软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(3):670−681 [doi: 10.13328/j.cnki.jos.004988]

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

二维逻辑 PPTL^{SL} 的可满足性检查*

陆 旭[1,2], 段振华[1,2], 田 聪[1,2]

[1](西安电子科技大学 计算理论与技术研究所,陕西 西安 710071)

[2](综合业务网理论及关键技术国家重点实验室(西安电子科技大学),陕西 西安 710071)

通讯作者: 段振华,田聪, E-mail: {zhhduan,ctian}@mail.xidian.edu.cn

摘 要: 由于指针的灵活性以及别名现象的存在,程序的运行可能会出现悬空指针引用、内存泄漏等诸多问题. PPTL^{SL} 是一种二维(时间和空间)时序逻辑,它结合了分离逻辑(separation logic)与命题投影时序逻辑 PPTL (propositional projection temporal logic),能够描述和验证操作链表的指针程序的时序性质.简要回顾了 PPTL^{SL} 的相关理论,并详细介绍了工具 SAT-PPTL^{SL} 的工作原理.该工具主要利用 PPTL^{SL} 与 PPTL 之间构建起来的同构关系进行 PPTL^{SL} 公式的可满足性检查.此外,结合一些实例展示了 SAT-PPTL^{SL} 的执行过程,并通过实验分析了关键参数对 SAT-PPTL^{SL} 执行效率的影响.

关键词: 时序逻辑;分离逻辑;指针;二维逻辑;可满足性

中图法分类号: TP301

中文引用格式: 陆旭,段振华,田聪.二维逻辑 PPTL^{SL} 的可满足性检查.软件学报,2016,27(3):670−681. http://www.jos.org.cn/1000-9825/4988.htm

英文引用格式: Lu X, Duan ZH, Tian C. Checking satisfiability of two-dimensional logic PPTL^{SL}. Ruan Jian Xue Bao/Journal of Software, 2016,27(3):670−681 (in Chinese). http://www.jos.org.cn/1000-9825/4988.htm

Checking Satisfiability of Two-Dimensional Logic PPTL^{SL}

LU Xu[1,2], DUAN Zhen-Hua[1,2], TIAN Cong[1,2]

[1](Institute of Computing Theory and Technology, Xidian University, Xi'an 710071, China)

[2](State Key Laboratory of Integrated Service Networks (Xidian University), Xi'an 710071, China)

Abstract: Programs become more error-prone with inappropriate management of memory because of pointer aliasing, e.g., dereferencing null or dangling pointers, and memory leaks. PPTL^{SL} is a two-dimensional (spatial and temporal) logic which integrates separation logic with PPTL (propositional projection temporal logic). It is useful to describe and verify temporal properties of list manipulating programs. This paper first gives an overview of PPTL^{SL}, and then introduces the foundation of the tool SAT-PPTL^{SL} in detail. SAT-PPTL^{SL} can be used to check the satisfiability of PPTL^{SL} formulas according to the "isomorphic" relationship between PPTL^{SL} and PPTL. In addition, the paper presents examples to show the checking process of SAT-PPTL^{SL} and analyze the effect of some key parameters on the performance of SAT-PPTL^{SL}.

Key words: temporal logic; separation logic; pointer; two-dimensional logic; satisfiability

指针作为程序设计语言不可或缺的基本构成,广泛应用于各种软硬件程序的编制,如 C 语言中指针的显式声明和使用,或者像 Java 语言那样隐式的指针操作.指针程序的验证是目前学术界研究的热点和难点之一.如果程序中存在不当的指针操作,将会导致程序运行中出现严重的内存安全性错误,例如内存泄漏、空指针或者悬

* 基金项目: 国家自然科学基金(61322202, 61133001, 61420106004, 91418201)
Foundation item: National Natural Science Foundation of China (61322202, 61133001, 61420106004, 91418201)
收稿时间: 2015-07-30; 修改时间: 2015-10-20; 采用时间: 2015-11-27 ; jos 在线出版时间: 2016-01-05
CNKI 网络优先出版: 2016-01-05 16:40:02, http://www.cnki.net/kcms/detail/11.2560.TP.20160105.1640.014.html

空指针的引用等.并且由于指针的灵活性所引发的别名问题,加大了程序的检测难度.此外,除了内存安全性,指针程序的时序性质也是亟需验证的一类重要性质.

分离逻辑[1]是霍尔逻辑的变种和扩展,由 Reynolds 于 2002 年提出,其通过引入表达显式分离的逻辑操作符以及相应的推导规则,简化了对指针程序的验证工作.虽然基于分离逻辑的程序验证技术是目前学术界普遍认可的一种重要方法,但需要指出的是,分离逻辑只是在推理方式方面优于其他方法,即局部推理.而从表达能力的角度看,分离逻辑的某些片段并未突破一阶逻辑的范畴[2],描述和验证内存时序性质的能力欠缺.另一方面,以线性时序逻辑(linear temporal logic,简称 LTL)[3]、计算树逻辑(computation tree logic,简称 CTL)[4]为代表的时序逻辑虽然能够表达时序性质,其缺点也显而易见,即,不能描述带有指针的程序的性质.

近些年来,学者们提出了一些用于指针程序验证的时序逻辑.Yahav 等人提出的演化时序逻辑(evolution temporal logic,简称 ETL)[5]是一阶线性时序逻辑,用于描述指针程序的动态行为,包括内存的动态分配和释放.ETL 最大的特点在于能够描述大粒度的内存对象与高层次的线程.Distefano 等人用刻画指针的断言扩展了 LTL,得到的逻辑称为导航时序逻辑(navigation temporal logic,简称 NTL)[6],并提出了基于 Tableau 的模型检测算法.Rieger 提出了一种表达能力较强的指针时序逻辑(temporal pointer logic,简称 TPL)[7],该逻辑也扩展自 LTL.然而,TPL 是不可判定的,Rieger 需要使用抽象技术来建立相应的模型检测算法.文献[8]把 LTL 与 CTL 结合起来得到了一种二维逻辑,LTL 用于描述时序(时间),而 CTL 用于描述内存(空间).由于两个维度都是利用时序逻辑来实现的,使得时间与空间维度之间的区别变得不够清晰,而且性质描述不够简单直观.Brochenin 等人的研究[9]将分离逻辑与 LTL 相结合,得到一种描述内存时序性质的逻辑 LTL^{mem}.但是,Brochenin 选用的分离逻辑片段不包含量词,这限制了 LTL^{mem} 的表达能力,使其不能描述复杂的指针结构.此外,文献[9]主要从逻辑可判定性和计算复杂度的角度展开研究,缺少了相应的支持工具和实验结果.

本文介绍的二维逻辑 PPTL^{SL} 结合了分离逻辑的一个可判定子集与命题投影时序逻辑 PPTL[10],用于描述指针程序的时序性质,并且开发了工具 SAT-PPTL^{SL} 来检查任意 PPTL^{SL} 公式的可满足性.PPTL^{SL} 的优点在于:

(1) 继承了分离逻辑的优势,能够以一种简洁直观的方式表达内存性质.同时,PPTL^{SL} 使用的分离逻辑子集可以描述复杂的指针结构,如链表;

(2) 继承了 PPTL 的优势,PPTL 能够表达完全正则语言[11],其表达能力要强于 LTL 与 CTL.PPTL^{SL} 含有时序操作符";"和"*",前者可以刻画顺序事件的发生,而后者可以刻画循环性质.

SAT-PPTL^{SL} 首先解析 PPTL^{SL} 公式,将其转换为等价可满足的 PPTL^{SL} 的严格子集,称为 RPPTL^{SL};然后,根据 RPPTL^{SL} 与 PPTL 的同构性质,重用 PPTL 的相关理论(逻辑规则以及判定过程等),从而完成公式的检查.工具 SAT-PPTL^{SL} 不仅得到了 PPTL 理论的支持,还通过调用 SMT(satisfiability modulo theories)求解器实现了自动化检查,提高了工具的实际应用价值.

本文以一些简单的 PPTL^{SL} 公式为例,详细地阐述了 PPTL^{SL} 到 PPTL 的同构转换.另外,还以一个经典的指针程序(含有链表操作)为例,说明 PPTL^{SL} 如何描述该程序的时序性质,并详细地展示了 SAT-PPTL^{SL} 的执行过程;同时,通过实验分析了同构转换过程中所使用的一些参数对工具执行效率的影响.

本文第 1 节给出 PPTL^{SL} 的语法和语义.第 2 节简要介绍如何构建 PPTL^{SL} 与 PPTL 之间的同构关系以及 PPTL^{SL} 的判定过程.第 3 节详细介绍工具 SAT-PPTL^{SL},给出若干实例说明同构关系的构建和 PPTL^{SL} 公式的判定.第 4 节给出实验以及分析结果.最后一节总结全文并对研究内容提出进一步展望.

1 二维逻辑 PPTL^{SL}

1.1 分离逻辑

本节简要介绍分离逻辑的一个可判定子集(简称为 SL)[12],SL 的表达式与文献[12]中的定义略有区别,但这并不影响其判定性的结论.令 Var 表示可数无穷的变量集合,Loc 表示有限的内存地址集合,该集合的元素为大于 0 的自然数.常量 0 用于表示空指针,Var=Loc∪{0}表示值的集合.根据分离逻辑的标准语义,内存状态 s 由栈 I_s 和堆 I_h 两部分构成,表示为二元组(I_s,I_h):

栈 $I_s:Var\rightharpoonup Var$;

堆 $I_h:Loc\rightharpoonup Var$.

表达式 e 和 SL 公式 ϕ 的语法定义如下:

$$e::=n|x,$$
$$\phi::=e_1=e_2|e_1\mapsto e_2|\neg\phi|\phi_1\vee\phi_2|\phi_1\#\phi_2|\exists x:\phi,$$

其中,n 为自然数,x 为变量.公式 $e_1\mapsto e_2$ 描述了仅包含一个内存单元的堆,其地址是 e_1,存储的内容为 e_2.公式 $\phi_1\#\phi_2$ 断言整个堆可以被分成两个不相交的子堆,其中,一个子堆使得 ϕ_1 成立,而另外一个子堆使得 ϕ_2 成立.

对于任意表达式 e,其在状态 (I_s,I_h) 下的取值表示为 $(I_s,I_h)[e]$,具体定义为 $(I_s,I_h)[n]=n$ 与 $(I_s,I_h)[x]=x$.

下面给出 SL 公式的语义,采用可满足性关系符号 $\vDash_{SL}$ 来定义.

$$I_s,I_h\vDash_{SL} e_1=e_2 \quad \text{iff} \quad (I_s,I_h)[e_1]=(I_s,I_h)[e_2]$$
$$I_s,I_h\vDash_{SL} e_1\mapsto e_2 \quad \text{iff} \quad dom(I_h)=\{(I_s,I_h)[e_1]\}\text{并且}I_h((I_s,I_h)[e_1])=(I_s,I_h)[e_2]$$
$$I_s,I_h\vDash_{SL} \neg\phi \quad \text{iff} \quad I_s,I_h\nvDash_{SL}\phi$$
$$I_s,I_h\vDash_{SL} \phi_1\vee\phi_2 \quad \text{iff} \quad I_s,I_h\vDash_{SL}\phi_1\text{或者}I_s,I_h\vDash_{SL}\phi_2$$
$$I_s,I_h\vDash_{SL} \phi_1\#\phi_2 \quad \text{iff} \quad \exists I_{h_1},I_{h_2}:I_{h_1}\perp I_{h_2}\text{并且}I_{h_1}\cdot I_{h_2}=I_h\text{并且}I_s,I_{h_1}\vDash_{SL}\phi_1\text{并且}I_s,I_{h_2}\vDash_{SL}\phi_2$$
$$I_s,I_h\vDash_{SL} \exists\phi \quad \text{iff} \quad \exists v\in Val,\text{使得}I_s[x\rightarrow v],I_h\vDash_{SL}\phi$$

函数 f 的定义域记作 $dom(f)$,对于部分函数 f,$dom(f)$ 指的是使得 $f(x)$ 有定义的那些变量 x 的集合.给定两个函数 f_1 和 f_2,符号 $f_1\perp f_2$ 表示 f_1 和 f_2 的定义域不相交,符号 $f_1{\cdot}f_2$ 表示 f_1 和 f_2 的并.函数更新符号 $f[x\rightarrow v]$ 定义为 $f[x\rightarrow v](x)=v$ 并且 $\forall y\neq x;f[x\rightarrow v](y)=f(y)$.

SL 还引入以下简写表示一些形式较为复杂的公式,在实际应用中能够描述较为常见的内存性质:

$$e_1\rightsquigarrow e_2\triangleq e_1\mapsto e_2\#\text{true}$$
$$alloc(e)\triangleq\exists x:e_1\rightsquigarrow e_2$$
$$emp\triangleq\neg\exists x:alloc(x)$$
$$least(n)\triangleq\overbrace{\neg emp\#...\#\neg emp}^{n\ times}$$
$$\natural e\geqslant n\triangleq e\neq 0\wedge(\overbrace{(\exists x:x\mapsto e)\#...\#(\exists x:x\mapsto e)}^{n\ times}\#\text{true})$$
$$\natural e\leqslant n\triangleq e\neq 0\wedge\neg(\overbrace{(\exists x:x\mapsto e)\#...\#(\exists x:x\mapsto e)}^{n+1\ times}\#\text{true})$$
$$\natural e=n\triangleq e\neq 0\wedge(\overbrace{(\exists x:x\mapsto e)\#...\#(\exists x:x\mapsto e)}^{n\ times}\#\text{true})\wedge\neg(\overbrace{(\exists x:x\mapsto e)\#...\#(\exists x:x\mapsto e)}^{n+1\ times}\#\text{true})$$
$$e_1\leadsto e_2\triangleq alloc(e_1)\wedge(e_2\neq e_1\rightarrow\neg alloc(e_2)\wedge\natural e_1=0)\wedge(\forall x:x\neq e_2\rightarrow(\natural x=1\rightarrow alloc(x)))\wedge(\forall x:x\neq 0\rightarrow\natural x\leqslant 1)$$
$$ls(e_1,e_2)\triangleq e_1\leadsto e_2\wedge\neg(e_1\leadsto e_2\#\neg emp)$$
$$e_1\rightarrow^+ e_2\triangleq\text{true}\#ls(e_1,e_2)$$
$$e_1\rightarrow^* e_2\triangleq e_1=e_2\vee e_1\rightarrow^+ e_2$$

这里选择几个比较重要的公式进行解释.$e_1\rightsquigarrow e_2$ 弱化了 $e_1\mapsto e_2$ 的含义,表示除了 e_1 描述的内存单元外,堆中还有可能包含其他内存单元.$alloc(e)$ 说明当前堆中 e 指向的地址是已经被分配的.emp 表示空堆,其中不含有任何有效内存地址.堆中至少含有 n 个有效的内存单元才能使得 $least(n)$ 成立.$ls(e_1,e_2)$ 描述了一段以 e_1 为头部,e_2 为尾部的链表片段.$e_1\rightarrow^+e_2$ 和 $e_1\rightarrow^*e_2$ 属于指针可达性谓词,表示从指针 e_1 开始,可以通过指针的指向关系到达 e_2.如果将单域指针公式 $e_1\mapsto e_2$ 扩展为多域指针公式 $e\mapsto\{e_1,...,e_n\}$,则 SL 可以描述其他指针结构,如双链表和二叉树等.

1.2 分离逻辑的时序扩展

PPTL^{SL} 将 SL 作为状态公式,而把 PPTL 的时序算子作为时序公式的连接算子,从而能够描述指针程序的时序性质.设 $Prop$ 为无限可数的原子命题集合,PPTL^{SL} 公式 P 以及 PPTL 公式 Q 的语法定义如下:

$$P::=\phi|\neg P|P_1\vee P_2|\bigcirc P|(P_1,\ldots,P_m)\ prj\ P|P^*,$$

$$Q::=q|\neg Q|Q_1\vee Q_2|\bigcirc Q|(Q_1,\ldots,Q_m)\ prj\ Q|Q^*,$$

其中,$P\in Prop$,ϕ为 SL 公式,$P_1,\ldots,P_m(Q_1,\ldots,Q_m)$都是良好形式的 PPTL^{SL}(PPTL)公式.“○”(next),“*prj*”(projection)和“*”(star)为时序算子.仅含时序算子的公式称为时序公式,否则称为状态公式.

区间$\sigma\triangleq\langle s_0,s_1,\ldots\rangle$为一个状态序列,$|\sigma|$表示区间长度.如果区间长度是无限的,则$|\sigma|=\omega$;否则,它等于状态数减 1.$N_0$ 指代非负整数集合,并定义 $N_\omega\triangleq N_0\cup\{\omega\}$,$\preceq\triangleq\leqslant\setminus\{(\omega,\omega)\}$.区间$\sigma$的子区间$\langle s_i,\ldots,s_j\rangle$用$\sigma_{(i\ldots j)}(0\leqslant i\preceq j\leqslant|\sigma|)$表示.此外,用$\sigma\cdot\sigma'$表示由区间$\sigma$的末状态和$\sigma'$的首状态连接而成的新区间.PPTL 的解释是三元组$\mathcal{I}\triangleq(\sigma,k,j)$,其中,$\sigma$是区间,$k\leqslant j\leqslant|\sigma|$为整数.解释$\mathcal{I}=(\sigma,k,j)$满足 P,记作$\mathcal{I}\vDash P$.为定义投影算子的语义,需要辅助算子↓.设$\sigma=\langle s_0,s_1,\ldots\rangle$是一个区间,$r_0,\ldots,r_h(h\geqslant 0)$是整数,且有 $0\leqslant r_0\leqslant\ldots\leqslant r_h\leqslant|\sigma|$,则$\sigma$在 $r_0,\ldots,r_h$ 上的投影 $\sigma\downarrow\langle r_0,\ldots,r_h\rangle=\langle s_{t_0},\ldots,s_{t_l}\rangle$.其中,$t_0,\ldots,t_l$是通过删除 $r_0,\ldots,r_h$ 中重复元素所得.例如,$\langle s_0,s_1,s_2\rangle\downarrow(0,0,2,2,2,=\langle s_0,s_2\rangle$.PPTL^{SL} 的可满足关系定义如下:

$\mathcal{I}\vDash\phi$ iff $I_s^k,I_h^k\vDash_{SL}\phi$

$\mathcal{I}\vDash\neg P$ iff $\mathcal{I}\nvDash P$

$\mathcal{I}\vDash P_1\vee P$ iff $\mathcal{I}\vDash P_1$或者$\mathcal{I}\vDash P_2$

$\mathcal{I}\vDash\bigcirc P$ iff $\mathcal{I}=(\sigma,k,j),k<j$并且$(\sigma,k+1,j)\vDash P$

$\mathcal{I}\vDash(P_1,\ldots,P_m)\ prj\ P$ iff 存在整数$k=r_0\leq r_1\leqslant\ldots\leqslant r_m\preceq j$使得$(\sigma,r_0,r_1)\vDash P,(\sigma,r_{l-1},r_l)\vDash P_l(1<l\leqslant m)$,
并且对于符合以下两个条件的σ'来说,$(\sigma',0,|\sigma''|)\vDash P$:
(a) $r_m<j$并且$\sigma'=\sigma\downarrow(r_0,\ldots,r_m)\cdot\sigma_{(r_m+1\ldots j)}$;(b) $r_m=j$并且$\sigma'=\sigma\downarrow(r_0,\ldots,r_h),0\leqslant h\leqslant m$

$\mathcal{I}\vDash P^*$ iff 存在有限多个整数$r_0,\ldots,r_n\in N_\omega$,使得$k=r_0\leqslant r_1\leqslant\ldots\leqslant r_{n-1}\circ r_n$并且$(\sigma,r_0,r_1)\vDash P$
并且对于所有的$1<l\leqslant n,(\sigma,r_{l-1},r_l)\vDash P$;或者存在无限多个整数$k=r_0\leqslant r_1\leqslant r_2\leqslant\ldots$,
使得 $\lim_{i\to\infty}r_i=\omega$并且$(\sigma,r_0,r_1)\vDash P$,并且对于所有的$l>1,(\sigma,r_0,r_1)\vDash P$

下面列出部分常用的 PPTL^{SL} 的导出公式:

$$\varepsilon\triangleq\neg\bigcirc\text{true},$$

$$P_1;P_2\triangleq(P_1,P_2)\ prj\ \varepsilon,$$

$$P^+\triangleq P;P^*,$$

$$\Diamond P\triangleq\text{true};P,$$

$$\Box P\triangleq\neg\Diamond\neg P,$$

$$\bigcirc^nP\triangleq\bigcirc(\bigcirc^{n-1}P),n\geqslant 1.$$

◇与□类似于 LTL 和 CTL 中的含义,只是多了有限模型的情况.ε表示区间的结束,没有后续状态.$P_1;P_2$表示区间可以分为时间先后的两部分:第 1 部分满足 P_1,第 2 部分满足 P_2.P^+和 P^*描述 P 在区间上反复成立,前者要求 P 至少成立一次,后者 P 可以成立 0 次.

2 PPTL 与 PPTL^{SL} 之间的同构关系

本节概要介绍如何构建 PPTL^{SL} 与 PPTL 之间的同构关系.从语法结构的角度看,PPTL^{SL} 与 PPTL 非常相似,只是状态公式有所区别.因此,二者之间存在着紧密的联系.为了构建这种联系,下面将定义 PPTL^{SL} 的一个严格子集 RPPTL^{SL},使其成为连接 PPTL 与 PPTL^{SL} 的纽带.RPPTL^{SL} 公式 P_s 的语法定义如下:

$$P_s::=e_1=e_2\mid\neg P_s\mid P_{s_1}\vee P_{s_2}\mid\bigcirc P_s\mid(P_{s_1},\ldots,P_{s_m})\ prj\ P_s\mid P_s{}^*.$$

给定 PPTL^{SL} 公式 P,从 PPTL^{SL} 到 PPTL 的转换需要两步:第 1 步,将 P 转换为与其等价可满足的 RPPTL^{SL} 公式 P_s;第 2 步,把 P_s 再映射到与其同构的 PPTL 公式.第 1 步从 PPTL^{SL}~RPPTL^{SL} 的等价可满足转换的意义在于:使得我们只需要研究 RPPTL^{SL} 的性质即可,而且 RPPTL^{SL} 的形式更为简单,有助于问题的简化.第 2 步说明 RPPTL^{SL} 完全可以重用 PPTL 的理论,从而证明 PPTL^{SL} 是可判定的.

令 C(可包含上下标)表示含有 n 个变量对的向量,即 $C=((C_{1,1},C_{1,2}),\ldots,(C_{n,1},C_{n,2}))$.第 1 步转换需要利用 C 来模拟内存单元的地址和内容,通过函数 f 和 F 将 PPTL^{SL} 公式映射到 RPPTL^{SL} 公式,f 负责状态公式 ϕ 的转换,而 F 负责公式 P 的整体转换.

$$f(e_1=e_2,C)\triangleq e_1=e_2$$

$$f(e_1\mapsto e_2,C)\triangleq \bigvee_{i\in\{1,\ldots,|C|\}}\left(C_{i,1}\neq 0\wedge\left(\bigwedge_{\substack{j\in\{1,\ldots,|C|\}\\ i\neq j}}C_{j,1}=0\right)\wedge C_{i,1}=e_1\wedge C_{i,2}=e_2\right)$$

$$f(\neg\phi,C)\triangleq\neg f(\phi,C)$$

$$f(\phi_1\vee\phi_2,C)\triangleq f(\phi_1,C)\vee f(\phi_2,C)$$

$$f(\exists x:\phi,C)\triangleq\bigvee_{v\in Val}f(\phi,C)[v/x]$$

$$f(\phi_1\#\phi_2,C)\triangleq\bigvee_{c_2\in Val^{2|C|}}\left(\bigvee_{c_1\in Val^{2|C|}}(C=C'\Theta C''\wedge f(\phi_1,C')\wedge f(\phi_2,C'')\right)[c_1/C'])[c_2/C'']$$

$$F(\varphi,C)\triangleq f(\varphi,C)$$

$$F(\neg P,C)\triangleq\neg F(P,C)$$

$$F(\bigcirc P,C)\triangleq\bigcirc F(P,C)$$

$$F(P_1\vee P_2,C)\triangleq F(P_1,C)\vee F(P_2,C)$$

$$F((P_1,\ldots,P_m)\ prj\ P,C)\triangleq(F(P_1,C),\ldots,F(P_m,C))\ prj\ F(P,C)$$

$$F(P^*,C)\triangleq F(P,C)^*$$

其中,$C=C'\Theta C''\triangleq\bigwedge_{i\in\{1,\ldots,|C|\}}((C'_{i,1}=C_{i,1}\wedge C''_{i,1}=0\wedge C'_{i,2}=C_{i,2})\vee(C''_{i,1}=C_{i,1}\wedge C'_{i,1}=0\wedge C''_{i,2}=C_{i,2}))$,$c_1$ 和 c_2 是与 C 的基数相同的值对向量,φ 指代公式 $e_1=e_2,e_1\mapsto e_2,\exists x:\phi$ 或 $\phi_1\#\phi_2$.

借助文献[2]中的定理 1,可以证明经过 F 转换所得的公式保持了原有公式的可满足性.接下来第 2 步的转换比较直观,很容易看出,RPPTL^{SL} 比 PPTL^{SL} 更接近于 PPTL,因为二者只是原子公式有所不同.由此,根据公式的语法结构,下面给出公式同构的定义.直观的讲,公式同构是 RPPTL^{SL} 公式与 PPTL 公式在保证原子公式一一对应的前提下,语法结构上的同构.

定义 1. 给定 RPPTL^{SL} 公式 P_s 和 PPTL 公式 Q,称 P_s 与 Q 同构,记作 $P_s\cong Q$,当且仅当:

(1) $P_s\cong e_1=e_2,Q\equiv q,q$ 唯一对应于 $e_1=e_2$,或

(2) $P_s\equiv\neg P_{s_1},Q\equiv\neg Q_1,P_{s_1}\cong Q_1$,或

(3) $P_s\equiv P_{s_1}\vee P_{s_2},Q\equiv Q_1\vee Q_2,P_{s_1}\cong Q_1,P_{s_2}\cong Q_2$,或

(4) $P_s\equiv\bigcirc P_{s_1},Q\equiv\bigcirc Q_1,P_{s_1}\cong Q_1$,或

(5) $P_s\equiv(P_{s_1},\ldots,P_{s_m})\ prj\ P_{s_0},Q\equiv(Q_1,\ldots,Q_m)\ prj\ Q_0,P_{s_i}\cong Q_i,0\leqslant i\leqslant m$,或

(6) $P_s\equiv P_{s_1}{}^*,Q\equiv Q_1{}^*,P_{s_1}\cong Q_1$.

基于公式的语法结构,通过构造映射原子公式和完整公式的双射函数,可以证明同构关系确实存在于 RPPTL^{SL} 与 PPTL 之间,从而使得 PPTL 的理论可以被 RPPTL^{SL} 所重用,特别是 PPTL 的判定过程、逻辑规则和相关定义.例如,若 $\bigcirc Q_1;Q_2\equiv\bigcirc(Q_1;Q_2)$ 是 PPTL 中有效的逻辑规则,那么 $\bigcirc P_{s_1};P_{s_2}\equiv\bigcirc(P_{s_1};P_{s_2})$ 也是 RPPTL^{SL} 中有效的逻辑规则.下面简要概述 PPTL^{SL} 公式的判定过程.

定义 2. 给定 RPPTL^{SL} 公式 $P_s,P_=$ 表示 P_s 中出现的原子公式.称 P_s 处于范式(normal form)形式,如果 P_s 能够等价地转换为 $P_s\equiv\bigvee_{j=1}^{n'}(P_{e_j}\wedge\varepsilon)\vee\bigvee_{i=1}^{n}(P_{c_i}\wedge\bigcirc P'_{s_i})$.其中,$P_{e_j}\equiv\bigwedge_{k=1}^{m'}e_{j_k}\doteq e'_{j_k},P_{c_i}\equiv\bigwedge_{h=1}^{m}e_{i_h}\doteq e'_{i_h},e_{j_k}=e'_{j_k},e_{i_h}=e'_{i_h}\in P_=$.对于任意 $e_1=e_2\in P_=,e_1\doteq e_2$ 表示 $e_1=e_2$ 或 $e_1\neq e_2$,P'_{s_i} 是一个 RPPTL^{SL} 公式.

PPTL^{SL} 公式的判定很大程度上依赖于范式这一良好的公式形式.大体上范式分为两部分,分别称作当前部分和未来部分.顾名思义,当前部分刻画的是区间的当前状态,未来部分刻画的则是除了当前状态的后续状态,

如果没有后续状态,则用ε表示.可以证明,任意 RPPTL^{SL} 公式都能够被转换为范式形式,证明过程类似文献[13,14]中的范式证明.此外,范式转换的算法也可以借鉴文献[13,14]中的算法.基于范式,还可以构建与 RPPTL^{SL} 公式相应的范式图(normal form graph,简称 NFG),其定义和构造方法同样可以借鉴 PPTL 理论.范式图的节点是 RPPTL^{SL} 公式,边是范式的当前部分,通过不断构造未来部分的范式,直至没有新的节点和边产生为止.简单地说,RPPTL^{SL} 公式的判定过程就是构造公式的范式图,然后从其范式图中寻找从根节点出发的有效的有限或者无限路径的过程.一旦找到,说明公式可满足;反之,则公式不可满足.

3 原型工具 SAT-PPTL^{SL}

3.1 工具架构

基于 PPTL^{SL} 理论,我们开发了求解 PPTL^{SL} 公式可满足性的原型工具,命名为 SAT-PPTL^{SL}.SAT-PPTL^{SL} 主要包括 4 个模块:转换器、范式图构造器、范式生成器以及路径查找器.这些模块共同完成了 PPTL^{SL} 公式的检查过程.工具将 PPTL^{SL} 公式 P 作为输入,解析后,利用转换器将 P 转换为等价的 RPPTL^{SL} 公式 R.接着,R 被传递给范式图构造器,构造其相应的范式图.在构造的过程中,SMT 求解器 Z3 被调用,主要作用是检查范式图中各个边的可满足性,如果不满足,则不会生成相应的边.第 3.3 节将会详细介绍如何调用 SMT 求解器 Z3.范式生成器也是范式图构造的时候所需要调用的,根据产生的范式来决定是否需要生成新的边和节点,并不断递归调用范式生成器,直至没有新的边和节点出现为止.最终,范式图交由路径查找器处理,由其负责判定是否存在有效的路径,从而给出肯定或否定的判断.如果结果是肯定的,说明 P 是可满足的;否则,P 是不可满足的.

3.2 转换实例

本节通过几个实例详细地说明如何根据同构关系完成 PPTL^{SL} 公式到 PPTL 公式的转换,以及 RPPTL^{SL} 公式的范式及范式图构造.

例 1:令 PPTL^{SL} 公式 $P\equiv\Diamond x=y\vee\Box x\mapsto 0$,参照第 2 节的转换过程,利用函数 F 将 P 转换成等价可满足的 RPPTL^{SL} 公式.首先,观察 P 的结构可知,P 包含子公式 $x\mapsto 0$,说明若要满足该子公式,堆的大小至少为 1.设变量向量 $C=((h,h'))$,转换过程如下:

$$\begin{aligned}f(x\mapsto 0,C)&=f(x\mapsto 0,C)\\&=f(x\mapsto 0,(h,h'))\\&=f(h\neq 0\wedge h=x\wedge h'=0)\\&=h\neq 0\wedge h=x\wedge h'=0,\end{aligned}$$

$$\begin{aligned}F(P,C)&=F(\Diamond x=y\vee\Box x\mapsto 0,C)\\&=F(\Diamond x=y,C)\vee F(\Box x\mapsto 0,C)\\&=\Diamond F(x=y,C)\vee\Box F(x\mapsto 0,C)\\&=\Diamond f(x=y,(h,h'))\vee\Box f(x\mapsto 0,(h,h'))\\&=\Diamond x=y\vee\Box(h\neq 0\wedge h=x\wedge h'=0).\end{aligned}$$

容易看出,$F(P,C)$是一个 RPPTL^{SL} 公式,然后可以找到与其同构的 PPTL 公式 Q.设命题 p_1,p_2,p_3,p_4 分别唯一对应于公式 $x=y,h=0,h=x,h'=0$,那么 $Q\equiv\Diamond p_1\vee\Box(\neg p_2\wedge p_3\wedge p_4)$.$F(P,C)$与 Q 在语法结构上完全相同,唯一的区别在于原子公式.图 1 给出了 $F(P,C)$(图左)和 Q(图右)的范式图.图中采用黑色公式标记节点,粗体公式标记边,双环圆形节点为起始节点,黑色实心节点为ε节点.从图形的角度看,两个图也是同构的,揭示了两个公式的同构关系.另外,边上对应的状态公式也是同构的.

需要指出的是:PPTL^{SL} 的等价可满足转换 F 含有量词展开的过程,这大大增加了转换所得公式的长度.为了解决这一问题,在实际范式的生成过程中,状态公式并未进行彻底转换.原因在于:范式的生成主要由时序操作符所主导,而状态公式只用于刻画当前状态的公式,且量词只在状态公式中出现.

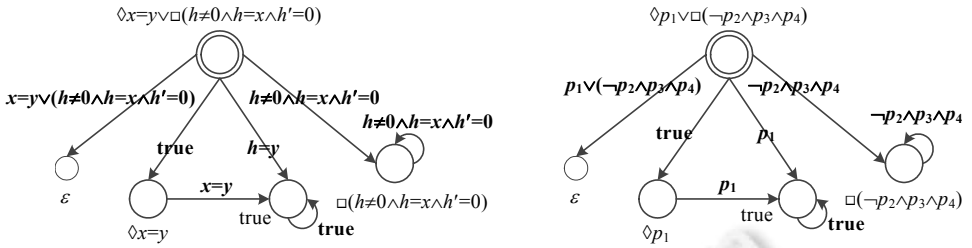

Fig.1 Example normal form graph 1

图 1 范式图示例 1

例 2:令 PPTL^{SL} 公式 $P\equiv(x\neq 0\vee x=y)\wedge\Box(x\mapsto 0\#y\mapsto 0)$,参照第 2 节的转换过程,利用函数 F 将 P 转换成等价可满足的 RPPTL^{SL} 公式 $F(P,C)$,然后构造 $F(P,C)$的范式图.首先,观察 P 的结构可知,P 包含子公式 $x\mapsto 0\#y\mapsto 0$,说明若要满足子公式,堆的大小至少为 2.设变量向量 $C=((h_1,h_1'),(h_2,h_2'))$,转换过程如下:

$$\begin{aligned}F(P,C) &= F((x\neq 0\vee x=y)\wedge\Box(x\mapsto 0\#y\mapsto 0),C)\\ &= F(x\neq 0\vee x=y,C)\wedge F(\Box(x\mapsto 0\#y\mapsto 0),C)\\ &= F(x\neq 0\vee x=y,C)\wedge\Box F(x\mapsto 0\#y\mapsto 0,C)\\ &= (F(x\neq 0,C)\vee F(x=y,C))\wedge\Box F(x\mapsto 0\#y\mapsto 0,C)\\ &= (f(x\neq 0,C)\vee f(x=y,C))\wedge\Box f(x\mapsto 0\#y\mapsto 0,C)\\ &= (x\neq 0\vee x=y)\wedge\Box f(x\mapsto 0\#y\mapsto 0,C).\end{aligned}$$

需要注意的是:这里并未对$\Box f(x\mapsto 0\#y\mapsto 0,C)$做进一步的转换,目的是避免展开量词.类似于 PPTL 构建范式图的方法,通过不断地递归构造未来部分的范式来得到范式图.首先,将 $P\equiv(x\neq 0\vee x=y)\wedge\Box(x\mapsto 0\#y\mapsto 0,C)$转换为范式形式,转换过程如下:

$$\begin{aligned}&(x\neq 0\vee x=y)\wedge\Box f(x\mapsto 0\#y\mapsto 0,C)\\ &\equiv(x\neq 0\vee x=y)\wedge(f(x\mapsto 0\#y\mapsto 0,C)\wedge\varepsilon\vee f(x\mapsto 0\#y\mapsto 0,C)\wedge\bigcirc\Box f(x\mapsto 0\#y\mapsto 0,C))\\ &\equiv f(x\neq 0\wedge(x\mapsto 0\#y\mapsto 0),C)\wedge\varepsilon\vee f(x\neq 0\wedge(x\mapsto 0\#y\mapsto 0),C)\wedge\bigcirc\Box f(x\mapsto 0\#y\mapsto 0,C)\vee\\ &\quad f(x=y\wedge(x\mapsto 0\#y\mapsto 0),C)\wedge\varepsilon\vee f(x=y\wedge(x\mapsto 0\#y\mapsto 0),C)\wedge\bigcirc\Box f(x\mapsto 0\#y\mapsto 0,C).\end{aligned}$$

有新的节点$\Box f(x\mapsto 0\#y\mapsto 0,C)$产生,紧接着,将其转换为范式形式,转换过程如下:

$$\Box f(x\mapsto 0\#y\mapsto 0,C)\equiv f(x\mapsto 0\#y\mapsto 0,C)\wedge\varepsilon\vee f(x\mapsto 0\#y\mapsto 0,C)\wedge\bigcirc\Box f(x\mapsto 0\#y\mapsto 0,C).$$

因为已经没有新的节点产生,由此得到公式 $F(P,C)$的范式图,如图 2(图左)所示.

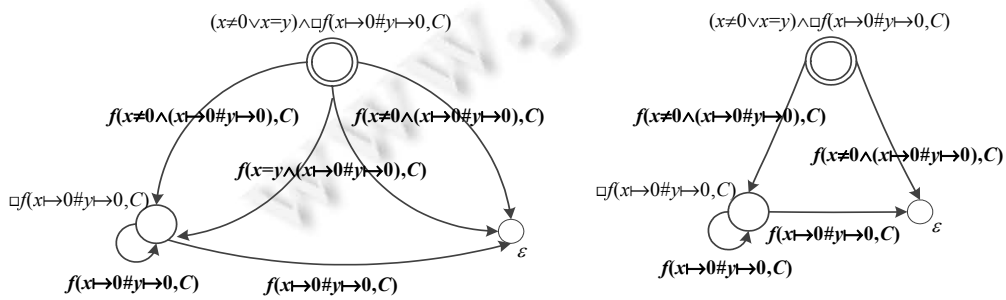

Fig.2 Example normal form graph 2

图 2 范式图示例 2

然而,标记为 $f(x=y\wedge(x\mapsto 0\#y\mapsto 0),C)$的边是不可满足的,因为 x 与 y 别名,即指向同一个内存单元,但是同一个内存单元不可能分离为两个不相交的部分$(x\mapsto 0\#y\mapsto 0)$.因此,凡是标记为 $f(x=y\wedge(x\mapsto 0\#y\mapsto 0),C)$的边都应该从范

式图中删除,其对应的未来部分将不会参与继续构造范式图,正确的范式图如图 2(图右)所示.

3.3 调用SMT求解器Z3

为了能够让工具自动地删除范式图中那些不可满足的边,SAT-PPTL^{SL} 会调用 SMT 求解器完成这项工作.本节主要结合 SAT-PPTL^{SL} 介绍如何使用 SMT 理论检查边的可满足性.

比特向量逻辑(quantifier bit-vector logic,简称 QBVL)[15]是一阶逻辑的特例,它要求每个变量都是比特向量类型.由于每个全称量词(或存在量词)能够等价地展开为有限长度的合取(或析取)公式,因此,QBVL 的可满足性问题是可判定的.但是在实际处理的过程中,量词展开的复杂度是指数级的,目前很多方法都尽量保留量词而不做量词的展开.构建范式的关键在于清晰地将当前部分与未来部分做一个区隔.为了在范式图中找到有限或者无限的从根节点出发的有效路径,需要判定当前部分(边)是否可满足.

PPTL^{SL} 中变量的定义域是有限的,设每个变量为固定长度为 n 的比特向量类型,则变量的取值范围是 $[0,2^n-1]$.因此,范式的当前部分属于 QBVL 的范畴,而 SMT 理论能够很好地支持 QBVL 的求解,构造范式图时便可以结合 SMT 理论来判定每条边是否可满足.

SMT 求解器 Z3[16]是目前业界比较认可的性能优秀的求解器,工具 SAT-PPTL^{SL} 集成了 Z3.下面简要介绍怎样将范式图的边转换为 SMT 求解器所能够接受的语言 SMT-LIB 2.0.整个转换分为 3 个步骤:表达式的转换、当前部分公式的转换以及脚本文件的生成.

(1) 首先介绍表达式的转换.

表达式包括两部分:变量 x 与常量 n.前者的转换比较直接,后者的转换需要将常量重写为二进制、十进制或十六进制的向量表示形式.符号 Bv 是一个函数,用于把常量转换为等价的比特向量形式.具体的转换实现为函数 $Texp(e)$:

Function $Texp(e)$
  case
    e is n: return $Bv(n)$; e is x: return x;
  end case
end Function

(2) 然后介绍范式当前部分公式的转换.

变量向量 C 可能会表示非法的堆,因为相同的地址可能会出现多于一次,而堆中的地址都是唯一的.例如,设向量 $C=((C_{1,1},C_{1,2}),(C_{2,1},C_{2,2}))$,满足 $(C_{1,1}=C_{2,1})\neq 0$,该向量就不能表示合法的堆.合法堆 $VH(C)$ 的定义为

$$VH(C)\triangleq \bigwedge_{\substack{i,j\in\{1,\ldots,|C|\}\\ i\neq j}}(C_{i,1}=0\vee C_{j,1}=0\vee C_{i,1}\neq C_{j,1}).$$

简单地说,$VH(C)$ 通过限制变量的取值使得堆中的地址唯一而保证 C 可以表示合法的堆.

范式当前部分转换的核心主要是状态公式的转换,算法 1 和算法 2 实现了该过程.

算法 1. 将状态公式 ϕ 转换为相应的 QBVL.

Function $TQBVL(\phi,C)$
  case
    ϕ is true: return true;
    ϕ is $e_1=e_2$: return $e_1=e_2$;
    ϕ is $e_1\mapsto e_2$: return $\bigvee_{i\in\{1,\ldots,|C|\}}\left(C_{i,1}\neq 0\wedge\left(\bigwedge_{\substack{j\in\{1,\ldots,|C|\}\\ i\neq j}}C_{j,1}=0\right)\wedge C_{i,1}=e_1\wedge C_{i,2}=e_2\right)$;
    ϕ is $\phi_1\#\phi_2$: return $\exists C',C'':C'\Theta C''\wedge f(\phi_1,C')\wedge f(\phi_2,C'')$;
    ϕ is $\neg\phi_1$: return $\neg TQBVL(\phi_1,C)$;
    ϕ is $\phi_1\vee\phi_2$: return $TQBVL(\phi_1,C)\vee TQBVL(\phi_2,C)$;

ϕ is $\phi_1 \wedge \phi_2$: return $TQBVL(\phi_1,C) \wedge TQBVL(\phi_2,C)$;

ϕ is $\exists x:\phi_1$: return $\exists x:TQBVL(\phi_1,C)$;

ϕ is $\forall x:\phi_1$: return $\forall x:TQBVL(\phi_1,C)$;

end case

end Function

算法 2. 将状态公式ϕ转换为相应的 QBVL,同时要求变量向量 C 表示合法堆.

Function $TQBVL\_VH(\phi,C)$

retrun $TQBVL(\phi,C) \wedge VH(C)$;

end Function

(3) 最后介绍如何生成 Z3 可接受的脚本文件,伪代码见算法 3 和算法 4.

除了状态公式的转换,为了让 Z3 能够成功执行,脚本中还需要一些额外的信息(set-option 命令)来配置 Z3 的行为.算法中,*Loc* 是自定义类型,声明为长度为 n 的比特向量类型,即(_BitVec n).在 SMT 语言中,变量必须声明后才能被使用,因此,首先声明所有出现在公式中的自由变量;接着,利用函数 write 将配置信息、变量声明以及转换后的公式写入文件 *sat-formula.smt*2.

算法 3. 将 QBVL 公式ϕ_b转换为 SMT 语言.

Function $QBVL2SMT(\phi_b)$

case

ϕ_b is true : return true;

ϕ_b is $e_1=e_2$: return (=$Texp(e_1)$ $Texp(e_2)$);

ϕ_b is $e_1 \neq e_2$: return (distinct $Texp(e_1)$ $Texp(e_2)$);

ϕ_b is $\neg\phi_{b_1}$: return (not $QBVL2SMT(\phi_{b_1})$) ;

ϕ_b is $\phi_{b_1} \vee \phi_{b_2}$: return (or $QBVL2SMT(\phi_{b_1})$ $QBVL2SMT(\phi_{b_2})$) ;

ϕ_b is $\phi_{b_1} \wedge \phi_{b_2}$: return (and $QBVL2SMT(\phi_{b_1})$ $QBVL2SMT(\phi_{b_2})$) ;

ϕ_b is $\exists x:\phi_{b_1}$: return (exists ((x *Loc*)) $QBVL2SMT(\phi_{b_1})$) ;

ϕ_b is $\forall x:\phi_{b_1}$: return (forall ((x *Loc*)) $QBVL2SMT(\phi_{b_1})$) ;

end case

end Function

算法 4. 将状态公式ϕ转换为 SMT 脚本文件,$fv(\phi)$表示公式ϕ中的所有自由变量.

Function $TSMT(\phi,C)$

ϕ_b:=$TQBVL\_VH(\phi,C)$;

$\{x_1,\ldots,x_m\}$:=$fv(\phi_b)$;

R:=QBVL2SMT(ϕ_b);

*sat-formula.smt*2:=write(*sat-formula.smt*2,((set-option: print-success true) (set-option:
print-models true) (set-option: print-proofs true) (set-option: mbqi true)
(define-sort *Loc*() (_BitVec n)) (declare-fun x_1 () *Loc*)…
(declare-fun x_m () *Loc*) (assert R) (check-sat) (get-value (x_1,…,x_m))));

return *sat-formula.smt*2;

end Function

4 实验与分析

当前版本的工具可以在 Windows 系统中运行,采用 C++语言开发.我们将例 2 讨论的 $PPTL^{SL}$ 公式以文本的形式输入给工具提供的编辑器,运行后生成的范式图如图 3 所示.

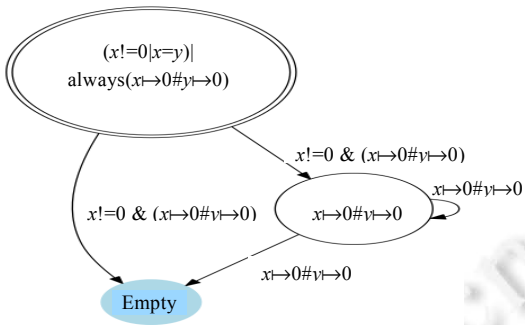

Fig.3 NFG generated by the tool

图 3 工具生成的范式图

正如所期望的,范式图中并未包含标记为 $f(x=y\wedge(x\mapsto 0\#y\mapsto 0),C)$的边.最后,工具检测到范式图中存在有效的有限路径和无限路径:

- 前者如路径$\langle(x\neq 0\vee x=y)\wedge\Box f(x\mapsto 0\#y\mapsto 0,C),x\neq 0\wedge(x\mapsto 0\#y\mapsto 0),\varepsilon\rangle$;
- 后者如路径$\langle(x\neq 0\vee x=y)\wedge\Box f(x\mapsto 0\#y\mapsto 0,C),x\neq 0\wedge(x\mapsto 0\#y\mapsto 0),x\mapsto 0\#y\mapsto 0,x\mapsto 0\#y\mapsto 0,\ldots\rangle$.

从而判定公式是可满足的.

下面给出一个指针程序的伪代码,其主要功能是创建长度为 n 的链表,然后再释放其占用的内存.

本小节将展示 PPTL^{SL} 能够描述该程序的一些时序相关的性质,接着,将会用 SAT-PPTL^{SL} 工具求解这些性质的可满足性.

```
struct Node {struct Node *next;};
Function cre_dis() {
  Node *x:=NULL, *t:=NULL;
  while (n−−) {t:=new(Node); t→next:=x; x:=t;}
  while (x!=NULL) {t:=x; x:=x→next; free(t);}
}
```

(1) $\Diamond(ls(x,0)\wedge least(n));\Diamond emp$:此性质刻画了两个顺序事件,程序首先将会到达一个状态,该状态的堆中存在一个链表,并且至少包含 n 个内存单元,然后释放链表,最终内存不存在任何已经分配的内存单元;

(2) $\Box(ls(x,0)\#\text{true}\vee x=0)$:在程序运行的过程中,指针 x 总会指向一个链表,或者为空指针;

(3) $\Diamond(\exists y:(ls(x,y)\#y\mapsto 0)\wedge y=t)$:指针 t 将会与链表的最后一个节点别名;

(4) $\Box(\forall y:alloc(y)\rightarrow(x\rightarrow^* y\vee t\rightarrow^* y))$:此性质利用指针可达性谓词描述了程序不存在内存泄漏的情况;

(5) $(\Diamond(ls(x,0))^*$:此性质描述了一个循环事件的发生,程序中两个循环的每次完整执行,都会使得所有的内存单元都会组成一个以 x 为头指针的链表.更精确地,假设每个赋值语句的执行占用一个单位区间的长度,则此性质可以写做$(\bigcirc^3 ls(x,0))^*$(每隔 3 个状态循环性质成立).

逻辑 maLTL[8]使用 LTL 描述程序的时序性质,同时利用 CTL 表达每个全局状态的指针结构.虽然文献[8]可以用 CTL 描述链表,但是没有分离逻辑简单和直观,而且无法表达分离逻辑中的分离性质,不能精确地刻画内存的状态.从另外一个角度,LTL 的表达能力要弱于 PPTL,上述程序的某些时序性质难以用 LTL 表示,如顺序和循环性质.类似地,LTL^{mem}[9]在描述时序性质方面与 PPTL^{SL} 相比也略显不足,而且其结合的分离逻辑片段不含有量词,缺少描述复杂指针结构的能力.NTL[6]与 TPL[7]采用相似的方法对操作单链表的指针程序进行抽象,并构建了相应的模型检测算法.由于二者属于一阶时序逻辑的范畴,表达能力要优于 PPTL^{SL},但是抽象技术使得某些情况下不能确定程序是否满足指定的性质.此外,文献[6,7]的抽象技术只能针对单链表程序,而 PPTL^{SL} 则可以描述更为复杂的指针结构(需扩展 $e_1\mapsto e_2$ 为 $e\mapsto\{e_1,\ldots,e_n\}$).

我们通过实验来证明工具的可用性以及分析堆的大小对工具性能的影响,实验环境为 Intel 双核 i3 CUP, 3.20GHz,8GB 内存.表 1 展示了实验结果,第 1 列为上面提到的用 PPTL^{SL} 公式表达的 6 个时序性质,它们描述了操作链表的指针程序的一些典型性质;第 2 列为公式转换时所需变量向量的大小,即对应于堆的大小;第 3 列记录了工具运行时间(包括 SMT 求解器调用时间和总计运行时间);最后一列给出了公式可满足性的检测结果.可以看出:随着堆的增长,工具的运行时间变得越长.这一现象与转换过程相一致,因为变量向量越大,转换的结果越复杂.公式转换的时间比较短,大部分时间花费在 SMT 求解器调用上.理论上,所有的公式都是可满足的,然而工具在某些情况下给出了否定的结果,这是由于堆的大小并不足以使得公式成立.例如,$\Diamond(ls(x,0)\wedge least(3));\Diamond emp$ 要求堆必须至少存在 3 个已分配的内存单元,因此在堆小于 3 的情况下是不可满足的.

Table 1 Checking satisfiability of PPTL^{SL} formulas

表 1 PPTL^{SL} 公式可满足性检查

公式	堆大小	运行时间(s)		可满足性
		SMT 调用	总计	
$\Diamond(ls(x,0)\wedge least(3));\Diamond emp$	1	0.156	0.234	否
	2	0.203	0.328	否
	3	35.211	35.443	是
$\Box(ls(x,0)\#\text{true}\vee x=0)$	1	0.093	0.141	是
	3	0.266	0.406	是
	7	1.466	1.856	是
$\Diamond(\exists y{:}(ls(x,y)\#y\mapsto 0)\wedge y=t)$	1	0.126	0.172	否
	3	0.265	0.390	是
	7	4.133	4.493	是
$\Box(\forall y{:}alloc(y)\rightarrow(x\rightarrow{*}y\vee t\rightarrow{*}y))$	1	0.077	0.156	是
	3	0.156	0.296	是
	7	0.344	0.702	是
$(\Diamond(ls(x,0))^*$	1	0.716	0.920	是
	2	1.403	1.809	是
	3	18.364	18.876	是
$(\bigcirc^3 ls(x,0))^*$	1	0.562	0.655	是
	2	0.873	1.092	是
	3	12.543	12.855	是

5 结束语

本文结合二维(空间和时间)逻辑 PPTL^{SL} 介绍了支持工具 SAT-PPTL^{SL}.相比现有的时序逻辑,PPTL^{SL} 的表达能力较强,描述方式简单直观,能够描述操作链表的指针程序的时序性质,并且有着 PPTL 理论的强力支持.SAT-PPTL^{SL} 利用 SMT 求解器实现了 PPTL^{SL} 公式可满足性的自动化检查.此外,本文还选取一些典型逻辑公式实例进行了实验,分析了关键参数对 SAT-PPTL^{SL} 执行效率的影响.在今后的工作中,我们将会研究基于 PPTL^{SL} 的统一模型检测方法[17],并进一步扩展 PPTL^{SL},使其能够描述比单链表更为复杂的指针结构,从而验证现实世界的指针程序,如 Windows 驱动程序[18],以及并发生产者-消费者算法[6]、非阻塞栈(non-blocking stack)算法[19]和锁耦合链表(lock-coupling list)算法[20]等.

References:

[1] Reynolds JC. Separation logic: A logic for shared mutable data structures. In: Proc. of the 17th Annual IEEE Symp. on Logic in Computer Science. IEEE Computer Society, 2002. 55−74. [doi: 10.1109/LICS.2002.1029817]

[2] Calcagno C, Gardner P, Hague M. From separation logic to first-order logic. In: Sassone V, ed. Proc. of the Foundations of Software Science and Computational Structures. LNCS 3441, Berlin, Heidelberg: Springer-Verlge, 2005. 395−409. [doi: 10.1007/978-3-540-31982-5_25]

[3] Manna Z, Pnueli A. The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer Science & Business Media, 2012. [doi: 10.1007/978-1-4612-0931-7]

[4] Ben-Ari M, Pnueli A, Manna Z. The temporal logic of branching time. Acta Informatica, 1983,20(3):207−226. [doi: 10.1007/BF01257083]

[5] Yahav E, Reps T, Sagiv M, Wilhelm R. Verifying temporal heap properties specified via evolution logic. Logic Journal of IGPL, 2006,14(5):755−783. [doi: 10.1093/jigpal/jzl009]

[6] Distefano D, Katoen JP, Rensink A. Safety and liveness in concurrent pointer programs. In: de Boer FS, *et al*., eds. Proc. of the Formal Methods for Components and Objects. LNCS 4111, Berlin, Heidelberg: Springer-Verlag, 2006. 280−312. [doi: 10.1007/11804192_14]

[7] Rieger S. Verification of pointer programs [Ph.D. Thesis]. Aachen: RWTH Aachen University, 2009.

[8] del Mar Gallardo M, Merino P, Sanán D. Model checking dynamic memory allocation in operating systems. Journal of Automated Reasoning, 2009,42(2-4):229−264. [doi: 10.1007/s10817-009-9124-y]

[9] Brochenin R, Demri S, Lozes E. Reasoning about sequences of memory states. Annals of Pure and Applied Logic, 2009,161(3):305−323. [doi: 10.1016/j.apal.2009.07.004]

[10] Duan ZH. An extended interval temporal logic and a framing technique for temporal logic programming [Ph.D. Thesis]. Newcastle upon Tyne: University of Newcastle upon Tyne, 1996.

[11] Tian C, Duan ZH. Propositional projection temporal logic, Büchi automata and ω-regular expressions. In: Agrawal M, *et al*., eds. Proc. of the Theory and Applications of Models of Computation. LNCS 4978, Berlin, Heidelberg: Springer-Verlag, 2008. 47−58. [doi: 10.1007/978-3-540-79228-4_4]

[12] Brochenin R, Demri S, Lozes E. On the almighty wand. Information and Computation, 2012,211:106−137. [doi: 10.1016/j.ic.2011.12.003]

[13] Tian C, Duan ZH. Complexity of propositional projection temporal logic with star. Mathematical Structures in Computer Science, 2009,19(1):73−100. [doi: 10.1017/S096012950800738X]

[14] Duan ZH, Tian C, Zhang L. A decision procedure for propositional projection temporal logic with infinite models. Acta Informatica, 2008,45(1):43−78. [doi: 10.1007/s00236-007-0062-z]

[15] Wintersteiger CM, Hamadi Y, De Moura L. Efficiently solving quantified bit-vector formulas. Formal Methods in System Design, 2013,42(1):3−23. [doi: 10.1007/s10703-012-0156-2]

[16] De Moura L, Bjørner N. Z3: An efficient SMT solver. In: Ramakrishnan CR, Rehof J, eds. Proc. of the Tools and Algorithms for the Construction and Analysis of Systems. LNCS 4963, Berlin, Heidelberg: Springer-Verlag, 2008. 337−340. [doi: 10.1007/978-3-540-78800-3_24]

[17] Duan ZH, Tian C. A unified model checking approach with projection temporal logic. In: Liu S, Maibaum TSE, Araki K, eds. Proc. of the Formal Methods and Software Engineering, Vol.5256. Berlin, Heidelberg: Springer-Verlag, 2008. 167−186. [doi: 10.1007/978-3-540-88194-0_12]

[18] Berdine J, Cook B, Ishtiaq S. SLAyer: Memory safety for systems-level code. In: Gopalakrishnan G, Qadeer S, eds. Proc. of the Computer Aided Verification. LNCS 6806, Berlin, Heidelberg: Springer-Verlag, 2011. 178−183. [doi: 10.1007/978-3-642-22110-1_15]

[19] Parkinson M, Bornat R, O'Hearn P. Modular verification of a non-blocking stack. In: Proc. of the ACM SIGPLAN Notices, Vol.42. ACM Press, 2007. 297−302. [doi: 10.1145/1190216.1190261]

[20] Vafeiadis V, Parkinson M. A marriage of rely/guarantee and separation logic. In: Proc. of the CONCUR 2007—Concurrency Theory. Springer-Verlag, 2007. 256−271. [doi: 10.1007/978-3-540-74407-8_18]

陆旭(1985－),男,河北承德人,博士,CCF 学生会员,主要研究领域为分离逻辑,模型检测.

田聪(1981－),女,博士,教授,CCF 会员,主要研究领域为形式化方法,时序逻辑,模型检测.

段振华(1948－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络计算,高可信软件理论和技术.