计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201100080

基于程序转化的SCADE模型检测

冉 丹 陈 哲 孙 毅 杨志斌
南京航空航天大学计算机科学与技术学院 南京 211106
华东师范大学上海市高可信计算重点实验室 上海 200062
南京大学计算机软件新技术国家重点实验室 南京 210093
(RDan456@163.com)

摘 要 SCADE同步语言是一种常用的嵌入式系统程序设计语言。在航空、航天、交通等安全关键领域的装备研发中,SCADE同步语言通常被用于实现实时嵌入式自动控制系统。SCADE语言是工业级的开发工具,它源于Lustre语言,并在其基础上增加了更多的语言结构来精简代码。目前,相比Lustre语言,SCADE程序模型检测的学术研究相对落后。为此,文中提出了一种对SCADE程序进行模型检测的方法并实现了一款SCADE模型检测工具,该方法的核心思想是基于程序转化,即把SCADE程序经过词法分析、语法分析、抽象语法树生成与化简等操作最终转化为等价的Lustre程序,然后用JKind与SMT求解器完成模型检测。此外,通过理论推导和大量实验证明了工具的模型检测的正确性。实验结果表明,功能相同的两个SCADE和Lustre测试用例模型的检测结果相同,但SCADE程序的模型检测效率相对较低。

关键词:模型检测;安全有限状态机;词法分析;语法分析;抽象语法树;JKind
中图法分类号 TP311

SCADE Model Checking Based on Program Transformation

RAN Dan,CHEN Zhe,SUN Yi and YANG Zhi-bin
College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China
Shanghai Key Laboratory of Trustworthy Computing,East China Normal University,Shanghai 200062,China
State Key Laboratory for Novel Software Technology,Nanjing University,Nanjing 210093,China

Abstract SCADE synchronization language is a common programming language for embedded system. It is often used to realize real-time embedded automatic control system in the research of equipment in aviation,aerospace,transportation and other safety critical fields. SCADE synchronization language is derived from the Lustre language,which adds more language structure to simplify source code. However,compared with Lustre language,the development of model checking technology of SCADE synchronous language is relatively backward. In this paper,we introduce a method and suite for model checking of SCADE programs. The core idea of the method is based on program transformation,that is,the SCADE program is finally transformed into equivalent Lustre program after lexical analysis,syntax analysis,abstract syntax tree generation,simplification,and then use JKind to complete the model checking. Moreover,we prove the correctness of the suite's model checking result by theoretical deduction and a large number of experiments. Experimental results show that the SCADE and Lustre programs with the same function produce the same model checking results,but the efficiency of SCADE is lower.

Keywords Model checking,Safe state machine,Lexical analysis,Syntax analysis,Abstract syntax tree,JKind

1 引言

同步语言[1]是一种常用的嵌入式系统程序设计语言。常见的同步语言包括Esterel,Lustre,SIGNAL[2],SCADE[3]等,其中SCADE源于Lustre,并在其基础上新增了很多语法规则(包括安全状态机)来简化程序开发。在航空、航天、交通等安全关键领域的研发中,SCADE通常被用于实现实时嵌入式自动控制系统[4],这些领域的系统对安全性与

到稿日期:2020-11-09 返修日期:2021-04-12 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然基金委员会-中国民航局民航联合研究基金(U1533130);中央高校基本科研业务费人工智能+专项(NZ2020019);上海市高可信计算重点实验室开放课题,南京大学计算机软件新技术国家重点实验室开放课(KFKT2020B10)
This work was supported by the Joint Research Funds of National Natural Science Foundation of China and Civil Aviation Administration of China (U1533130),Fundamental Research Funds for AI(NZ2020019) and Open Project of Shanghai Key Laboratory of Trustworthy Computing,Open Project of State Key Laboratory for Novel Software Technology(KFKT2020B10).
通信作者:陈哲(zhechen@nuaa.edu.cn)

可靠性有很高的要求。

传统的软件测试技术对系统的安全性与可靠性评估是不可靠的，软件测试只能保证系统在测试用例上的正确性，无法保证测试用例之外的情况的正确性。使用模型检测方法可以弥补软件测试技术的不足，因为模型检测可以检测系统的某种特性在所有的系统状态上是否成立[5]。

目前，Lustre 同步语言程序的模型检测技术比较完善，有多个开源的模型检测工具，如 Kind，Kind2[6]，PKind[7]，JKind[8]等。

关于 SCADE 同步语言程序模型检测，Gacek 等于 2012 年开源了一个 SCADE 模型检测工具[9]。其基于程序转化的思想，首先将 SCADE 程序转化为 LAMA 程序，然后将 LAMA 程序转化为可满足性求解器的输入，最后将转化结果输入到可满足性求解器中完成模型检测。这是目前唯一针对 SCADE 程序的模型检测工具，是基于 Haskell 语言实现的，但自从发布后就没再维护过，安装时出现了不可修复的错误。通过剖析其源代码，发现其存在以下问题：1）它只支持 SCADE 的语法子集，不能解析 clock，merge，group，onrest 语句以及各种细分数据流类型等；2）LAMA 是一种非常简单的自动机语言，导致部分 SCADE 语句无法转化为 LAMA 程序；3）它只使用了有界模型检测和 K-归纳法两种模型检测算法，且整个求解过程使用单线程完成，因此模型检测的效率低。

综上，本文重点研究 SCADE 程序的模型检测，其实现思路是：基于程序转化的思想，将 SCADE 程序转化为 Lustre 程序，然后将 Lustre 程序输入到 JKind 完成模型检测。

本文的主要工作包括：1）提出了一种从 SCADE 到 Lustre 的转化算法，尤其是安全状态机的转化算法；2）从理论上证明了安全状态机转化算法的正确性；3）实现了 SCADE 模型检测的原型系统；4）开发了包括 887 个 SCADE 程序的测试集，并在该测试集上验证了本文实现的工具对 SCADE 程序模型检测的正确性，同时从理论上对比了该工具与已存在的工具的性能，通过实验还评估了工具的性能，与功能相同的 Lustre 测试用例相比，该工具的效率稍微比 JKind 低。

2 理论知识

2.1 词法分析、语法分析与抽象语法树

若要实现对 SCADE 程序的模型检测，首先需要保证程序符合 SCADE 语言的语法规则。在编译器中，词法分析器与语法分析器的一个功能是检测源程序是否有语法错误[10]。抽象语法树[11]（Abstract Syntax Tree，AST）是源程序的一种抽象表示。抽象语法树上每个节点都代表一种语法结构，它的数据结构需根据源语言的语法结构进行设计。

手动设计词法、语法分析器是比较困难的，而且开源的词法、语法分析器生成工具都很成熟。常见的词法分析器生成工具如 Lex 和 Flex 等，常见的语法分析器生成工具如 Bison 等[12]。

2.2 安全状态机

安全状态机（Safe State Machine，SSM）是特殊的有限状态自动机，简称有限自动机（Finite State Machine，FSM）。FSM 可以用五元组 $M=(Q,\Sigma,\delta,q_0,F)$表示[13]，其中，$Q$ 表示 FSM 的所有状态集合；Σ 是字母表，表示 FSM 输入字符的集合；$\delta:Q\times\Sigma\rightarrow Q$ 表示状态转化函数集合；$Q_0\subseteq Q$ 表示 FSM 的初始状态集合；$F\subseteq Q$ 表示 FSM 的终止状态集合。

Moore 自动机和 Mealy 自动机是一种特殊的 FSM，它们的每个状态都可以有数据输出。Moore 自动机用六元组 $M=(Q,\Sigma,\Delta,\delta,\lambda,q_0)$表示，其中 Q,Σ,δ 和 FSM 中的 Q,Σ,δ 意义相同；Δ 表示 Moore 自动机的输出字母表；$\lambda:Q\rightarrow\Delta$ 为输出函数，$\lambda(q)=a$，其中 $q\in Q,a\in\Delta$，表示 Moore 自动机在状态 q 时输出 a；$q_0\in Q$ 表示 Moore 自动机的初始状态集合。

Mealy 自动机用六元组 $M=(Q,\Sigma,\Delta,\delta,\lambda,q_0)$表示，其中 $Q,\Sigma,\delta,\Delta,q_0$ 和 Moore 自动机中 $Q,\Sigma,\delta,\Delta,q_0$ 的意义相同；$\lambda:Q\times\Sigma\rightarrow\Delta$ 是输出函数，$\lambda(q_i,in)=a$，其中 $q\in Q,in\in\Sigma,a\in\Delta$，表示 Mealy 自动机处于状态$q_i$且读入 in 时，输出 a。

SSM 中有且仅有一个初始状态，没有终止状态，它的状态间迁移包括强迁移与弱迁移两种方式。强迁移和弱迁移的区别是：假设 SSM 的状态为S_0，从S_0出发只有一个可达状态为S_1。对于强迁移，如果迁移条件满足，则 SSM 状态变为S_1，执行S_1状态中的代码。对于弱迁移，如果迁移条件满足，则 SSM 状态变为S_1，但是执行S_0状态中的代码。如果条件不满足，则 SSM 的状态不变，执行S_0状态中的代码。

2.3 模型检测

模型检测技术是一种自动化软件验证技术，它通过探索所有可能的系统状态[14]来证明某种安全属性在系统中是否一直被满足，也就是证明系统的所有状态都满足该属性。但是，处理庞大的系统状态是一个真正的挑战，如 Java 程序中，一个 int 类型的数据状态空间有 2^{32} 个，因此很多学者在处理系统状态空间方面进行了很多优化，如引入二元决策图[15]（Binary Decision Diagram，BDD）等。

在模型检测器开发中，为了避免手动处理程序的状态空间，通常采用另一种方法来实现模型检测。通过对源代码建模，将验证的特性和源程序提取为逻辑公式[16]。然后把逻辑公式输入可满足性求解器，利用模型检测算法实现模型检测。常见可满足性求解器有 Z3，C4V4，Yices2 等。

常用的模型检测算法有有界模型检测（Bound Model Checking，BMC）、k-归纳法（k-induction）[17]。BMC 采用穷举的方式判断逻辑公式在所有的系统状态上是否可满足。但是穷举所有的系统状态基本上不可能，因为资源（时间、服务器的内存等）是有限的，所以会规定求解的次数。使用 BMC 时会将整个逻辑公式取反，通过反证法的思想来判断该特性在系统中是否成立。k-induction 法采用数学归纳法来求证。通常模型检测器会结合两种方法，先使用 BMC，没有结果时再使用 k-归纳法求解。

3 SCADE 模型检测的实现

3.1 词法分析、语法分析的实现

使用 Flex 和 Bison 工具分别生成词法、语法分析器，开发者只需提供 scade_lexer. l，scade_parser. y 文件，scade_lexer. l 文件的内容包括 SCADE 的词法 Token 以及接受词法的正则表达式，scade_parser. y 中定义 SCADE 语言的语法规则和抽象语法树节点来生成代码。该文件经过 Flex 和 Bison 工具编

译器后，会生成词法、语法分析器 scade_lexer.cpp 和 scade_parser.cpp。

3.2　SCADE 到 Lustre 的转化算法

为了清楚说明 SCADE 到 Lustre 的转化算法，下文结合一个简单 SCADE 程序来进行阐述，该程序的伪代码如算法 1 所示。

算法 1

```
1. node even_odd(c:bool;i,n:int8) returns(o:int last=0;s:int;t:
                    real;r:bool)
2. sig a;
3. let
4.     assume n:i>0;
5.     r='a;
6.     s=fby(s,2,0);
7.     t=restart add every c();
8.     automaton even_times
9.       initial state EVEN
10.      unless if c resume ODD;
11.      let
12.      tel
13.      state ODD
14.      unless if c resume EVEN;
15.      let
16.        emit 'a;
17.      tel
18.      returns;
19. tel
20. node add() returns(x:real)
21. let
22.    x=1.00+0-> pre(x);
23. tel
```

SCADE 程序转化为 Lustre 的具体算法如下：

(1)数据流类型转化。Lustre 只包括 *bool*，*int* 和 *real* 3 种数据流类型。SCADE 对数据流类型进行了更详细的划分，包括整数类型 *int*，*int*8，*int*16，*int*32，*int*64，*uint*32，*uint*64 和浮点数类型 *float*，*float*32，*float*64。因此，转化算法保持程序中的 *bool* 类型不变，将整型转化为 *int* 类型，浮点数类型转化为 *real* 类型。

本例中，将数据流定义 *i*:*int*8 转化为 *i*:*int*。

(2)*fby* 语句转化。语句 *fby*(*b*;*n*;*a*)的含义是使用数据流 *a* 和 *b* 构造一个新的数据流，该数据流的前 *n* 个时钟的值与数据流 *a* 的初始值相等，$n+i$ 个时钟的值与数据流 *b* 的第 *i* 个时钟的值相等，其中整数 $n>0$，$i>0$。因此，转化算法将 *fby* 语句简化为包含 *n* 个 *pre* 操作符的语句：$a \rightarrow pre(a \rightarrow pre(\cdots pre(a \rightarrow pre b)\cdots))$，其中 *pre* 操作符返回数据流前一个时钟的值。

本例中，将 $s=fby(s;2;0)$ 转化为 $s=0 \rightarrow pre(0 \rightarrow pre(s))$。

(3)*when* 语句转化。它的语法是 *do when c*，其中 *c* 是一个 *bool* 类型的表达式，*do* 是一个表达式。该语句表示：如果 *c* 为 true，则执行 *do*，否则什么也不做。因此，转化算法将 *when* 语句转化为 *if-then-else* 语句：*if c then do else*。

本例中，将 $s=1$ *when c* 转化为 $s=$*if c then* 1 *else*。

(4)*merge* 语句转化。它的语法是 *merge*(*c*，do_1 *when c*，do_2 *when not c*)，其中 *c* 是一个 *bool* 类型的表达式，do_1 和 do_2 是表达式。该语句表示：如果 *c* 为 true，则执行 do_1，否则执行 do_2。因此，转化算法将 *merge* 语句转化为：*if condition then* do_1 *else* do_2。

本例中，将 $s=$*merge*(*c*，1 *when c*，2 *when not c*)转化为 $s=$*if c then* 1 *else* 2。

(5)*restart-every* 语句转化。它的语法是(*restart node every c*)(*params*)，其 *node* 为调用节点的标识符，*c* 是一个 *bool* 类型的表达式，*params* 为 *node* 节点调用的参数。该语句表示：如果 *c* 为 true，则让 *node* 回到初始状态，否则执行 *node*。因此，转化规则将 *restart-every* 语句转化为：*if c then initial else node*(*params*)，其中 *initial* 表示 *node* 节点的输出流的初始值。

本例中，将 $t=$*restart add every c*()转化为 $t=$*if c then* 1.00 *else add*()。

(6)*activity-every-initial-default* 语句转化。它的语法是 *i*.(*activity node every c*)(*params*)，*ii*.(*activity node every c default v*)(*params*)，*iii*.(*activity node every c initial default v*)(*params*)，其中 *node* 为调用节点的标识符，*c* 是一个 *bool* 类型的表达式，*params* 为 *node* 节点调用参数，*v* 表示语句的初始值。语句 *i* 表示：如果 *c* 为 true，则执行 *node*，否则就让 *node* 回到初始状态。语句 *ii* 表示：如果 *c* 为 true，则执行 *node*，否则值为 *v*。语句 *iii* 表示：如果 *c* 为 true，则执行 *node*，否则如果上一个时钟的 *c* 为 true，该时钟的 *c* 为 false，则该语句的值为 *v*；如果上一个时钟和该时钟的 *c* 都为 false，则该语句的值等于上一个时钟的值。因此，转化算法将语句 *i* 转化为：*if c then node*(*params*)*else initial*。其中，*initial* 表示 *node* 节点输出流的初始值。语句 *ii* 转化为 *if c then node*(*params*) *else v*。语句 *iii* 转化为 *if c then node*(*params*)*else* $v \rightarrow pre(value)$。其中，*value* 为上一个时钟该语句的值。

本例中，将 $t=$(*activity add every c*)()转化为 $t=$*if c then add*() *else* 1.00。

(7)*package-end* 语句转化。它的语法是 *package id* 〈*code*〉*end*，其中 *id* 是包的标识符，〈*code*〉表示包中的代码。SCADE 允许源代码位于不同的包中，但 Lustre 不允许。因此，转化算法将不同包中的程序移动到同一个包中，为包中的所有标识符加上 *id* 作为前缀，并将源程序中的 *package id* 〈*code*〉*end* 语句删除。

(8)*open* 语句转化。它的语法是 *open id*，其中 *id* 表示包的标识符。在 SCADE 程序中，如果一个包中的程序需要调用另一个包中的程序，则需要在该程序中声明要引用程序所在的包。在 *package-end* 语句转化中，已经将不同包中的程序移动到了同一个包中。因此，转化算法将 *open id* 语句删除。

(9)*last* 语句转化。它的语法是 *last'id*，其中 *id* 为被 *last* 语句使用的输出数据流标识符，该数据流的值在 SSM 的不同状态中共享。因此，转化算法将 SCADE 程序 *id* 数据流的定义转化为 *id* ：*type*，其中 *type* 是数据流类型，新增数据流定

义 *var last_id*：*type*,其中 *var* 是数据流定义的关键字,*last_id* 是数据流标识符;新增等式 *last_id*=*value*→*pre*(*id*),其中 *value* 是初始值,最后将程序中所有的 *last'id* 替换为*last_id*。

本例中,被 *last* 语句使用的输出数据流的定义为 *o*：*int last*=0,转化算法将其转化为新的定义 *o*：*int*;新增数据流定义为 *var last_o*：*int*,新增等式 *last_o*=0→*pre*(*o*),最后将所有的 *last'o* 替换为 *last_o*。

(10)*assume* 语句和 *guarantee* 语句转化。它的语法是 *assume*/*guarantee name*：*c*,其中 *name* 为断言的标识符,*c* 为 *bool* 类型的表达式,用于表示 *c* 必须为 true。*assume* 只能断言输入流,*guarantee* 只能断言输出流。Lustre 的断言用 *assert* 语句表示,它可以断言输入流和输出流。因此,转化算法将 *assume* 语句和 *guarantee* 语句转化为 *assert*：*c*。

本例中,将 *assume n*:*i*>0 转为 *assert*:*i*>0。

(11)*group* 语句转化。它的语法是 *group id* =(t_1,t_2,…,t_n),其中 *id* 是组的标识符,t_1-t_n 是数据流类型。*group* 语句用于定义一组输入数据流或输出数据流类型,组中的数据流类型和顺序与定义中的类型和顺序一致。因此,转化算法将 *group* 语句转化为多个数据流定义:*var* a_1：t_1,…,a_n：t_n。其中,a_1-a_n是新的数据流标识符。

(12)*sig* 语句转化。它只能用在 SSM 中,语法是 *sig* v_1,…,v_n,其中v_1-v_n为 *bool* 类型的信号流,信号流指在所有时钟上的默认值都为 false 的数据流,如果 SSM 的某个状态中有 *emit*‘v_i语句($1\leqslant i\leqslant n$),且当程序达到该状态时,v_i值为 true,否则为 false。因此,转化算法将 *sig* 语句转化为一组数据流定义 *var* v_1,…,v_n:*bool*,将涉及信号流的等式 *r* ='v_i转化为 $r=v_i$。

本例中,将 *sig a* 转化为 *var a*：*bool*,将 *r*='*a* 转化为 *r*=*a*。

(13)*emit* 语句转化。它的语法是 *emit* 'v_1,…,'v_n,其中 v_1-v_n是 *sig* 语句定义的信号流,用于将这些信号流的值变为 true。因此,转化算法将 *emit* 语句转化为:v_1=true;…;v_n=true。

本例中,将 *emit'a* 转化为 *a*=true。

(14)*where* 语句转化。它用于动态确定数据流类型,语法是 *node id* (v_1:tv_1;…;v_i:'*T*;…;v_m:tv_m) *returns* (r_1:tr_1;…;r_j:'*T*;…;r_n:tr_n) *where*' *T type*,其中 *id* 为节点标识符,v_1-v_m为输入数据流,tv_1-tv_m为类型,r_1-r_n为输出数据流,t_1-t_n为类型,*type* 可以为 *float*,*integer*,*numeric*。因此,转化算法将 *where* 语句转化为 *node id_type*(v_1:tv_1;…;*v_i type*;…;v_m:tv_m)*returns*(r_1:tr_1;…;r_j:*type*;…;r_n:tr_n)。

(15)SSM 的转化,它是 SCADE 特有的语法结构,将其转化为 Lustre 程序的转化算法,如算法 2 所示。

算法 2 安全状态机转化算法

输入:transform_state_machine(node * sm)

输出:Lustre 程序

1. begin
2. 从 sm 节点中提六元组 M=(Q,Σ,Δ,δ,λ,q_0)。其中,集合 Q,δ 中的元素数量为 n,m。
3. for i=0→n do
4. 给q_i状态编号q_i=i;q_i∈Q
5. end for
6. for i=0→n do
7. 构建 $node_i$,$node_i$的输入数据流为 Σ,输出数据流为 Δ,$node_i$中的代码为q_i中的代码。
8. end for
9. δ:Q×Σ→Q 构造“迁移矩阵”**Mr**,**Mr**[f,c,t,a,flag]。/* f 为出发状态,c 为迁移条件,t 为目的状态,a 为执行状态,flag 为目的状态的编码,针对于弱迁移 */
10. for i=0→m do
11. **Mr**[i]=[q_j,c_i,q_k,$action_i$,$flag_i$],0⩽j,k⩽n,q_j×c_i→q_k。
12. end for
13. 在 sm 节点中构建 int 类型数据流 temp_id,id 和等式 temp_id=0→pre(id)。
14. 在 sm 中构建等式 Δ=
15. for i=0→m do
16. if temp_id=**Mr**[i].f.id &&**Mr**[i].cond then a_i else
17. end if
18. end for
19. 在 sm 节点中构建等式 id=
20. for i=0→m do
21. if temp_id=**Mr**[i].f.id &&**Mr**[i].cond then **Mr**[i].flag else
22. end if
23. end for
24. 在 sm 节点中删除状态机语句
25. end

4 安全状态机转化正确性证明

经过转化算法,将 SSM 转化为两组 *if-then-else* 语句、多个节点和节点调用。FSM 是控制流语言,其本质可看作一组 *if-then-else* 语句。因此,可将转化后的两组 *if-then-else* 语句看作两个 FSM,其中判断下一个状态的自动机为 Moore 状态机。

定理 1[18] 如果 $M=(Q,\Sigma,\Delta,\delta,\lambda,q_0)$是 Moore 自动机,则一定存在一个与它等价的 Mealy 自动机。

定理 2[18] $\mathscr{L}$是某一个不确定自动机接受的语言,那么一定存在一个能接受 $\mathscr{L}$ 的确定自动机。

定理 3[18] 两个正则集的交集是闭包的。

现在需要证明经过 SSM 转化算法后,转化前的 SACDE 的 SSM 和转化后的结果等价。

证明:将 SCADE 程序中的 SSM 用五元组$M_s=(Q,\Sigma,\delta,q_{s0},F)$表示,把转化后的两个自动机分别用$M_1=(Q_1,\Sigma,\delta_1,q_1,F_1)$,$M_2=(Q_2,\Sigma,\delta_2,q_2,F_2)$表示。将$M_1$和$M_2$合并后表示为 $M=M_1\times M_2=(Q_1\times Q_2,\Sigma,\delta,[q_1,q_2],F_1\times F_2)$,其中 $\delta([q_i,q_j],x)=[\delta_1(q_i,x),\delta_2(q_j,x)]$,$q_i\in Q_1$,$q_j\in Q_2$,$x\in\Sigma$。合并后是一个不确定 FSM,引入状态$q_0$和∈迁移条件,*M* 的变化如图 1 所示。

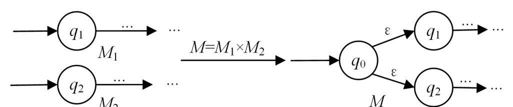

图 1 不确定自动机 $M=M_1\times M_2$

Fig.1 Nondeterministic automaton $M=M_1\times M_2$

M_2的每个状态中的代码为获取下一个状态的编号值，它是一个 Moore 自动机，而每个 Moore 自动机都有一个等价的 Mealy 自动机，将M_2转化为 Mealy 自动机后，每个状态不再有代码。Moore 自动机转化为 Mealy 自动机的过程如图 2 所示。

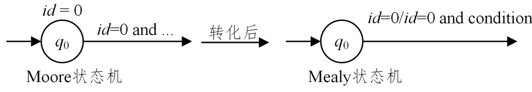

图 2　Moore 自动机与 Mealy 自动机的转化

Fig. 2　Moore automaton translate to Mealy automaton

M 的状态转化函数为 $\delta([q_i, q_j], x) = [\delta_1(q_i, x), \delta_2(q_j, x)]$，由转化算法可知，$M_1$，$M_2$对应状态的迁移条件完全一样，因此可以将对应的两个状态合并为一个状态。而M_2转化为 Mealy 机后，每个状态中的代码为空，因此合并后的状态中只包含M_1状态中的代码，而M_1状态中的代码与状态机M_s对应状态中的代码相同，因此$Q_1 \times Q_2 = Q$。至此，M 与M_s的区别为状态转化函数δ。在M_s中，δ只包含图 2 中 Mealy 自动机的 condition 部分，而 M 中 $id = x / id = y$ and condition，表示在状态q_i上，FSM 接收了输入 $id = y$ and condition，输出 id 为 x。输入的 id 值等于上一个时钟的 id 值，初始值为 0，它只是 FSM 所处状态的标识，由上一个时钟 FSM 的状态确定，M 的状态转化关系与它无关。因此，转化后的结果与 SCADE 程序中的 SSM 等价。

5　SCADE 模型检测的实现

SCADE 模型检测工具由 3 个模块组成，即编译器前端、SCADE 到 Lustre 的转化模块、模型检测后端。编译器前端基于 Flex，Bison 实现，结合定义的 AST 节点数据结构，将 SCADE 程序转化为对应的 AST；转化模块中实现了所有的转化算法，通过该模块可以将 SCADE 程序转化为等价的 Lustre 程序；模型检测后端基于 JKind 和 SMT 求解器实现。使用 JKind 工具从转化后的 Lustre 程序中提取一阶逻辑公式，然后将该公式输入到 Z3 求解器中，最后结合模型检测算法，使用 Z3 求解器完成模型检测。

6　实验与分析

为了验证本文开发的 SCADE 模型检测工具的正确性，我们构建了一个 SCADE 标准测试集。目前该测试集包含 887 个测试用例，这些测试用例都是通过 SCADE Suite 建模而来，与 Lustre 标准测试[6]的功能等价。该测试集覆盖了 SCADE 的常用语法结构。SCADE Suite 工具构建的模型中，每个模型单元都会定义新的数据流来保存输出值，因此在 SCADE 程序中会定义大量的数据流。

在指定的实验环境（每个测试用例的最大运行时间为 120 s，服务器内存为 8 GB，处理器为 Intel(R) Core(TM) i7-4510 CPU@2.00 GHz，操作系统为 Ubuntu16.04）下，该工具输出的模型检测结果有 valid，invalid，timeout 3 种。valid 表示检测属性在所有系统状态中一直可被满足；invalid 表示检测属性在所有系统状态中不能被一直满足，同时会输出不满足的系统状态；timeout 表示在有限资源条件下，不能确定检测属性在所有系统状态中一直被满足。

通过对比本文开发的工具对 SCADE 测试集与 JKind 对 Lustre 测试集的模型检测结果，来验证本文开发的工具的模型检测的正确性。运行结果如表 1 所列，两个工具的模型检测结果都是 437 个 invalid、371 个 valid 以及 79 个 timeout，通过进一步分析实验结果可知，每个 SCADE 测试用例与和它功能相同的 Lustre 测试用例模型的检测结果相同。因此，验证了本文实现的 SCADE 模型检测工具的正确性。

表 1　SCADE 和 Lustre 测试集的模型检测时间

Table 1　Model checking time of SCADE and Lustre benchmark

	SCADE 测试集	Lustre 测试集
invalid	437/339.0 s	437/289.9 s
valid	371/1 574.8 s	371/989.7 s
timeout	79/7 110.0 s	79/7 110.0 s

目前供学术研究的 SCADE 模型检测工具 LAMA 自 2012 年发布后就没再维护，在安装时发生了不可修复的错误。因此只能从理论上分析对比该工具与本文实现的工具差距：关于覆盖的 SCADE 语法，LAMA 工具不支持 clock，group，merge 等语句，而本文实现的工具支持所有的 SCADE 语法。关于模型检测效率，本文工具利用了 JKind 的优点，实现了 BMC，k-induction，PDR 算法，同时还使用了抽象技术、多线程等来优化模型检测过程，而 LAMA 工具只实现了 BMC 和 k-induction 算法，且没有实现任何优化技术。因此，本文实现的工具模型检测效率更高。

本文最后对比了 SCADE 测试集与 Lustre 测试集的模型检测效率。如图 3 所示，横坐标表示 Lustre 测试集模型检测的运行时间，纵坐标表示 SCADE 测试集模型检测的运行时间。两个等价（SCADE 测试用例和 Lustre 测试用例）的测试用例的模型检测结果用一个点表示，如果这个点位于坐标轴对角线下方，则表示 Lustre 测试用例模型的检测效率更高，否则表示 SCADE 测试用例模型的检测效率更高。从图 3 可以看出，对于相同功能的两个测试用例，Lustre 测试用例模型的检测效率更高。同时，由表 1 可知，SCADE，Lustre 测试集中的模型检测结果为 invalid 时，花费时间分别为 339.0 s，289.9 s，valid 花费时间分别为 1 574.8 s，989.7 s。原因之一是 SCADE Suite 建模过程中，每个模型单元会定义新的本地数据流来保存其输出，导致 SCADE 提取的逻辑公式长度比等价的 Lustre 测试用例的逻辑公式长，从而使得模型的检测效率相对较低。

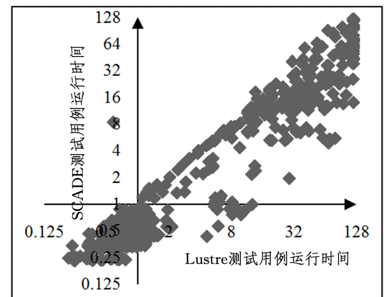

图 3　SCADE 测试集与 Lustre 的模型检测效率对比图

Fig. 3　Comparison between SCADE and Lustre benchmark

结束语　本文提出了一种 SCADE 程序的模型检测方法

并实现了一款 SCADE 模型检测工具。基于程序转化的思想，将 SCADE 程序转化为 Lustre 程序，然后使用 JKind 工具与 Z3 求解器完成模型检测。SCADE 作为工业级语言，拥有功能强大的 SCADE Suite 工具，但是关于它的学术研究成果较少。本文提出的 SCADE 模型检测方法与实现的 SCADE 模型检测器在一定程度上对 SCADE Suite 工具的模型设计和代码验证自动化验证具有一定的促进作用。同时，对 SCADE 模型检测的学术研究有一定的促进作用。

参 考 文 献

[1] YANG P, WANG S Y. Survey on Trustworthy Compilers for Synchronous Data-flow Languages[J]. Computer Science, 2019, 46(5): 21-28.

[2] YANG Z B, YUAN S H, XIE J, et al. Multi-threaded code generation toolfor synchronous language[J]. Journal of Software, 2019, 30(7): 1980-2002.

[3] COLAÇO J L, PAGANO B, POUZET M. SCADE 6: A formal language for embedded critical software development[C]// 2017 International Symposium on Theoretical Aspects of Software Engineering (TASE). IEEE, 2017: 1-11.

[4] DURAK U, D'AMBROGIO A, BOCCIARELLI P. Safety-critical simulation engineering[C]// Proceedings of the 2020 Summer Simulation Conference. 2020: 1-12.

[5] SIRJANI M, LEE E A, KHAMESPANAH E. Model checking software in cyberphysical systems[C]// 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). IEEE, 2020: 1017-1026.

[6] HAGEN G, TINELLI C. Scaling up the formal verification of Lustre programs with SMT-based techniques[C]// 2008 Formal Methods in Computer-Aided Design. IEEE, 2008: 1-9.

[7] CHAMPION A, MEBSOUT A, STICKSEL C, et al. The kind 2 model checker[C]// International Conference on Computer Aided Verification. Cham: Springer, 2016: 510-517.

[8] KAHSAI T, TINELLI C. PKind: A parallel k-induction based model checker[J]. Electronic Proceedings in Theoretical Computer Science, 2011, 72: 55-62.

[9] GACEK A, BACKES J, WHALEN M, et al. The JKind model checker[C]// International Conference on Computer Aided Verification. Cham: Springer, 2018: 20-27.

[10] BASOLD H, GÜNTHER H, HUHN M, et al. An open alternative for SMT-based verification of SCADE models[C]// International Workshop on Formal Methods for Industrial Critical Systems. Cham: Springer, 2014: 124-139.

[11] BARRETT C, TINELLI C. Satisfiability modulo theories[M]// Handbook of Model Checking. Cham: Springer, 2018: 305-343.

[12] AHO A V, SETHI R, ULLMAN J D. Compilers, principles, techniques[J]. Adson Wesley, 1986, 7(8): 9.

[13] NEAMTIU I, FOSTER J S, HICKS M. Understanding source code evolution using abstract syntax tree matching[C]// Proceedings of the 2005 International Workshop on Mining Software Repositories. 2005: 1-5.

[14] JOHN L. Flex & Bison: Text Processing Tools [M]. O'Reilly Media, Inc: Cambridge, 2009: 1.

[15] HOPCROFT J E, MOTWANI R, ULLMAN J D. Introduction to automata theory, languages, and computation[J]. Acm Sigact News, 2001, 32(1): 60-65.

[16] ANDRÉ C. Semantics of SSM(safe state machine) [R]. Technical Report, Esterel Technologies, 2003.

[17] MCMILLAN K L. Symbolic model checking [M]// Symbolic Model Checking. Boston, MA: Springer, 1993: 25-60.

[18] DE MOURA L, RUEß H, SOREA M. Bounded model checking and induction: From refutation to verification[C]// International Conference on Computer Aided Verification. Berlin: Springer, 2003: 14-26.

RAN Dan, born in 1996, postgraduate. His main research interests include verification of software and model checking.

CHEN Zhe, born in 1981, associate professor, is a member of China Computer Federation. His main research interests include verification of software, software engineering and network security.