 P 的结构归纳定义如下：

1) 当 $P = x := f(x_1, \ldots, x_n)$ 时：

$\phi(x := f(x_1,\ldots,x_n)) \triangleq \left\{\hat{x} = x_i \mid 1 \le i \le n\right\}$

2) 当 $P = x := x_1 \ \$ \ init \ c$ 时：

$\phi(x := x_1 \ \$ \ init \ c) \triangleq \left\{\hat{x} = x_1\right\}$

3) 当 $P = x := x_1 \ when \ b$ 时：

$\phi(x := x_1 \ when \ b) \triangleq \left\{\hat{x} = x_1 \wedge \hat{b}\right\}$

4) 当 $P = x := x_1 \ default \ x_2$ 时：

$\phi(x := x_1 \ default \ x_2) \triangleq \left\{\hat{x} = x_1 \vee x_2\right\}$

5) 当 $P = P_1 \mid P_2$ 时：$\phi(P_1 \mid P_2) \triangleq \phi(P_1) \cup \phi(P_2)$

子句集构造函数 $Clause(P)$ 用于将时钟关系等式进一步转换为对应子句集，以便后续进行时钟演算。$Clause(P) \triangleq \bigcup_{(\hat{x}=x_i)\in\phi(P)} (\{(\neg\hat{x}), x_i)\}, \{(\neg x_i), \hat{x}\} \cup \bigcup_{(\hat{x}=x_1\wedge b)\in\phi(P)} (\{\neg\hat{x}, x_1, b)\}, \{\neg x_1, \hat{x}\}, \{\neg b, \hat{x}\}) \cup \bigcup_{(\hat{x}=x_1\vee x_2)\in\phi(P)} (\{\neg\hat{x}, x_1, x_2)\}, \{\neg x_1, \hat{x}\}, \{\neg x_2, \hat{x}\})$ 。形成的子句集记为 Ω 。

同时，后续时钟演算中涉及到子句集操作定义如下：

函数 $POS(\Omega, v) \triangleq \{cl \mid v \in cl \wedge cl \in \Omega\}$ 表示变量 $v \in P_{Var}$ 在子句集 Ω 中正出现的子句集；同理 $NEG(\Omega, v) \triangleq \{cl \mid \neg v \in cl \wedge cl \in \Omega\}$ 表示 v 在 Ω 中负出现的子句集。

集合 $POS_v \triangleq \{cl - \{v\} \mid cl \in POS(\Omega, v)\}$，表示将 v 正出现的子句集中减去 v，$NEG_v \triangleq \{cl - \{\neg v\} \mid cl \in NEG(\Omega, v)\}$ 表示将 v 负出现的子句集中减去 $\neg v$ 。

求解时钟等式系统算法描述了时钟演算的核心步骤，根据输入的 kSIGNAL 程序 P 中的时钟关系，计算出所有有定义的时钟等式，形成时钟等式方程 Eq 并输出。算法首先计算出子句集 Ω 并初始化有关变量(第 01 行)，然后进入循环，依次计算出所有有定义的时钟等式(第 03-12 行)：如果子句集中存在空子句(记为{}或 $\varnothing$)，则表示时钟关系中存在矛盾，报错并终止算法(第 03 行)；否则任选一个未定义变量 v (第 05 行)，计算其对应的 POS_v 和 NEG_v 形成的新子句集，新生成的子句集中永真子句被忽略(第 07 行)，其余子句添加到 Ω 中(第 08 行)；最后将 v 对应的时钟等式加入到 Eq 中，并在中删去与 v 有关的子集。当子句集 Ω 为空时，循环结束(第 02 行)，并返回时钟等式方程 Eq (第 13 行)。

算法 1. 求解时钟等式系统算法.

输入：kSIGNAL 进程 P

输出：时钟等式方程 Eq

01：$\Omega := Clause(P)$；$\Omega_0 := \varnothing$；$Eq := \varnothing$

02：WHILE $\Omega \neq \varnothing$ DO

03：　IF ‘ $(\bigwedge_{C\in\Omega}(\bigvee_{e\in C} e))$ THEN ERROR

04：　ELSE

05：　　select a $v \in P_{Var} - Def(Eq)$

06：　　FOR EACH $<P, N> \in POS_v \times NEG_v$ DO

07：　　IF ‘ $\bigvee_{e\in P\vee N} e$ THEN SKIP

08：　　ELSE $\Omega_0 := \Omega_0 \cup (P \cup N)$

09：　　END FOR

10：　　$Eq := Eq \cup \{v \triangleq (\bigwedge_{C\in NEG_v}(\bigvee_{e\in C} e))\}$

11： $\Omega := \Omega_0 \cup (\Omega - POS(\Omega, v) - NEG(\Omega, v))$

12： END WHILE

13： RETURN Eq

根据时钟等式系统 Eq，可按如下方式定义时钟依赖图 $<V, E>$：

- $V = Def(Eq) \cup \{P_{Var} - Def(Eq)\}$
- $E \subseteq V \times V$，其中 $<v_1, v_2> \in E$ 当且仅当至少以下条件之一成立：
  - $v_1, v_2 \in Def(Eq)$ 并且存在 $eq \in Eq(v_1 \in right(eq) \wedge v_2 = left(eq))$
  - $v_1 \in P_{Var} - Def(Eq), v_2 \in Def(Eq)$ 并且存在 $eq \in Eq, q \in k$ $r(i\ g$ $(k \in v_1 \wedge v_2 = left(eq))$。

其中 $\{P_{Var} - Def(Eq)\}$ 记为根节点 $root$，函数 $left$ ($right$)用于计算时钟等式 eq 左侧变量(右侧变量集合)。

易证上述定义方式形成时钟依赖图为有向无环图，以及具有根时钟。

3.2.2 clockedSIGNAL形式语义

基于 kSIGNAL 语义对应的变迁系统 $<S, A, \rightarrow>$，clockedSIGNAL 语义对应的变迁系统 $<S', A, \rightarrow'>$ 定义为： $S' = \{s_i \mid s_i \in S$ $and\ s_i$ 惯$Eq\ and\ s_i$ $root\}$ 以及 $\rightarrow' = S'^2 \cap \rightarrow$。

其中 s_i 惯$Eq \Leftrightarrow s_i$ $eq\ for\ all\ eq \in Eq$。

clockedSIGNAL 语义定义表明，只有当消解后的程序同时满足根节点存在且时钟等式方程约束时，程序执行才有意义。

3.3 C子集

本节介绍一个包含源语言特性的C语言子集作为目标语言，并形式化定义 clockedSIGNAL 到 C 之间翻译函数，最后基于标签变迁系统定义所选 C 子集的形式语义。

3.3.1 C子集形式语法

首先，选择的 C 子集由三类条件语句和并发语句组成，其抽象语法如下所示：

$$
\begin{aligned}
Pc ::= &\ if\ b_1\ then\ x_1 := e_1 \\
&| if\ b_1\ then\ x_1 := e_1; x_2 := e_2 \\
&\quad\ else\ if\ b_2\ then\ x_3 := e_3 \\
&| if\ b_1\ then\ x_1 := e_1 \\
&\quad\ else\ if\ b_2\ then\ x_2 := e_2 \\
&| eq \,||\, eq
\end{aligned}
$$

其中 x_i 为变量，e_i 为表达式，b_i 为布尔表达式。

3.3.2 clockedSIGNAL2C编译过程

其次，本文选择基本的任务划分算法[10]，其基本原则是每个基本结构对应于一个划分，并转换为对应的目标线程。这里本文将其抽象为 clockedSIGNAL2C 翻译函数，该翻译函数由计算函数 $\varphi(P)$ 和内存更新函数 $\psi(P)$ 组成。

根据 clockedSIGNAL 程序 P 的结构，$\varphi(P)$ 归纳定义为：

1) 当 $P = x := f(x_1, \ldots, x_n)$ 时：

$$\varphi(x := f(x_1, \ldots, x_n)) = if \underset{1 \le i \le n}{\wedge} x_i\ then\ x := f(x_1, \ldots, x_n)$$

2) 当 $P = x := x_1\ \$\ init\ c$ 时：

$$
\begin{aligned}
&\varphi(x := x_1\ \$\ init\ c) = \\
&if\ \ x_1 \wedge flag_x\ then\ Mem_x := c; flag_x := false \\
&\qquad\qquad else\ if\ \ x_1\ then\ Mem_x := x_1
\end{aligned}
$$

3) 当 $P = x := x_1\ when\ b$ 时：

$$\varphi(x := x_1\ when\ b) = if\ \ \hat{x}_1 \wedge b\ then\ x := x_1$$

4) 当 $P = x := x_1 \ default \ x_2$ 时：

$\varphi(x := x_1 \ default \ x_2) \triangleq$
$if \ \hat{x}_1 \ then \ x := x_1 \ else \ if \ (\neg\hat{x}_1) \wedge \hat{x}_2 \ then \ x := x_2$

5) 当 $P = Eq$ 时：$\forall eq \in Eq \quad \varphi(eq) \triangleq eq$

6) 当 $P = P_1 \mid P_2$ 时：$\varphi(P_1 \mid P_2) \triangleq \varphi(P_1) \| \varphi(P_2)$

内存更新函数 $\psi(P)$ 与延迟动作相关，负责更新内存变量。因此 $\psi(P)$ 定义为：

1) 当 $P = x := x_1 \ \$ \ init \ c$：

$\psi(x := x_1 \ \$ \ init \ c) \triangleq if \ \ x_1 \ then \ x := Mem_x$

2) 当 $P = P_1 \mid P_2$：

$\psi(P_1 \mid P_2) \triangleq \psi(P_1) \| \psi(P_2)$

对于一个 clockedSIGNAL 进程 P，其生成的命令式程序为 $Pc = \varphi(P) + \psi(P)$，其执行顺序为每次计算函数结束后，执行一次内存更新函数。

3.3.3 C子集形式语义

最后，C 子集的操作语义定义为变迁系统 $< S_c, A, \rightarrow >$，状态 $s \in S_c$ 包括时钟、变量、内存变量和内存标志：

$$s : T + \{P_{Var}, P_{Mem}\} + \{flag_{Mem}, flag_P, port\}$$
$$\rightarrow \omega_0 + Value_\perp + Boolean$$

特别地， clockSIGNAL 中变量是时钟到变量取值的函数映射空间，而 C 程序中变量直接取为当前时钟下的变量取值，因此需要额外定义内存变量 P_{Mem} 和内存标志 $flag_{Mem}$ 用于记录历史数据。S_c 中初始状态 $s_0 \triangleq \{T : \emptyset\} \ \cup \{v : v \in V_a \mid P\} \cup \{v : False \mid v \in P_M\} \cup \{v : True \mid v \in flag_{Mem} \cup flag_P \cup \{port\}\}$。

类似地，在定义 C 子集变迁规则之前，需归纳定义对应的谓词 $precC$：

1) $precC(if \ \ b_1 \ then \ x_1 := e_1) \triangleq$
$(\neg port) \wedge b_1 \wedge flag_{if \ b_1 \ then \ x_1 := e_1}$

2) $precC^1(if \ \ b_1 \ then \ x_1 := e_1; x_2 := e_2$
$else \ if \ \ b_2 \ then \ x_3 := e_3) \triangleq (\neg port) \wedge b_1$
$\wedge \ flag_{if \ b_1 \ then \ x_1 := e_1; x_2 := e_2 \ else \ if \ b_2 \ then \ x_3 := e_3}$

3) $precC^2(if \ \ b_1 \ then \ x_1 := e_1; x_2 := e_2$
$else \ if \ \ b_2 \ then \ x_3 := e_3) \triangleq (\neg port) \wedge (\neg b_1)$
$\wedge b_2 \wedge flag_{if \ b_1 \ then \ x_1 := e_1; x_2 := e_2 \ else \ if \ b_2 \ then \ x_3 := e_3}$

4) $precC^1(if \ \ b_1 \ then \ x_1 := e_1 \ else \ if \ \ b_2$
$then \ x_2 := e_2) \triangleq (\neg port) \wedge b_1$
$\wedge \ flag_{if \ b_1 \ then \ x_1 := e_1 \ else \ if \ b_2 \ then \ x_2 := e_2}$

5) $precC^2(if \ \ b_1 \ then \ x_1 := e_1 \ else \ if \ \ b_2$
$then \ x_2 := e_2) \triangleq (\neg port) \wedge (\neg b_1) \wedge$
$b_2 \wedge flag_{if \ b_1 \ then \ x_1 := e_1 \ else \ if \ b_2 \ then \ x_2 := e_2}$

C 子集对应操作语义规则按如下方式定义：

设 C 程序为 $Pc = if \ \ b_1 \ then \ x_1 := e_1$，则对应语义规则 R2.1 为：

$$\frac{S \ `\ precC(if \ \ b_1 \ then \ x_1 := e_1)}{S \triangleright if \ \ b_1 \ then \ x_1 := e_1 \rightarrow S[e / x, \ False / flag_{if \ b_1 \ then \ x_1 := e_1}] \triangleright if \ \ b_1 \ then \ x_1 := e_1}$$

设 C 程序为 $Pc = if \ \ b_1 \ then \ x_1 := e_1; x_2 := e_2$ $else \ if \ \ b_2 \ then \ x_3 := e_3$，则对应语义规则 R2.2 和 R2.3 分别定义为：

$$\frac{S \text{ ‘ } precC^1(if\ b_1\ then\ x_1 := e_1; x_2 := e_2\ else\ if\ b_2\ then\ x_3 := e_3)}{S \triangleright if\ b_1\ then\ x_1 := e_1; x_2 := e_2\ else\ if\ b_2\ then\ x_3 := e_3 \to S[e_1/x_1, e_2/x_2, False/flag_{if\ b_1\ then\ x_1:=e_1;x_2:=e_2\ else\ if\ b_2\ then\ x_3:=e_3}] \triangleright if\ b_1\ then\ x_1 := e_1; x_2 := e_2\ else\ if\ b_2\ then\ x_3 := e_3}$$

$$\frac{S \text{ ‘ } precC^2(if\ b_1\ then\ x_1 := e_1; x_2 := e_2\ else\ if\ b_2\ then\ x_3 := e_3)}{S \triangleright if\ b_1\ then\ x_1 := e_1; x_2 := e_2\ else\ if\ b_2\ then\ x_3 := e_3 \to S[e_3/x_3, False/flag_{if\ b_1\ then\ x_1:=e_1;x_2:=e_2\ else\ if\ b_2\ then\ x_3:=e_3}] \triangleright if\ b_1\ then\ x_1 := e_1; x_2 := e_2\ else\ if\ b_2\ then\ x_3 := e_3}$$

设 C 程序为 $Pc = if\ b_1\ then\ x_1 := e_1\ else\ if\ b_2\ then\ x_2 := e_2$，则对应语义规则 R2.4 和 R2.5 分别定义为：

$$\frac{S \text{ ‘ } precC^1(if\ b_1\ then\ x_1 := e_1\ else\ if\ b_2\ then\ x_2 := e_2)}{S \triangleright if\ b_1\ then\ x_1 := e_1\ else\ if\ b_2\ then\ x_2 := e_2 \to S[e_1/x_1, False/flag_{if\ b_1\ then\ x_1:=e_1\ else\ if\ b_2\ then\ x_2:=e_2}] \triangleright if\ b_1\ then\ x_1 := e_1\ else\ if\ b_2\ then\ x_2 := e_2}$$

$$\frac{S \text{ ‘ } precC^1(if\ b_1\ then\ x_1 := e_1\ else\ if\ b_2\ then\ x_2 := e_2)}{S \triangleright if\ b_1\ then\ x_1 := e_1\ else\ if\ b_2\ then\ x_2 := e_2 \to S[e_2/x_2, False/flag_{if\ b_1\ then\ x_1:=e_1\ else\ if\ b_2\ then\ x_2:=e_2}] \triangleright if\ b_1\ then\ x_1 := e_1\ else\ if\ b_2\ then\ x_2 := e_2}$$

对于并发结构，设 C 程序为 $Pc = Pc_0 \| Pc_1$，则对应语义规则 R2.6 和 R2.7 分别定义为：

$$\frac{S \triangleright Pc_0 \to S' \triangleright Pc_0}{S \triangleright Pc_0 \| Pc_1 \to S' \triangleright Pc_0 \| Pc_1}$$

$$\frac{S \triangleright Pc_1 \to S' \triangleright Pc_1}{S \triangleright Pc_0 \| Pc_1 \to S' \triangleright Pc_0 \| Pc_1}$$

对于外部环境交互，对应语义规则 R2.8 和 R2.9 分别定义为：

$$\frac{S \text{ ‘ } port}{S \triangleright P \xrightarrow{?(\vec{v},\overrightarrow{value})} S[\overrightarrow{value}/\vec{v}, \overrightarrow{True}/\overrightarrow{flag_P}] \triangleright P}$$

$$\frac{S \text{ ‘ } (\underset{p \in P_{Set}}{\wedge}(flag_p \to (\neg precC(p)))) \wedge (\neg port)}{S \triangleright P \xrightarrow{!(\vec{v},\overrightarrow{value})} S[\overrightarrow{\perp}/\overrightarrow{P_{Var} - P_{Mem}}, True/port, Succ(T)/T] \triangleright P}$$

注意 clockedSIGNAL 中端口(port)等标记，可以通过在实际物理设备上增加若干个从输出端口到输入端口的反馈信号来实现。因此在语义保持证明过程中，可以忽略 clockedSIGNAL 语义中端口等标记对证明过程的影响。

3.4 语义保持证明方法

本节主要基于互模拟等价思想证明编译前后的语义一致性。首先给出互模拟的定义，其次介绍语义保持证明过程中的核心引理，最后基于核心引理完成语义保持证明。

定义 3 **互模拟** 令 $TS =< S, A, \to >$ 为一个标记转换系统。互模拟定义为一种关系 $R \subseteq S \times S$ 使得，如果 $< q_1, q_2 > \in R$，则对任意动作 $\mu \in A$，以下关系成立：

1) $\forall q_1'(q_1 \xrightarrow{\mu} q_1' \Rightarrow \exists q_2'(q_2 \xrightarrow{\mu} q_2' \ and\ q_1'\ R\ q_2'))$

2) $\forall q_2'(q_2 \xrightarrow{\mu} q_2' \Rightarrow \exists q_1'(q_1 \xrightarrow{\mu} q_1' \ and\ q_1'\ R\ q_2'))$

如果上述关系有且只有一个被满足，则称为单向模拟关系。

由于 clockedSIGNAL 操作语义是基于 kSIGNAL 操作语义构造而成，易证时钟演算过程只

存在单向模拟关系。因此，本节主要讨论clockedSIGNAL2C编译过程的语义保持证明。

该过程语义保持证明主要涉及两类引理：

引理 1. clockedSIGNAL2C 编译过程具有单向模拟关系(1)。

引理 2. clockedSIGNAL2C 编译过程具有单向模拟关系(2)。

引理 1 和引理 2 证明思路相似，这里仅介绍前者的主要证明过程。

证明 (引理 1). 设 P 为 clockedSIGNAL 程序，对应操作语义为 $<S, A, \rightarrow>$，Pc 为经过翻译程序生成的目标程序，对应操作语义为 $<Sc, A, \rightarrow>$。

模拟关系 $R \subseteq S \times Sc$ 定义为 $<s, sc> \in R$ 当且仅当 $\forall v \in P_{Var}(s \text{ 惯} v \Leftrightarrow sc \quad v)$。这里我们考虑程序内部执行过程（不考虑和外部交互行为），因此变迁动作仅包括不可见动作 τ。

假设 $<q_1, q_2> \in R$，且 $q_1 \rightarrow q_1'$：

根据证明树的深度进行归纳证明，按证明树上应用的最后一条规则进行分类讨论。

情形 1 当应用语义规则 R1.1 时：

$\therefore q_1 \text{ 惯} preC(P), q_1' \quad x[T] = f(x_1[T], \ldots, x_n[T])$ 且 $q_1' \text{‘} \ x$。

根据谓词 $preC$ 可得 $q_1 \text{‘} \underset{1 \le i \le n}{\wedge} ((x_i[T] \neq \perp))$。

$\therefore q_2 \text{‘} \underset{1 \le i \le n}{\wedge} ((x_i[T] \neq \perp))$，即 $q_2 \text{‘} \underset{1 \le i \le n}{\wedge} (x_i)$。

根据函数 φ 得到（$Pc = if \underset{1 \le i \le n}{\wedge} x_i \ then \ x := f(x_1, \ldots, x_n)$）：

$\therefore$ 根据语义规则 R2.1 可得：

存在 $q_2' = q_2[f(x_1, \cdots, x_n) / x]$ 使得 $q_2 \rightarrow q_2'$。

易证 $q_2' \text{ 惯} x \Leftrightarrow q_1' \quad x$。

$\therefore <q_1', q_2'> \in R$。

情形 2 当应用语义规则 R1.2 时：

$\therefore q_1 \text{‘} \ preC(P)^1$，$q_1' \text{‘} \ x[T] = c$ 且 $q_1' \text{‘} \ x$。

根据 $preC$ 可得 $q_1 \text{‘} \ x_1$。

$\therefore q_2 \text{‘} \ x_1$，即 $q_2 \text{‘} \ x_1$。

根据 φ 得到（$Pc = if \ x_1 \wedge flag_x \ then \ Mem_x := c; flag_x := false \ else \ if \ x_1 \ then \ Mem_x := x_1$）：

$\therefore$ 由语义规则 R2.2 成立可得：

存在 $q_2' = q_2[c / x]$ 使得 $q_2 \rightarrow q_2'$。

易证 $q_2' \text{ 惯} x \Leftrightarrow q_1' \quad x$。

$\therefore <q_1', q_2'> \in R$。

情形 3 当应用语义规则 R1.3 时：

$\therefore q_1 \text{ 惯} preC(P)^2, q_1' \quad x[T] = x_1[i], q_1' \text{？} x$

根据 $preC$ 可得 $q_1 \text{‘} \ x_1$。

$\therefore q_2 \text{‘} \ x_1$，即 $q_2 \text{‘} \ x_1$。

根据 i 的定义可知，x_1 前一个有取值时刻不是 0 时刻。

$\therefore$ 根据 R2.2 可知，$q_2 \text{‘} \ \neg flag_x$。

$\therefore$ 语义规则 R2.3 成立

$\therefore$ 存在 $q_2' = q_2[x_1 / Mem_x]$ 使得 $q_2 \rightarrow q_2'$。

根据 ψ 可得 $q_2'' = q_2[x_1 / x]$ 使得 $q_2 \rightarrow q_2''$。

易证 $q_2'' \text{ 惯} x \Leftrightarrow q_1' \quad x$。

$\therefore <q_1', q_2''> \in R$。

易证，对于 R1.4-R1.6（**情形 3**–**情形 6**），存在 q_2' 使得 $q_2 \rightarrow q_2'$ 且 $<q_1', q_2'> \in R$。

情形 7 当应用语义规则 R1.7 时：

$\therefore q_1 \triangleright P_1 \to q_1' \triangleright P_1'$。

根据归纳假设，存在对应的 $q_2 \triangleright Pc_1 \to q_2' \triangleright Pc_1'$ 且 $< q_2, q_2' > \in R$。

∴由语义规则 R2.6 可得：

$q_2 \triangleright Pc_1 \| Pc_2 \to q_2' \triangleright Pc_1' \| Pc_2$。

情形 8：当应用语义规则 R1.8 时：

$\therefore q_1 \triangleright P_2 \to q_1' \triangleright P_2'$。

根据归纳假设，存在对应的 $q_2 \triangleright Pc_2 \to q_2' \triangleright Pc_2'$ 且 $< q_2, q_2' > \in R$。

∴由语义规则 R2.7 可得：

$q_2 \triangleright Pc_1 \| Pc_2 \to q_2' \triangleright Pc_1 \| Pc_2'$。

综上所述，因此可得 $\forall q_1'(q_1 \to q_1' \Rightarrow \exists q_2'(q_2 \to q_2' \; and \; q_1' \; R \; q_2'))$。

证毕.

最后，利用定理 1 完成语义保持证明。

定理 1. clockedSIGNAL2C 编译前后语义保持。

证明. 根据引理 1 可知，$\forall q_1'(q_1 \to q_1' \Rightarrow \exists q_2'(q_2 \to q_2' \; and \; q_1' \; R \; q_2'))$。

根据引理 2 可知，$\forall q_2'(q_2 \xrightarrow{\mu} q_2' \Rightarrow \exists q_1'(q_1 \xrightarrow{\mu} q_1' \; and \; q_1' \; R \; q_2'))$

∴即证明 clockedSIGNAL 程序和翻译后的目标 C 程序两者语义互模拟等价。

证毕.

4 相关工作

在设计实现安全关键系统时，代码自动生成技术是一种提高系统安全性的重要手段。代码自动生成方法本身的正确性需要经过形式化的证明，即证明翻译前后保持程序正确性等性质，以确保不会出现误编译的情况(即一个错误的编译器可能在编译过程中插入错误)。

本文将已有编译器验证工作按照接收源程序不同划分为语言层面和模型层面：

4.1 语言层面

CompCert[17]是经过形式化验证的可信编译器。该编译器将 C 的一个核心子集翻译为机器代码，生成的代码性能在 PowerPC 上可达到 GCC4-1 性能的 90%。CompCert 基于数学的方式提出一种消除误编译的根本性解决方案(即携带证明编译 Certified Compiler 方法)：通过辅助定理证明工具对编译器本身进行形式化证明，在数学上证明编译器生成的可执行机器代码和 C 源程序的语义上完全一致。

在 CompCert 的基础上，Gordon 等人提出一种经过验证支持分离编译的编译器 Compositional CompCert。Compositional CompCert 支持共享内存交互，基于交互语义（interaction semantics）对模块和外部环境的交互进行建模，并通过逻辑模拟关系证明与交互语义相关的编译步骤的正确性。

4.2 模型层面

基于 CompCert 编译器，法国 Inria Paris 的 PARKAS 小组实现了经过验证的 Lustre 编译器 Vélus[18]。Vélus 接收一个 Lustre 语言子集，经过若干编译步骤，最终生成符合 CompCert 语法的 Clight 程序，并基于 CompCert 进一步编译生成可执行机器代码。

类似于 Vélus，国内清华大学 L2C 项目提出一个 L2C 编译器[19]支持将 Lustre 语言子集可信翻译到 C 程序。L2C 编译器的编译过程中涉及到类型和时钟检查，处理时序算子 fby 和 pre(类似于本文中的时钟演算)，并经过一系列翻译步骤，生成符合 CompCert 的 Clight 程序。

此外，A。Pnueli 等人[20]提出了一种转换确认(Translation Validation)的方法，并验证从 SIGNAL 语言到 C 程序的翻译过程。该方法针对每一次具体的编译，并且要求源程序和翻译后的目标程序具有共同的语义基础（基于变迁系统）。然后证明翻译前后是否语义保持，采用的行为等价关系为单向模拟关系(Simulation)。

综上所示，相比于转换确认方法，本文采用了 CompCert 中提出的携带证明编译技术，可以更为彻底地对编译器内部翻译步骤依次进行形式化证明，并正在开展在定理证明器 Coq 中完成证明过程的检

查工作。此外，目前已有基于携带证明技术的同步语言语义保持证明研究主要侧重于串行代码生成的语义保持证明，而本文主要关注多线程代码生成的语义保持证明。

5 总结与展望

随着多核处理器逐渐被用于在安全关键领域，同步语言的多线程代码生成方法及其语义保持证明方法成为学术界和工业界共同关注的重要问题。在前期研究中，我们提出了基于 OCaml 的同步语言 SIGNAL 多线程代码生成工具 MiniSIGNAL。本文则进一步提出一种 SIGNAL 核心编译过程的语义保持证明方法，通过形式化定义编译过程中多种语言的操作语义和抽象代码生成过程，并基于互模拟等价思想证明了多线程代码生成核心步骤的语义一致性。

MiniSIGNAL 的最终目标是生成一个新的 SIGNAL 验证编译器(Verified Compiler)，即在 Coq 中规约并证明编译规则的语义保持，然后从 Coq 证明中自动抽取编译器的实现。本文目前主要考虑 MiniSIGNAL 核心编译步骤的语义保持证明，并选择 C 子集作为目标语言。为生成最终的 SIGNAL 验证编译器，我们正在 Coq 中开展对 MiniSIGNAL 中其余编译过程如标准化、任务划分(涉及多种划分算法)、虚拟多任务结构和多线程代码生成(涉及 Ada 和 Java 等多种目标语言)等阶段的语义一致性进行证明。前期研究中我们主要关注 SIGNAL 多线程代码生成方法，在未来的工作中，我们将进一步考虑在执行平台机制方面的优化工作[21]及负载均衡等问题。SIGNAL 主要用于描述系统的同步数据流行为，难以对复杂信息物理系统进行建模。因此我们还将研究基于架构描述语言 AADL[22]和 SIGNAL 的异构建模方法。

参 考 文 献

[1] Leveson N. Engineering A Safer World: Systems Thinking Applied to Safety. Cambridge, USA:MIT Press, 2011.

[2] Colaço J L, Pagano B, Pasteur C, et al. Scade 6: from a Kahn Semantics to a Kahn Implementation for Multicore//Proceedings of the Forum on Specification & Design Languages (FDL). Munich, Germany, 2018: 5-16.

[3] Halbwachs N, Caspi P, Raymond P, Pilaud D. The synchronous data-flow programming language Lustre. Proceedings of the IEEE, 1991, 79(9):1305–1320.

[4] Boussinot F, de Simone R. The Esterel language. Proceedings of the IEEE, 1991,79(9):1293–1304.

[5] Benveniste A, Le Guernic P, Jacquemot C. Synchronous programming with events and relations: The signal language and its semantics. Science of Computer Programming, 1991,16:103–149.

[6] Schneider K. The synchronous programming language QUARTZ. Germany: Department of Computer Science, University of Kaiserslautern, Internal Report:375, 2010.

[7] Besnard L, Gautier T, Le Guernic P, et al. Compilation of polychronous data flow equations//Proceedings of the Synthesis of Embedded Software. Boston, USA, 2010: 1-40.

[8] Besnard L, Gautier T, Talpin J P. Code generation strategies in the Polychrony environment. France: INRIA, Research Report:RR-6894, 2009.

[9] Yang Zhi-Bin, Zhao Yong-Wang, Huang Zhi-Qiu, Hu Kai, Ma Dian-Fu, Bodeveix Jean-Paul, Filali Mamoun. Time-Predictable multi-threaded code generation with synchronous languages. Journal of Software, 2016,27(3):611−632 (in Chinese).
(杨志斌，赵永望，黄志球，胡凯，马殿富，BODEVEIX Jean-Paul, FILALI Mamoun. 同步语言的时间可预测多线程代码生成方法.软件学报,2016, 27(3): 611-632.)

[10] Yang Zhi-Bin, Yuan Sheng-Hao, Xie Jian, Zhou Yong, Chen Zhe, Xue Lei, Bodeveix Jean-Paul, Filali Mamoun. Multi-threaded Code Generation Tool for Synchronous Language. Journal of Software, 2019, 30(7): 1980-2002(in Chinese).
(杨志斌，袁胜浩，谢健，周勇，陈哲，薛垒，Jean-Paul BODEVIX, Mamoun FILALI. 一种同步语言多线程代码自动生成工具. 软件学报, 2019, 30(7): 1980-2002.)

[11] Yang Z, Bodeveix J P, Filali M, et al. Towards a verified compiler prototype for the synchronous language SIGNAL. Frontiers of Computer Science, 2016, 10(1):37-53.

[12] Yang Z, Bodeveix J P, Filali M. Towards a simple and safe Objective Caml compiling framework for the synchronous language SIGNAL. Frontiers of Computer Science, 2019, 13(4): 715-734.

[13] Jiang Wen-Yi Pang Li-Ping, Gao Lan, Han Zongfen. Serial Program Parallelism Algorithm. Journal of Huazhong University of Science and Technology(Natural Science Edition). 2000, 28(12): 30-32(in Chinese)
(江文毅，庞丽萍，高兰. 串行程序的并行划分算法研究. 华中科技大学学报(自然科学版), 2000,28(12):30-32.)

[14] Stewart G, Beringer L, Cuellar S, et al. Compositional compcert//Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages(POPL). Mumbai, India, 2015, 50(1): 275-287.

[15] Gamati é A. Designing embedded systems with the Signal programming language: synchronous, reactive specification. Berlin, Germany: Springer Science & Business Media, 2009.

[16] Plotkin G D. A structural approach to operational semantics. Aarhus, Denmark: Computer Science Department, Aarhus University Denmark.

1981.

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

[18] Bourke T, Brun L, Dagand P É, et al. A formally verified compiler for Lustre//Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation(PLDI). Barcelona, Spain, 2017, 52(6): 586-601.

[19] Kang Yue-Xin, Gan Yuan-Ke, Wang Sheng-Yuan. Comparison of Two Trustworthy Compilers Vélus and L2C for Synchronous Languages. Journal of Software, 2019, 30(7): 2003-2017(in Chinese).
(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. 软件学报, 2019, 30(7): 2003-2017.)

[20] Pnueli A, Siegel M, and Singerman E. Translation validation//Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Germany, 1998:151-166.

[21] Tran H N, Honorat A, Talpin J P, et al. Efficient Contention-Aware Scheduling of SDF Graphs on Shared Multi-bank Memory//Proceedings of the 24th International Conference on Engineering of Complex Computer Systems. Hong Kong, China. 2019: 114-123.

[22] Feiler P H, Gluch D P, Hudak J J. The architecture analysis & design language (AADL): An introduction. Technical Report: CMU/SEI-2006-TN-011. Carnegie-Mellon University Pittsburgh PA Software Engineering Institute,USA,2006.

附录X.

YUAN Sheng-Hao, M.S. His research interests include formal method and theorem proving.

YANG Zhi-Bin, Ph.D., associate professor. His research interests include safety-critical software and formal verification.

ZHANG Bo-Lin, M.S. His research interests include formal verification and software engineering.

ZHOU Yong, Ph.D., associate professor. His research interests include software engineering, formal methods.

XUE Lei, senior engineer. His research interests include embedded software verification, embedded software system design.

Bodeveix Jean-Paul, Ph.D., professor. His research interests include real-time system and formal method.

Filali Mamoun, Ph.D., senior research fellow. His research interests include real-time system and formal method.

Background

Compiler verification is a fundamental problem of computer science, especially in the computer system. It also plays a vital role in the design of safety-critical software in order to guarantee the correctness of all kinds of transformation processes, including code generation. For instance, CompCert is one of the most classical verified compilers, which comes with a mathematical, machine-checked proof that the generated executable code behaves exactly as prescribed by the semantics of the source program. Recently, many research focus on the compilation verification of synchronous languages, such as Vélus and L2C. However, the existing studies mainly concern the semantic preservation of sequential code generation from synchronous languages, they pay a little attention to the semantic preservation of multi-threaded code generation.

In this paper, we present a verified multi-threaded code generation method for the synchronous language SIGNAL. The main advantage of the method is to support multi-threaded code generation, comparing with the existing related works. Firstly, the method defines the structured operational semantics of multiple intermediate languages in code generation; secondly, it formalizes the algorithms and mapping rules in the code generation process; thirdly, it proves the semantic consistency between the SIGNAL program after clock calculus and the target C program after translation based on the bisimulation equivalent relation.

The research in the paper is a part of the National Natural Science Foundation of China (61502231); The National Key Research and Development Program of China (2016YFB1000802); National Defense Basic Scientific Research Project under Grant of China (JCKY2016203B011); Natural Science Foundation of Jiangsu Province (BK20150753); the Fundamental Research Funds for the Central Universities(NP2017205); Foundation of Graduate Innovation Center in NUAA(kfjj20181603).