时钟为 *base*.根据时钟演算规则,变量 *c* 的时钟为 *base* on *x*(与 *counter* 调用中的两个参数的时钟相同),而表达式(*pt* when not *x*)的时钟为 *base* on not *x*;*x*,*t*,*pt* 等变量的时钟均为 *base*.表达式 merge *x c* (*pt* when not *x*)的作用是将时钟分别为 *base* on *x* 和 *base* on not *x* 的两个表达式 *c* 和(*pt* when not *x*)归并为一个时钟为 *base* 的表达式.这里,需要注意的是,when 算子会使时钟变慢,而 merge 算子会使时钟变快.

在得到各变量以及表达式的时钟后,when 算子实际上就可以消去了.翻译的结果就是将等式(左边的变量和右边的表达式具有相同的时钟)的执行加上时钟为真的条件,生成 if 条件语句.这一条件的构成是将时钟表达式中所有的时钟变量(或者其否定)进行合取,而基本时钟 *base* 可认为是 true.例如,若时钟为 *base* on not *x*,则条件即为 not *x*.

对于 merge 算子,则会翻译为 if-else 条件语句或 case 语句.例如,图 3 中的 merge *x c* (*pt* when not *x*),Vélus 会翻译为类似于 if (*x*) {*c*=…} else {*pt*=…}之类的语句.在 L2C 中也类似,不同的是,L2C 的设计要支持枚举型时钟,所以目前选择了一种更为一般的情况,会将 merge 表达式翻译至 case 语句.具体细节可分别对应于附录中图 4 所示右半边的 if 语句和图 5 所示右半边的 case 语句.

顺便,Vélus 在生成 Obc 表示后会针对这些 if 语句进行 Fusion Optimization 的优化处理(如图 1 所示),使条件相容(多为时钟相同)的分支语句序列进行合并,这样仅需一次条件判断即可.目前,L2C 尚未开展此类优化工作.这个问题,后面还会提到.

(b) 时态算子的消去

Vélus 经过 Normalization 变换,或者 L2C 经过 LustreVGen 和 LustreSGen 过程,已确保了每个 fby 表达式只出现在专用等式的右端,而没有嵌在其他表达式中.图 3 中的所有 fby 表达式均已满足这一要求,对应的等式为:节点 *counter* 中的 *f*=true fby false 和 *c*=1 fby *n*,节点 *rising* 中的 *ps*=true fby *s*,以及节点 *tracker* 中的 *pt*=0 fby *t*.我们以 *tracker* 中的 *pt* = 0 fby *t* 为例解释 fby 算子的消去.

每个 fby 对应一个时态变量,用于记录历史状态.对于图 3 中节点 *tracker* 的含 fby 等式 *pt*=0 fby *t*,这一变量为 *pt*.无论是 Vélus 还是 L2C,都会将 *pt* 包装为节点 *tracker* 内可访问的特殊变量.下一小节会看到,Vélus 会将 *pt* 看作节点 *tracker* 所对应的类的属性变量,每个 *tracker* 实例都会维护一份 *pt* 的存储,可以保存以前周期的信息,而其他普通变量(比如 *t*)只能看作节点 *tracker* 所对应的类的方法的内部变量.类似地,L2C 会将 *pt* 看作与每个 *tracker* 的实例相关联的某个全局结构体的域变量,而其他普通变量(比如 *t*)不在这个全局结构体的域中.

若从对应到 Vélus 翻译后的 Clight 代码角度来看,图 4 所示的结构体类型 *tracker* 包含了域变量 *pt*,对应于上述节点 *tracker* 所对应的类的属性变量 *pt*.若从对应到 L2C 翻译后的 Clight 代码角度来看,图 5 所示的结构体类型 *outC_tracker* 包含的域变量 *acg_fby*4,对应于上述节点 *tracker* 所对应的类的属性变量 *pt*.这种时态变量在 L2C 中是自动产生的新变量,比如对于图 3 所示的程序,将会产生 *acg_fby*1、*acg_fby*2、*acg_fby*3 和 *acg_fby*4,分别对应于该程序中的 4 个 fby 的时态变量 *f*、*c*、*ps* 和 *pt*.

对于每个 *tracker* 实例中 *pt*=0 fby *t* 的执行过程,Vélus 和 L2C 的处理是类似的.在该实例执行的第 1 个激活周期开始时,会将相应于这一实例的时态变量 *pt* 的值置为 0,而在之后的每个激活周期开始时会将 *pt* 的值置为前一个激活周期结束时 *pt* 的值.在每个激活周期的结束阶段,会将在该周期中已经计算好的 *t* 的值存入 *pt* 的存储空间,存入之后在该周期不会再有对 *pt* 的读取.

若从对应到翻译后的 Clight 代码角度来看,图 4 中省略了 reset 相关的部分,而从图 5 右半边上部的 if 语句来看,我们会发现:第 1 个激活周期(*acg_init* 为 1 时)开始时,*pt* 会被置为 0,而在之后的每个激活周期开始时会将 *pt* 的值置为相应时态变量 *acg_fby*4 在前一个激活周期结束时的值.关于每个激活周期结束时对时态变量的修改,图 4 中是由最后一句(* self).*pt*=(* out).*t* 来完成的,而在图 5 中是由倒数第 2 句(*outC).*acg_fby*4=(*outC).*t* 来完成的.

另外,L2C 还支持三元的 fby 算子,其实现更加复杂一些.L2C 的实现方案是将自动产生的新增时态变量(类似上述的 *acg_fby*4)扩展为数组,其大小即为三元 fby 比起前面讨论的二元 fby 多出来的常量参数值.因 Vélus 目前不支持三元 fby 算子,所以我们不在这里讨论相应的翻译细节.

除了二元和三元 fby 算子,L2C 还支持传统 Lustre 的时态算子 pre.虽然 Vélus 目前未涵盖 pre 算子,但其实现与 fby 雷同.

(c) 同步数据流范型转换为串行命令式范型

在处理好前述几个核心同步数据流语言特征翻译的基础上,再解决好流数据对象的表示及其周期性处理的环节,基本上就实现了同步数据流范型向串行命令式范型的转换.在 Vélus 中,这种范型转换体现在生成 Obc 中间表示.相应地,在 L2C 中,这种转换导致了 LustreC 中间表示的产生.

Vélus 的 Obc 是一种基于对象的中间表示,用于封装从同步数据流代码到命令式代码的翻译结果.在翻译到 Obc 时,每一个节点会对应到一个类.这个类中包含的属性有两类:一类是该节点中的所有时态变量(每个 fby 对应一个这样的变量);另一类是该节点中调用其他节点的实例(注:不允许调用到本节点的实例).这两类属性都是有状态的,即要维护其以前周期的历史信息(注:节点实例内部也可能有自己的时态变量).该类中包含的方法有两个:一个是 reset,用于初始化时态变量属性以及调用实例,仅在第 1 个激活周期执行;另一个是 step,用来执行该节点内部的所有等式,每个激活周期都会被重复调用.每个节点的输入输出参数以及局部变量均在 step 方法中得以体现.

与 Vélus 不同,L2C 从同步数据流代码到命令式代码的翻译结果(LustreC 中间表示),并未将每个节点对应的方法及 reset 方法与它们要处理的状态数据(时态变量和调用实例)封装在一起,而是采用了类似于 C 程序的组织方式.每个节点对应 C 的一个函数,它的每个实例(对应于程序中实际调用该节点的位置)对应于一个结构体,其中封装了对应的时态变量和要调用的其他节点实例.为方便每个周期的循环调用,每个节点的输入输出参数变量也会被封装在一个该节点专用的结构体中,对于主节点要封装所有输入和输出变量,而对于其他节点仅封装输出.因为有可能有多个输出,所以对应到 C 时用一个结构体类型的变量存放多个返回值,所以,所有节点的输出变量均被分装在结构体中.主节点的输入变量要作为外部输入,为方便起见,也被封装在结构体中.同时,每个节点均有一个对应的 reset 函数.由此可见,虽然没有对每个节点对应的函数和 reset 函数以及它们要处理的数据对象进行封装,但这些内容均包含在生成的 LustreC 代码中,如同 C 语言的代码结构.

从翻译后的 Clight 代码可以看到一些具体的细节,如图 4 和图 5 所示.

(d) 翻译至 Clight

综合前面几个核心同步数据流语言特征的翻译步骤,应该能够看明白从图 3 所示的 Lustre 源程序分别经 Vélus 和 L2C 翻译至图 4 和图 5 所示的 Clight 代码(二者虽然形式上不同,但体现的行为是相通的),从而也可以了解最后阶段的这一核心翻译步骤所起到的作用.

在这个阶段,Vélus 就是将 Obc 中间代码的语法设法与翻译后的 Clight 代码的语法对应起来.L2C 也类似,通过 CtempGen 完成从 LustreC 到 Ctemp 的翻译,又通过 ClightGen 将 Ctemp 中的 memcpy 语义翻译为内存拷贝函数,最终翻译至 Clight 代码.L2C 后面这一步骤(从 Ctemp 到 Clight)在 Vélus 目前的版本中没有对应的需求.这些翻译步骤的图示可分别如图 1 和图 2 所示.

5 同步数据流语义与翻译正确性验证

这部分内容的准确论述需采用形式化或半形式化方式,但受限于篇幅,本文仅就其要点进行非形式描述.

(a) 同步数据流语义

在从 Lustre 到 C 的转换过程中,经历了同步数据流语义逐步向串行命令式语义的过渡.在现阶段,Vélus 的动态语义的形式化定义是从 SN-Lustre 中间表示开始的,在 Obc,中间表示被变换至一种基于对象的串行命令式语义;相应地,L2C 的动态语义形式化定义是从 LustreS 中间表示开始的,在 LustreC,中间表示彻底变换为一种类 C 的串行命令式语义.这些中间层语言所处位置,如图 1 和图 2 所示.

可以看出,Vélus 的同步数据流语义消去仅在一个阶段完成,而 L2C 则经历了多个阶段逐步消去.由于 L2C 的源语言特征相比 Vélus 要复杂很多,实践过程决定了跨多层的语义过渡是必然的选择.

Vélus 在定义 SN-Lustre 语义时已完成排序(scheduling),因此无需考虑同步数据流语言的数据流并发特

性.L2C 是在 LustreS 层进行排序(toposort),在 LustreS 层有串行语义定义(节点内部的所有等式相互之间存在全序关系),同时也定义有超粗粒度的并发语义(以等式为粒度,等式操作是不可分割的原子操作,等式之间存在由数据依赖所确定的偏序关系).

Vélus 和 L2C 的同步数据流语义都选择了基于操作语义来定义(虽然也存在看似很不错的基于指称语义的基础工作[33]).在 Lustre 语言的原始论文[11]中,作者给出了 Lustre 语言的基本操作语义规则.Vélus 和 L2C 的操作语义定义首先要与学界公认的这些工作相符,同时也应考虑实现的因素(交互式定理证明的方便性).比如,采用 Coinductive 来定义流看似是非常自然的(因为要处理的是 infinite 对象),然而无论是 Vélus 还是 L2C 均放弃选择这种方法,L2C 团队当初遇到的问题主要是因为采用 Coq 实现整体任务需求的技术难度无法预测.

在 Vélus 和 L2C 的同步数据流语义中,对于流数据对象的基本理解是一致的,均将其看作是从自然数到取值论域的函数.比如,可以将一个流 s 在第 n 个周期的值记为 $s(n)$.然而,在具体实现上,L2C 采取了一些不同的措施.特别是针对同步数据流语义中最具代表性的 fby 算子.

在定义 fby 算子的语义时,Vélus 使用了一个辅助算子 hold,会将所处理的流数据对象在上一个激活周期的取值保持到其后续的无效周期,这样,若当前周期是激活周期,则 fby 运算的结果就相当于当前周期 hold 运算的结果.这一思想在 L2C 项目开展的初期用来实现一种时钟归一化(clock unifying)算法[34],其中定义了类似于 hold 的算子,并用来刻画 fby 算子的语义.然而,在后来的工作中,L2C 摈弃了这种做法.主要原因是,这种方法仅适合于定义二元 fby 算子,而难以推广到实际应用中很有用的三元 fby 算子.L2C 对二元和三元 fby 算子的语义定义采用了一致的框架,定义一个大小为 n 的循环缓冲区,并设法限定仅在激活周期可访问缓冲区,这里,n 即为三元 fby 算子的第 3 个参数(对于二元 fby 算子,n=1).

定义同步数据流语义的另一个重要方面,就是语义环境中要保留子环境的状态,这些子环境对应于当前环境下调用其他节点而形成的实例环境.保存这些子环境的原因是因为存在时态变量(见第 4.2 节),需要保存历史状态,这就导致了在调用的节点实例返回时不能像传统的函数调用那样释放掉变量的存储空间.因而,我们必须要定义并维护一个树状的语义环境,这对语义定义以及相应的证明来说,难度增加了许多,但这是我们必须要应对的.L2C 在定义语义时对是否是时态变量进行了区分,以方便对应到 Clight 的语义环境中.Vélus 在文献[15]中对这种树状语义环境也有许多描述,但仅从论文无法了解其进一步的实现详情.

另外,L2C 因支持高阶迭代算子以及更多的数组和结构体算子,使语义定义的难度增大许多,但相关细节不属本文所要讨论的范围.L2C 的多阶段翻译架构(如图 2 所示)正是为了适应各种各样的语义复杂度以及与 Clight 语义之间的巨大差异,同样,图 2 所示的许多翻译步骤也不在本文所讨论的议题之内.

(b) 翻译正确性验证

在形式化定义动态语义阶段,Vélus 和 L2C 的各阶段翻译正确性的验证目标与 CompCert 所建议的阶段翻译正确性目标从形式上是一致的,都是要证明一种从高层到低层单向的语义模拟等价关系.这种模拟等价关系之所以可以是单向的,是由于每一层语言均具有确定的语义(deterministic semantics).

对于同步数据流语义来说,这种单向的语义模拟等价关系可大致描述为:任意数据流程序 G 的每个节点 f,若可以将输入流 xs 映射到输出流 ys,则对于翻译后得到的程序 G'中与 f 对应的节点或函数 f'以及这一层次中相应于 G'的初始化函数 reset,如果 G'在其第 1 个激活周期先执行 reset 后执行 f',而在其他后续激活周期中都重复执行 f',那么同样可以将 xs 映射到输出流 ys.注意:这一命题中的 G、f、xs 和 ys 都是任意的.

虽然各阶段要证明的单向语义模拟等价关系形式上看都是相似的,但具体到不同阶段,其含义却都有所不同.首先,各个阶段的语法和语义互不相同;其次,每个阶段可能有或没有语法层面的 reset 函数,如果有的话,不同阶段的 reset 函数也可能有所不同;以及还有其他各种不同,这里不再赘述.

一般来说,从 Lustre 源程序到 Clight 程序的各阶段翻译过程中,同步数据流语言的语法及语义特征逐渐被消去,C 语言语法及语义特征逐渐增加.在定义上述单向语义模拟等价关系时,所消失的语义特征描述通常会由新增加的语法特征来弥补.比如前面提到的 reset 特征,在较高的语言层次,语法层面并没有对应的函数,所以是在语义定义中体现其行为,而在靠后的语言层次(比如图 2 所示的 LustreE2 层)已有语法层面的 reset 函数,就不

会也不可能通过语义定义来体现 reset 行为.

因 Vélus 和 L2C 的编译框架之间差异较大(如图 1 和图 2 所示),可以想象到二者在翻译正确性验证方面存在较大的差异.这种差异体现在各个方面.这里仅列举一例.如果考虑第 4.1 节~第 4.3 节提到的核心翻译步骤,在 Vélus 中仅由 Translation 一个阶段完成(如图 1 所示),而在 L2C 中则是分散到从 LustreR1 到 LustreD 的许多阶段实现的(如图 2 所示).一般来说,一个翻译阶段所负责的工作越多,其正确性证明的难度也越大,重用性也越差.然而,另一方面,翻译阶段也不能太多,否则会导致语言层次增加太多,反而加重维护的负担.因此,应根据源语言的规模合理设计翻译阶段的数量.

Vélus 中 SN-Lustre 之前的阶段未定义动态语义,这些阶段所保持的特性主要是静态方面的.多数这些阶段,甚至从 SN-Lustre 到 Obc 的过程(Translation),基本上是继承前人(Bertails、Biernacki 和 Auger 等人)的工作.Bourke 等人的论文[15]对这些静态语义相关阶段的介绍很少,同时,Vélus 未开源,所以具体继承情况的详情未能很好地得到了解,从 Biernacki 和 Auger 等人的论文[14,27]仅可了解到其中的一些.

Vélus 的语法分析器是 Bourke 等人重新写的,它基于 Jourdan 等人提出的对 LR(1)自动机进行确认并对确认程序进行验证的方法[35],因此可以认为是一个经过形式化验证的语法分析器.L2C 中,最近刚完成与此平行的工作[32],并以编译选项的方式集成到了开源 L2C 编译器[31]中.

对于 Vélus 中的类型、时钟检查(elaboration)以及接下来的规范化过程(normalization),仅了解 Bertails、Biernacki 和 Auger 等人在 Coq 中实现了这些过程以及进行了验证,但他们的论文中未提到是如何进行验证的.L2C 中与此对应的过程(LustreWGen、LustreVGen 和 LustreSGen)的验证在目前的开源版本中尚未发布,相关工作仍在完善和整理之中.这些翻译过程在 Vélus 和 L2C 编译框架中所处位置,如图 1 和图 2 所示,下同.

Vélus 对于排序过程(Scheduling)的验证是基于翻译确认的方法[16],具体实现在相关论文[14,15,27]中未加介绍.L2C 中的排序过程(Toposort)是经过严格形式化验证的,证明了经过排序后的 LustreS 程序的串行语义行为,与排序前 LustreS 程序的等式粒度并发语义所刻画的行为是兼容的.从实现角度来看,对排序过程的形式化验证要比翻译确认的工作量大很多,所以从通常的观点来看,是否有必要这样做是值得商榷的.但 L2C 的这项工作,对于在项目开展初期积累团队成员在使用 Coq 进行编程和验证的经验以及检查 LustreS 操作语义定义的合理性等方面起着重要的作用.

6 实现与性能

Vélus 和 L2C 的开发环境与 CompCert[2,36]一致,有验证需求的部分均在 Coq[3,37]工具之中实现,并自动抽取得到相关的 Ocaml 代码.词法分析器的 Ocaml 代码由 Ocamllex 自动产生.语法分析器使用了 Ocamlyacc,或者 Menhir[38],前者针对未经验证的语法分析器选项,后者针对经过形式化验证的语法分析器选项.驱动程序以及其他辅助工具(如各层的语法树或代码的格式输出程序)通过手工编写 Ocaml 代码实现.

由于 Lustre 没有标准测试集,在 Bourke 等人的论文[15]中从前人相关工作中选取了 14 个有代表性的程序进行性能测试,在开源 OTAWA 框架[39]下,对 Vélus 以及其他一些 Lustre 编译器(Lustre V6 和 Heptagon 与 CompCert 或者 GCC 的组合)的最坏执行时间(worst-case execution time,简称 WCET)指标进行了测试评估.CompCert 的版本是 2.6,GCC 的版本是 4.8.4,目标体系结构是 armv7-a(含硬件浮点单元 vfpv3-d16).

我们从 Bourke 那里获得了文献[15]中使用的全部 14 个测例程序源码,并在与上述测试环境基本相同的配置下(配置选项为-march=armv7-a-mfloat-abi=softfp-mfpu=vfpv3-d16),针对 L2C 完成了测例程序的评估测试.我们将得到的测试结果汇总于表 2.

表 2 中的第 1 列是 14 个测例程序的名称,第 2 列是 L2C 和 CompCert 2.6 组合编译器的测试结果(WCET 值),第 3 列是从文献[15]中摘录的 Vélus 测试结果,第 4 列是 L2C 和 GCC4.8.4(加-O1 优化选项)组合编译器的测试结果,最后一列是 Lustre V6 和 GCC4.8.4(加-O1 优化选项)组合编译器的测试结果.

Table 2 WCET estimates in cycles for an armv7-a/vfpv3-d16 target

表 2 周期为单位的 WCET 评估值(目标处理器:armv7-a/vfpv3-d16)

测例程序名	L2C+CompCert	Vélus	L2C+gcc-O1
avgvelocity	515	315	275
count	90	55	55
tracker	1 035	680	580
pip_ex	6 620	4 415	2 745
mp_longitudinal	7 415	5 525	3 140
cruise	3 200	1 760	1 645
risingedgeretrigger	640	285	265
chrono	250	410	80
watchdog3	1 080	610	435
functionalchain	17 565	11 550	7 795
landing_gear	14 325	9 660	5 955
minus	1 555	890	560
prodcell	3 315	1 020	1 190
ums_verif	4 210	2 590	855

从表 2 的结果粗略来看,L2C+CompCert 的性能比 Vélus 低 50%左右.L2C 性能不如 Vélus 的原因,目前能想到的主要是因为 Vélus 在 Obc 中间表示进行了 if 语句的 Fusion Optimization 优化(如图 1 所示),而目前 L2C 未作任何优化,CompCert 目前也没有支持有关的优化.

附件 A.1 和 A.2 分别给出了测例程序 tracker(源码如图 3 所示)经 Vélus 和 L2C 编译生成的 Clight 代码,如图 4 和图 5 所示.从图 5 可以看出,L2C 生成的代码中包含了大量可融合的 if 语句.从图 4 可以看出,Vélus 生成的代码中时钟相同的操作已被合并到相同的条件分支下.

表 2 的第 4 列给出了 L2C 和 GCC4.8.4(加-O1 优化选项)组合编译器的测试结果.从中可以看出,测例程序 tracker 的 WCET 值比起第 2 列有显著降低(由 1 035 降至 580),其他测例程序的情况也是如此,性能也优于 Vélus 的测试结果(见第 3 列).在 GCC 的“-O1”优化选项中,包含了针对 if 语句的“-fif-conversion”优化,可以缩短 if-then 语句所花费的时间.

Lustre V6 编译器设计时也未过多考虑相关优化,主要是寄望于 C 编译器会负责这些优化工作(L2C 也曾有类似的考虑,但更主要的是在开发 L2C 企业版时,用户方出于可追溯性的硬性要求,原则上不允许对代码进行编译优化,这种理念一直延续至今,但不一定适合各种场合).从文献[15]的测试结果(如该文中图 12 第 6 列所示)看,Lustre V6 和 CompCert 组合的编译器对于各测例程序的 WCET 性能指标均不如 L2C 和 CompCert 组合的编译器.目前,我们对 Lustre V6 和 CompCert 组合编译器的性能测试工作也在进行之中.

7 总 结

Vélus 和 L2C 都是基于辅助定理证明器(Coq)构造 Lustre 可信编译器的长线项目,它们都是先将源语言翻译至 Clight,并与 CompCert 编译器实现衔接.Vélus 和 L2C 课题组在立项目标、发展过程和方向、源语言需求以及工作基础等方面都不同,因而导致 Vélus 和 L2C 编译器在许多方面有明显的差异,形成了各自的特色.

本文从基本情况、源语言特性、编译器结构、核心翻译步骤、同步数据流语义与翻译正确性验证以及实现与性能等多个角度对 Vélus 和 L2C 进行了较为深入的分析与比较.当然,也不排除漏掉一些比较重要的方面,今后会通过其他机会加以补充.

据 Bourke 介绍,Vélus 项目组近期在开展将有限自动机融入 Vélus 的研究工作.Scade[8]工具中包含对有限自动机的支持,其目的是增加控制流特性.L2C 项目组目前尚无计划开展类似的工作.

References:

[1] Knight, JC. Safety critical systems: Challenges and directions. In: Proc. of the 24th Int'l Conf. on Software Engineering, ICSE 2002. 2002. 547–550.

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

[3] Bertot Y, Castéran P. Interactive theorem proving and program development—Coq'Art: The calculus of inductive constructions. In:

Proc. of the EATCS Series on Theoretical Computer Science. Springer-Verlag, 2004.

[4] Morrisett G. Technical perspective: A compiler's story. Communications of the ACM, 2009,52(7):106.

[5] Yang XJ, Chen Y, Eide E, Regehr J. Finding and understanding bugs in C compilers. In: Proc. of the 2011 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2011). 2011. 283–294.

[6] Leroy X. Formal certification of a compiler back-end or: Programming a compiler with a proof assistant. In: Proc. of the 33rd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. 2006,41(1):42–54.

[7] Simulink home. http://www.mathworks.com/products/simulink/

[8] Scade home. http://www.ansys.com/products/embedded-software/ansys-scade-suite

[9] Berry G, Gonthier G. The Esterel synchronous programming language: Design, semantics, implementation. Science of Computer Programming, 1992,19(2):87–152.

[10] Maraninchi F. The ARGOS language: Graphical representation of automata and description of reactive systems. In: Proc. of the IEEE Workshop on Visual Languages. Kobe, 1991.

[11] Caspi P, Pilaud D, Halbwachs N, Plaice J. Lustre: A declarative language for programming synchronous systems. In: Proc. of the 14th ACM Symp. on Principles of Programming Languages, POPL'87. Munchen, 1987.

[12] Halbwachs N, Caspi P, Raymond P, Pilaud D. The synchronous dataflow programming language LUSTRE. Proc. of the IEEE, 1991,79(9):1305–1320.

[13] Le Guernic P, Gautier T, Le Borgne M, Le Maire C. Programming real time applications with SIGNAL. Proc. of the IEEE, 1991, 79(9):1321–1336.

[14] Biernacki D, Colaco J, Hamon G, Pouzet M. Clock-directed modular code generation of synchronous data-flow languages. In: Proc. of the ACM Int'l Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES). Tucson, 2008.

[15] Bourke T, Brun L, Dagand PÉ, Leroy X, Pouzet M, Rieg L. A formally verified compiler for Lustre. In: Proc. of the Programming Language Design and Implementation. 2017. 586–601.

[16] Pnueli A, Siegel M, Singerman E. Translation validation. In: Proc. of the TACAS'98. Lecture Notes in Computer Science 1384, 1998. 151–166.

[17] Milner R. Calculi for synchrony and asynchrony. TCS, 1983,25(3).

[18] Caspi P, Halbwachs N. A functional model for describing and reasoning about time behaviour of computing systems. Acta Informatica, 1986,22:595–697.

[19] Halbwachs N. A synchronous language at work: The story of Lustre. In: Proc. of the 2nd ACM/IEEE Int'l Conf. on Formal Methods and models for Co-Design, MEMOCODE 2005. Washington: IEEE Computer Society, 2005.

[20] Lustre-V6 home. http://www-verimag.imag.fr/Lustre-V6.html

[21] Wadge WW, Ashcroft EA. Lucid, the Dataflow Programming Language. San Diego: Academic Press Professional Inc., 1985.

[22] Lucid Synchrone home. http://www.di.ens.fr/~pouzet/lucid-synchrone/

[23] Caspi P, Pouzet M. A functional extension to LUSTRE. In: Proc. of the 8th Int'l Symp. on Languages for Intensional Programming, ISLIP'95. Sydney, 1995.

[24] Paulin C, Pouzet M. Certified compilation of scade/Lustre. 2006. http://www.lri.fr/paulin/lustreincoq.pdf

[25] Bertails A, Biernacki D, Paulin C, Pouzet M. A certified compiler for the synchronous language Lustre. In: Proc. of the Presentation of TYPES 2007. 2007. http://users.dimi.uniud.it/types07/slides/Bertails.pdf

[26] Auger C. Compilation Certifiée de SCADE/LUSTRE [Ph.D. Thesis]. Université Paris Sud, 2013.

[27] Auger C, Colaçob J-L, Hamon G, Pouzet M. A formalization and proof of a modular Lustre compiler. 2012. http://www.di.ens.fr/~pouzet/cours/mpri/cours4/scp12.pdf

[28] Shi G, Wang SY, Dong Y, JI ZY, Gan YK, Zhang LB, Zhang YC, Wang L, Yang F. Construction for the trustworthy compiler of a synchronous data-flow language. Ruan Jian Xue Bao/Journal of Software, 2014,25(2):341–356 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4542.htm [doi: 10.13328/j.cnki.jos.004542]

[29] Liu Y, Gan YK, Wang SY, Dong Y, Yang F, Shi G, Yan X. Trustworthy translation for eliminating high-order operation of a synchronous dataflow language. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):332–347 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4785.htm [doi: 10.13328/j.cnki.jos.004785]

[30] Shang S, Gan YK, Shi G, Wang SY, Dong Y. Key translations of the trustworthy compiler L2C and its design and implementation.

Ruan Jian Xue Bao/Journal of Software, 2017,28(5):1233–1246 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5213.htm [doi: 10.13328/j.cnki.jos.005213]

[31] Open L2C home. http://soft.cs.tsinghua.edu.cn:8000/

[32] Yang P, Wang SY. Survey on trustworthy compilers for synchronous data-flow languages. Computer Science, 2019,46(5):21–28 (in Chinese with English abstract).

[33] Paulin-Mohring C. A constructive denotational semantics for Kahn networks in Coq. In: From Semantics to Computer Science. Essays in Honour of Gilles Kahn, 2009. 383–414.

[34] Shi G, Zhang YC, Shang S, *et al*. A formally verified transformation to unify multiple nested clocks for a Lustre-like language. Science China-Information Sciences, 2019,62(1). http://scis.scichina.com/en/2019/012801.pdf

[35] Jourdan JH, Pottier F, Leroy X. Validating LR(1) parsers. In: Proc. of the Programming Languages and Systems—the 21st European Symp. on Programming, ESOP 2012. Lecture Notes in Computer Science 7211, Springer-Verlag, 2012. 397–416.

[36] CompCert home. http://compcert.inria.fr/

[37] Coq home. https://coq.inria.fr/

[38] Pottier F, Régis-Gianas Y. Menhir Reference Manual. 2018. http://gallium.inria.fr/~fpottier/menhir/manual.pdf

[39] Ballabriga C, Cassé H, Rochange C, Sainrat P. OTAWA: An open toolbox for adaptive WCET analysis. In: Proc. of the 8th IFIP WG 10.2 Int'l Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS 2010). Lecture Notes in Computer Science 6399, Springer-Verlag, 2010. 35–46.

附中文参考文献:

[28] 石刚,王生原,董渊,嵇智源,甘元科,张玲波,张煜承,王蕾,杨斐.同步数据流语言可信编译器的构造.软件学报,2014,25(2):341–356. http://www.jos.org.cn/1000-9825/4542.htm [doi: 10.13328/j.cnki.jos.004542]

[29] 刘洋,甘元科,王生原,董渊,杨斐,石刚,闫鑫.同步数据流语言高阶运算消去的可信翻译.软件学报,2015,26(2):332–347. http://www.jos.org.cn/1000-9825/4785.htm [doi: 10.13328/j.cnki.jos.004785]

[30] 尚书,甘元科,石刚,王生原,董渊.可信编译器 L2C 的核心翻译步骤及其设计与实现.软件学报,2017,28(5):1233–1246. http://www.jos.org.cn/1000-9825/5213.htm [doi: 10.13328/j.cnki.jos.005213]

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

附录

A.1

例:Vélus 生成的 Clight 代码.如图 4 所示.

```
struct tracker {
  int pt;
  struct d_integrator s;
  struct rising x;
  struct counter c;
  f=true fby false;
  c=1 fby n;
};

struct tracker$step {
  int p;
  int t;
};
```

```
void tracker$step(struct tracker *self,
    struct tracker$step *out, int acc, int limit)
{
    struct d_integrator$step out$s$step;
    register int step$n, s, c;
    register bool step$edge, x;
    d_integrator$step(&(*self).s, &out$s$step, acc);
    s=out$s$step.speed;
    (*out).p=out$s$step.position;
    step$edge = rising$step(&(*self).x, s>limit);
    x=step$edge;
    if (x) { step$n=counter$step(&(*self).c, 1, 1, 0);
             c=step$n; (*out).t=c;}
    else { (*out).t=(*self).pt;}
    (*self).pt=(*out).t;
}
```

Fig.4 Clight code procuded from Fig.3's *tracker* node by Vélus (the reset function being left out)

图 4 用 Vélus 编译图 3 中 *tracker* 节点生成的 Clight 代码(略去了 reset 函数)

A.2

例:L2C 生成的 Clight 代码.如图 5 所示.

```
typedef struct {
   int acc;
   int limit;
}inC_tracker;

typedef struct {
   int p;
   int t;
   bool acg_init;
   int acg_fby4;
   outC_counter acg_context1;
   outC_d_integrator acg_context3;
   outC_rising acg_context2;
}outC_tracker;

void tracker_reset(outC_tracker *outC)
{
   (*outC).acg_init=1;
   rising_reset(&(*outC).acg_context2);
   d_integrator_reset(&(*outC).acg_context3);
   counter_reset(&(*outC).acg_context1);

}

void tracker(inC_tracker *inC, outC_tracker *outC)
{
   int s;
   int pt;
   bool x;
   int c;
   int acg_L1;
   int acg_L2;
   bool acg_L3;
   int acg_L4;
   d_integrator((*inC).acc, &(*outC).acg_context3);
   s=(*outC).acg_context3.speed;
   (*outC).p=(*outC).acg_context3.position;
   if ((*outC).acg_init) {
      pt=0;
    } else {
      pt=(*outC).acg_fby4;
    }
   rising(s>(*inC).limit, &(*outC).acg_context2);
    x=(*outC).acg_context2.edge;
    if (x) {
       acg_L1=1;
    }
    if (x) {
      acg_L2=1;
     }
     if (x) {
       acg_L3=0;
     }
     if (!x) {
       acg_L4=pt;
   }
   if (x) {
     counter(acg_L1, acg_L2, acg_L3,
       &(*outC).acg_context1);
     c=(*outC).acg_context1.n;
   }
              switch (x) {
                 case 1:
        (*outC).t=c;
        break;
      case 0:
        (*outC).t=acg_L4;
        break;
     default:
         /*skip*/;
    }
   (*outC).acg_fby4=(*outC).t;
   (*outC).acg_init=0;
}
```

Fig.5 Clight code procuded from Fig.3's *tracker* node by L2C (with the reset function)

图 5 用 L2C 编译图 3 中 *tracker* 节点生成的 Clight 代码(含 reset 函数)

康跃馨(1993－),男,河南安阳人,硕士,主要研究领域为形式化验证.

甘元科(1983－),男,硕士,主要研究领域为形式化验证.

王生原(1964－),男,博士,副教授,CCF 高级会员,主要研究领域为程序语言与系统,程序验证,Petri 网应用.