软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1595−1606 [doi: 10.13328/j.cnki.jos.005469]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 SMT 的时钟约束语言 CCSL 的形式化分析方法与工具*

应云辉[1,2], 张 民[1,2]

[1](华东师范大学 计算机科学与软件工程学院,上海 200062)

[2](高可信软件国家重点实验室(华东师范大学),上海 200062)

通讯作者: 张民, E-mail: zhangmin@sei.ecnu.edu.cn

摘 要: 时钟约束语言 CCSL 是一种用于描述实时嵌入式系统中事件之间约束的形式化语言,它是 UML 针对实时嵌入式系统建模的扩展包 MARTE(modeling and analysis of real-time and embedded systems)中用于对时间建模的一个子语言.给定一组由 CCSL 定义的时钟约束条件,需要判断是否存在某种调度策略满足约束、是否所有满足这些约束的行为都不会导致系统死锁等分析.目前已经有一定的针对 CCSL 的形式化分析研究工作,如基于状态迁移系统与时间自动机的方法等.但这些方法要么只针对某种特定的分析,要么只适用于部分 CCSL 约束,要么分析效率较低.提出了基于 SMT 的统一且高效的 CCSL 形式化分析方法.统一性体现在其可用于有效性证明、迹分析、死锁检测、LTL 模型检测等方面的验证与分析.基于该方法开发了原型工具,同时支持上述 4 种验证功能.工具集成了当前最高效的 SMT 求解器 Z3 和 CVC4.得益于 SMT 求解器的高效性,实验中大部分的验证可以在短时间内完成.

关键词: CCSL;SMT;有效性证明;迹分析;死锁检测;LTL 模型检测;工具

中图法分类号: TP311

中文引用格式: 应云辉,张民.基于 SMT 的时钟约束语言 CCSL 的形式化分析方法与工具.软件学报,2018,29(6):1595−1606. http://www.jos.org.cn/1000-9825/5469.htm

英文引用格式: Ying YH, Zhang M. SMT-Based approach to formal analysis of CCSL with tool support. Ruan Jian Xue Bao/ Journal of Software, 2018,29(6):1595−1606 (in Chinese). http://www.jos.org.cn/1000-9825/5469.htm

SMT-Based Approach to Formal Analysis of CCSL with Tool Support

YING Yun-Hui[1,2], ZHANG Min[1,2]

[1](School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China)

[2](National Key Laboratory of Trustworthy Software (East China Normal University), Shanghai 200062, China)

Abstract: CCSL (abbreviated for clock constraint specification language) is a formal language for specifying the constraints on the occurrences of events in real-time and embedded systems. CCSL is a companion language of MARTE (abbreviated for modeling and analysis of real-time and embedded systems), a UML profile which provides support for specification, design, and verification/validation stages for model-driven development of real time and embedded Systems. Given a set of CCSL constraints, it is necessary to check if there exist schedules that satisfy all the constraints and if all the behaviors that conforming to the constraints will never incur deadlock of systems. Many approaches have been proposed based on state transition system, timed automata, etc. However, most of the existing approaches have some drawbacks, such as being ad hoc to specific formal analysis, and being suited to only subsets of CCSL constraints or inefficient. In this paper, a unified and efficient SMT-based approach to formal analysis of CCSL is proposed. This approach is unified in that it can be applied to various analysis such as validity proving, trace analysis, deadlock detection and LTL model checking. A prototype tool for the new approach is implemented to support the four types of formal analysis, utilizing the state-of-the-art SMT solvers

* 基金项目: 国家自然科学基金(61502171)
Foundation item: National Natural Science Foundation of China (61502171)
本文由形式化方法的理论基础专题特约编辑傅育熙教授、李国强副教授、田聪教授推荐.
收稿时间: 2017-07-01; 修改时间: 2017-09-01; 采用时间: 2017-11-06; jos 在线出版时间: 2017-12-28
CNKI 网络优先出版: 2017-12-29 13:19:28, http://kns.cnki.net/kcms/detail/11.2560.TP.20171229.1318.010.html

such as Z3 and CVC4. With efficient SMT solvers, verification can be completed in a reasonable time, as demonstrated by the experimental results in the case studies.

Key words: CCSL; SMT; validity proving; trace analysis; deadlock detection; LTL model checking; tool

MARTE(modeling and analysis of real-time and embedded system)是统一建模语言 UML 在实时和嵌入式方面的一个扩展,提供了规范、设计和验证的支持,这个新的扩展将要取代 UML 现有的在可调度、性能和时间上的扩展[1].在 MARTE 中,时钟是访问模型时间结构的元素,使用 MARTE 时间模型要定义互相依赖的时钟,而时钟之间的这种依赖关系用专门的语言进行描述.该语言被称为 CCSL(clock constraint specification language),其定义在 UML 规范附录 C3 中[2,3].CCSL 逐渐被应用于各种实时嵌入式系统的设计,如车载操作系统[4]以及信息物理系统[5].

CCSL 中的时钟为逻辑时钟,用于抽象地表示系统事件的发生情况,即,一次时钟滴答代表其对应的事件发生一次.CCSL 约束可以描述系统中不同事件之间的关系,用于限制系统的行为.所有的系统行为都应当满足预定义的约束条件,同时还要保证这些约束条件能够保证系统不产生死锁.死锁是指在某一时刻,所有时钟对应的事件都由于当前的时钟约束条件而无法发生.因此,如何验证与分析 CCSL 约束是否满足上述条件具有理论与实际意义.

作为一个新兴的建模语言,CCSL 吸引了许多研究人员寻找对 CCSL 约束进行形式化分析的有效方法,如可调度性、死锁检测(deadlock detection).Frederic 等人在其文献[6]中详细讨论了已有的关于 CCSL 的分析方法,主要方法包括利用状态迁移系统[6]、Promela 语言[7]、Büchi 自动机[8]、时间自动机[9]、重写逻辑[10]等对 CCSL 约束进行转化.然而,这些转换方法都存在一定的缺陷.例如,基于状态迁移系统或自动机的方法如文献[7–9]中方法只能应用于对无状态的约束关系的建模,无状态约束关系是指其时钟的当前行为与其之前的行为无关.而 CCSL 约束关系中的有些关系是有状态的,如优先关系(precedence)和因果关系(causality),因而无法利用状态迁移系统进行建模.同时,每个约束用一个状态迁移系统表示,这些系统合成统一的系统容易导致状态爆炸问题.另外,这些方法需要复杂的中间变换且对集合中某个约束的修改需要对整个集合的约束进行重新转换而导致效率低下.基于重写逻辑的方法通过定义 CCSL 的操作语义及执行 CCSL 约束条件仿真所有满足这些条件的行为,然而同样遇到状态爆炸问题.另外,已有的这些方法也只针对某种特定的分析.

早期的工作中,我们提出了一种基于 SMT 技术[12]的 CCSL 线性时序逻辑(LTL)模型检测方法[19].通过该方法,将 CCSL 约束与 LTL 公式直观地转换成 SMT 公式,减少了转换过程的复杂度.众所周知,SMT 能够有效缓解模型检测的状态爆炸问题,这种方法可以实现对 CCSL 高效的 LTL 模型检测(LTL model checking).然而,模型检测依然受限于有限状态模型,而 CCSL 中的调度策略是定义在整个自然数集合上的,因此对于一些性质只能通过设定边界值验证其是部分正确的.在该工作的基础上,本文利用 SMT 可满足性求解的方法证明 CCSL 约束之间的蕴含关系,称之为有效性证明.有别于模型检测,有效性证明不受状态空间的限制,这是当前已知的大部分 CCSL 形式化分析方法缺少的.此外,我们还将这种基于 SMT 的验证方法用于死锁检测与迹分析(trace analysis),并将所有的功能集成到一个原型验证与分析工具.

综上,本文的主要贡献是在原有工作[19]的基础上提出了一种基于 SMT 的统一的 CCSL 形式化分析方法,用于有效性证明、迹分析、死锁检测及 LTL 模型检测,同时开发了对应的原型验证与分析工具.具体描述如下:

(1) 提出如何利用有效性证明的方法证明 CCSL 约束之间的关系;

(2) 提出死锁对应的 SMT 公式的定义,并利用一个具体的案例演示方法的有效性及高效性;

(3) 提出从迹(trace)到 SMT 公式的转化方法,并利用具体的实例介绍转化的过程及迹分析的具体应用;

(4) 针对上述功能开发了原型验证工具,同时支持有效性证明、迹分析、死锁检测以及 LTL 模型检测.

本文第 1 节主要介绍 CCSL 语法语义及其可调度性问题.第 2 节介绍从 CCSL 约束到 SMT 公式的转换方法.第 3 节分别介绍基于 SMT 的 4 种 CCSL 形式化分析方法的应用,包括有效性证明、迹分析、死锁检测以及 LTL 模型检测.第 4 节介绍并展示原型工具.第 5 节将本文工作与相关工作进行比较.第 6 节总结本文并对未来工作加以展望.

1 预备知识

本节主要介绍 CCSL 的语法语义和 CCSL 的可调度问题,这些内容为 CCSL 约束集转换成 SMT 公式和基于 SMT 的 CCSL 形式化分析方法提供了理论支持.

1.1 CCSL的语法和语义

我们先给出与 CCSL 相关的 3 个概念:逻辑时钟(logical clock)、调度(schedule)和历史(history)的定义[6].

定义 1(逻辑时钟).逻辑时钟 c 是一个无穷序列 $(c^i)_{i\in\mathbb{N}^+}$,其中每个 c^i 表示 *tick* 或者 *idle*.

其中,$\mathbb{N}^+$是大于 0 的自然数集合.如果 c^i 的值为 *tick*,则表示 c 代表的事件在第 i 步时发生;否则没发生.

定义 2(调度).逻辑时钟集合 C 上的调度是一个函数映射 $\delta:\mathbb{N}^+\to 2^C$,对于所有的 $i\in\mathbb{N}^+$,都有 $\delta(i)=\{c\mid c\in C\wedge c^i=tick\}$且$\delta(i)\neq\varnothing$.

很明显,$\delta(i)$表示在第 i 步时所有活动的时钟的集合.当集合为空时称该步为空步.条件$\delta(i)\neq\varnothing$要求每一步都至少有一个活动时钟.通过该条件,我们只考虑每一步都至少有一个活动时钟的调度.这是因为在这样的调度中插入一些空的步并不影响时钟之间的逻辑关系,因此只需考虑不含有任何非空步的调度即可.

定义 3(历史). 一个调度$\delta:\mathbb{N}^+\to 2^C$的历史(history)也是一个函数映射$\chi:\mathbb{N}^+\to\mathbb{N}$,对于每一个 $c\in C$ 和 $i\in\mathbb{N}^+$,χ定义如下:

$$\chi(c,i)=\begin{cases}0, & \text{if } i=1\\ \chi(c,i-1), & \text{if } i>1\wedge c\notin\delta(i-1)\\ \chi(c,i-1)+1, & \text{if } i>1\wedge c\in\delta(i-1)\end{cases}.$$

显然,$\chi(c,i)$表示时钟 c 在到达第 i 步前的滴答次数.

CCSL 包含 12 类约束的定义,包括优先关系(*Precedence*)、因果关系(*Causality*)、从属关系(*Subclock*)、互斥关系(*Exclusion*)、并关系(*Union*)、交关系(*Intersection*)、下确界关系(*Infimum*)、上确界关系(*Supremum*)、延迟关系(*Delay*)、相对延迟关系(*DelayFor*)、周期关系(*Periodicity*)与取样关系(*SampledOn*).利用上述 3 个概念,用调度满足约束的形式定义 CCSL 约束的语义.$\delta\models\phi$表示调度δ满足 CCSL 约束ϕ.表 1 给出了 12 类 CCSL 约束的语法和语义.

Table 1 Syntax and semantics of CCSL

表 1 CCSL 的语法和语义

$c_1[b]\prec c_2$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,\chi(c_2,n)-\chi(c_1,n)=b\Rightarrow c_2\notin\delta(n)$	(*Precedence*)
$c_1\preccurlyeq c_2$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,\chi(c_2,n)\geqslant\chi(c_1,n)$	(*Causality*)
$c_1\subseteq c_2$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,c_1\in\delta(n)\Rightarrow c_2\in\delta(n)$	(*Subclock*)
$c_1\#c_2$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,c_1\in\delta(n)\Rightarrow c_2\in\delta(n)$	(*Exclusion*)
$c_1\triangleq c_2+c_3$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,c_1\in\delta(n)\Leftrightarrow(c_2\in\delta(n)\vee c_3\in\delta(n))$	(*Union*)
$c_1\triangleq c_2*c_3$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,c_1\in\delta(n)\Leftrightarrow(c_2\in\delta(n)\wedge c_3\in\delta(n))$	(*Intersection*)
$c_1\triangleq c_2\wedge c_3$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,\chi(c_1,n)=\max(\chi(c_2,n),\chi(c_3,n))$	(*Infimum*)
$c_1\triangleq c_2\vee c_3$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,\chi(c_1,n)=\min(\chi(c_2,n),\chi(c_3,n))$	(*Supremum*)
$c_1\triangleq c_2\$d$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,\chi(c_1,n)=\max(\chi(c_2,n)-d,0)$	(*Delay*)
$c_1\triangleq c_2\$d$ on c_3	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,c_1\in\delta(n)\Leftrightarrow(c_3\in\delta(n)\wedge\exists m\in\mathbb{N}^+,(c_2\in\delta(m)\wedge\chi(c_3,n)-\chi(c_3,m)=d))$	(*DelayFor*)
$c_1\triangleq c_2\propto p$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,c_1\in\delta(n)\Leftrightarrow(c_2\in\delta(n)\wedge\exists m\in\mathbb{N}^+,\chi(c_3,n)=m\times p-1)$	(*Periodicity*)
$c_1\triangleq c_2\lightning c_3$	$\Leftrightarrow$	$\forall n\in\mathbb{N}^+,c_1\in\delta(n)\Leftrightarrow(c_3\in\delta(n)\wedge\exists m\in\mathbb{N}^+,c_3\in\delta(m)\wedge\chi(c_3,n)-\chi(c_3,m)=1\wedge\chi(c_2,n)-\chi(c_2,m)\geqslant 1))$	(*SampledOn*)

以优先关系为例,解释 CCSL 约束的含义.假设有两个时钟 c_1 与 c_2,约束 $c_1[b]\prec c_2$ 表示 c_2 可以优先比 c_1 发生 b 次.当 c_2 已经优先比 c_1 发生 b 次后,c_2 必须等到 c_1 发生之后才能发生.一个优先关系的应用场景是对一个初始非空队列的存取.假设队列中预先放置了 b 个元素,c_1 代表存放一个元素,c_2 代表取出一个元素.则 c_1 与 c_2 的关系可表示为 $c_1[b]\prec c_2$.特别地,如果 b=0,可简写为 $c_1\prec c_2$.关于其他约束类型的详细解释见文献[6].对于一个 CCSL 约

束集合Φ,用δ表示调度满足约束集Φ,即:$\delta\models\Phi$当且仅当$\forall\phi\in\Phi,\delta\models\phi$.

1.2 CCSL的可调度问题

对于给定的 CCSL 约束集,一个最基本的问题是判断是否存在调度满足这个约束集中的所有约束,这便是 CCSL 的可调度问题.目前所知,还没有一个算法能够对任意给定的一组 CCSL 约束集判断是否存在一个满足所有约束的调度.一方面是因为所求调度是无限长的,另一方面是因为一些约束如 Precedence 所定义的时钟行依赖时钟的历史,即时钟的滴答次数.若限定调度的长度,则使问题易解.

定义 4(限界调度). 给定正整数 n 和调度δ,如果从第 1 步到第 n 步都满足 CCSL 约束集Φ,则称δ为上界为 n 的限界调度.

然而在实际应用中,限界调度对于实时嵌入式系统过于限制.因为嵌入式系统的执行一般不假设时间上限,我们期望能够求解在无限长度下满足约束条件的调度.在无限调度中存在一个特殊的情况,即周期调度.周期调度的特点是在有限长步数后,所有的时钟的行为都周期性发生.

定义 5(周期调度). 若一个调度δ中存在两个自然数 l,p,使得对任意 $k\geqslant l$ 都满足$\delta(k+p)=\delta(k)$,那么称δ为周期调度.其中,p 是δ的周期.

判断周期调度的存在性目前也是一个公开问题,其原因与判断调度的存在性一样,关于周期调度问题的详细分析见文献[20].一个折衷的方法是,把限界调度扩展成为周期调度.根据周期调度的定义可知:它进入周期循环之前的步数是有限的,且周期的长度也是有限的.

定义 6. 给定一个上界为 n 的限界调度δ和两个正整数 l,k,且 $l<k\leqslant n$,若$\delta(l)=\delta(k)$,那么δ可被扩展成一个周期调度$\delta_{l,k}$:

$$\delta_{l,k}(i)=\begin{cases}\delta(i), & \text{if } i\leqslant k\\ \delta(l+(i-l)\bmod(k-l)), & \text{if } i>k\end{cases}.$$

然而,由一个满足Φ的限界调度扩展的周期调度却不一定满足Φ,因为即使$\delta_{l,k}$在 l 到 k 的区间内满足Φ,在后续的周期循环中并不能保证满足Φ.如因果关系 $c_1\preccurlyeq c_2$,要保证对$\forall n\in\mathbb{N}^+$都满足$\chi(c_2,n)\geqslant\chi(c_1,n)$.如果在从 l 到 k 的整个区间内 c_1 发生的次数大于 c_2 发生的次数,则在第 k 步之后的某个时刻 c_1 滴答的累积次数将大于 c_2 的次数.因此,保证因果关系始终被满足的一个额外条件是在从 l 到 k 的整个区间内 c_2 发生的次数应大于或等于 c_1 发生的次数,即$\chi(c_2,k)-\chi(c_2,l)\geqslant\chi(c_1,k)-\chi(c_1,l)$.其他的约束关系也需要类似的条件保证,所需条件与证明可参考文献[11,19],在此不再赘述.此外,像从属关系、互斥关系、并关系与交关系这类不依赖历史的约束,因为各步之间没有关联,则不需要额外的条件就可以保证$\delta_{l,k}$始终满足这些约束.

2 CCSL 约束集到 SMT 公式的转换

CCSL 约束集转换成 SMT 公式是非常直观的.给定一个时钟集合 C 以及在 C 上的 CCSL 约束集Φ,为每一个时钟 $c\in C$ 声明一个函数 $t_c:\mathbb{N}^+\to Bool$.其中,$Bool$ 表示布尔值.对于每一步 $n\in\mathbb{N}^+$,如果 $t_c(n)$为真,则表示时钟 c 在第 n 步滴答一次;否则,表示空闲.因此,对任意 $n\in\mathbb{N}^+$,$t_c(n)\Leftrightarrow c\in\delta(n)$成立,即,一个调度$\delta$就可由一个函数集合 $T=\{t_c|c\in C\}$表示.再为每一个时钟 c 声明一个函数 $h_c:\mathbb{N}^+\to\mathbb{N}$,使得对任意 $n\in\mathbb{N}^+$有下式成立:

$$h_c(n)=\begin{cases}0, & \text{if } n=1\\ h_c(n-1), & \text{if } n\geqslant 2\wedge\neg t(n-1).\\ h_c(n-1)+1, & \text{if } n\geqslant 2\wedge t(n-1)\end{cases}$$

易证明:对于任意 $n\in\mathbb{N}^+$,都有 $h_c(n)=\chi(c,n)$.

根据表 1 中定义的 CCSL 的语义,代入 h_c 和 t_c,得到 CCSL 对应的 SMT 公式,见表 2.对于一个 CCSL 约束关系ϕ,其对应的 SMT 公式记为$[\![\phi]\!]$.另外,根据定义 1,对每一步 $n\in\mathbb{N}^+$,δ都应满足$\delta(i)\neq\varnothing$.该条件对应的 SMT 公式可定义为:$\forall n\in\mathbb{N}^+,\vee_{c\in C}t_c(n)$.

Table 2 Encoding CCSL constraints into SMT formulas

表 2 CCSL 约束到 SMT 公式的转换

CCSL 约束 ϕ	SMT 公式
$c_1[b]\prec c_2$	$\forall n\in\mathbb{N}^+, h_{c_2}(n)-h_{c_1}(n)=b\Rightarrow\neg t_{c_2}(n)$
$c_1\preccurlyeq c_2$	$\forall n\in\mathbb{N}^+, h_{c_2}(n)\geqslant h_{c_1}(n)$
$c_1\subseteq c_2$	$\forall n\in\mathbb{N}^+, t_{c_1}(n)\Rightarrow t_{c_2}(n)$
$c_1\#c_2$	$\forall n\in\mathbb{N}^+, \neg t_{c_1}\vee\neg t_{c_2}$
$c_1\triangleq c_2+c_3$	$\forall n\in\mathbb{N}^+, t_{c_1}(n)\Leftrightarrow(t_{c_2}(n)\vee t_{c_3}(n))$
$c_1\triangleq c_2*c_3$	$\forall n\in\mathbb{N}^+, t_{c_1}(n)\Leftrightarrow(t_{c_2}(n)\wedge t_{c_3}(n))$
$c_1\triangleq c_2\wedge c_3$	$\forall n\in\mathbb{N}^+, h_{c_1}(n)=\max(h_{c_2}(n),h_{c_3}(n))$
$c_1\triangleq c_2\vee c_3$	$\forall n\in\mathbb{N}^+, h_{c_1}(n)=\min(h_{c_2}(n),h_{c_3}(n))$
$c_1\triangleq c_2\$d$	$\forall n\in\mathbb{N}^+, h_{c_1}(n)=\max(h_{c_2}(n)-d,0)$
$c_1\triangleq c_2\$d$ on c_3	$\forall n\in\mathbb{N}^+, t_{c_1}(n)\Leftrightarrow(t_{c_3}(n)\wedge\exists m\in\mathbb{N}^+,(t_{c_2}(m)\wedge h_{c_3}(n)-h_{c_3}(m)=d))$
$c_1\triangleq c_2\propto p$	$\forall n\in\mathbb{N}^+, t_{c_1}(n)\Leftrightarrow(t_{c_2}(n)\wedge\exists m\in\mathbb{N}^+, h_{c_3}(n)=m\times p-1)$
$c_1\triangleq c_2\curlywedge c_3$	$\forall n\in\mathbb{N}^+, t_{c_1}(n)\Leftrightarrow(t_{c_3}(n)\wedge(\exists m\in\mathbb{N}^+, t_{c_3}(m)\wedge h_{c_3}(n)-h_{c_3}(m)=1\wedge h_{c_2}(n)-h_{c_2}(m)\geqslant 1))$

综合以上提到的 t_c、h_c、t_c 与 h_c 之间关系以及任意一个 CCSL 约束 ϕ 对应的 SMT 公式,给定一个时钟集合 C 以及定义在 C 上的约束关系 Φ,则得到如下 SMT 公式,记为 $[\![\Phi]\!]$:

$$[\![\Phi]\!]=\wedge_{\phi\in\Phi}[\![\phi]\!]\wedge(\forall n\in\mathbb{N}^+,\vee_{c\in C}t_c(n)).$$

在第 1.2 节中提到的 CCSL 可调度问题,等价于公式 $[\![\Phi]\!]$ 的可满足性问题.根据 SMT 标准库 SMT-LIB 的定义,$[\![\Phi]\!]$ 由于包含量词、未解释函数(unintepreted function)和线性整数演算(linear arithmatic),因而属于 UFLIA 逻辑.而 UFLIA 逻辑公式的可满足性问题是不可判定的.因此,对于某些 CCSL 约束集合 Φ,其对应的 $[\![\Phi]\!]$ 的可满足性可能无法判定.

一个解决办法是,通过设置一个边界值 n 消去公式中的量词.消去量词后的公式记为 $[\![\Phi]\!]_n$,其可满足性问题是可判定的.一个满足 $[\![\Phi]\!]_n$ 的解即为一个上界为 n 的限界调度.若 $[\![\Phi]\!]_n$ 是不可满足的,则可推出不存在一个调度满足约束集 Φ,即 Φ 是不可调度的.然而,该方法无法判定 Φ 是否是可无限调度的.

判定 Φ 是否是无限可调度的不完全的方法是,判定是否存在一个可被扩展为周期调度的限界调度.即:对于上界为 n 的限界调度,判断是否存在 $l,k(l<k\leqslant n)$,使得 $\wedge_{c\in C}t_c(l)\Leftrightarrow t_c(k)$;且对于 Φ 中某些约束,应满足其对应的额外条件.这些额外条件向 SMT 公式的转化已经在前期工作[11]中做过介绍,这里不再赘述.转化的 SMT 公式与 $[\![\Phi]\!]_n$ 合取公式记为 ${}^{l}_{k}[\![\Phi]\!]_n$.同样地,${}^{l}_{k}[\![\Phi]\!]_n$ 的可满足性是可判断的.若存在一个解,则说明 Φ 是无限可调度的.然而,如果不存在这样的解,并不意味着 Φ 是不可无限调度的.

3 基于 SMT 的 CCSL 形式化分析方法

基于 SMT 的方法除了上述用于判断 CCSL 约束的可调度性之外,对 CCSL 形式化分析还有更广泛的应用.本节给出了基于 SMT 方法的对 CCSL 约束形式化分析方法,包括定理证明和模型检测.其中,基于定理证明的方法包括有效性证明和迹分析,基于模型检测的方法包括死锁检测和 LTL 模型检测.以下给出这些方法应用的具体内容,并通过案例验证和分析其有效性与高效性.案例分析中的实验环境为 Windows 10 家庭版,硬件配置为 Intel® Core™ i5-5200U 处理器与 4GB 内存.

3.1 有效性证明(validity proving)

对于 CCSL 的分析,一个重要的内容是约束之间的关系,比如优先关系蕴含因果关系,即:如果两个时钟满足优先关系,那么他们一定满足因果关系.相关的证明可见文献[6].更一般地,问题可描述为给定一个 CCSL 约束

集Φ与另一个约束ϕ,证明任何满足Φ的调度都满足ϕ.若Φ与ϕ满足这种关系,称Φ可推出ϕ,记为$\Phi\vdash\phi$.

定义 7. 对于一个 CCSL 约束集Φ和一个约束ϕ,$\Phi\vdash\phi$成立当且仅当于每个调度$\delta\in\Delta$,$\delta\models\Phi\Rightarrow\delta\models\phi$.

其中,$\Delta=\mathbb{N}^+\times 2^C$,$C$表示$\Phi$与$\phi$中所有时钟的集合.

然而,目前证明$\Phi\vdash\phi$成立的方法还仅限于手动证明.有些证明不仅过程繁杂,而且再推导过程中容易犯错.

证明$\Phi\vdash\phi$成立的问题在 CCSL 转化为 SMT 公式的基础上转化为 SMT 求解问题.根据第 2 节介绍的 CSL 约束集在时钟集 C 上的调度可以转换成一个函数集合 T,δ满足Φ当且仅当 T 是$[\![\Phi]\!]$的一个解.同样地,δ满足ϕ当且仅当 T 是$[\![\phi]\!]$的一个解.根据定义 7,$\Phi\vdash\phi$等价于$[\![\Phi]\!]\Rightarrow[\![\phi]\!]$成立.因此,有如下命题:

命题. 对于一个 CCSL 约束集Φ和一个约束ϕ,$\Phi\vdash\phi$成立当且仅当$[\![\Phi]\!]\Rightarrow[\![\phi]\!]$成立.

根据该命题,我们知道,证明$\Phi\vdash\phi$成立等价于证明公式$[\![\Phi]\!]\Rightarrow[\![\phi]\!]$是永真的,即,它的非$[\![\Phi]\!]\wedge\neg[\![\phi]\!]$是永假的.对于证明$[\![\Phi]\!]\wedge\neg[\![\phi]\!]$是永假的,现有的 SMT 求解器如 Z3 和 CVC4 对有些公式可直接返回结果,对有些公式则必须通过设定一个上界 n 消除量词之后返回结果.对于这两种情况,如果返回的结果为不可满足的,则说明$[\![\Phi]\!]\wedge\neg[\![\phi]\!]$是永假的,即证明了$\Phi\vdash\phi$成立.如果返回的结果是可满足的,此时求解器会返回 T 的一个解,对于第 1 种情况,则说明$\Phi\vdash\phi$不成立,且返回的解是说明$\Phi\vdash\phi$不成立的一种情况.对于第 2 种情况,并不能证明$\Phi\vdash\phi$是不成立的,需要适当增加上界 n 的值继续求解.

表 3 列出了一些通过自动定理证明的方法证明的一些性质,证明所需要的时间以及是否需要设定上界值.首先考虑不设置上界值得情况,如果求解器可返回结果,则证明结束.如果在限定的时间内无法返回结果,则设置一个边界值(默认为 100)继续求解.如果在设置边界值的情况下返回的结果为可满足的,则继续增加边界值直到结果为不满足,或者超时.

Table 3 Automatic proof of some properties of CCSL constraints

表 3 一些 CCSL 约束集属性的自动证明

性质	定义	边界值	时间(s)
优先关系蕴含因果关系	$c_1\prec c_2\vdash c_1\preccurlyeq c_2$	100	3.70
优先关系具有传递性	$c_1\prec c_2,c_2\prec c_3\vdash c_1\prec c_3$	100	12.50
因果关系具有传递性	$c_1\preccurlyeq c_2,c_2\preccurlyeq c_3\vdash c_1\preccurlyeq c_3$	∞	0.01
从属关系具有反对称性	$c_1\subseteq c_2,c_2\subseteq c_1\vdash c_1=c_2$	∞	0.01
下确界关系蕴含因果关系	$c_1\triangleq c_2\wedge c_3\vdash c_1\preccurlyeq c_2,c_1\preccurlyeq c_3$	∞	0.01
上确界关系蕴含因果关系	$c_1\triangleq c_2\vee c_3\vdash c_2\preccurlyeq c_1,c_3\preccurlyeq c_1$	∞	0.01
延迟关系蕴含优先关系	$\forall d\in\mathbb{N}^+,c_1\triangleq c_2\$d\vdash c_2\prec c_1$	∞	0.01

由此可见,通过设定边界值的证明方法是不完全的.但该方法的优势在于:通过 SMT 公式的转换,借助 SMT 求解器的高效性实现某些定理的证明,方法具有一定的实用性.另外,实验结果表明:对于一些定理证明,所需要的边界值都比较小.

3.2 迹分析

迹又被称为执行迹(execution trace),指执行路径上每一步发生的事件集合.在实时嵌入式系统中,相应的插桩代码经过一段时间产生一个有限长度的事件序列.在 CCSL 中,事件由时钟表示,因此,一个迹本质上是一个有界的调度.有别于定理证明与死锁检测,因为迹是有界的,判断一个迹是否满足对应的 CCSL 约束是一个可判定问题.

一段长度为 n 的迹可定义为上界为 n 的限界调度δ,其对应的 SMT 公式可定义为

$$\wedge_{i\in(1,\dots,n)}\wedge_{c\in C}t_c(i)=v.$$

其中,如果 $c\in\delta(i)$,则 v 表示 true;否则为 false.

对于给定的 CCSL 约束集Φ,验证这段迹是否满足Φ等价于验证上述公式与$[\![\Phi]\!]_n$是否是同时可满足的.如果是,则说明迹满足Φ;否则则不满足.

以红绿闪烁灯为例,其 CCSL 约束集合为{*green*≺*red*,*tmp*≜*green*$1,*red*≺*tmp*},表示红绿灯交替地出现.图 1 表示在边界值为 100 时的一个迹.将迹按照上述公式转换成 SMT 公式并连同约束对应的公式一起验证其可满足性.求解器返回的结果为可满足.为了模拟不满足的情况情况,对部分时钟的行为进行修改,例如,将时钟 *green* 在 95 步时改为不发生.再次对修改后的迹分析,求解器返回的结果为不满足.

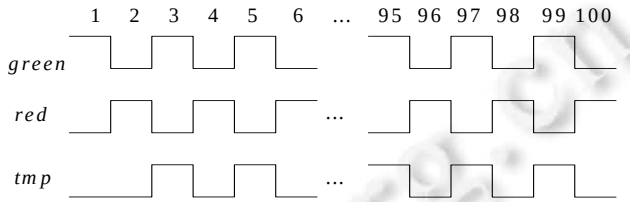

Fig.1 Result of the example with bound 100

图 1 该例子在边界值为 100 下的结果

然而,上述方法并不能显示迹在哪一步违反了约束.为了定位错误出现的位置,将原来的公式$[\![\Phi]\!]_n$修改为如下公式:

$$\exists d\in\{1,\ldots,n\},[\![\Phi]\!]_{d-1}\wedge\neg[\![\Phi]\!]_d.$$

即:存在 1~*n* 的整数 *d*,使得迹在 *d* 步前都满足Φ而在 *d* 步时不满足.因此,找出出错位置只需要用 SMT 求解器求解同时满足上述两个公式的 *d* 的值即可.

表 4 展示了长度为 200 的迹进行验证的结果.实验分别对不同时钟进行不同位置的错误修改,结果显示,求解时间随着出错位置增大而增长.对于上述公式来说,*d* 值增大,因此 SMT 求解器需要花费更多时间.综上来说,本小节提出的两个验证公式对于迹的分析来说都是有效的,适用于判断迹的满足性与分析迹不满足的情况.

Table 4 Verification result on the unsatisfied trace

表 4 迹不满足情况下的验证结果

出错位置	*green*	*red*	*tmp*
	时间(s)	时间(s)	时间(s)
45	2.10	2.40	2.40
95	8.50	7.60	9.50
145	18.80	19.00	23.20
195	37.70	36.10	38.00

3.3 死锁检测

对于任何系统来说,无死锁都是系统的基本需求.对 CCSL 约束而言,所有满足约束的系统行为都被视为合法行为.因此,CCSL 约束需要保证任何的合法行为都不会导致系统进入死锁状态.这里的死锁指在某个时刻没有任何一个时钟对应的事件可以发生,每个时钟都在等待其他的时钟.

本节介绍利用 SMT 求解的方式检测死锁.对于给定的 CCSL 约束集Φ,存在死锁等价于存在上界为某个自然数 *n* 的限界调度满足Φ,但在 *n*+1 步时,任何时钟产生滴答都会造成Φ中的某个ϕ不被满足.其对应的 SMT 公式如下:

$$\exists n\in\mathbb{N}^+,[\![\Phi]\!]_n\wedge(\forall c\in C.\forall t_c(n+1)\in\mathcal{B}\Rightarrow\vee_{\phi\in\Phi}\neg[\![\phi]\!]_{n+1}^{n+1}),$$

其中,$\mathcal{B}$表示布尔值的集合,$[\![\phi]\!]_{n+1}^{n+1}$表示公式$[\![\phi]\!]$中的量词被消去后且 *n*+1 替换 *n* 之后的公式.如果对上面的公式 SMT 求解器返回一个解,则说明公式是可满足的,即证明Φ存在死锁的情况,且返回的解则为具体的死锁实例.然而,该公式的可满足性问题是不可判定的,需要设置一个边界值消去量词后使其可判定.如果在这个边界内公式为真,则找到死锁;否则,意味着在该边界内不存在死锁.

举例来检测 CCSL 约束集的死锁.这个例子是从 AADL 规范的延时分析抽象出来的实际应用组件[6],如图 2 所示.这个组件有两个输入 in_1 和 in_2,在 $step_1$ 和 $step_2$ 中分别计算,然后在 $step_3$ 中计算,最后结果由 *out* 输出.这其

中不断地获取新的输入和产生结果.这个组件中的事件约束可以用 CCSL 描述,图 3 描绘了这些事件的 9 个约束.虚线箭头部分表示了两个中间事件 tmp_1 和 tmp_2 与其他事件的之间的约束.譬如,我们可以得到:

$$tmp_1 \triangleq in_1+in_2.$$

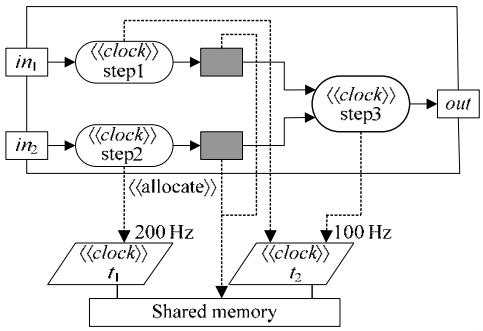

Fig.2 A component in a practical application

图 2 一个实际应用的组件

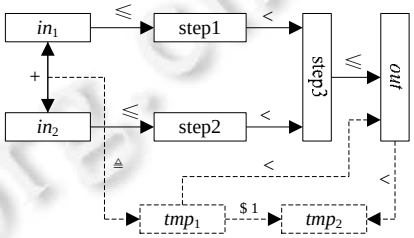

Fig.3 An abstraction as a clock constraint specification

图 3 抽象成时钟约束规范

通过设置边界值 n,我们将这 9 个约束转换成上述的求解公式检测其可满足性.表 5 展示了用 Z3 求解的结果,若公式满足,则表示存在一个死锁.否则表示不存在.另外,我们分别用 $tmp_1 \triangleq in_1 \wedge in_2$ 和 $tmp_1 \triangleq in_1 \vee in_2$ 代替 $tmp_1 \triangleq in_1+in_2$ 检测该公式的满足性.

Table 5 Verification result on the deadlock in the example

表 5 验证死锁的结果

上界	$tmp_1 \triangleq in_1+in_2$		$tmp_1 \triangleq in_1 \wedge in_2$		$tmp_1 \triangleq in_1 \vee in_2$	
	死锁?	时间(s)	死锁?	时间(s)	死锁?	时间(s)
1	Yes	4.40	No	0.01	No	0.01
10	Yes	4.20	No	0.60	No	0.30
20	Yes	14.20	No	2.40	No	1.70
30	Yes	98.60	No	6.10	No	4.10
40	Yes	50.00	No	7.00	No	7.40
50	Yes	141.10	No	10.00	No	12.40

从表中我们看到:$tmp_1 \triangleq in_1+in_2$ 造成了死锁,而 $tmp_1 \triangleq in_1 \wedge in_2$ 和 $tmp_1 \triangleq in_1 \vee in_2$ 则没有检测到死锁.图 4 表示了在用 $tmp_1 \triangleq in_1+in_2$ 关系且上界设为 10 时,相关的时钟从第 1 步~第 10 步的行为及在第 11 步时发生死锁的情景.

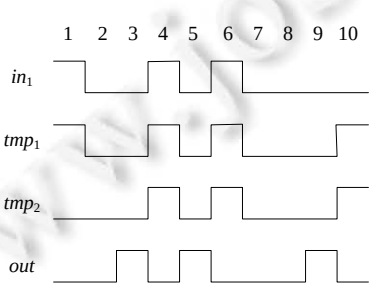

Fig.4 Clocks with deadlock

图 4 存在死锁的时钟集合

在第 11 步时,若 in_1 发生,因为 $tmp_1 \triangleq in_1+in_2$ 约束,tmp_1 必须发生,从而导致 tmp_2 也必须发生.由于 $out \triangleq tmp_1\$1$, out 在此步不会发生.而 tmp_2 与 out 存在优先关系约束,out 要比 tmp_2 先发生,从而导致死锁.死锁产生的主要原因是 $tmp_1 \triangleq in_1+in_2$ 约束制约太弱,不能保证 in_1 和 in_2 在 out 之前都要发生.而通过下确界与下确界关系约束,则可以

保证这一点,从而避免死锁发生.效率方面,随着界的增大,执行时间增大,然而都可以在较短的时间内完成检测.另外,从表中可以看出,无死锁的求解时间明显较短.

3.4 LTL模型检测

借助 SMT 方法实现 CCSL 的 LTL 模型检测的主要思想是将一个 LTL 公式ψ,根据既定的边界值 n 判断其与由 CCSL 转化成的 SMT 公式的可满足性.若结果是可满足的,则证明ψ是不可满足的,所求的解是ψ不被满足的实例.若验证结果是不满足的,则说明在 n 步之前ψ是被 CCSL 约束满足的.

LTL 公式转换成 SMT 公式分为两种.

- 第 1 种是δ为限界调度,根据 LTL 的语义,对于 LTL 公式ψ,以 $G(\psi_1 \wedge \psi_2)$为例,我们用公式$[\![\psi]\!]_n^1$表示满足ψ的上界为 n 的限界调度δ,通过判断ψ中的连接词来分解公式,得到公式$\forall i \geqslant 1, ([\![\psi_1]\!]_n^1 \wedge [\![\psi_2]\!]_n^1)$.再对子公式$[\![\psi_1]\!]_n^1 \wedge [\![\psi_2]\!]_n^1$进行判断分解直到子公式只有原子公式为止.递归这个过程,最终就得到了该 LTL 公式的 SMT 公式;
- 第 2 种是δ为周期调度,比第 1 种情况较为复杂.其主要思想如下:用公式${}_l[\![\psi_1]\!]_k^1$表示满足 LTL 公式ψ的周期调度$\delta_{l,k}$.因为$\delta_{l,k}$到了 k 步后就会重复 l 步~k 步间的内容,因此逻辑连接词并不受影响.而对于时态连接词则不同,比如 X(next),验证 $X\psi$在 k 步时的满足性就等价于验证ψ在 l+1 步的满足性;又如 G(globally),在无限调度(周期调度)下才有意义,验证 $G\psi$在 l 步前的满足性就等价验证ψ是否在 l 步前处处满足.而在 l 步后,因为周期的存在我们只需验证在一个周期内的满足性即可.

结合第 3.1 节,对于 CCSL 约束集Φ,我们可以通过 LTL 公式对其时间属性进行模型检测.令ψ表示该 LTL 公式,若$[\![\Phi]\!] \wedge \neg [\![\psi]\!]$是不可满足的,则证明该约束集具有$\psi$表示的时间属性.当然,我们通常设置一个边界值 n,用公式$[\![\Phi]\!]_n \wedge \neg [\![\psi]\!]_n^1$来验证限界调度下的满足情况.但在实际系统验证中,周期调度更有实际意义,我们用公式${}_k^l[\![\Phi]\!]_n \wedge \neg {}_l[\![\psi]\!]_k^1$来表示.我们在文献[19]中给出了从 LTL 公式到 SMT 公式的转换过程,并且以一个交通灯控制系统[14]和车载系统中的车窗控制系统为例,验证了该方法的有效性.有兴趣的读者可见文献[19]了解更多细节.

4 工 具

为了使以上对于CCSL形式化验证与分析的方法有更好的可重复性和扩展性,我们将其集成于一个原型工具,其 UML 活动图如图 5 所示.

工具的核心思想是:根据各个形式化方法将给定的 CCSL 约束、LTL 公式转换成对应的 SMT 公式,并放入 SMT 求解器中求解,最终得到结果.它具有以下几点功能.

(1) 在转换前需要输入运行参数:程序超时时间,防止运行时间过长而不能终止的情况;SMT 求解器的选择,可以选择使用 Z3 还是 CVC4 求解 SMT 公式;边界值的设定,当为 0 时表示无界的检测,一般我们会输入个正整数作为检测的界;

(2) 对于输入的 CCSL 约束集合,选择需要进行的操作,如果需要 LTL 模型检测,则要输入 LTL 公式.根据不同操作,工具自动进行转换成对应的 SMT 公式并求解;

(3) 对于 SMT 求解器返回的结果进行处理,不仅提示有没有解,并且对于满足的情况画出结果,整个工具采用图形化界面,具有较强的可操作性;

(4) 采用基于 nodejs 的跨平台框架 electron 进行开发,使得工具便于维护与移植.

图 6 是工具在死锁检测时的页面截图,展示了该原型工具在参数设置、公式输入和结果处理上的交互逻辑.

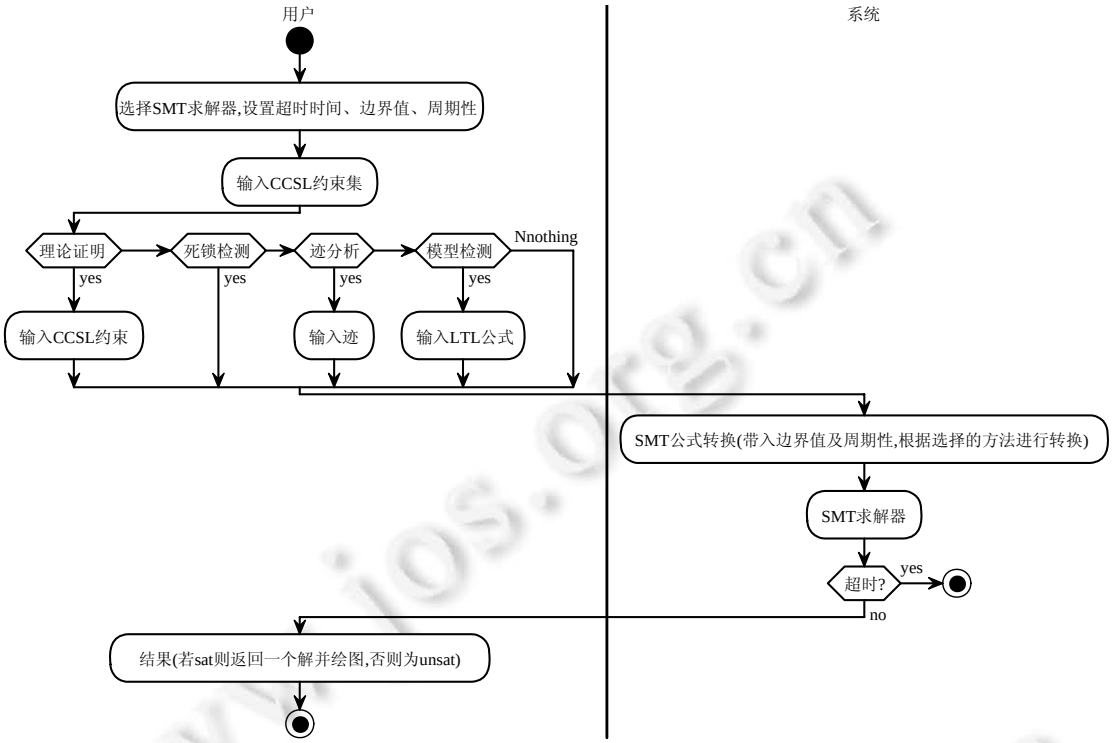

Fig.5 UML activity diagrams of the tool

图 5 该工具的 UML 活动图

Fig.6 Screenshot of the tool on deadlock detection

图 6 死锁检测时的工具截图

5 相关工作

对于CCSL约束的形式化分析方法的相关工作大多基于状态迁移系统与自动机原理.这些方法的优势在于可以利用现有的验证工具对CCSL验证分析.然而受限于目标模型的表达能力,有些工作中只考虑了部分CCSL约束关系.另外,由于对应的验证工具只能实现某种验证与分析,无法对CCSL进行其他方面的分析.例如:Yin等

人提出在 Promela 里对 CCSL 操作符进行转换[7],然后用 Spin 工具进行模型检测;Gaston 等人是将其转换成 Büchi 自动机[8],然后比较 CCSL 与时态逻辑的表现.但这两个方法仅考虑了 CCSL 安全子集.Yu 等人提出用 SINGAL 做 CCSL 的转换[18],但也仅包含一部分,并且 SINGAL 依赖三界逻辑.Suryadevara 等人是将其转换为时间自动机[9],但 UPPAAL 只能验证 CCSL 安全和物理时间的那部分约束.Mallet 等人提出基于状态的 CCSL 语义并将每个约束转换为一个迁移系统[6],然而一些如 Precedence,Supremun 这样的约束并不能用有限状态迁移系统表示.Zhang 等人用 Maude 定义了一套 CCSL 可执行语义[10]进行仿真和模型检测.

本文提出的方法是将 CCSL 约束转换为 SMT 公式并进行统一化求解.本文提出的方法不是单一的针对某个问题而展开,而是系统性地将多个问题统一而有效的解决.虽然该方法在不设置界的情况下满足性问题同样不能完全得到解决,但是基于 SMT 求解器的高效性及其在有界条件下在不同分析中的应用,基于 SMT 的方法依然是对 CCSL 形式化分析的有效方法之一.

已经有工作[16,17]将 LTL 公式转换成 SAT 公式,它们将有限状态自动机模型转换成命题逻辑公式,LTL 公式的转换考虑到两种情况:一种是路径里没有循环,另一种是存在循环,同本文提出的对限界调度和周期调度的验证方法思路基本一致.不同之处有两点:其一是模型不同,本文基于 CCSL 的调度模型;其二是转换成公式为 SMT 公式.

6 总结以及未来工作

本文提出了一种基于 SMT 的统一的 CCSL 形式化分析方法并集成于一个原型工具,介绍了该方法在 CCSL 形式化分析中多种不同的应用,包括有效性证明、迹分析、死锁检测以及 LTL 模型检测.同时展示了他们在具体实例中的应用.实验结果和数据说明了基于 SMT 的方法对 CCSL 形式化分析的有效性及高效性.

在本文工作的基础上有更多的工作值得研究:如何自动确定合适的边界值、如何将描述 CCSL 约束时间属性的 CTL 公式转换成 SMT 公式等.另外,拟将该方法应用于工业案例中更复杂系统的验证,以检验该方法在实际工业案例中的有效性.此外,原型工具也需要进一步完善用于复杂系统的 CCSL 建模与分析,如引入图形化操作实现对 CCSL 约束的定义、对验证结果进行图形化表示使其更容易理解等.

本文中,CCSL 约束有效性证明采用的是对证明的公式取反证明其不可满足的方法.受限于转化的公式的可满足性问题的不可判定性,这种方法是不完全的.*K*-induction 是另一个潜在的方法,是基本数学归纳证明方法的扩展,形式化验证工具 SAL 实现了基于 *K*-induction 方法的定理证明[21].利用 *K*-induction 实现 CCSL 某些定理的证明可避免设定边界值证明的局限性,这又是一个值得深入研究的工作.

References:

[1] Object Management Group. UML profile for MARTE: Modeling and analysis of real-timeembedded systems. 2011.

[2] André C. Syntax and semantics of the clock constraint specification language (CCSL). Research Report, RR-6925, INRIA, 2009.

[3] Mallet F, André C. On the semantics of UML/MARTE clock constraints. In: Proc. of the IEEE Int'l Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC 2009). IEEE, 2009. 305-312. [doi: 10.1109/ISORC.2009.27]

[4] Kang EY, Schobbens PY. Schedulability analysis support for automotive systems: From requirement to implementation. In: Proc. of the 29th Annual ACM Symp. on Applied Computing. ACM Press, 2014. 1080–1085. [doi: 10.1145/2554850.2554929]

[5] Mallet F. MARTE/CCSL for modeling cyber-physical systems. In: Proc. of the Formal Modeling and Verification of Cyber-Physical Systems. Springer Fachmedien Wiesbaden, 2015. 26–49. [doi: 10.1007/978-3-658-09994-7_2]

[6] Mallet F, De Simone R. Correctness issues on MARTE/CCSL constraints. Science of Computer Programming, 2015,106:78–92. [doi: 10.1016/j.scico.2015.03.001]

[7] Yin L, Mallet F, Liu J. Verification of MARTE/CCSL time requirements in Promela/SPIN. In: Proc. of the 2011 16th IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS). IEEE, 2011. 65–74. [doi: 10.1109/ICECCS.2011.14]

[8] Gascon R, Mallet F, Deantoni J. Logical time and temporal logics: Comparing UML MARTE/CCSL and PSL. In: Proc. of the 2011 18th Int'l Symp. on Temporal Representation and Reasoning (TIME). IEEE, 2011. 141–148. [doi: 10.1109/TIME.2011.10]

[9] Suryadevara J, Seceleanu C, Mallet F, *et al*. Verifying MARTE/CCSL mode behaviors using UPPAAL. In: Proc. of the Int'l Conf. on Software Engineering and Formal Methods. Berlin, Heidelberg: Springer-Verlag, 2013. 1–15. [doi: 10.1007/978-3-642-40561-7_1]

[10] Zhang M, Mallet F. An executable semantics of clock constraint specification language and its applications. In: Proc. of the Int'l Workshop on Formal Techniques for Safety-Critical Systems. Cham: Springer-Verlag, 2015. 37–51. [doi: 10.1007/978-3-319-29510-72]

[11] Zhang M, Mallet F, Zhu H. An SMT-based approach to the formal analysis of MARTE/CCSL. In: Proc. of the Int'l Conf. on Formal Engineering Methods. Springer Int'l Publishing, 2016. 433–449. [doi: 10.1007/978-3-319-47846-3_27]

[12] Barrett C, Stump A, Tinelli C. The smt-lib standard: Version 2.0. In: Proc. of the 8th Int'l Workshop on Satisfiability Modulo Theories. Edinburgh, 2010. 13–14.

[13] De Moura L, Bjørner N. Z3: An efficient SMT solver. In: Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. 2008. 337–340. [doi: 10.1007/978-3-540-78800-3_24]

[14] Cohen B, Venkataramanan S, Kumari A. System Verilog Assertions Handbook—For Formal and Dynamic Verification. Vhdlcohen Publishing, 2005.

[15] Biere A, Cimatti A, Clarke EM, *et al*. Bounded model checking. Advances in Computers, 2003,58:117–148.

[16] Cimatti A, Pistore M, Roveri M, *et al*. Improving the encoding of LTL model checking into SAT. In: Proc. of the Int'l Workshop on Verification, Model Checking, and Abstract Interpretation. Berlin, Heidelberg: Springer-Verlag, 2002. 196–207. [doi: 10.1007/3-540-47813-2_14]

[17] Zhang W. SAT-Based verification of LTL formulas. In: Proc. of the FMICS/PDMC. 2006. 277–292. [doi: 10.1007/978-3-540-70952-7_18]

[18] Yu H, Talpin JP, Besnard L, *et al*. Polychronous controller synthesis from MARTE CCSL timing specifications. In: Proc. of the 9th ACM/IEEE Int'l Conf. on Formal Methods and Models for Codesign. IEEE Computer Society, 2011. 21–30. [doi: 10.1109/MEMCOD.2011.5970507]

[19] Zhang M, Ying Y. Towards SMT-based LTL model checking of clock constraint specification language for real-time and embedded systems. In: Proc. of the 18th ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems. ACM Press, 2017. 61–70. [doi: 10.1145/3078633.3081035]

[20] Zhang M, Dai F, Mallet F. Periodic scheduling for MARTE/CCSL: Theory and practice. In: Proc. of the Science of Computer Programming. 2017. [doi: 10.1016/j.scico.2017.08.015]

[21] De Moura L, Owre S, Rueß H, *et al*. SAL 2. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin, Heidelberg: Springer-Verlag, 2004. 496–500. [doi: 10.1007/978-3-540-27813-9_45]

[22] Jin JW, Ma FF, Zhang J. Brief Introduction to SMT solving. Journal of Frontiers of Computer Science and Technology, 2015,9(7):769–780 (in Chinese with English abstract).

附中文参考文献:

[22] 金继伟,马菲菲,张健.SMT 求解技术简述.计算机科学与探索,2015,9(7):769–780.

应云辉(1992－),男,浙江宁海人,硕士,主要研究领域为形式化方法.

张民(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为形式化方法,软件工程.