基于多引擎并行协作的 SCADE 模型检测

方雨瑶,张 聪
(南京航空航天大学 计算机科学与技术学院,江苏 南京 211106)

摘 要:SCADE 语言是一种同步数据流语言,通常被用于实时嵌入式自动控制系统的开发,在航空航天、交通、核工业等领域有广泛的应用。已有的 SCADE 同步语言模型检测工具存在无法验证部分复杂程序和验证效率低下的问题。为了解决现有的问题,该文提出了多引擎并行协作的方法,通过并行执行 BMC 引擎、归纳法引擎和程序抽象引擎三个模型检测引擎来实现对 SCADE 同步语言程序验证的协作,其中程序抽象引擎通过反例引导的抽象精化方法解决了大型复杂程序验证效率低下的问题。实现了一款针对 SCADE 同步语言程序的模型检测工具 PSMC,该工具采用多引擎并行协作方法来提升 SCADE 同步语言程序模型检测的效率。手动构造了 887 个 SCADE 同步语言程序用于对 PSMC 进行实验验证,结果表明提出的优化方法可以有效地对 SCADE 同步语言程序进行自动的验证,并且可以提升模型检测的验证效率(约 31%)。

关键词:同步语言;形式化验证;一阶逻辑;反应系统;可满足性模理论
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2023)11-0086-05
doi:10.3969/j.issn.1673-629X.2023.11.013

SCADE Model Checking Based on Multi-engine Parallel Collaboration

FANG Yu-yao, ZHANG Cong
(School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

Abstract: SCADE language is a synchronous data flow language that is commonly used for the development of real-time embedded automatic control systems and has a wide range of applications in the aerospace, transportation and nuclear industries. The existing SCADE synchronous language model checking tools suffer from the inability to verify some of the complex programs and the inefficiency of verification. In order to solve the existing problems, we propose a multi-engine parallel collaboration approach, in which three model checking engines, namely the BMC engine, the induction engine and the program abstraction engine, are executed in parallel to collaborate on the verification of SCADE synchronous language programs, where the program abstraction engine solves the problem of inefficient verification of large complex programs by means of counterexample-guided abstraction refinement. We have implemented a model checking tool, PSMC, for SCADE synchronous language programs, which uses a multi-engine parallel collaboration approach to improve the efficiency of model checking for SCADE synchronous language programs. We manually construct 887 SCADE synchronous language programs for experimental verification of PSMC, and the results show that the proposed optimization method can effectively and automatically verify SCADE synchronous language programs, and can improve the verification efficiency of model checking by about 31%.

Key words: synchronous languages; formal verification; first-order logic; reactive systems; satisfiability mode theory

0 引 言

随着科技的发展,反应式系统[1]在航空航天、交通、核工业等关键性领域应用越来越广泛,它能够以确定的时间对其所处的环境做出连续反应,具有时间约束性、可预测性、与外部环境的交互性等。为了方便反应式系统的设计与实现,领域专家与学者提出了同步语言[2]。其中 SCADE(Safety Critical Application Development Environment)语言[3-5]是这些领域的事实标准开发语言,已经被广泛应用于实现实时嵌入式自动控制系统[6]。这些领域对系统的安全性与可靠性要求非常高,因此这些系统需要经过严格的形式化验证[7]。

现有的 SCADE Suite 工具可以对 SCADE 程序进

收稿日期:2022-12-09 **修回日期**:2023-04-12
基金项目:国家自然科学基金(62172217);国家自然科学基金委员会-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)
作者简介:方雨瑶(1998-),女,通讯作者,硕士,研究方向为软件验证。

行模型检测,但该工具是一款价格昂贵的商业软件,且已经限制在国内使用和研究,并且验证方法单一、验证效率不高[8-9]。而由 Henning Basold 等人开发的免费工具不支持完整的 SCADE 语言,并且验证效率较低[10]。

基于现状,该文提出了基于多引擎并行协作的 SCADE 模型检测方法。将 SCADE 程序和待验证的安全属性转化为一阶逻辑公式,并对一阶逻辑公式使用 SMT(Satisfiability Modulo Theories)求解器,通过多引擎并行协作的方法进行验证。主要工作如下:

(1)提出并行执行 BMC(Bounded Model Checking)引擎、归纳法引擎,将原本 k-Induction 算法中串行的基础步骤和归纳步骤改为并行执行。

(2)提出程序抽象引擎,对大型复杂程序进行抽象,并反复进行检测、确认、精化,加快对无效属性的验证速度。

(3)开发了一款 SCADE 模型检测工具 PSMC,实现 BMC 引擎、归纳法引擎和程序抽象引擎并行协作的验证架构,经过实验证明了相比于不使用优化时能够提高验证效率,从而可以对较大规模的程序进行验证。

1　相关工作

传统的 k-Induction 算法[11]以 1-Induction 标准归纳法为基础,当标准归纳法无法证明“性质在系统执行的第 i 步成立”蕴含“性质在系统执行的第 $i+1$ 步成立”时,对待验证公式进行扩展,尝试证明“性质在系统执行的第 $n+1$ 到 $n+k$ 步内成立”蕴含“性质在系统执行的第 $n+k+1$ 步成立”。假设一个有限状态迁移系统 $S=<X,T,I>$,其中 X 是系统所有状态变量的集合,x_i 表示系统所有状态变量在时钟 i 下的瞬时值,I 表示系统的初始状态,T 是系统在前一个时钟到当前时钟的迁移关系表达式。要验证属性 P 是有效的,即证明 P 在有限迁移系统 S 中是不变式(在有限迁移系统 S 的所有可达状态都满足 P)。k-Induction 算法所依赖的验证公式包括公式(1)和公式(2)。如果它们在某个时刻 k($k\geqslant 0$)都成立,即可证明 P 是不变式。

$$T(0)\wedge\cdots\wedge T(k)\Rightarrow P(0)\wedge\cdots\wedge P(k) \tag{1}$$

$$T(n)\wedge\cdots\wedge T(n+k+1)\wedge P(n)\wedge\cdots\wedge P(n+k)\Rightarrow P(n+k+1) \tag{2}$$

传统的 k-Induction 算法中基础步骤公式(1)和递归步骤公式(2)是串行协作的关系,也就是说,公式(2)的检查需要依赖于公式(1)的检查结果。由此带来的一个后果是,当属性无效且在 k 值较大才能被证伪的情况下,串行验证会消耗大量时间在递归步骤公式(2)的验证上。为了提高验证效率,将串行协作优化为并行协作,将基础步骤作为单独线程,将递归步骤也作为单独线程,再通过消息传递机制控制线程之间的关系,这样算法引擎就可以在各自的线程上并行地同时运行,可以使得有效属性和无效属性的验证速度大大加快。接下来,详细描述多引擎并行协作算法的实现。

2　多引擎并行协作方法

图 1 展示了多引擎并行协作的架构设计。首先,词法语法解析器将 SCADE 同步语言程序解析为抽象语法树[12](AST),然后逻辑公式生成器将 AST 转化为求解器可求解的逻辑公式,最后 BMC 引擎、归纳法引擎和程序抽象引擎分别调用独立的 SMT 求解器(例如 Z3)[13]同时对逻辑公式进行求解,得到验证结果。注意,在求解阶段设定了一个固定的 k 值,在 k_opt 步

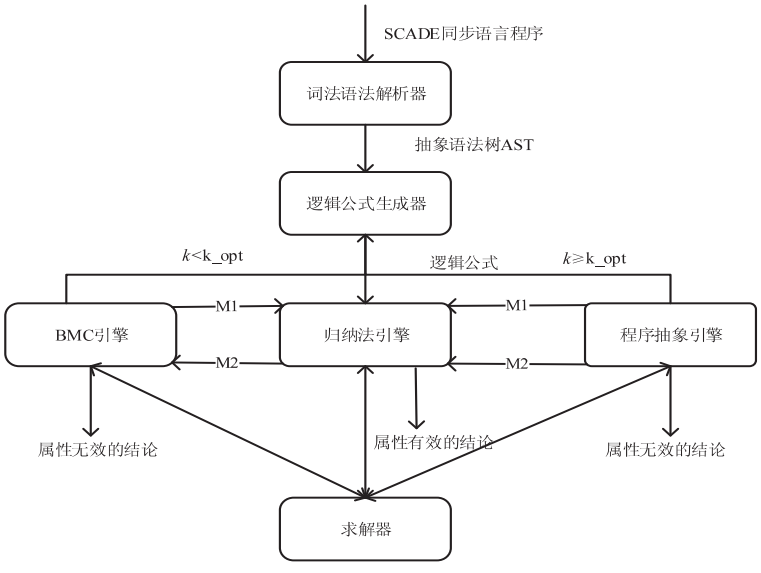

图 1　PSMC 的架构设计

之前由 BMC 引擎和归纳法引擎并行执行,在 k_opt 步之后由程序抽象引擎和归纳法引擎并行执行,并且通过共享变量进行消息传递和协作。

2.1 BMC 引擎

BMC 引擎从 $k=0$ 开始验证公式(1),并逐渐递增 k 的值,如果在某个 k 值下公式不成立,求解器会返回一组可以证明公式不成立的变量值,称之为反例。再将无效属性和反例输出后终止该线程,同时将终止线程信号 M_1 = Abort 传递给归纳法引擎所处的线程。否则,将继续递增 k 值,并将成功通过该公式验证的 k 值通过信号 $M_1 = k$ 传递给归纳法引擎线程,直到接收到由归纳法引擎所处线程传递来的终止线程信号 M_2 = Abort,终止本线程。

2.2 归纳法引擎

线程休眠,直到接收到来自 BMC 引擎线程或程序抽象引擎线程消息 M_1。如果 M_1 = Abort 是终止线程信号,也就是说,BMC 引擎或程序抽象引擎已经证明了属性是不成立的,则终止本线程。如果 $M_1 = k$ 是自然数,也就是说,BMC 引擎或程序抽象引擎发现属性在第 k 个时钟以内是成立的,则对于所有的自然数 $i \leqslant k$,调用 SMT 求解器验证归纳公式(2)是否成立。如果对于某个 i 归纳公式(2)成立,那么对于所有的时钟 c ($c>0$),基础公式(1)都是成立的,即证明了属性是成立的。此时输出有效属性,并将终止线程信号 M_2 = Abort 传递给 BMC 引擎线程和程序抽象引擎线程,然后终止本线程。如果归纳公式(2)不成立,归纳法引擎则继续递增 k 值对公式(2)进行验证,直到 k 值递增到大于 M_1 传递来的自然数,归纳法引擎继续休眠直到再次接收到来自 BMC 引擎线程或程序抽象引擎线程消息 M_1。

2.3 程序抽象引擎

在多引擎并行协作的架构上设定一个控制 BMC 引擎和程序抽象引擎的临界值 k_opt,当 BMC 引擎的变量 k < k_opt 时,程序抽象引擎所处的线程休眠;当 k >k_opt 时,BMC 引擎所处线程进入休眠状态并且启动程序抽象引擎,对程序进行抽象后,进行 BMC 检测,如果没有报告反例,说明该抽象程序在 k 个时钟以内满足待验证属性,则通过信号 $M_1 = k$ 将通过检查的 k 值传递给归纳法引擎;如果报告了反例,且经过确认该反例确实为真实反例,可以得出该属性无效的结论,则输出反例所违反的属性(即无效属性)和反例,并将终止线程信号 M_1 = Abort 传递给归纳法引擎所处线程,然后终止本线程。如果接收到归纳法引擎所处线程传递来的终止线程信号 M_2 = Abort,则终止本线程。否则, k 值加 1 继续进行抽象程序算法检测。

图 2 是程序抽象引擎采用的程序抽象算法流程。首先,对目标程序进行初始化抽象,将抽象程序进行 BMC 算法检测,如果检测过程中出现反例,那需要对该反例进行确认,判断该反例是否为伪反例,如果是,那么需要对抽象程序进行精化操作使抽象程序更靠近原程序;如果不是,则得出该属性无效的结论。接下来将对反例引导的抽象精化算法进行详细的说明。

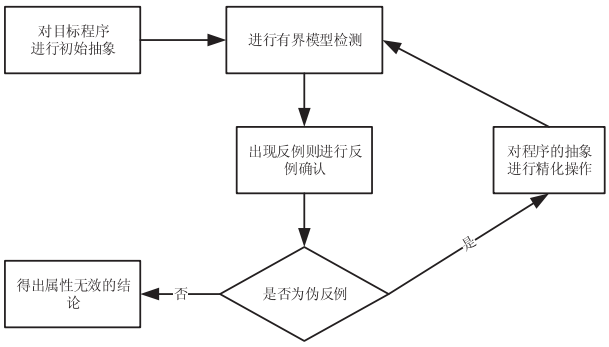

图 2 程序抽象算法流程

如图 2,首先将对逻辑公式转化器处理完的程序 N 进行抽象处理,得到初始抽象程序 N'。原程序四元组 N 中有输入数据流 I,局部数据流 L,输出数据流 O。用 set 集合存储了程序的等式依赖关系,通过等式依赖关系集合,得到待验证属性 P 依赖与 y_a、y_b,注意,这里提到的依赖为等式等号左侧和右侧直接的关系,比如有 $a = b + c$,就有 $< a \rightarrow < b, c >>$, a 依赖于 b 和 c 的值;接下来将阐述构建抽象程序的过程。原始程序 N 记为四元组 $N = < I, O, L, E >$,其中的输入数据流 I、局部数据流 L、输出数据流 O 都作为抽象程序 N' 的输入数据流 I',将输出数据流 O' 置为空集,将待验证属性和其依赖的变量作为抽象程序 N' 的局部数据流 L',将 P 的等式定义 eq_p,以及 P 所依赖变量 y_a、y_b 的等式定义 eq_{y_a}、eq_{y_b} 作为新的等式集合 E'。按照这样的规则构建了新的抽象程序四元组 $N = < I', O', L', E' >$。

在对原程序进行初始抽象后,对抽象程序进行 BMC 算法检测,如果检测出反例,需要判断该反例是真实的还是虚假的。SMT 求解器返回的反例由步数 k,变量以及变量对应的值构成,将源程序同样提取一阶逻辑公式,去检测反例所显示的变量状态是否是源程序可达的变量状态,也就是说,用 SMT 求解器去检测反例是否是由源程序在 k 步下可达的状态,如果求解器返回 unsat 证明反例表示的状态不可达,该反例是伪反例,同时可以从求解器中获得 unsatcore(unsatcore 为使得检测的式子不满足的关键变量的集合),再根据获得的 unsatcore 对抽象程序进行抽象精化;如果求解器返回 sat 证明反例表示的状态可达,该反例是真实反例,得出该检测属性无效,停止检测。

算法 1:refine 算法

```
Inputs:unsatcore , map m:V→2^V
priorty_queue Q;
foreach var in unsatcore
Q. push(var);
while(Q. size>0)
  node = Q. pop();
if ( is_undefined(node) )
refine_var(node);
else if ( is_fully_refined(node) )
continue;
else
dependencies dep_set = map. get(node);
foreach temp_var in dep_set
Q. push(temp_var);
```

经过反例确认后,如果该反例是伪反例,那需要根据获得的 unsatcore 对抽象程序进行精化操作,具体的过程在算法 1 中展示。将所有的变量用 refined,undefined,ori-input 标记;标记为 refined 或 undefined 的变量对应源程序中的非输入变量,标记为 ori-input 的变量对应源程序的输入变量;refined 标记的变量指该变量在抽象程序中已得到等式定义的约束,也就是说该变量的等式定义已在抽象程序的等式集合中定义;undefined 标记的变量指的是还未得到等式定义约束的变量;其中,如果某一变量和它的依赖变量都得到了等式定义的约束,那称这个变量是 fully-refined。

3　实验分析

在 Linux 环境下使用 C++语言开发实现了一款 SCADE 同步语言程序模型检测工具 PSMC(Parallel Scade Model Checker),该工具实现了多引擎并行协作模式和程序抽象引擎。在实验过程中,首先采用未进行优化的串行验证工具,之后再开启 PSMC 的 BMC 引擎和归纳法引擎对比串行验证和并行协作的验证效率,再开启程序抽象引擎,观察程序抽象引擎对 PSMC 工具带来的效率提升,以证明多引擎并行协作可以有效提升 SCADE 同步语言程序验证的效率。

笔者构建了包含 887 个 SCADE 同步语言程序(分为七个文件夹)的测试集,该测试集是根据 Lustre 标准测试集[14]、利用 SCADE Suite 工具[15]人工手动建模而来,与 Lustre 标准测试集的功能等价。将测试集按照最终的验证结果划分为三种,无效属性(invalid)测试集、有效属性(valid)测试集、验证超时(timeout)测试集。其中无效属性测试集中有 359 个无效属性,有效属性测试集中有 305 个无效属性,设置了一个运行时间限制(100 秒),导致还有一种验证结果为超时,所以测试集中有包含 223 个超时属性的测试集。

表 1 展示了针对无效属性测试集逐个增加优化算法的验证时间对比,Size 指的是测试集的大小,也就是无效属性的数量,Depth 指的是测试集中检测出无效属性产生反例时 k 的最大值,统计的运行时间是一个测试集的时间总和。表 2 展示了针对有效属性测试集逐个增加优化算法的验证时间对比,Size 表示测试集中有效属性的数量,Depth 表示测试集中有效属性被证明成立时 k 的最大值。两个表格中的 M_1 指的是加入了 BMC 引擎和归纳法引擎双引擎并行协作的方法,M_2 指在 M_1 的基础上加入程序抽象引擎。

表 1　无效测试集运行时间对比

Program class	Size	Depth	无优化	+M1/s	+M2/s
functional	0	–	–	–	–
large	45	23	309.49	289.67	277.12
memory1	109	4	16.17	14.49	14.45
memory2	98	5	8.41	8.56	8.46
misc	40	11	3.65	3.60	3.61
protocol	14	12	3.97	3.83	3.81
simulation	72	43	365.27	343.94	180.50
total	378		706.96	664.09	487.95

从表 1 可见在无效属性测试集中,表格中双引擎并行协作能提速 6% 左右,抽象程序引擎的加入能在之前的基础上提速 26% 左右,四种优化算法一共能提速 31% 左右,PSMC 的整体运行速度明显快于优化前,在 Depth 较大的测试集中 PSMC 的提速更加明显。其中 functionalchain 测试集中验证结果为无效的属性个数为 0。

表 2　有效测试集运行时间对比

Program class	Size	Depth	无优化	+M1/s	+M2/s
functional	38	1	103.26	75.27	74.34
large	34	1	2.69	2.50	2.16
memory1	85	1	11.65	10.98	10.88
memory2	54	1	5.05	5.96	5.16
misc	31	4	3.45	3.41	3.38
protocol	17	1	2.37	1.99	2.1
simulation	78	9	9.93	9.86	8.96
total	337		138.37	109.97	106.98

从表 2 可见在有效属性测试集中,多引擎并行协作的效果在大部分测试集上耗时区别不大,甚至并行协作架构的加入使得部分测试集验证时间变长了,经过分析实验结果和实验代码,发现有效性大多都在 k 递增到一步或者两步时被检出,其耗时在微秒级别。而多引擎并行协作时,线程之间的交互以及线程协作过程中线程的等待也会消耗一定的时间,但单个程序微妙级别耗时的增加在实际工具使用中不会带来特别大的影响,实际操作中更注重工具在复杂问题上带来

的显著提速效果,通过后续优化算法的加入可以尽量抵消这部分时间的消耗。

4 结束语

由于当前模型检测工具对 SCADE 程序验证的支持存在不足,该文提出了一种基于多引擎并行协作的 SCADE 程序模型检测方法,并开发了 PSMC。实验表明,该方法不仅能够很好地对 SCADE 同步语言程序进行自动、直接地验证,而且相比于未使用优化时能明显提高验证效率。在未来的工作中,会在多引擎并行协作架构的基础上加入更多先进的算法引擎,进一步提高 SCADE 同步语言程序的验证效率,并在这一背景下验证这些算法的效率。

参考文献:

[1] HALBWACHS N. Synchronous programming of reactive systems[M]. Berlin: Springer Science & Business Media, 2013:22-47.

[2] FORGET J, BONIOL F, LESENS D, et al. A multi-periodic synchronous data-flow language[C]//2008 11th IEEE high assurance systems engineering symposium. Nanjing: IEEE, 2008:251-260.

[3] SCHMID K N. Safety critical application development environment[J]. A Survey of Tools for Model Checking and Model-Based Development, 2006, 17:45-52.

[4] DORMOY F X. Scade 6: a model based solution for safety critical software development[C]//Proceedings of the 4th European congress on embedded real time software (ERTS' 08). Toulouse: HAL, 2008:1-9.

[5] ANICULAESEI A, VORWALD A, RAUSCH A. Using the scade toolchain to generate requirements-based test cases for an adaptive cruise control system[C]//2019 ACM/IEEE 22nd international conference on model driven engineering languages and systems companion (MODELS-C). Munich: IEEE, 2019:503-513.

[6] HOBBS C. Embedded software development for safety-critical systems[M]. Florida: CRC Press, 2019:1-39.

[7] HAGEN G E. Verifying safety properties of Lustre programs: an SMT-based approach[D]. Iowa: The University of Iowa, 2008.

[8] ABDULLA P A, DENEUX J, STÅLMARCK G, et al. Designing safe, reliable systems using scade[C]//International symposium on leveraging applications of formal methods, verification and validation. Paphos: Springer, 2004:115-129.

[9] 冉 丹. 基于模型转化的 SCADE 模型检测[D]. 南京:南京航空航天大学, 2021.

[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. Zurich: Springer, 2014:124-139.

[11] DONALDSON A F, HALLER L, KROENING D, et al. Software verification using k-induction[C]//18th international symposium on static analysis. Venice: Springer, 2011:351-368.

[12] 冉 丹, 陈 哲, 孙 毅, 等. 基于程序转化的 SCADE 模型检测[J]. 计算机科学, 2021, 48(12):125-130.

[13] DE MOURA L, BJØRNER N. Z3: an efficient SMT solver[C]//Tools and algorithms for the construction and analysis of systems: 14th international conference, TACAS 2008, held as part of the joint european conferences on theory and practice of software, ETAPS 2008. Budapest: Springer, 2008:337-340.

[14] CHAMPION A, MEBSOUT A, STICKSEL C, et al. The Kind 2 model checker[C]//International conference on computer aided verification. Hangzhou: Springer, 2016:510-517.

[15] GUDEMANN M, ANGERER A, ORTMEIER F, et al. Modeling of self-adaptive systems with SCADE[C]//2007 IEEE international symposium on circuits and systems. New Orleans: IEEE, 2007:2922-2925.