计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20150370
53(9):2067-2084,2016

具有程序的静态结构和动态行为语义的时序逻辑

陈冬火[1] 刘 全[1,2] 金海东[1] 朱 斐[1,2] 王 辉[1]
[1](苏州大学计算机科学与技术学院 江苏苏州 215006)
[2](符号计算与知识工程教育部重点实验室(吉林大学) 长春 130012)
(dhchen@suda.edu.cn)

A Temporal Logic with a Semantics Defined on the Static Structure and Dynamic Behavior of Program

Chen Donghuo[1], Liu Quan[1,2], Jin Haidong[1], Zhu Fei[1,2], and Wang Hui[1]
[1](*School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu* 215006)
[2](*Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun* 130012)

Abstract The paper presents an interval temporal logic—CFITL(control flow interval temporal logic) which is used to specify the temporal properties of abstract model of program, for example control flow graph, generally abbreviated to CFG. The targeted logic differs from the general sense of temporal logics, typically CTL and LTL, whose semantical models are defined in term of the state-based structures. The semantics of CFITL is defined on an ordered CFG structure, which encodes the static structure and dynamic behavior of program. The ordered CFGs are a subset of CFGs, and their topology can be summarized by partially ordered sets, such that the induced natural number intervals can be mapped onto the well-formed structures of program. In other words, the CFITL formulae have the ability of specifying the properties related to not only the dynamic behavior but also static structure of programs. After the syntax and semantics of CFITL are expounded, the problem of model checking over CFITL is detailedly discussed. Furthermore, two types of algorithms are designed: one is based on the computation of reachable state space as well as the another is based on bounded model checking employing the SMT(satisfiability modulo theories) solvers power. Because programs implemented by advanced programming languages inevitably involve complex abstract data types with unbounded domains and operators, and the semantics of CFITL is more complex than the one of CTL, the method of SMT based model checking is more practical than the method of direct search of state space. In the sequel, a prototype tool is implemented, and some case studies are conducted.

收稿日期:2015-05-14;**修回日期**:2016-02-16
基金项目:国家自然科学基金项目(61272005,61303108,61373094,61472262,61502323,61502329);江苏省自然科学基金项目(BK2012616);福建省自然科学基金项目(2014J01221);江苏省高校自然科学研究项目(13KJB520020);吉林大学符号计算与知识工程教育部重点实验室项目(93K172014K04);苏州市应用基础研究计划项目(SYG201422)
This work was supported by the National Natural Science Foundation of China (61272005, 61303108, 61373094, 61472262, 61502323, 61502329), the Natural Science Foundation of Jiangsu (BK2012616), the Natural Science Foundation of Fujian Province (2014J01221), the High School Natural Foundation of Jiangsu (13KJB520020), the Project Funded by the Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University (93K172014K04), and the Suzhou Industrial Application of Basic Research Program (SYG201422).
通信作者:刘全(quanliu@suda.edu.cn)

Key words interval temporal logic; control flow graph (CFG); static structure of program; model checking; satisfiability modulo theories (SMT)

摘 要 提出一种区间分支时序逻辑——控制流区间时序逻辑(control flow interval temporal logic, CFITL),用于规约程序的时序属性.不同于计算树逻辑(computation tree logic, CTL)和线性时序逻辑(linear temporal logic, LTL)等传统的时序逻辑,CFITL 公式的语义模型不是基于状态的类 Kripke 结构,而是以程序的抽象模型控制流图(control flow graph, CFG)为基础所构建的含序 CFG 结构.含序 CFG 是 CFG 的一种受限子集,它们的拓扑结构可映射为偏序集,这样诱导产生的自然数区间可自然地用于描述定义良好的程序结构.这种结构含有程序的静态结构信息和动态行为信息,换而言之,CFITL 具有规约程序实现结构属性和程序执行动态行为属性的能力.在定义 CFITL 的语法和语义的基础上,详细讨论了 CFITL 的模型检验问题,包括基于值状态空间可达性计算的模型检验方法和基于 SMT (satisfiability modulo theories)的 CFITL 有界模型检验方法.现代程序都含有复杂且具有无限值域的抽象数据类型及各种复杂的操作,CFITL 语义定义相比 CTL 等时序逻辑更复杂,因此,基于显示状态搜索的方法难以有效进行,而基于 SMT 的 CFITL 有界模型检验方法更易实现、更具有可行性.最近开发相关的原型工具,并进行相关的实例研究.

关键词 区间时序逻辑; 控制流程图; 程序静态结构;模型检验;可满足性模理论

中图法分类号 TP311

时序逻辑是一种模态逻辑,广泛用于描述和推理各种计算机系统状态随时间演化所应满足的性质,例如计算树逻辑(computation tree logic, CTL)[1]、线性时序逻辑(linear temporal logic, LTL)[2]、概率计算树逻辑(probabilistic computation tree logic, PCTL)[3]和连续时间随机逻辑(continuous time stochastic logic, CSL)[4]等.各种时序逻辑具有足够的表达能力,能用于描述各类计算机系统包括安全性属性及活性属性的各类性质.自从 20 世纪 80 年代模型检验[5]思想提出以来,基于时序逻辑的模型检验方法成为形式化验证领域的重要研究方向[6-11].有别于传统的测试方法,模型检验方法用算法自动检测系统的行为(数学结构)是否满足某条属性(时序逻辑公式),并且当系统不满足相应的属性时,给出反例作为证据帮助设计者调试系统.对于系统某个属性而言,测试方法通过设计测试用例、运行系统、观察输出等环节来验证系统是否满足这个属性,但这种方法是不完备的;而模型检验方法则通过算法证明系统是否满足这个属性,针对属性而言该方法是完备和可靠的.经过许多学者的努力,该方向的研究工作取得了重要的进展.除了计算机领域的研究人员,各种系统设计者和开发者乃至整个工业界都日益重视在软硬件开发项目中应用时序逻辑模型检验方法,以提高所开发系统功能和性能方面的质量[12-16].

传统的时序逻辑,例如 CTL 和 LTL,它们的语义模型是基于状态,例如 Kripke 结构[1],因此它们适合描述状态基结构的性质.在进行相应的模型检验时,不得不建立各种抽象层次的状态基模型.时序逻辑的这种语义定义方法,没有考虑系统设计和实现的结构因素.此处强调的结构因素指设计和实现的控制结构,不包括普通意义上的数据结构,例如图 1 所示程序中的循环结构和选择结构.但是对于开发者而言,在许多情况下,基于系统设计和实现的结构特征描述所开发系统的属性是一种更为自然直观的方式.各种集成开发环境所提供的程序点断言机制就是一种简单的基于程序结构的属性描述方

```
int find(int a[n]) {
①  int[] b=new int[n];
②  int i=0;
③  int result=0,spot=0;
④  spot=a.length;
⑤  while (i<a.length) {
⑥    if (spot=a.length && a[i]≠0)
⑦      spot=i;
⑧    b[i]=(a[i]≠0);
⑨    i=i+1; }
⑩  assertion("spot≠a.length→a[spot]≠0");
⑪  if (spot=a.length)
⑫    result=0;
⑬  else {
⑭    i=2;
⑮    while (spot<a.length) {
⑯      result=result|i;
⑰      i=i×2;
⑱      spot=spot+1;}
⑲    delete[] b;}
⑳  return result;}
```

Fig. 1 The source code of the function *find*.

图 1 函数 *find* 源代码

法,更进一步,一些语言提供了比程序点断言更为复杂的基于结构的属性描述方法[17-19],例如 Java 模型语言(Java modeling language, JML)[20]. JML 在 Java 语言的基础上设计了一些特殊语言机制,可以描述循环不变式及成员方法输入和输出应该满足的条件等,但是这些方法的描述能力具有很大的局限性.对于图 1 所示程序,必须满足下面 4 个属性:1)对于任意一条执行路径,数组 b 所分配的内存资源一定在将来某个时刻会被回收;2)假如返回值为 0,则数组 a 所有元素的值一定为 0;3)在程序点 5 和程序点 9 之间属性 $i \leqslant a.length \wedge spot \leqslant a.length$ 为真;4)在程序点 5 和程序点 9 之间,若 $spot \neq a.length$,则对于 $0 \leqslant i \leqslant spot, a[i]=0$. 这些属性具有显著的特点:它们都与程序静态结构相关,并且包含动态行为时序性.

上面的实例说明构造基于系统设计和实现的时序逻辑系统及设计相关的模型检验算法具有很大的必要性和价值.鉴于 CTL 具有足够的表达能力和可判定的逻辑属性,本文提出一种区间分支时序逻辑——控制流区间时序逻辑(control flow interval temporal logic, CFITL),用于描述抽象程序模型——控制流图(control flow graph, CFG)的时序性质.该逻辑的语义解释集成了程序基于状态的执行语义及静态结构语义,可以描述基于结构和动态行为的各种时序属性,例如程序点断言及循环结构不变式属性等.本文在给出 CFITL 详细语法的基础上,基于 CFG 的一种受限子集,定义状态集的偏序集及相关的位阶函数 *rank*,用以解释 CFITL 公式所涉及的结构因素,进而基于符号状态路径对 CFITL 完整的语义进行定义.基于可达状态空间计算的方法讨论了 CFITL 的模型检验问题.考虑 CFITL 模型检验是路径敏感的,比 CTL 模型检验具有更高的复杂性以及问题域的源程序在数据与操作方面的复杂性,本文基于符号执行技术[21-23]及时序逻辑公式重写技术[24]定义了 CFITL 公式的重写规则,提出基于可满足性模理论(satisfiability modulo theories, SMT)[25]的有界模型检验方法.详细设计了 CFITL 模型检验器的体系架构,实现了原型系统 VerTPCFG(verifica-tion of temporal property of CFGs),实施相关的实例研究,对所提算法的有效性和效率进行初步评估.

1 控制流区间时序逻辑 CFITL

控制流区间时序逻辑 CFITL 是计算树逻辑 CTL 的一种扩展,通过在语法上引进区间的概念来描述目标的结构因素.基于 CFITL 所描述的属性把系统基于状态的功能性属性与系统设计或实现的结构性因素相关联.本节将详细介绍该逻辑系统的语法和语义模型.

1.1 语 法

给定严格的偏序集 $\mathbb{D}=\langle D, \prec \rangle$,其中 $\prec \subseteq D \times D$ 是严格偏序关系,令 $[d_1, d_2]$ 是 $\mathbb{D}$ 上的区间,其中 $d_1, d_2 \in D, d_1 \preceq d_2$. $d \in D$,若 $d_1 \preceq d \preceq d_2$,称 d 属于区间 $[d_1, d_2]$,简记为 $d \in [d_1, d_2]$. 当 $D=\mathbb{N}$($\mathbb{N}$ 为自然数集合),$\prec = <$ 时,可以得到自然数区间,下文自然数区间一般用符号 $I, I_1, I_2, \cdots$ 表示.

给定原子命题集合 AP 及自然数集合区间 I,下面给出 CFITL 的抽象语法定义:

$$\begin{aligned}\psi \triangleq\ & \text{true} \mid p \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \mathbb{EX}\psi \mid \mathbb{EF}\psi \mid \\ & \mathbb{EG}\psi \mid \mathbb{E}(\psi_1 \mathbb{U} \psi_2) \mid \mathbb{EF}^I\psi \mid \mathbb{EG}^I\psi \mid \mathbb{E}(\psi_1 \mathbb{U}^I \psi_2) \mid \\ & \mathbb{AX}\psi \mid \mathbb{AF}\psi \mid \mathbb{AG}\psi \mid \mathbb{A}(\psi_1 \mathbb{U} \psi_2) \mid \mathbb{AF}^I\psi \mid \mathbb{AG}^I\psi \mid \\ & \mathbb{A}(\psi_1 \mathbb{U}^I \psi_2).\end{aligned}$$

上面抽象语法定义中,$p \in AP$;$\mathbb{E}$ 和 $\mathbb{A}$ 称为路径量词,需要注意的是此处路径量词的语义与 CTL 中量词的语义有较大的区别,这一点将在逻辑公式的语义定义中得到说明. $\mathbb{X}$,$\mathbb{F}$,$\mathbb{G}$,$\mathbb{U}$ 是时序算子,分别表示“下一”、“将来”、“总是”、“直到”的时间概念.式中 true 表示逻辑常量“真”,各种逻辑运算符 $\wedge$,$\neg$,$\vee$ 符合标准的逻辑解释,包括没有明确定义的逻辑运算符 $\rightarrow$ 和逻辑真值 false.

1.2 语 义

1.2.1 控制流图和位阶函数

为了使得所定义的逻辑框架不与具体的程序语言相关,本节定义程序的抽象模型——控制流图,该结构将作为 CFITL 语义解释的基础. CFG 只关注程序的控制流,忽略程序的数据流方面的特性.

定义 1. CFG 是多元组结构 $\langle \Sigma, Start, Act, \rightarrow, Final \rangle$,其中 Σ, $Start$, Act, $\rightarrow$, $Final$ 分别解释如下:

1) Σ 是有限的状态集合,非空集合 $Start \subseteq \Sigma$ 为初始状态集合,非空集合 $Final \subseteq \Sigma$ 为终止状态集合;

2) Act 称为动作集合;

3) $\rightarrow \subseteq \Sigma \times Act \times \Sigma$ 为变迁关系,对于 $\langle l, a, l' \rangle \in \rightarrow$ 可简记为 $l \xrightarrow{a} l'$.

令集合 $Vars$ 为所有程序变量的集合,$v \in Vars$,$\mathcal{D}_v$ 表示变量 v 的取值范围,$V=\{v_1, v_2, \cdots, v_n\} \subseteq$

$Vars$, $\mathcal{D}_V=\mathcal{D}_{v_1}\times\mathcal{D}_{v_2}\times\cdots\times\mathcal{D}_{v_n}$. 此处的状态集合 Σ 类似于程序计数器,不同于系统的状态空间 $\mathcal{D}_{Vars}$,是抽象的位置标示符集合,用来记录程序的静态结构信息. 元素 $l\in\Sigma$ 被视为字,也即字符串类型. 如果程序模块具有单入口和单出口的特性,集合 $Start$ 和 $Final$ 可用 $init\in\Sigma$ 和 $final\in\Sigma$ 代替,分别表示开始状态和结束状态. Act 包含 2 种类型的动作: $v=e$ 和 $assume(c)$,其中 e 表示各种满足程序语言语义约束的运算表达式,c 为不含量词的一阶逻辑公式. 需要注意的是,此处的 e 和 c 中不含各种可能引进副作用的程序表达式,例如许多程序语言允许 $v=++x$ 这样的操作,如果程序出现这种操作,在构建程序相关的 CFG 时,应把该操作执行分解为操作 $x=x+1$ 和 $v=x$ 的顺序执行,并引进相应的状态标签. 符号序列 $l_0a_1l_1a_2\cdots l_{n-1}a_nl_n$ 称为 CFG 的执行路径,其中 $\forall 0\leqslant i\leqslant n\cdot l_i\in\Sigma$, $i_0\in Start$, $\forall 1\leqslant j\leqslant n\cdot a_j\in Act$, $\forall 0\leqslant i\leqslant n-1\cdot\langle l_i,a_{i+1},l_{i+1}\rangle\in\rightarrow$,此处所谓的执行路径没有要求 $l_n\in Final$,n 为路径的长度. 在不需要强调动作的情况下,路径可以直接用状态序列表示,例如 $l_0l_1\cdots l_n$.

基于 Σ,下面给出前缀闭合字串集的概念. 令 Σ^+ 表示 Σ 中元素构成的所有字串的集合,不包含空串.

定义 2. 令 $ws\subseteq\Sigma^+$,如果对于任意 $str\in ws$,str 的任意非空串的前缀 pre 满足 $pre\in ws$,则 ws 被称为前缀闭合字串集.

令 $Path_{\mathcal{G}}\subseteq\Sigma^+$ 为 CFG $\mathcal{G}=\langle\Sigma,Start,Act,\rightarrow,Final\rangle$ 的所有路径的集合,根据定义 2 可知 $Path_{\mathcal{G}}$ 为前缀闭合字串集合. 对于某程序的控制流模型 $\mathcal{G}$,状态集合 Σ 是有限集合,但由于循环结构的存在,$Path_{\mathcal{G}}$ 可以是无限的. 路径集合 $Path_{\mathcal{G}}$ 是根据 $\mathcal{G}$(程序)的静态结构所得到的,没有考虑程序的语义,因此包含程序任意可能的执行路径. 程序中的任意循环结构都假定循环体能执行任意次,即使循环次数为定值的循环结构,换句话说,在计算 $Path_{\mathcal{G}}$ 时,对循环条件不做任何解释,条件 true,$n\leqslant 10$ 和 $x^2+y^3\neq a^4+b^5$ 在任何环境里,具有一样的效果,仅是一个未解释的循环条件. $p,p'\in Path_{\mathcal{G}}$,当 p 为 p' 的前缀时可记为 $p\sqsubseteq p'$,显然结构 $\langle Path_{\mathcal{G}},\sqsubseteq\rangle$ 为偏序集.

为了讨论问题需要,下面给出函数 $last$ 和函数 $MinPath_{\mathcal{G}}$ 的说明. 函数 $last:Path_{\mathcal{G}}\rightarrow\Sigma$ 用于计算给定路径的终止状态,即 $last(l_0l_1\cdots l_n)=l_n$,$last$ 是部分函数,如果 $p\in Path_{\mathcal{G}}$ 是无限路径,$last(p)$ 没有定义. $MinPath_{\mathcal{G}}:\Sigma\rightarrow\mathbb{P}(Path_{\mathcal{G}})$ 为给定状态的最短路径计算函数,$\mathbb{P}(Path_{\mathcal{G}})$ 表示 $Path_{\mathcal{G}}$ 的幂集,当状态 $l\in\Sigma$ 和路径 $S\subseteq Path_{\mathcal{G}}$ 满足以下条件 1 和条件 2 时,$MinPath_{\mathcal{G}}(l)=S$:1)对于 $s\in S$,$last(s)=l$;2)对于 $s\in S$,当 $s'\sqsubset s$,$last(s')\neq l$. 路径之间的偏序关系 $\sqsubset$ 可扩展至路径集合的情形,给定 $l,l'\in\Sigma$,$MinPath_{\mathcal{G}}(l)\sqsubset MinPath_{\mathcal{G}}(l')$ 等价于 $\forall p\in MinPath_{\mathcal{G}}(l)\cdot\exists p'\in MinPath_{\mathcal{G}}(l')\cdot p\sqsubset p'$.

定义 3. $\langle\Sigma,\lhd\rangle$. 令 $\mathcal{G}=\langle\Sigma,Start,Act,\rightarrow,Final\rangle$,$\lhd\subseteq\Sigma\times\Sigma$:

1) 对于任意的 $l_1,l_2\in\Sigma$,如果 $MinPath_{\mathcal{G}}(l_1)\sqsubset MinPath_{\mathcal{G}}(l_2)$,那么 $l_1\lhd l_2$;

2) 假如存在路径集合 $\mathcal{P}=l_0l_1\cdots l_i(l_{i+1}\mid i_{i+2}\mid l_{i+2}\mid i_{i+3}\cdots\mid l_{i+j})^*l_{i+j+1}\subseteq Path_{\mathcal{G}}$,并且 $l_{i+j+1}\neq l_{i+1}$, $l_{i+j+1}\neq l_{i+2}$, …, $l_{i+j+1}\neq l_{i+j}$,则 $l_{i+1}\lhd l_{i+j+1}$, $l_{i+2}\lhd l_{i+j+1}$, …, $l_{i+j}\lhd l_{i+j+1}$.

在定义 3 中,$\mathcal{P}=l_0l_1\cdots l_i(l_{i+1}\mid i_{i+2}\mid l_{i+2}\mid i_{i+3}\mid\cdots\mid l_{i+j})^*l_{i+j+1}$ 使用了正规式语言的描述方法,式中 $\mid$ 和 * 运算代表“或”运算和“闭包”运算. 根据正规式语言的解释,正规式实质上描述的是语言,即字串(路径)的集合. 闭包运算用于描述程序中循环结构的控制流模型,代表循环体执行任意次,包括 0 次. 图 2(a)给出了图 1 所示程序的控制流模型,根据定义 3 可知 $\langle\{l_0,l_1,\cdots,l_{18}\},\lhd\rangle$ 是严格偏序集. 实质上 $\langle\Sigma,\lhd\rangle$ 反映了相关 CFG 的拓扑结构所关联程序的控制结构信息及程序执行语义信息,但并不是根据任意的 CFG 构造出的结构 $\langle\Sigma,\lhd\rangle$ 都是完备格. 图 2(b)所示的 CFG 由于存在 $goto$ 动作($goto$ 相当于 Act 中的 $assume$(true)),并进行了特殊的无条件跳转,导致该 CFG 相关的 $\langle\Sigma,\lhd\rangle$ 为非偏序集结构. 从图 2(b)可以看出,该 CFG 存在路径集合 $\mathcal{P}_1=l_0l_1(l_2l_4l_5)^*l_8l_3l_6l_7$ 和 $\mathcal{P}_2=l_0l_1l_3l_6l_9l_5l_4(l_2l_4l_5)^*l_8l_3$,根据 $\lhd$ 的定义,从 $\mathcal{P}_1$ 和 $\mathcal{P}_2$ 分别可以得出以下的关系式:$l_6\rhd l_2$, $l_6\rhd l_4$, $l_6\rhd l_5$ 和 $l_6\lhd l_2$, $l_6\lhd l_4$, $l_6\lhd l_5$,显然 $\langle\Sigma,\lhd\rangle$ 不是偏序集. 当然,并不是所有含有 $goto$ 动作的 CFG 相关的 $\langle\Sigma,\lhd\rangle$ 都不是偏序集. 实质上图 2(a)中的 $assume$(true)也是一种无条件转移操作,相当于 $goto$ 的作用,但由于 $assume$(true)转移的目标状态受到限制,例如仅仅限于循环的开始状态,不影响关系 $\lhd$ 的结构. 另外,结合动作 $assume(c)$,其中 c 不为 false 和 true,$assume$(true)能实现大多数高级程序设计语言循环结构经常使用的“break”和“continue”语句的功能,根据关系的定义,CFG 中这种类型的 $assume$(true)动作不影响 $\lhd$ 的结构.

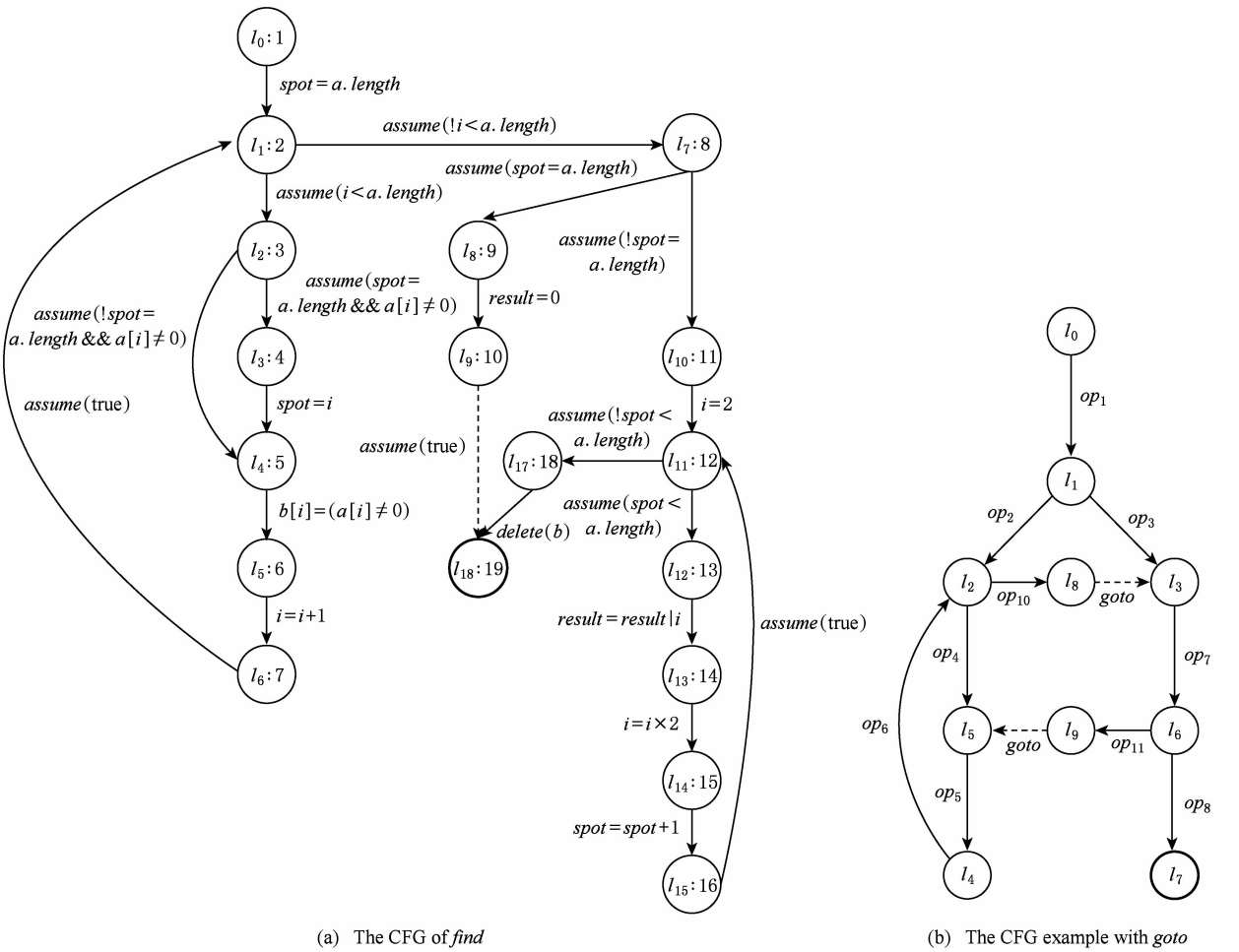

Fig. 2 The CFG examples.

图 2 CFG 实例

从 Böhm-Jacopini 定理[26-27]可知,CFG 能表达由顺序结构、选择结构和循环结构构成的可计算函数,并且含有"goto"语句的可计算函数可通过算法等价转换为不含"goto"、由 3 种结构构成的函数.根据程序语言理论,对于描述程序的 CFG,当 $assume(\text{true})\in Act$ 的作用仅限于循环结构的无条件转换、"break"和"continue"时,如果 $l,l'\in\Sigma$ 满足 $l\lhd\cdots\lhd l'$,则 $l'\not\lhd\cdots\not\lhd l$.本文把描述不含有"goto"语句的程序的 CFG 称为受限 CFG,后文没有特别说明,所涉及的 CFG 都局限于这种类型.

定理 1. $\mathcal{G}=\langle\Sigma,init,Act,\rightarrow,final\rangle$ 为不含"goto"语句的可计算函数构造的 CFG,$\langle\Sigma,\lhd\rangle$ 为严格偏序结构.

证明. 该定理的证明可基于程序的静态结构进行归纳证明,下面对证明过程做出详细的说明.可计算函数由顺序结构、选择结构和循环结构组成,这些结构从单纯的基础结构,通过嵌套构成任意复杂的结构.这 3 种结构分别用 $seq,branch,loop$ 表示,不同的同种结构可用下标进行区分,$seq(str_1,str_2,\cdots,str_n)$ 表示由结构 $str_1,str_2,\cdots,str_n$ 顺序组成的顺序结构,$branch(c,str_1,str_2)$ 是选择条件为 c 且真分支和假分支结构分别为 str_1 和 str_2 的选择结构,$branch(c,str)$ 表示只有真分支的选择结构,$loop(c,str)$ 是循环条件为 c、循环体为 str 的循环结构,显然,可计算函数也可表示为多层嵌套的 3 种结构之一.因此只要说明每种结构所涉及的状态集与关系 $\lhd$ 构成偏序集即可.令 $\Sigma[str]$ 表示结构 str 所涉及的状态集,根据程序语言理论,当 str_1 和 str_2 没有嵌套关系时,$\Sigma[str_1]\bigcap\Sigma[str_2]=\varnothing$.

1) 基础步.当 str,str_1,str_2 为基本顺序结构时,$branch(c,str_1,str_2)$ 和 $loop(c,str)$ 分别为基本选择结构和基本循环结构.显然 $\langle\Sigma[str],\lhd\rangle$ 为线性偏序集,$\langle\{l_c\}\cup\Sigma[str_1],\lhd\rangle$,$\langle\{l_c\}\cup\Sigma[str_2],\lhd\rangle$,$\langle\{l_c\}\cup\Sigma[str],\lhd\rangle$ 分别为线偏序集,因此 $\langle\{l_c\}\cup\Sigma[str_1]\cup\Sigma[str_2],\lhd\rangle$ 为偏序集,即 $\langle\Sigma[branch(c,str_1,str_2)],\lhd\rangle$ 为偏序集.并且这些偏序集都存在最小元素,结构

str 如果存在最小元素,则记为 $\min(str)$. l_c 为选择或循环结构的入口状态.

2) 假设步和归纳步. 假设 $\langle \Sigma[str_1], \lhd \rangle$, $\langle \Sigma[str_2], \lhd \rangle$, …, $\langle \Sigma[str_n], \lhd \rangle$ 都为偏序集,且 $str_1, str_2, \cdots, str_n$ 都存在最小元素. 下面说明 $\langle \Sigma[seq(str_1, str_2, \cdots, str_n)], \lhd \rangle$ 为偏序集. 根据顺序结构的构造原理,对于 $1 \leqslant i \leqslant n-1$,任意 $l \in \Sigma[str_i]$, $l \lhd \min(str_{i+1})$,所有 $\langle \Sigma[str_i] \cup \Sigma[str_{i+1}], \lhd \rangle$ 为偏序集,且有最小元 $\min(str_i)$. 因此 $\langle \Sigma[str_1] \cup \Sigma[str_2] \cup \cdots \cup \Sigma[str_n], \lhd \rangle$ 为偏序集,最小元为 $\min(str_1)$,即 $\langle \Sigma[seq(str_1, str_2, \cdots, str_n)], \lhd \rangle$ 为偏序集,且 $\min(seq(str_1, str_2, \cdots, str_n)) = \min(str_1)$. 采用类似的方法,容易证明 $\langle \Sigma[branch(c, str_1, str_2)], \lhd \rangle$ 和 $\langle \Sigma[loop(c, str_1)], \lhd \rangle$ 分别为偏序集并且存在最小元 l_c. 证毕.

定义 4. 令 $\mathcal{G} = \langle \Sigma, init, Act, \rightarrow, final \rangle$, $\langle \Sigma, \lhd \rangle$ 是偏序集,函数 $rank_{\mathcal{G}}: \Sigma \rightarrow \mathbb{N}$ 满足以下性质时,称为 CFG $\mathcal{G}$ 的位阶函数:

1) 对于任意的 $l_1, l_2 \in \Sigma$,如果 $l_1 \lhd l_2$, $rank_{\mathcal{G}}(l_1) < rank_{\mathcal{G}}(l_2)$;

2) $rank_{\mathcal{G}}$ 是一一映射函数,即对于任意 $l \neq l'$, $rank_{\mathcal{G}}(l) \neq rank_{\mathcal{G}}(l')$.

图 2(a)所示的 CFG 相关的位阶函数 $rank_{\mathcal{G}}(l_i) = i+1$,其中 $0 \leqslant i \leqslant 18$,状态值及相应位阶函数值标记在相应的状态里. 位阶函数作为一种特殊的函数,把程序的控制结构信息及程序执行语义信息与自然数集进行关联,用于解释 CFITL 公式的区间信息. 位阶函数是由 $\langle \Sigma, \lhd \rangle$ 诱导的,且映射结果在 $<$ 上保序,如果 CFG 相关的 $\langle \Sigma, \lhd \rangle$ 不为偏序集,这意味着该 CFG 不能定义相应的位阶函数;但如果位阶函数存在的话,则位阶函数不是唯一的. 例如:任意 $n \geqslant 1$, $rank_{\mathcal{G}}(l_i) = n \times (i+1)$ 都是图 2(a)所示 CFG 的位阶函数. 图 2(b)所示的 CFG 由于存在 *goto* 动作,进行了特殊的无条件跳转,导致该 CFG 的拓扑结构没有相应的位阶函数.

1.2.2 CFITL 的解释模型

CFITL 公式从语法层次描述了程序的静态结构及动态行为的属性,因此对 CFITL 公式的解释不仅需要 CFG 的状态 Σ 的变迁关系,还需要对动作集合 Act 进行解释,建立程序变量的符号状态和值状态的变迁关系. 符号执行技术[21]是一种在程序静态分析领域常用的分析方法,下面将基于符号状态关联程序的静态结构和动态行为,定义 CFITL 的语义. 符号状态 s 为三元组 $\langle l, A, \Pi \rangle$,其中 $l \in \Sigma$ 为结构分量; $A: Vars \rightarrow Terms$ 为赋值操作, $Terms$ 为由各种类型符号变量、$\bigcup_{v \in Vars} D_v$ 中的常量及受限的程序语言运算符所构成的表达式,该表达式符合语法规范及语义约束要求的运算表达式集合; Π 称为路径条件,一般情况下为不含量词的一阶逻辑公式. 下面基于符号状态对 CFG $\mathcal{G} = \langle \Sigma, init, Act, \rightarrow, final \rangle$ 的变迁关系 $\rightarrow$ 作出进一步解释,令 $s = \langle l, A, \Pi \rangle$, $\langle l, op, l' \rangle \in \rightarrow$,则:

$$s' = s[op] = \begin{cases} \langle l', A, \Pi \wedge [c]_s \rangle, & op = \text{“}assume(c)\text{”}, \\ \langle l', A' = A[v \leftarrow e], \Pi \rangle, & op = \text{“}v = e\text{”}, \end{cases}$$

其中, $s[op]$ 表示符号状态 s 执行动作 $op \in Act$ 的后继状态, $[c]_s$ 表示基于符号状态 s 评估条件 c 的真假值, $A[v \leftarrow e]$ 表示 $A': Vars \rightarrow Terms$,对于 $v' \in Vars$:当 $v' \neq v$, $A'(v') = A(v')$;当 $v' = v$, $A'(v') = [e]_s$, $[e]_s$ 与 $[c]_s$ 类似,表示在符号状态 s 评估表达式 e 的值. 对于符号状态 s,如果 Π 是不可满足的,则表示 s 是不可达的,即程序的静态结构中存在死代码.

根据符号状态的变迁关系,执行路径 $p = l_0 l_1 \cdots \in Path_{\mathcal{G}}$ 可用符号状态路径 $s_0 s_1 \cdots$ 表示. 其中 $s_0 = \langle l_0, A_0, \Pi_0 \rangle$ 为初始符号状态, A_0 用符号变量或常量初值对所有的程序变量进行初始化; Π_0 则记录程序变量之间的约束关系,假如程序变量之间没有任何约束, $\Pi_0 = \text{true}$. 如果 s_0 是唯一的,每条路径 $p \in Path_{\mathcal{G}}$ 对于唯一的符号状态路径. 对于 $i \geqslant 0$, $\langle l_i, op, l_{i+1} \rangle \in \rightarrow$, $s_{i+1} = s_i[op]$. 为了表达方便,令 $SP(p)$ 为执行路径所对应的符号状态路径, $S_{\mathcal{G}}$ 为 $\mathcal{G}$ 相关的符号状态集合; $Loc(s) = l$, $Eval(s) = A$, $Cond(s) = \Pi$,其中 $s = \langle l, A, \Pi \rangle$. 如果对出现在符号状态中的符号变量用相应类型的常量进行代换得到 D_{Vars} 类型的程序变量值状态及相应的值状态变迁路径. 令 $SV(s)$ 表示出现在符号状态 s 里的符号变量, $D_{SV(s)}$ 同 D_{Vars} 意义类似,表示符号变量值域的笛卡儿积, $E(s) = SV(s) \rightarrow D_{SV(s)}$ 为对符号变量进行常量赋值的映射集合,被称为符号状态 s 的环境. 一般情况下,集合 Act 中的动作不会引进新的符号变量,因此任何符号状态路径中的符号变量集合为 $SV(s_0)$,所有符号状态 s 的环境满足 $E(s) \subseteq E(s_0)$. 对任意环境 $\mathcal{E}$,记 $\mathcal{E}(\Pi) = \{\mathcal{A} \in \mathcal{E} \mid \mathcal{E} \models \Pi\}$, $\mathcal{E}(s) = \{\mathcal{A} \in \mathcal{E}_0 \mid \mathcal{A} \models Cond(s)\}$, $\mathcal{E}(l) = \bigvee_{s \in S_{\mathcal{G}}} (\mathcal{E}_0(Cond(s)) \wedge Loc(s) = l)$,其中 $\mathcal{E}_0 = E(s_0)$, $l \in \Sigma$, $s \in S_{\mathcal{G}}$.

定义 5. 给定 CFG $\mathcal{G}=\langle \Sigma, init, Act, \rightarrow, final\rangle$ 和 $rank_{\mathcal{G}}$,二元组 $\mathcal{M}=\langle \mathcal{G}, rank_{\mathcal{G}}\rangle$ 称为含序 CFG 结构. 下面基于含序 CFG 结构 $\mathcal{M}$ 及符号状态路径建立区间时序逻辑 CFITL 的解释模型. $(\mathcal{M},l_0)\models_{\mathcal{E}}\psi$ 表示在环境 $\mathcal{E}$ 下结构 $\mathcal{M}$ 的状态 l_0 满足公式 ψ,具体定义如下:

$(\mathcal{M},l_0)\models_{\mathcal{E}}$ true iff $\mathcal{E}\neq\varnothing$.

$(\mathcal{M},l_0)\models_{\mathcal{E}} p$ iff $\forall s=\langle l_0,A_0,\Pi_0\rangle\in S_{\mathcal{G}}\cdot\forall\mathcal{A}\in\mathcal{E}(\Pi_0)\cdot A_0\wedge\mathcal{A}\models p$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\psi_1\wedge\psi_2$ iff $(\mathcal{M},l_0)\models_{\mathcal{E}}\psi_1$ and $(\mathcal{M},l_0)\models_{\mathcal{E}}\psi_2$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\psi_1\vee\psi_2$ iff $\forall s=\langle l_0,A_0,\Pi_0\rangle\in S_{\mathcal{G}}\cdot\forall\mathcal{A}\in\mathcal{E}\cdot((\mathcal{M},l_0)\models_{\{\mathcal{A}\}}\psi_1\vee(\mathcal{M},l_0)\models_{\{\mathcal{A}\}}\psi_2)$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\neg\psi$ iff $(\mathcal{M},l_0)\not\models_{\mathcal{E}}\psi$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{EX}\psi$ iff $\exists\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge(\mathcal{M},l_1)\models_{\mathcal{E}(Cond(s_1))}\psi)$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{AX}\psi$ iff $\forall\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge(\mathcal{M},l_1)\models_{\mathcal{E}(Cond(s_1))}\psi)$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{EG}\psi$ iff $\exists\mathcal{P}=l_0l_1\cdots l_\infty\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\forall i\geqslant 0\cdot(\mathcal{M},Loc(s_i))\models_{\mathcal{E}(Cond(s_i))}\psi)$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{AG}\psi$ iff $\forall\mathcal{P}=l_0l_1\cdots l_\infty\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\forall i\geqslant 0\cdot(\mathcal{M},Loc(s_i))\models_{\mathcal{E}(Cond(s_i))}\psi)$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{EF}\psi$ iff $\exists\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\exists i\geqslant 0\cdot(\mathcal{M},Loc(s_i))\models_{\mathcal{E}(Cond(s_i))}\psi)$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{AF}\psi$ iff $\forall\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\exists i\geqslant 0\cdot(\mathcal{M},Loc(s_i))\models_{\mathcal{E}(Cond(s_i))}\psi)$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{E}(\psi_1\mathbb{U}\psi_2)$ iff $\exists\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\exists j\geqslant 0\cdot(\forall 0\leqslant i<j\cdot(\mathcal{M},l_i)\models_{\mathcal{E}(Cond(s_i))}\psi_1\wedge(\mathcal{M},l_j)\models_{\mathcal{E}(Cond(s_j))}\psi_2))$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{A}(\psi_1\mathbb{U}\psi_2)$ iff $\forall\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\exists j\geqslant 0\cdot(\forall 0\leqslant i<j\cdot(\mathcal{M},l_i)\models_{\mathcal{E}(Cond(s_i))}\psi_1\wedge(\mathcal{M},l_j)\models_{\mathcal{E}(Cond(s_j))}\psi_2))$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{EF}^I\psi$ iff $\exists\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\exists i\geqslant 0\cdot(rank_{\mathcal{G}}(l_i)\in I\wedge(\mathcal{M},l_i)\models_{\mathcal{E}(Cond(s_i))}\psi))$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{AF}^I\psi$ iff $\forall=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\exists i\geqslant 0\cdot(rank_{\mathcal{G}}(l_i)\in I\wedge(\mathcal{M},l_i)\models_{\mathcal{E}(Cond(s_i))}\psi))$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{EG}^I\psi$ iff $\exists\mathcal{P}=l_0l_1\cdots l_\infty\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\forall i\geqslant 0\cdot(rank_{\mathcal{G}}(l_i)\in I\wedge(\mathcal{M},l_i)\models_{\mathcal{E}(Cond(s_i))}\psi))$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{AG}^I\psi$ iff $\forall\mathcal{P}=l_0l_1\cdots l_\infty\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\forall i\geqslant 0\cdot(rank_{\mathcal{G}}(l_i)\in I\rightarrow(\mathcal{M},l_i)\models_{\mathcal{E}(Cond(s_i))}\psi))$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{E}(\psi_1\mathbb{U}^I\psi_2)$ iff $\exists\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\exists j\geqslant 0\cdot(rank_{\mathcal{G}}(l_j)\in I\wedge(\mathcal{M},l_j)\models_{\mathcal{E}(Cond(s_j))}\psi_2\wedge\forall 0\leqslant i<j\cdot(\mathcal{M},l_i)\models_{\mathcal{E}(Cond(s_i))}\psi_1))$.

$(\mathcal{M},l_0)\models_{\mathcal{E}}\mathbb{A}(\psi_1\mathbb{U}^I\psi_2)$ iff $\forall\mathcal{P}=l_0l_1\cdots\in Path_{\mathcal{G}}\cdot(SP(\mathcal{P})=s_0s_1\cdots\wedge\exists j\geqslant 0\cdot(rank_{\mathcal{G}}(l_j)\in I\wedge(\mathcal{M},l_j)\models_{\mathcal{E}(Cond(s_j))}\psi_2\wedge\forall 0\leqslant i<j\cdot(\mathcal{M},l_i)\models_{\mathcal{E}(Cond(s_i))}\psi_1))$.

关于 CFITL 语义模型的解释,做出如下进一步的 4 点说明:

1) 语义定义中 $p\in AP$, $\mathcal{P}=l_0l_1\cdots l_\infty$ 表示 2 种类型的路径:无限路径或满足 $l_\infty=final$ 的有限路径. 当 $\mathcal{P}$ 为无限路径时,l_∞ 不表示具体的状态,只是表达无限的概念. 标准的 CTL 语义解释是基于无限路径的,路径只与系统的动态行为相关;CFITL 的语义解释是基于符号状态路径,既可能为无限路径,也可能为有限路径,并且符号状态路径集成了系统动态行为信息及程序静态结构信息.

2) 当环境 $\mathcal{E}=\varnothing$,对于任意 ψ, $(\mathcal{M},l)\not\models_{\mathcal{E}}\psi$. 当 $l_0=init$ 时,$(\mathcal{M},l_0)\models_{\mathcal{E}}\psi$ 可简写为 $(\mathcal{M},l_0)\models\psi$,此时环境为初始环境,即 $E(s_0)$;当 $l_0\neq init$ 时,由于从初始状态 $init$ 开始,可能存在多条路径到达 l_0,根据定义 5 可知,每条路径可以产生环境映射,因此 $(\mathcal{M},l_0)\models\psi$ 意味着对于所有可能的环境 $\mathcal{E}$ 而言,$(\mathcal{M},l_0)\models_{\mathcal{E}}\psi$ 成立. 利用当 $\mathcal{E}=\varnothing$, $(\mathcal{M},l)\not\models_{\mathcal{E}}$ true 的性质,可通过检验 $(\mathcal{M},init)\models\mathbb{AG}$true 是否成立来判断程序是否存在死代码. 当该属性不成立,说明存在死代码.

3) 由于 CFITL 的语义模型与 CTL 语义模型有着根本的差异,因此一些在 CTL 里很自然的、符合直觉的逻辑关系在 CFITL 里不一定成立,例如 $\mathbb{EX}\psi\neq\neg\,\mathbb{AX}\,\neg\psi$, $\mathbb{AG}\psi\neq\neg\,\mathbb{EF}\,\neg\psi$,如果 $(\mathcal{M},l_0)\models_{\mathcal{E}}\psi_1\vee\psi_2$ 并不等价于 $(\mathcal{M},l_1)\models_{\mathcal{E}}\psi_1\vee(\mathcal{M},l_0)\models_{\mathcal{E}}\psi_2$ 等.

4) 集合 Act 含有 2 种类型的动作:赋值动作 $v=e$ 和条件判定动作 $assume(c)$. 在程序语言层面,

赋值动作改变程序变量的值,程序的执行语义模型的状态迁移描述了这种迁移前后状态之间的关系;条件判定动作不会改变程序变量的值,一般隐式或显式地作为前一种动作的约束条件体现在程序的语义模型的状态变迁中.用于解释 CFITL 公式中时序算子的路径 $l_0l_1\cdots$包含 2 种操作所导致的 CFG 结构中的状态变迁,这种时序的观念集成了程序静态结构的"序"概念和体现程序动态行为的程序变量值状态变化的"序"概念,这一点与 CTL 公式时序算子的"序"的解释不一样.当然,通过简单的转换,可以去掉路径 $l_0l_1\cdots$中操作 $assume(c)$所导致的状态变迁,建立起与 CTL 标准解释模型 Kripke 结构一致的"序"的观念.

按照 CFITL 公式的语义解释,下面分别用 CFITL 公式描述图 1 所示程序应该满足的 4 条典型性质:

1) $\mathbb{AF}delete(b)$;

2) $\mathbb{AF}^{19}(result=0\rightarrow Zero(a))$, $\mathbb{AG}^{19}(result=0\rightarrow Zero(a))$;

3) $\mathbb{AG}^{[2,7]}(i\leqslant a.length\wedge spot\leqslant a.length)$;

4) $\mathbb{AG}^{[2,7]}(spot\neq a.length\rightarrow Zero(a,0,spot)\wedge Zero(b,0,spot))$.

以上的公式中,$delete(b)$表示命题"数组 b 所占用的内存空间被释放",$Zero(a)$表示命题"数组 a 所有的元素值为 0",$Zero(a,0,spot)$表示命题"数组 a 从第 1 个元素至第 $spot$ 个元素的值为 0".性质 2 中 2 个公式等价,描述的是同一性质,19 是[19,19]的简写.根据公式的语义解释可知,图 1 所示程序满足性质 2~4,不满足性质 1.

1.2.1 节论及对于给定的 *CFG* $\mathcal{G}$,如果存在位阶函数,则存在无穷个满足定义 4 的位阶函数.这意味着同一描述程序结构特征和动态行为特征的 CFITL 公式可以构造不同的含序 CFG 结构$\langle\mathcal{G},rank_{\mathcal{G}}\rangle$,对公式给出不同的解释.为了消除 CFITL 公式相对于 CFG 所呈现的多义性,下面提出规范化位阶函数的概念.正如定理 1 的证明所述,一个可计算函数(程序)是由顺序结构、选择结构及循环结构 3 种类型的程序块通过嵌套的方式所产生,并且程序本身也属于这 3 种结构之一.为了表述方便,引进表达式 $P[str_1,str_2,\cdots,str_n]$,其中 $P,str_1,str_2,\cdots,str_n$ 是 $seq,branch$ 或 $loop$ 类型的结构. $P[str_1,str_2,\cdots,str_n]$结构表示程序 P 包含 n 个各种结构的程序块 $str_1,str_2,\cdots,str_n$,表达式中的逗号",",与 $seq(str_1,str_2,\cdots,str_n)$或 $branch(c,str_1,str_2)$中的逗号不一样,没有"序"的概念.任意 2 个不同的程序块 B_i 和 B_j 可以是嵌套关系或非嵌套关系. $\mathcal{G}=\langle\Sigma,init,Act,final,\rightarrow\rangle$为与 P 相关的 CFG;$rank_{\mathcal{G}}$ 为 $\mathcal{G}$ 的某个位阶函数,如果 $rank_{\mathcal{G}}$ 满足下列 4 个条件,则被称为规范化位阶函数(规范化位阶函数记为 $nrank_{\mathcal{G}}$):

1) 对于任意 $l\in\Sigma$, $1\leqslant rank_{\mathcal{G}}(l)\leqslant|\Sigma|$,即函数 $rank_{\mathcal{G}}$ 为类型 $\Sigma\rightarrow[1,|\Sigma|]$的双射函数.

2) 对于 P 的任意程序块 B_i,当 $l\in\Sigma[str_i]$, $rank_{\mathcal{G}}(l)\in[low,high]$,对于任意 $n\in[low,high]$,存在 $l\in\Sigma[str_i]$, $rank_{\mathcal{G}}(l)=n$.显然 $1\leqslant low\leqslant high\leqslant|\Sigma|$.满足此条件的位阶函数确保属于同一程序块的状态集合所对应的序数集合为某一完整的自然数区间.

3) 当 str_i 为选择结构程序块,str_t 和 str_f 分别为"真"分支程序块和"假"分支程序块,则对于任意 $l\in\Sigma[str_t]$, $l'\in\Sigma[str_f]$, $rank_{\mathcal{G}}(l)<rank_{\mathcal{G}}(l')$.

4) 对于任意基本程序块 str_i 和 str_j,如果它们之间没有嵌套关系,并且 $rank_{\mathcal{G}}(\Sigma[str_i])=[low_1,high_2]$, $rank_{\mathcal{G}}(\Sigma[str_j])=[low_2,high_2]$,则$[low_1,high_1]\cap[low_2,high_2]=\varnothing$,如果 str_i 嵌套于 str_j,则 $\Sigma[str_i]\subseteq\Sigma[str_j]$,式中 $rank_{\mathcal{G}}(\Sigma[str])=\{n\in\mathbb{N}\mid n=rank_{\mathcal{G}}(l),l\in\Sigma[str]\}$, str 为任意结构的程序块.

对于给定的 CFG $\mathcal{G}$,如果$\langle\Sigma,\lhd\rangle$是偏序集,则存在位阶函数 $rank_{\mathcal{G}}$ 及相应的规范化位阶函数 $nrank_{\mathcal{G}}$,且该规范化位阶函数是唯一的.定理 2 给出了规范化位阶函数存在性和唯一性的证明.如果用含序 CFG 结构$\langle\mathcal{G},nrank_{\mathcal{G}}\rangle$代替普通的含序 CFG 结构$\langle\mathcal{G},rank_{\mathcal{G}}\rangle$,则 CFITL 公式的解释具有唯一性.下文如果没有特别说明,CFITL 公式的解释都是基于规范化位阶函数.

给定程序 P、该程序诱导的偏序集$\langle\Sigma,\lhd\rangle$、规范化位阶函数 $nrank_{\mathcal{G}}$、程序 P 所包含的结构 str,并且任意 $l\in\Sigma[str]$, $nrank_{\mathcal{G}}(l)\in[low,high]$.

结论 1. 当 str 为顺序结构 $seq(str_1,str_2,\cdots,str_n)$,根据规范化位阶函数的定义可得以下性质:

1) $|\Sigma[str]|=\sum_{i=1}^{n}|\Sigma[str_i]|=high-low+1$;

2) 对于任意 $1\leqslant i<j\leqslant n$, $\forall l\in\Sigma[str_i]\cdot\forall l'\in\Sigma[str_j]\cdot(l\lhd l'\wedge nrank_{\mathcal{G}}(l)<nrank_{\mathcal{G}}(l'))$;

3) 对于任意 $1<i\leqslant n$, $\forall l\in\Sigma[str_i]\cdot\left(nrank_{\mathcal{G}}(l)\in\left[\sum_{j=1}^{i-1}|\Sigma[str_j]|+1,\sum_{j=1}^{i-1}|\Sigma[str_j]|+|\Sigma[str_i]|\right]\right)$,当 $l\in\Sigma[str_1]$, $nrank_{\mathcal{G}}(l)\in[low,low+|\Sigma[str_1]|-1]$.

结论 2. 当 str 为选择结构 $branch(c,str_t,str_f)$,同理可得以下性质:

1) $|\Sigma[str]|=|\Sigma[str_t]|+|\Sigma[str_f]|+2=high-low+1$,式中的 2 表示选择结构的开始状态和结束状态;

2) $nrank_{\mathcal{G}}(l_c)=low$, $nrank_{\mathcal{G}}(l_{end})=high$, l_c 和 l_{end} 分别为开始状态和结束状态;

3) 任意 $l\in\Sigma[str_t]$, $nrank_{\mathcal{G}}\in[low+1,low+|\Sigma[str_t]|]$, $l\in\Sigma[str_f]$, $nrank_{\mathcal{G}}(l)\in[low+\Sigma[str_t]+1,high-1]$. 当 str 为只有真分支的选择结构 $branch(c,str_t)$时,$|\Sigma[str]|=|\Sigma[str_t]|+1=high-low+1$,任意 $l\in\Sigma[str_t]$, $nrank_{\mathcal{G}}(l)\in[low+1,high]$, $nrank_{\mathcal{G}}(l_c)=loc$.

结论 3. 当 str 为循环结构 $loop(c,str)$,根据关系 $\lhd$ 的定义可知,循环结构中具有无条件转移作用,但转移目标状态受限的动作 $assume(c)$对构造 $\lhd$ 的拓扑结构没有实质性的影响,因此对于循环结构 $loop(c,str)$和选择结构 $loop(c,str)$而言,规范化位阶函数 $nrank_{\mathcal{G}}$ 在状态集合 $\Sigma[loop(c,str)]$上的性质是等价的.

定理 2. 对于 CFG $\mathcal{G}=\langle\Sigma,init,Act,final,\rightarrow\rangle$,如果$\langle\Sigma,\lhd\rangle$为偏序结构,则 $\mathcal{G}$存在规范化位阶函数 $nrank_{\mathcal{G}}$,且是唯一的.

证明. 如果$\langle\Sigma,\lhd\rangle$是偏序结构,$\mathcal{G}$显然存在位阶函数.令 P 为与 $\mathcal{G}$ 相关的程序,不失一般性,设其结构为 $seq(str_1,str_2,\cdots,str_n)$,其中 str_i 为顺序结构、选择结构或循环结构,$1\leqslant i\leqslant n$.下面通过一个构造性算法证明规范化位阶函数的存在性和唯一性.$\mathcal{G}$的位阶函数 $rank_{\mathcal{G}}$ 按以下 2 个步骤进行构造:

1) $Str\_Vector=\{str_1,str_2,\cdots,str_n\}$, $N=\sum_{i=1}^{n}|\Sigma[str_i]|$,任意 $l\in\Sigma[P]$, $rank_{\mathcal{G}}(l)\in[1,N]$,并且结构 str_i 所对应的区间按照结论 1 进行设定;

2) 假如 $Str\_Vector\neq\varnothing$,从集合 Str_Vector 中选择结构 $str(Str\_Vector=Str\_Vector/\{str\})$:

① 假如 str 为不含有选择或循环结构的顺序结构,则 $l\in\Sigma[str]$, $nrank_{\mathcal{G}}(l)\in I_{str}$,且在区间 I_{str}, $rank_{\mathcal{G}}$ 具有保序的性质,I_{str}是结构所属的区间信息;

② 假如 str 为含有嵌套结构的顺序结构、选择结构或循环结构,则按照结论 1~3 的方法,把相关的内嵌结构加入 str_Vector,并设置相关结构所属的区间,并返回步骤①.

显然按照以上步骤构造的 $rank_{\mathcal{G}}$ 符合规范化位阶函数的性质要求,并且交换任意 2 个互相不嵌套结构的区间,不满足规范化位阶函数的要求,即规范化位阶函数是唯一的. 证毕.

2 模型检验

2.1 基于显式状态枚举的 CFITL 模型检验

第 1 节通过环境、符号状态及符号状态路径对 CFITL 公式的语义进行了详细的定义,在此基础上,本节重点讨论基于 CFITL 的模型检验方法.模型检验的研究始于 20 世纪 80 年代[5],主要用于验证有限状态系统是否满足一些特定的性质.基本思想是:用迁移系统表示系统的行为,用模态/时序逻辑公式描述系统的性质,系统是否满足某个性质就转化为"迁移系统是否是相应的模态逻辑公式的模型"这样的数学问题.模型检验的核心问题是设计可行的算法,自动化检验迁移系统是否是相应的模态逻辑公式的模型.根据模态逻辑公式类型和表达能力以及不同的算法原理,可以构造各种不同类型的算法用于完成模型检验问题的求解.尽管这些算法类型各异,但根本上它们在执行同一任务:可达状态空间的计算.

在 CFITL 公式语义定义中表达推理的符号是 $\models_{\mathcal{E}}$,而不是 $\models$,$\mathcal{E}$被称为"环境",可理解为对公式进行解释的上下文,本质上,此处的上下文指 CFG 的路径信息,因此对于 CFITL 公式的模型检验问题 $((\mathcal{M},l)\models_{\mathcal{E}}\psi?)$也可表述为路径敏感的模型检验问题.在设计具体的模型检验算法时,不能简单归结为可达程序状态空间 D_{Vars} 的计算问题.定义 5 所给的语义是基于抽象的符号状态及符号状态路径,下面首先建立非抽象状态空间,即显式的值状态空间,在此基础上定义 CFITL 模型检验问题的可达状态空间的计算方法.给定含序结构 $\mathcal{M}=\langle\mathcal{G}=\langle\Sigma,init,Act,final,\rightarrow\rangle,nrank_{\mathcal{G}}\rangle$,令 $D_{nrank_{\mathcal{G}}}$ 为规范化位阶函数 $nrank_{\mathcal{G}}$ 的值域.根据 CFITL 的语义定义可知,某个具体的值状态需要包含程序变量的值信息(程序的动态行为信息)和位阶函数信息(程序的静态结构信息),基于这些考虑,值状态空间可表示为 $D_{nrank_{\mathcal{G}}}\times D_{Vars}$.进一步,为了把 $\models_{\mathcal{E}}$ 问题的求解转化为 $\models$ 问题的求解,引进状态环境的概念,为了与 CFITL 语义定义时的环境相区别,用 $\tilde{\mathcal{E}}$ 表示值状态的环境.状态 $\bar{s}\in D_{nrank_{\mathcal{G}}}\times D_{Vars}$ 的环境以下标的形式表示,即 $\bar{s}_{\tilde{\mathcal{E}}}$ 表示状态 $\bar{s}$ 的环境为 $\tilde{\mathcal{E}}$.下面任意给出的状态都具有环境,如果没有显式地以下标形式给出,则用 $\tilde{\mathcal{E}}(\bar{s})$ 获取 $\bar{s}$ 的环境.当 $\bar{s}=\langle nrank(init),val\rangle$时,$\tilde{\mathcal{E}}(\bar{s})=$

true. $\mathcal{E}$ 和 $\widetilde{\mathcal{E}}$ 都刻画程序的路径信息,两者具有内在的一致性. 对于 $\tilde{s}=\langle n,val\rangle$,必定存在唯一的 $s=\langle nrank_{\mathcal{G}}^{-1}(n),A,\Pi\rangle\in S_{\mathcal{G}}$,$\tilde{s}$ 和 s 满足 $val\models\widetilde{\mathcal{E}}(\tilde{s})\leftrightarrow\exists\mathcal{A}\in\mathcal{E}(s)\cdot(\mathcal{A}\models\Pi\wedge A[\mathcal{A}]=val)$,即对于确定的值状态 $\tilde{s}$,可对应唯一的推理环境 $\mathcal{E}(s)$,简记为 $\mathcal{E}(\tilde{s})$. 任意状态集合 $S\subseteq D_{nrank_{\mathcal{G}}}\times D_{Vars}$,$S$ 经过一步状态转换可达状态的集合记为 $Reach^{\rightarrow}(S)$,经过任意次状态转换可达状态的集合记为 $Reach^{*}(S)$:

$$Reach^{\rightarrow}(S)=S\cup\{\tilde{s}'=\langle n',val'\rangle\mid\exists\tilde{s}=\langle n,val\rangle\in D_{nrank_{\mathcal{G}}}\times D_{Vars}\cdot(\exists t=\langle l,op,l'\rangle\in\rightarrow\cdot(nrank_{\mathcal{G}}(l)=n\wedge nrank_{\mathcal{G}}(l')=n'\wedge\widetilde{\mathcal{E}}(\tilde{s}')=\widetilde{\mathcal{E}}(\tilde{s})[op]\wedge val'=val[op]))\},$$

其中,当 $op=$"$assume(c)$",$\widetilde{\mathcal{E}}(\tilde{s})[op]=\widetilde{\mathcal{E}}(\tilde{s})\wedge[c]_{\tilde{s}}$,$[c]_{\tilde{s}}$ 表示用状态 $\tilde{s}$ 里相应变量值对 c 的值进行评估;当 $op=$"$v=e$",$\widetilde{\mathcal{E}}(\tilde{s})[op]=\widetilde{\mathcal{E}}(\tilde{s})$. 如果把 val 和 val' 视作 $Vars\rightarrow D_{Vars}$ 类型函数,则 $val[op]$ 的结果也为 $Vars\rightarrow D_{Vars}$ 类型的函数. 当 $op=$"$assume(c)$",$val[op]=val$;当 $op=$"$v=e$",$val[op]=val[v\leftarrow e]$ 解释如下:

$$val[v=e](v')=\begin{cases}[e]_{val}, & v'=v,\\ val(v'), & v'\neq v,\end{cases}$$

其中,$[e]_{val}$ 表示由 val 获得相应变量的值代入表达式 e,对 e 进行计算,得到相应的常量值. 给定初始状态集合 $I\subseteq D_{nrank_{\mathcal{G}}}\times D_{Vars}$,程序的可达状态集合为 $Reach(I)$. $\tilde{s}_0=\langle n_0,val_0\rangle\in I$,$nrank_{\mathcal{G}}(init)=n_0$,根据 CFG $\mathcal{G}$ 可以得到一条执行路径 $\tilde{s}_0\tilde{s}_1\cdots$,其中 $\tilde{s}_i=\langle n_i,val_i\rangle$,对于 $i\geqslant 0$,存在 $op\in Act$,$\langle nrank^{-1}(n_i),op,nrank^{-1}(n_{i+1})\rangle\in\rightarrow$,并且 $\widetilde{\mathcal{E}}(\tilde{s}_{i+1})\Rightarrow\widetilde{\mathcal{E}}(\tilde{s}_i)$,$val_i\models\widetilde{\mathcal{E}}(\tilde{s}_i)$,$val_{i+1}=val_i[op]$. 显然 $Reach^{*}(Reach^{*}(S))=Reach^{*}(S)$,$Reach^{\rightarrow}(Reach(S)=Reach^{*}(S))$,即 $Reach^{*}(S)$ 是 $Reach^{\rightarrow}(\cdot)$ 的增量迭代计算的不动点.

通过 $Reach^{*}(\cdot)$ 可以构造基本的 CFITL 模型检验算法. 例如已知初始状态集 I 和 $Reach^{*}(I)$,假如所有的 $\tilde{s}=\langle nrank_{\mathcal{G}}(init),val\rangle\in Reach^{*}(I)$ 满足 $val\models p$,则 $(\mathcal{M},init)\models p$;当存在 $\langle init,op,l\rangle\in\rightarrow$,对所有长度为 1 的路径 $\tilde{s}_0\tilde{s}_1$,其中 $\tilde{s}_0,\tilde{s}_1=\langle nrank_{\mathcal{G}},val_1\rangle\in Reach^{*}(I)$,$val_1\models p$,则 $(\mathcal{M},init)\models\mathbb{EX}p$. 算法 1 和算法 2 给出了公式 $\mathbb{EG}f$ 和 $\mathbb{E}(f_1\ \mathbb{U}^{I}f_2)$ 模型检验算法的伪代码.

算法 1. 模型检验算法 $CheckEG(f)$.

输入:公式 f、程序 P;

输出:可达状态集合 $S_{\mathbf{EG}}$.

① $S=\{\tilde{s}=\langle n,val\rangle\in Reach^{*}(I)\mid(\mathcal{M},nrank_{\mathcal{G}}^{-1}(n))\models_{\mathcal{E}(\tilde{s})}f\}$;

② $SCC=\{C\mid C$ 是强连通图或具有终止状态 $\tilde{s}=\langle n,val\rangle$ 的有限路径,满足 $nrank_{\mathcal{G}}^{-1}(n)=final$ 且 $val\in D_{Vars}$ of $S\}$;

③ $S_{SCC}=\bigcup\limits_{C\in SCC}\{\tilde{s}\mid\tilde{s}\in C\}$;

④ $S_{\mathbf{EG}}=S_{SCC}$;

⑤ while ($S_{SCC}\neq\varnothing$) do

⑥ $\quad choose\ \tilde{s}=\langle n,val\rangle\in S_{SCC}$;

⑦ $\quad S_{SCC}=S_{SCC}/\{\tilde{s}\}$;

⑧ $\quad find\ \tilde{s}'=\langle n',val'\rangle\in S$,满足 $\exists op\in Act\cdot(\langle nrank_{\mathcal{G}}^{-1}(n'),op,nrank_{\mathcal{G}}^{-1}(n)\rangle\in\rightarrow\wedge\widetilde{\mathcal{E}}(\tilde{s})=\widetilde{\mathcal{E}}(\tilde{s}')[op]\wedge val=val'[op])$;

⑨ $\quad$ if ($\tilde{s}'\neq S_{SCC}$) $S_{SCC}=S_{SCC}\cup\{\tilde{s}'\}$;

⑩ $\quad S_{\mathbf{EG}}=S_{\mathbf{EG}}\cup\{\tilde{s}'\}$;

⑪ end while

算法 2. 模型检验算法 $CheckEU(I,f_1,f_2)$.

输入:公式 f_1、公式 f_2、区间 I;

输出:可达状态集合 $S_{\mathbf{EU}}$.

① $S=\{\tilde{s}=\langle n,val\rangle\in Reach^{*}(I)\mid(\mathcal{M},nrank_{\mathcal{G}}^{-1}(n))\models_{\mathcal{E}(\tilde{s})}f_2\wedge n\in I\}$;

② $S_{\mathbf{EU}}=S$;

③ while ($S\neq\varnothing$) do

④ $\quad choose\ \tilde{s}=\langle n,val\rangle\in S$;

⑤ $\quad S=S/\{\tilde{s}\}$;

⑥ $\quad\tilde{s}=\langle n',val'\rangle$,满足 $\exists op\in(\langle nrank_{\mathcal{G}}^{-1}(n'),op,nrank_{\mathcal{G}}^{-1}(n)\rangle\in\rightarrow\wedge\widetilde{E}(s)=\widetilde{\mathcal{E}}(s')[op]\wedge val=val'[op]\wedge(\mathcal{M},nrank_{\mathcal{G}}^{-1})\models_{\mathcal{E}(\tilde{s})}f_1$;

⑦ $\quad$ if ($\tilde{s}'\notin S$) $S=S\cup\{\tilde{s}'\}$;

⑧ $\quad S_{\mathbf{EU}}=S_{\mathbf{EU}}\cup\{\tilde{s}'\}$;

⑨ end while

按照算法 1 的描述,集合 SCC 含有 2 种类型的元素:一种元素是 CFG 所产生的有限值状态变迁结构中的强连通子图,并且该强连通图中的每个状态都满足 f;另外一种元素是有限路径,并且该路径是程序的完整执行路径,即程序处于终止状态,关于这一点 2.2 节将有更详细的讨论. 当存在 $\tilde{s}=\langle nrank_{\mathcal{G}}(init),val\rangle\in S_{\mathbf{EG}}$ 时,$(\mathcal{M},init)\models\mathbb{EG}f$ 成立,即程序满足属性 $\mathbb{EG}f$. 其余类型公式相关的模型检验算法没有详细给出. 当待验证的公式是类似于 $\neg\psi$ 时,传统的 CTL 模型检验算法都通过公式的等价变换,把 $\neg\psi$ 的模型检验问题转化为等价的公式 ψ'($\neg$ 只作用于原子命题)的模型检验问题. 在 CFITL 中,与路径量词和时序算子相关的许多自然的性质不再成立,因此一般都通过先获得满足属性 ψ 的状态空间 $\Sigma'\subseteq\Sigma$,再得到满足 $\neg\psi$ 的状态空间

Σ/Σ'这种方法求解 $\neg\psi$ 公式的模型检验问题.算法 1 和算法 2 的设计中,使用了$\models_{\varepsilon}$可满足性关系,例如算法 1 的行①在计算满足 f 属性的值状态空间涉及符号状态的变迁关系,而不是值状态变迁关系.如果用环境$\{\mathcal{A}\}\subseteq\mathcal{E}(\bar{s})$代替算法 1 行①中$\models_{\mathcal{E}(\bar{s})}$的环境$\mathcal{E}(\bar{s})$,并且在环境$\{\mathcal{A}\}$下与 $\bar{s}$ 相关的符号状态 s 满足 $Eval(S)=val$,这样 $\bar{s}=\langle nrank_{\mathcal{G}}(l),val\rangle\in S$ 意味着$(\mathcal{M},l)\models_{\{\mathcal{A}\}}f$,而不意味着$(\mathcal{M},l)\models_{\mathcal{E}(\bar{s})}f$.因此在计算出可达值状态集合后,还需要测试是否某符号路径所产生的值状态都包含在集合 $S_{\mathbf{EG}}$中,假如经过了测试,才能得出某符号状态路径满足属性$\mathbf{EG}f$的结论,算法 3 给出了一个不完备的算法.

算法 3. 路径测试函数 $TestSymPath(S,sp)$.

输入:S,sp;

输出:success/failure.

① if ($sp=l_0l_1\cdots l_n$ 为符号化路径,l_n 为终止状态)

② $\quad SymS=\{s_i\parallel 0\leqslant i\leqslant n,SP(sp)=s_0s_1\cdots s_n\}$;

③ $\quad$while ($SymS\neq\varnothing$)

④ $\quad\quad choose\ s=\langle l,A,\Pi\rangle\in SymS$;

⑤ $\quad\quad SymS=SymS/\{s\}$;

⑥ $\quad\quad\widetilde{S}=\{\bar{s}=\langle n',val\rangle\mid n'=nrank_{\mathcal{G}}(l),$
$\quad\quad\quad\exists\,\mathcal{A}\in\mathcal{E}(s)\cdot(val=A[\mathcal{A}])\}$;

⑦ $\quad\quad$if ($\widetilde{S}\not\subseteq S$) return failure;

⑧ $\quad$end while

⑨ $\quad$return success;

⑩ else /* $sp=l_0l_1\cdots l_i(l_{i+1}\cdots l_n)^*$ */

⑪ $\quad TestSymPath(S,l_0l_1\cdots l_i(l_{i+1}\cdots l_n)^{k_{\max}})$;

⑫ end if

函数 $TestSymPath$ 中参数 S 表示值状态集合,例如算法 1 中的 S,sp 为待测试符号路径,可以采用正规语言的方式表达.参数 $k_{\max}$是与 SCC 相关的一个参数,是反映 SCC 路径模式的一个参数,需要计算值状态路径与正规式表达的符号路径模式匹配问题,需要很大的计算开销.如果在系统的有界模型检验中存在阈值,则可取阈值作为 $k_{\max}$的保守值.当状态空间规模较小时,可采用模拟执行计算符号路径 sp 值状态集合不动点的方法进行计算.

下面通过与 CTL 的对比讨论 CFITL 模型检验的算法复杂度.当语义模型 Kripke 结构$\langle S,R,L\rangle$的状态集 S 为有限时,CTL 公式可满足性是可判定的,算法复杂度为 $O(len(f)(Card(S)+Card(R)))$.当系统实现的模型 CFG 所诱导的 Kripke 结构为有限结构时,由于刻画有序 CFG 拓扑结构的数值特征为整数区间,状态的数值属性为单点型的区间,在此条件下,CFITL 公式可满足性是可判定的.本节基于扩展后的状态空间 $D_{nrank_{\mathcal{G}}}\times D_{Vars}$构造了部分公式的判定算法,算法的复杂度为 $O(len(f)(Card(D_{nrank_{\mathcal{G}}}\times D_{Vars})+Card(R)))$.

CFITL 模型检验算法的复杂度较 CTL 模型检验算法高,但并不是指数级的升高,受 $D_{nrank_{\mathcal{G}}}$约束.对于大状态空间或无限状态空间系统的模型检验,传统的基于抽象技术,例如谓词抽象,经过重新设计后仍然可以应用于 CFITL 模型检验,以缓解状态空间爆炸问题;另外 2.2 节结合 CFITL 公式重写技术的有界模型检验也是一种实施途径,这也是原型系统采用的基本方法.

2.2 基于 SMT 的 CFITL 有界模型检验

基于状态枚举的模型检验算法一个首要的前提是构建系统的有限状态系统,例如算法 1 和算法 2 都只能应用于有限状态系统.现实中的各种类型程序涉及复杂的逻辑控制及具有无限值域的抽象数据类型,不能直接获得有限状态的系统结构,即使应用各种类型的抽象建模技术[28-29],但由于难以建立统一的、自动化的抽象机制及抽象精化方法,各种抽象技术也只能缓解,而不能从根本上克服各种模型检验方法所面临的可控规模状态空间的系统模型的构建问题,包括符号模型检验方法.另一方面,模型检验方法的完备性是相对于系统规约而言的,但开发人员不可能给出完全正确且完备的系统规约,以确保所开发的目标系统是完全"正确"的系统,而且,在资源受限的情况下,再先进的模型检验器也不可能对系统所有的属性进行验证.

基于以上这些因素,Clarke 等人提出有界模型检验的思想[9],这种方法不要求验证具有完备性,并且把基于可达状态集合的计算问题转化为命题逻辑的可满足性判定问题.这样,模型检验问题可以利用许多高效的可满足性判定器求解,例如 GRASP,SATO,ZCHAFF 等[30-31],从而避免可达状态集计算所带来的存储空间和计算时间的开销问题.从某种意义上,可以认为有界模型检验方法是介于测试和一般模型检验之间的一种方法,它采用定理证明的方法测试在系统的局部行为空间内规约是否成立,兼具测试和证明的特点,因此有界模型检验除了用于证明属性是否成立外,还经常用于查找程序的错误.

从上面的论述可以看出,有界模型检验方法的性能和能力取决于可满足性判定器.由于现代程序含有各种复杂的抽象数据类型及操作,包括 GRASP

和SATO等传统的命题逻辑可满足性判定器难以直接应用于程序属性验证.近年来,可满足性模理论及相关工具得到了快速地发展,例如:Z3[32],CVC,MathSAT等.SMT问题是命题逻辑公式可满足性判定问题的扩展:谓词逻辑公式的可满足性判定[25].纯逻辑领域不涉及任何领域性的理论,但在实际的应用中,大部分SMT判定器都内建有复杂的领域理论,包括整数理论、实数理论、未解释函数理论及一些处理复杂数据类型的理论,如向量、队列和数组理论等.这些背景理论的集成使得SMT判定器具有描述及推理复杂的特定领域问题的能力.许多程序分析与验证领域的问题:可验证的编译方法、谓词抽象、模型检验等,这些问题都可转化为SMT问题,利用集成多种不同背景理论的SMT判定器进行求解.SMT研究领域的众多的研究机构和研究人员提出了SMT工具的标准通讯语言和命令,被称为SMT-LIB,通过SMT-LIB可以方便地描述背景理论、各种复杂的待求解的问题及操纵各种SMT工具,并且这些SMT工具大都提供了丰富的人机接口和编程接口.

基于SMT的CFITL模型检验方法的关键环节是把有界模型检验问题归结于SMT,用集成了各种域理论的谓词逻辑语言进行描述.给定含序CFG结构$\mathcal{M}=\langle\mathcal{G},nrank_{\mathcal{G}}\rangle$,待验证时序属性$f$及界限$\mathcal{K}$($\mathcal{K}\geqslant 0$),用$[\![\mathcal{M},f]\!]_{\mathcal{K}}$表示长度为$\mathcal{K}$的符号状态路径满足$f$的充分必要条件,$[\![\mathcal{M},f]\!]_{\mathcal{K}}$由2部分组成:1)长度为$\mathcal{K}$的来自CFG $\mathcal{G}$的符号状态路径的行为及静态结构规范,记为$[\![\mathcal{M}]\!]_{\mathcal{K}}$;2)满足属性$f$且长度为$\mathcal{K}$的路径应该满足的条件,记为$[\![f]\!]_{\mathcal{K}}$,$[\![\mathcal{M},f]\!]_{\mathcal{K}}=[\![\mathcal{M}]\!]_{\mathcal{K}}\wedge[\![f]\!]_{\mathcal{K}}$.

$$[\![\mathcal{M}]\!]_{\mathcal{K}}=\bigwedge_{i=0}^{\mathcal{K}}(Eval(s_i)\wedge Cond(s_i))$$

,其中$s_i\in S_{\mathcal{G}}$,$Loc(s_0)=init$,对于$0\leqslant i\leqslant\mathcal{K}-1$,$s_i$,$s_{i+1}$满足$\exists op\in Act\cdot(\langle Loc(s_i),op,Loc(s_{i+1})\rangle\in\rightarrow)$.

$[\![f]\!]_{\mathcal{K}}$的编码形式取决于公式f和当前所处的状态$l\in\Sigma$,因此$[\![f]\!]_{\mathcal{K}}$的完整形式可表达为$[\![f]\!]_{\mathcal{K}}^{i}$,其中$i$表示符号状态路径$s_0s_1\cdots$的当前状态.当$i=0$时简记为$[\![f]\!]_{\mathcal{K}}$,此处由于$[\![f]\!]_{\mathcal{K}}^{i}$仅涉及单条符号状态路径,公式中的路径量词没有实质性的作用,例如$[\![\mathbb{E}\mathbb{G}p]\!]_{\mathcal{K}}^{i}=[\![\mathbb{A}\mathbb{G}p]\!]_{\mathcal{K}}^{i}$,因此可用$[\![\mathbb{G}p]\!]_{\mathcal{K}}^{i}$统一表达$[\![\mathbb{A}\mathbb{G}p]\!]_{\mathcal{K}}^{i}$和$[\![\mathbb{E}\mathbb{G}p]\!]_{\mathcal{K}}^{i}$.$[\![f]\!]_{\mathcal{K}}^{i}$可根据公式结构,用与LTL公式重写技术[24]类似的方法进行递归定义.根据定义5可知,在一些规约动态行为的时序逻辑里一些与路径量词及普通逻辑联结词相关的很自然、符合直觉感受的逻辑学原理在CFITL中是不成立的,此处为了简化问题的描述,假定公式里的否定词直接作用于原子命题.$[\![f]\!]_{\mathcal{K}}^{i}$详细的编码形式如下所示:

$[\![p]\!]_{\mathcal{K}}^{i}=[p]_{s_i}$;

$[\![\neg p]\!]_{\mathcal{K}}^{i}=[\neg p]_{s_i}$;

$[\![\psi_1\wedge\psi_2]\!]_{\mathcal{K}}^{i}=[\![\psi_1]\!]_{\mathcal{K}}^{i}\wedge[\![\psi_2]\!]_{\mathcal{K}}^{i}$;

$[\![\psi_1\vee\psi_2]\!]_{\mathcal{K}}^{i}=[\![\psi_1]\!]_{\mathcal{K}}^{i}\vee[\![\psi_2]\!]_{\mathcal{K}}^{i}$;

$[\![\mathbb{X}\psi]\!]_{\mathcal{K}}^{i}=[\![\psi]\!]_{\mathcal{K}}^{i+1}$;

$[\![\mathbb{G}\psi]\!]_{\mathcal{K}}^{i}=[\![\psi]\!]_{\mathcal{K}}^{i}\wedge[\![\mathbb{G}\psi]\!]_{\mathcal{K}}^{i+1}$;

$[\![\mathbb{F}\psi]\!]_{\mathcal{K}}^{i}=[\![\psi]\!]_{\mathcal{K}}^{i}\vee[\![\mathbb{F}\psi]\!]_{\mathcal{K}}^{i+1}$;

$[\![\mathbb{G}^I\psi]\!]_{\mathcal{K}}^{i}=[\![nrank_{\mathcal{G}}(Loc(s_i))\in I\rightarrow\psi]\!]_{\mathcal{K}}^{i}\wedge[\![\mathbb{G}^I\psi]\!]_{\mathcal{K}}^{i+1}$;

$[\![\mathbb{F}^I\psi]\!]_{\mathcal{K}}^{i}=[\![nrank_{\mathcal{G}}(Loc(s_i))\in I\wedge\psi]\!]_{\mathcal{K}}^{i}\vee[\![\mathbb{F}^I\psi]\!]_{\mathcal{K}}^{i+1}$.

由于符号状态路径$s_0s_1\cdots$的最大长度为$\mathcal{K}$,因此上式中$i\leqslant\mathcal{K}-1$,$[p]_{s_i}$表示基于符号状态s_i对命题p进行解释.根据语义定义可知,析取运算$\vee$的行为不具有一般意义上的那种直觉的性质,即$(\mathcal{M},l)\models_{\varepsilon}\psi_1\vee\psi_2$等价于$(\mathcal{M},l)\models_{\varepsilon}\psi_1\vee(\mathcal{M},l)\models_{\varepsilon}\psi_2$,上式中关于析取运算给出了一种更强的重写结果$[\![\psi_1]\!]_{\mathcal{K}}^{i}\vee[\![\psi_2]\!]_{\mathcal{K}}^{i}$,如果要得到等价SMT的编码形式,则需要引入量词.关于时序算子$\mathbb{G}$的重写规则需要特别的进一步的解释.根据定义5可知,$\mathbb{G}^I$和$\mathbb{G}$语义解释涉及2种类型的符号状态路径:一种是无限路径;一种是有限长度路径,且当程序执行完该路径后处于终止状态.当$s_{\mathcal{K}}$满足$Loc(s_{\mathcal{K}})=final$,且不存在$op\in Act$,$l\in\Sigma$,$\langle Loc(s_{\mathcal{K}}),op,l\rangle\in\rightarrow$时,即$s_0s_1\cdots s_{\mathcal{K}}$是程序的一条完整的执行路径,则只要迭代应用公式重写规则计算出$[\![\mathbb{G}\psi]\!]_{\mathcal{K}}^{i}$和$[\![\mathbb{G}^I\psi]\!]_{\mathcal{K}}^{i}$.当$s_0s_1\cdots s_{\mathcal{K}}$不是程序的完整路径,即$Loc(s_{\mathcal{K}})\neq final$,或者当$Loc(s_{\mathcal{K}})=final$,存在$op\in Act$,$l\in\Sigma$,$\langle Loc(s_{\mathcal{K}}),op,l\rangle\in\rightarrow$,那么可以分成以下2种情况:

1)当该路径为套索路径(lasso-shape path)[33]时,有限路径展现出无限路径的性质,该有限路径满足$[\![\mathbb{G}\psi]\!]_{\mathcal{K}}^{i}$或$[\![\mathbb{G}^I\psi]\!]_{\mathcal{K}}^{i}$意味着相关的无限路径满足$\mathbb{G}\psi$或$\mathbb{G}^I\psi$.当存在$0\leqslant i\leqslant\mathcal{K}$,并且存在$op\in Act$,$\langle Loc(s_{\mathcal{K}}),op,Loc(s_i)\rangle\in\rightarrow$,对于所有的$\mathcal{A}\in\mathcal{E}(s_{\mathcal{K}})$,$Eval(s_i)=Eval(s_{\mathcal{K}})[op]$,$s_0s_1\cdots s_i\cdots s_{\mathcal{K}}$为套索符号路径,与其相关的无限路径为$s_0s_1\cdots(s_i\cdots s_{\mathcal{K}})^{\omega}$,其中$\omega$表示无限的意思.

2)当由CFG $\mathcal{G}$产生的以$s_0s_1\cdots s_{\mathcal{K}}$为前缀的任意符号状态路径满足$\forall i>\mathcal{K}\cdot nrank_{\mathcal{G}}(Loc(s_i))\notin I$,则$s_0s_1\cdots s_{\mathcal{K}}$满足$[\![\mathbb{G}^I\psi]\!]_{\mathcal{K}}^{i}$意味着符合条件的无限路径满足$\mathbb{G}^I\psi$,这种情况称为区间排斥条件.$s_0s_1\cdots s_{\mathcal{K}}$是套索路径的条件为$LC=\bigvee_{i=0}^{\mathcal{K}}(\exists op\in Act\cdot$

$(\langle Loc(s_{\mathcal{K}}), op, s_i\rangle \in \rightarrow \wedge Eval(s_i) = Eval(s_{\mathcal{K}})[op]))$, $s_{\mathcal{K}}$ 是程序终止符号状态的条件为 $TC = (Loc(s_{\mathcal{K}}) = final \wedge \neg \exists op \in Act \cdot \exists l \in \Sigma \cdot \langle Loc(s_{\mathcal{K}}), op, l\rangle \in \rightarrow)$,更强的区间排斥条件为 $IE = \forall l \in \Sigma \cdot (nrank_{\mathcal{G}}^{-1}(upbound(I)) \lhd l \rightarrow \neg \exists op \in Act\, \exists 0 \leqslant i \leqslant \mathcal{K} \cdot (\langle l, op, Loc(s_i)\rangle \in \rightarrow))$,其中 $upbound(I)$ 表示区间 I 的上界. 基于 LC, TC, IE,下面对有关公式的编码进行改写:

$$[\![\mathcal{M}, \Delta \mathbb{G}\psi]\!]_{\mathcal{K}} = [\![\mathcal{M}]\!]_{\mathcal{K}} \wedge [\![\mathbb{G}\psi]\!]_{\mathcal{K}} \wedge (LC \vee TC),$$

$$[\![\mathcal{M}, \Delta \mathbb{G}^I\psi]\!]_{\mathcal{K}} = [\![\mathcal{M}]\!]_{\mathcal{K}} \wedge [\![\mathbb{G}^I\psi]\!]_{\mathcal{K}} \wedge (LC \vee TC \vee IE).$$

上式中 Δ 表示任意的路径量词. 需要强调的是上面所讨论的在有界模型检验的条件下证明 $\mathbb{G}\psi$ 和 $\mathbb{G}^I\psi$ 的方法可靠但并不完备,因为当程序所实现的系统为无限状态系统时,存在不含套索的无限路径满足 $\mathbb{G}\psi$ 或 $\mathbb{G}^I\psi$. 以一个没有上界的简单计数器为例,该系统是无限状态系统,显然容易得到一个不存在套索路径且满足 $\mathbb{AG}(count \geqslant 0)$ 的系统实现,$count$ 为计数器变量. 如果结合符号状态路径的结构特性和时序算子 $\mathbb{F}$ 的语义特性,同样可以得出一些关于 $[\![\mathbb{F}\psi]\!]_{\mathcal{K}}^i$ 和 $[\![\mathbb{F}^I\psi]\!]_{\mathcal{K}}^i$ 的更有意义的结论. 例如:如果有界符号路径 $s_0 s_1 \cdots s_{\mathcal{K}}$ 不满足 $[\![\mathbb{F}\psi]\!]_{\mathcal{K}}$ 或 $[\![\mathbb{F}^I\psi]\!]_{\mathcal{K}}$,并不能得出没有以 $s_0 s_1 \cdots s_{\mathcal{K}}$ 为前缀的路径满足 $[\![\mathbb{F}\psi]\!]_{\mathcal{K}}$ 或 $[\![\mathbb{F}^I\psi]\!]_{\mathcal{K}}$,但如果 $s_0 s_1 \cdots s_{\mathcal{K}}$ 满足 IE,则可以据此得到可靠且完备的结论,即 $s_0 s_1 \cdots s_{\mathcal{K}} \models \Delta \mathbb{F}^I\psi \Leftrightarrow s_0 s_1 \cdots s_{\mathcal{K}} \cdots \models \Delta \mathbb{F}^I\psi$.

从上面的相关分析可以看出,虽然有界模型检验方法不是完备的,但结合模型结构的分析,可以得到可靠且完备的结果,这方面的工作可参考文献[9].

实际上,在实施模型检验时,根据不同类型的属性 f,可以采用不同的验证策略. 例如对于安全性属性 $\mathbb{AG}(p_1 \wedge p_2)$,可以通过验证是否满足 $\mathbb{EF}(\neg(p_1 \wedge p_2))$ 的结果考察属性 $\mathbb{AG}(p_1 \wedge p_2)$. 假如满足 $\mathbb{EF}(\neg(p_1 \wedge p_2))$,则 $\mathbb{AG}(p_1 \wedge p_2)$ 不成立,并且满足 $\mathbb{EF}(\neg(p_1 \wedge p_2))$ 的符号状态路径是反例,进一步调用 SMT 工具获得值状态路径,结合 2 种类型的路径信息,可以更精确地对程序进行调试,定位程序错误;当在界限为 $\mathcal{K}$ 的条件下,$\mathbb{EF}(\neg(p_1 \wedge p_2))$ 不成立,则进一步根据系统的结构特性,例如完备的界限阈值及相关的路径特征属性信息 LC, TC, IE 等作出判断,决定下一步采取的策略. 对于形如 $\mathbb{AF}p$ 的活性属性同样可以采取 2 种验证策略.

3 VerTPCFG 模型检验器

CFITL 直接用于描述程序的各类与静态结构及动态行为相关的时序属性,为了使所设计的模型检验器系统开发成本尽可能低、运行自动化程度尽可能高、能处理的程序数据类型和结构尽可能丰富,尽量复用已有的高性能的开源软件工具,基于 SMT 的有界模型检验方法实现了相关的原型工具. CFITL 有界模型检验器总体架构如图 3 所示:

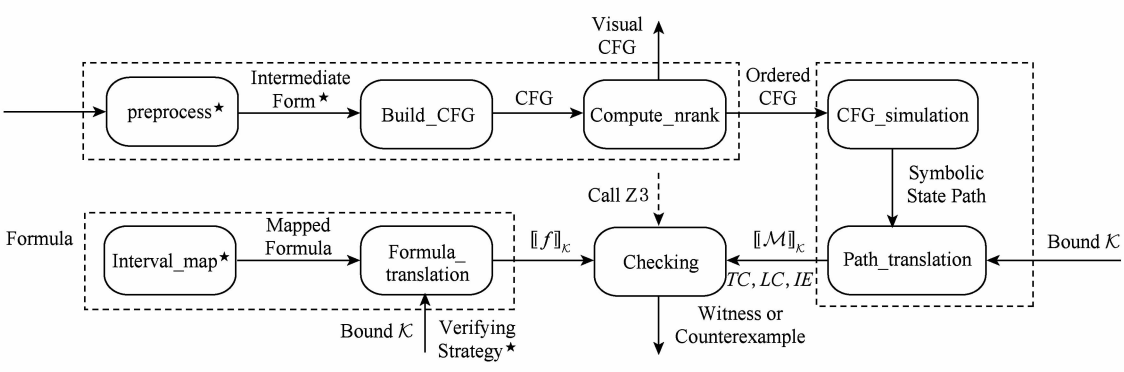

Fig. 3 The framework of bounded model checker over CFITL.

图 3 CFITL 有界模型检验器框架

图 3 所给模型检验器设计框架中的大部分内容在第 2 节都有所涉及,下面就图中标有★的部分内容作进一步的介绍. 为了增强验证工具的可扩展性,在程序和 CFG 之间设计了一个中间语言层,使得 CFG 独立于具体的程序设计语言. 具体的程序至中间语言的翻译由预处理过程"preprocess"负责处理,不同的程序语言对应不同的预处理过程. 当然,为了得到有序的 CFG 结构,预处理过程还需要对原程序的语法进行有效性检查及程序的等价转换,这方面的主要内容包括有副作用语句的等价变换和有害"goto"语句的消除. 中间语言采用 STG Format[34],这种语言由 IF 规约语言子集改编而成,用于描述

各种复杂的顺序及并发反应式系统. 目标系统的程序设计与实现是一种具有创新性的工作,深刻地体现开发人员的主观能动性,对于同样的功能需求,具有不同知识背景和思维能力的人所设计程序的静态语法结构差异极大,单条语句语法与语义可能存在极大的差别. 面临这种情况,在构造 CFITL 的语义模型有序 CFG 结构时有 2 种处理策略:1)允许动作集合 Act 的元素具有"副作用",与源程序的语句保持一致,这样使得 CFG 基本结构与源程序具有相同的静态语法结构;2)限制集合 Act 元素的复杂性,使得它在语法层面上具有"原子性",这样 CFG 静态结构与源程序的静态结构会存在某种程度的不一致性.

第 1 种处理方法的优点是开发人员可以依据自己的程序实现写出要求满足的属性公式,但这种方法有着显而易见的缺点,即系统的动态行为(状态空间及状态变迁关系)与语句形态高度耦合,CFITL 的"时序"的意义没有统一的解释;第 2 种方法保持了对动态行为模型的认识观点,使得 CFITL 的"时序"具有统一的解释,即一个在语法层面上不具有副作用的原子语句的执行就意味着系统变换至下一时刻点. 但第 2 种方法也有不足,由于程序与相关的 CFG 在结构上存在不一致的地方,因此在描述涉及与结构相关的属性时,需要利用"compute_nrank"处理过程输出有序 CFG 结构的详细信息. 图 3 所示的方法采用一种折中的方法,即 CFG 的构建按照第 2 种处理策略,但属性描述所涉及的结构因素(区间)按照源程序的结构特征进行解释. 这种处理方法的优点是属性的描述不需要完整的 CFG 信息,属性的结构要素与源程序保持一致,最符合开发人员的直觉认知. 这会产生一个关键性的问题需要处理:建立虚拟 CFG(Act 集合与源程序的语句保持一致)与 CFITL 有序 CFG 结构之间的结构映射关系,这是过程"Interval_map"的核心工作. 输入"Verifying Strategy"代表 2.2 节的验证策略. 给定一个待验证的属性,可以根据公式的类型和特点及界限参数 $\mathcal{K}$,选择"证明驱动"策略或"反例驱动"策略,不同的策略意味着不同的$[\![f]\!]_{\mathcal{K}}$ 输出.

基于图 3 的系统架构,设计了 VerTPCFG 原型系统,图 4 显示了该原型系统. 目前,该系统还缺乏真正的前端,仅实现了基于 STG Format 语言的系统规约及其静态结构和动态行为属性的分析和验证. 基于原型工具 VerTPCFG,选取汽车风挡刮水控制器(windscreen wiper controller)、饮料销售自动机(beverage machiner)和电梯控制器(elevator controller)作为基本实验对象,这些实验对象用 STG Format 语言进行描述. 每个实验对象各设计了 4 条属性,这些属性基本都含有静态结构因素,例如循环不变式属性,界限的值简单地根据由系统描述所产生的 CFG 中最长路径长度确定,实验中人为确定为路径长度的 3 倍. 这些属性大都是安全性属性,

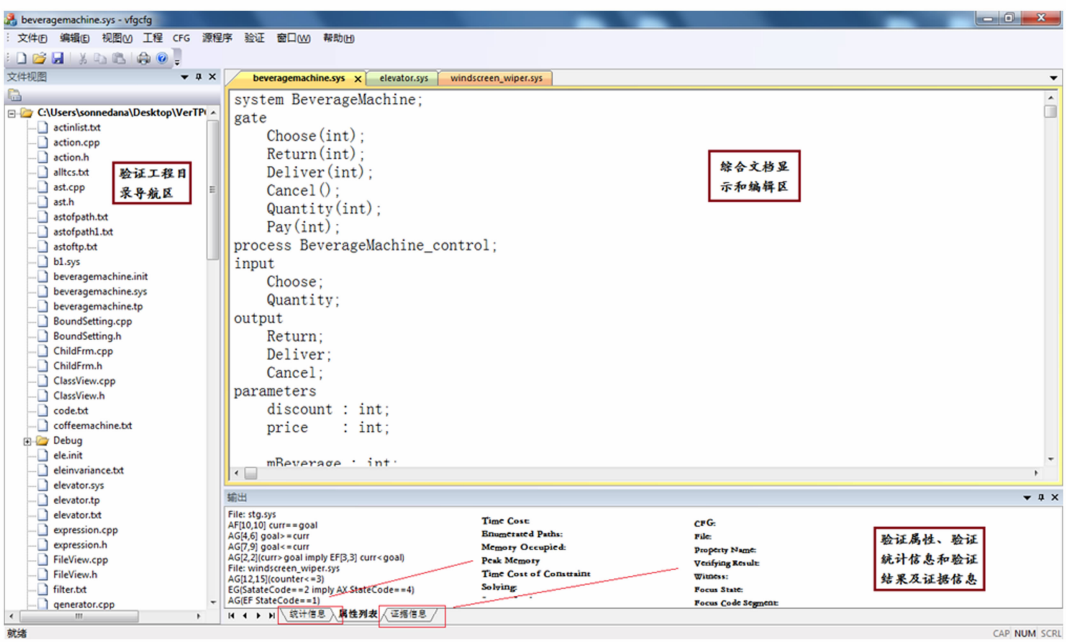

Fig. 4 The prototype system of VerTPCFG.

图 4 VerTPCFG 原型系统平台

因此除明确给出了反例的属性外,大部分属性的验证结果是不完备,原型工具没有计算 LC,TC,IE. 实验结果表明各条属性都在 30 s 以内得到证明,当然这与所给的属性公式的文本复杂度较低,不涉及复杂的时序算子的嵌套及 $\mathcal{K}$ 的取值相对较低等多种因素相关. 通过观察验证过程,发现较多亟待解决的问题,例如:验证$\mathbb{EG}^{[t_1,t_2]}\psi$类的属性时,容易给出平凡的路径,例如该路径所有状态的位阶不属于区间$[t_1,t_2]$;验证$\mathbb{AG}^{[t_1,t_2]}\psi$类属性时会出现大量无意义的路径信息的计算. 由于原型系统没有实现复杂数据类型的处理,且缺乏完整前端处理过程,即实现有效的程序实现语言到中间语言的翻译,因此仅选取了几个偏于控制类型的简单例子.

4 相关工作

时序逻辑和基于时序逻辑的软硬件系统的时序属性规约及验证是形式化方法领域的重要研究内容,鉴于 CTL 和 LTL 在表达能力、语法的易用性和判定算法的效率等方面所具有的平衡性,与之相关的逻辑理论、判定算法和数据结构及工具软件的开发和应用等方面的研究工作已经取得丰富的成果,并且这些成果已经成功地服务于工业界[1-2,5,8-12]. 国内唐稚松院士较早开展了关于时序逻辑理论方面的研究工作,提出了 XYZ/E 时序逻辑系统,该逻辑系统包括 2 个语言子集 XYZ/AE 和 XYZ/EE,分别用于描述程序的静态语义和动态行为语义的属性规约[35-36].

CTL 和 LTL 等这些时序逻辑仅能用于描述系统的动态语义属性. 显然,系统的动态行为属性由程序实现的静态结构属性决定,直接对实现的结构语义属性进行规约和推理具有重要的意义,是确保系统动态行为属性正确性的基础. 近年有多种逻辑被提出,用于处理程序实现的各种重要语法要素,例如指针和数组等,典型的包括分离逻辑[37]、范围逻辑[38]和动态逻辑[39]等. 分离逻辑和范围逻辑用于含有指针程序的属性分析;而动态逻辑则引入了类似于时序算子的模态算子,用于规约和推理程序的动态行为属性. 另外,陈意云教授研究团队提出了指针逻辑和形态图逻辑系统[40-41]用于对指针及递归数据结构的性质进行分析.

相比上面所提及的几类逻辑系统,CFITL 具有显著的特点和优势. 类似于 CTL 和 LTL 的时序逻辑系统,只具有规约和推理系统动态执行语义属性的能力,而缺乏规约系统实现的静态结构语义. CFITL 则具备在统一的框架内规约和推理静态结构语义和动态行为语义属性的能力. CFITL 的语法定义是基于 CTL 扩充而来的,但语义定义具有很大的差异,两者的表达能力具有相关性,又有很大的差异性. 例如对于只含有全称路径量词的 CTL 属性,在 CFITL 逻辑框架内能进行等价地表达;对于含有路径量词的公式,没有或不易构造相应的 CFITL 等价式,但 CFITL 有相应的一种更强的表达,即对于 CTL 公式 f_{ctl}、CFITL 公式 f_{cfitl},如果$(\mathcal{M},l_0)\models f_{\mathrm{cfitl}}$,则$(\mathcal{K},s_0)\models f_{\mathrm{ctl}}$,其中 $\mathcal{K}$ 为系统实现 $\mathcal{M}$ 所诱导的 Kripke 结构,f_{ctl}和 f_{cfitl}语法具有一定关联性. 当然,即使把这种更强的关系视为等价关系,也并不意味着 CFITL 语言包含 CTL. 显然,由于 CFITL 引进了区间的概念,关于静态结构属性的 CFITL 大部分描述在 CTL 框架里显然不能等价地表达. 当然 CTL 和 CFITL 语义解释之间的关系,下一步的工作将进行更加细致的研究.

分离逻辑和范围逻辑这类型的逻辑是 Hoare-Floyd 公理系统的扩展,用于推理具有指针类行为程序的属性,由于构造系统的目的与 CFITL 不一致,它们不具备系统化描述动态时序属性和静态结构属性的能力. XYZ/E 系统和动态逻辑系统可以直接用于规约和推理程序的行为属性,并且定义了类似的时序算子. 相对 CFITL 而言,XYZ/E 和动态逻辑系统具有较为复杂的语法体系,尤其是 XYZ/E 系统,它包含 2 个语言子系统,且定义了各种不同的原语,用于表达各种复杂时序属性所需要的时序、逻辑及程序结构相关的概念,动态逻辑系统基本不具备描述静态结构属性的能力;CFITL 通过引进整数区间作为程序结构因素的映射,使之具备对程序各种简单和复杂的结构属性进行规约的表达能力;另外 CFITL 通过继承 CTL 的语法和语义框架,使得许多 CTL 模型检验方法理论和算法思想可应用于 CFITL 模型检验领域.

区间时序逻辑包括一类具有数量约束指标范畴的时序逻辑,这些各种不同形式的数量指标大都用于复杂系统的性能属性,例如概率约束指标、数值奖赏指标、连续或离散时间约束指标和时钟约束等,这方面详情请参阅有关文献[3,7,14,18,42-44]. 这些各种不同类型具有数量约束指标的时序逻辑的目标显然还是系统的动态语义范畴,与 CFITL 的作用有着根本的差别. 另外还有一类典型的区间时序逻辑,例如 PITL[45]、Halpern-Shoham 逻辑[46](简称 HS

逻辑)及国内段振华教授提出的 PTL[47],它们的语义解释是线性离散时间结构,用于规约和推理具有离散时间刻度并发系统的离散时间相关的各类行为属性.基于 PITL,PTL 通过引进投影和投影星操作,使之具备规约和推理并发系统时间区间上的重复性属性,甚至类似于公正性要求的无穷次重复的属性以及进程同步性属性,文献[48]进一步把 PTL 用于单调速率调度算法可调度性的验证.基于不同的动机所构造的 CFITL,其语义解释是基于分支时间观念的非线性结构有序 CFG,并且有序 CFG 结构作为系统实现模型,决定了系统的行为模型——Kripke 结构的拓扑,因此 CFITL 公式集成了系统实现的静态结构语义和动态行为语义.计算机系统开发人员更习惯从系统实现的视面理解系统和规约系统,传统上经常采用谓词逻辑对系统实现进行规约,进而基于规则系统进行形式化的推理,而通过 CFITL 则可以采用自动的算法方法对实现的各类属性进行验证,例如循环不变式;另外,通过区间,CFITL 可以对系统实现结构在各种粒度上进行规约和推理.

5 总 结

系统的动态行为属性由系统实现的静态结构属性决定,研究关联系统实现静态结构和动态行为属性的规约形式和分析验证方法具有重要的应用价值.本文基于 CTL 构建了一种区间时序逻辑系统 CFITL,该逻辑系统具有描述复杂的程序静态结构和动态行为属性的表达能力,通过一类受限的 CFG 及 CFG 诱导的位阶函数给出了 CFITL 语义.位阶函数用于解释 CFG 的状态的拓扑信息,即程序的静态结构信息,定理 1 和定理 2 显示它的构造不单纯由程序的抽象视图 CFG 决定,还取决于 CFG 结构相关的程序结构因素.进一步,本文详细讨论了 CFITL 的模型检测问题,提出了具体的模型检测算法,包括基于可达值状态空间计算的模型检验算法和基于 SMT 及 CFITL 公式重写的有界模型检验算法,并基于后一种算法初步实现了 CFITL 模型检验器原型工具.后续工作将围绕以下 3 方面展开:1)对 CFITL 逻辑系统的逻辑性质进行更深入地研究,包括表达能力和证明系统等;2)研究可行的抽象技术,适用于路径敏感的 CFITL 模型检验方法;3)基于 C 语言编程环境,完善 VerTPCFG 系统,集成各种复杂数据类型的处理功能及更丰富的可视化环境的支撑,使之能处理各类复杂程序.

参 考 文 献

[1] Ben-Ari M, Pnueli A, Manna M. The temoral logic of branching time [J]. Acta Informatica, 1981, 20(3): 207-226

[2] Huth M, Ryan M. Logic in Computer Science: Modelling and Reasoning about System [M]. Cambridge, UK: Cambridge University Press, 2004

[3] Kwiatkowska M. Model checking for probability and time: From theory to practice [C] //Proc of the 18th IEEE Symp on Logic in Computer Science. Piscataway, NJ: IEEE, 2003: 351-360

[4] Panangaden P. Labelled Markov Processes [M]. London: Imperial College Press, 2009

[5] Clarke E M, Grumberg O, Peled D A. Model Checking [M]. Cambridge, MA: MIT Press, 1999

[6] Burch J R, Clarke E M, McMillan K L, et al. Symbolic model checking: 10^{20} states and beyond [J]. Information and Computation, 1992, 98(2): 142-170

[7] Alur R, Courcourbetis C, Dill D. Model checking for real-time systems [C] //Proc of the 5th Symp on Logic in Computer Science. Piscataway, NJ: IEEE, 1990: 414-425

[8] Ball T, Majumdar R, Millstein T D, et al. Automatic predicate abstraction of C programs [C] // Proc of the 22nd ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2001: 203-213

[9] Clarke E M, Biere A, Raimi R. Bounded model checking using satisfiability solving [J]. Formal Methods in System Design, 2001, 19(1): 7-34

[10] Miller S P, Whalen M W, Cofer D D. Software model checking takes off [J]. Communications of the ACM, 2010, 53(2): 58-64

[11] Cordeiro L, Fischer B, Marques-Silva J. SMT-based bounded model checking for embedded ANSI-C software [J]. IEEE Trans on Software Engineering, 2012, 38(4): 957-974

[12] Dill D L, Clarke E M. Automatic verification of asynchronous circuits using temporal logic [J]. IEE Proceedings E Computers and Digital Techniques, 1986, 133(5): 276-282

[13] Bujang S D A, Selamat A. Verification of mobile SMS application with model checking agent [C] //Proc of the 8th Int Conf on Intelligent Systems Design and Applications. Piscataway, NJ: IEEE, 2008: 217-222

[14] Alur R, Jagadeesan L J, Kott J J, et al. Model-checking of real-time systems: A telecommunications application: Experience report [C] // Proc of the 19th Int Conf on Software Engineering. Piscataway, NJ: IEEE, 1997: 514-524

[15] Witkowski T, Blanc N, Kroening D, et al. Model checking concurrent Linux device drivers [C] //Proc of the 22nd IEEE/ACM Int Conf on Automated Software Engineering. New York: ACM, 2007: 501-504

[16] Chen H, Dean D, Wagner D. Model checking one million lines of C code [C] // Proc of the 11th Symp on Network and Distributed System Security. Washington DC: Internet Society, 2004: 171-185

[17] Necula G C. Proof-carrying code [C] //Proc of the 24th ACM SIGPLAN-SIGACT Symp on Principles of Programming Languages. New York: ACM, 1997: 106-119

[18] Bozga M, Iosif R, Perarnau S. Quantitative separation logic and programs with lists [J]. Journal of Automated Reasoning, 2010, 45(2): 131-156

[19] Löding C, Lutz C, Serre O. Propositional dynamic logic with recursive programs [J]. Journal of Logic and Algebraic Programming, 2007, 73(1): 51-69

[20] Flanagan C, Leino C K R, Lillibridge M, et al. Extended static checking for Java [C] //Proc of the 23rd ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2002: 234-245

[21] King J C. Symbolic execution and program testing [J]. Communications of the ACM, 1976, 19(7): 385-394

[22] Ma K K, Phang K Y, Jeffrey S F. Directed symbolic execution [G] //LNCS 6887: Proc of the 18th Int Conf on Statis Analysis. Berlin: Springer, 2011: 95-111

[23] Saswat A, Godefroid P, Tillmann N. Demand-driven compositional symbolic execution [G] //LNCS 4963: Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2008: 367-381

[24] Rosu G, Havelund K. Rewriting-based techniques for running verification [J]. Journal of Automated Software Engineering, 2005, 12(2): 151-197

[25] Nieuwenhuis R, Oliveras A, Tinelli C. Solving SAT and SAT modulo theories: From an abstract DPLL procedure to DPLL(T)[J]. Journal of ACM, 2006, 53(6): 937-977

[26] Böhm C, Jacopini G. Flow diagrams, turing machines and languages with only two formation rules [J]. Communications of the ACM, 1966, 9(5): 366-371

[27] Harel D. On folk theorems [J]. Communications of the ACM, 1980, 23(7): 379-389

[28] Cousot P, Cousot R. Abstract interpretation: A unified lattice model for the static analysis of programs by construction or approximation of fixpoints [C] // Proc of the 4th ACM SIGPLAN-SIGACT Symp on Principles of Programming Languages. New York: ACM, 1977: 238-252

[29] Dams D, Gerth R, Grumberg O. Abstract interpretation of reactive systems [J]. ACM Trans of Program Language System, 1997, 19(2): 253-291

[30] Zhang L, Malik S. The quest for efficient boolean satisfiability solvers [G] //LNCS 2404: Proc of the 14th Int Conf on Computer Aided Verification. Berlin: Springer, 2002: 17-36

[31] Zhang H. SATO: An efficient propositional prover [G] // LNAI 1249: Proc of the 14th Int Conf on Automated Deduction. Berlin: Springer, 1997: 272-275

[32] Moura L D. Z3: An efficient SMT solver [G] //LNCS 4963: Proc of the 14th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2008: 337-340

[33] Fraser G, Wotawa F. Using LTL rewriting to improve the performance of model checker based test case generation [C] //Proc of the 3rd Workshop on Advances in Model Based Testing. New York: ACM, 2007: 64-74

[34] Clarke D, Jeron T, Rusu V. STG: A symbolic test generation tool [G] //LNCS 2280: Proc of the 8th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2002: 152-173

[35] Tang Zhisong, Zhao Chen. A temporal logic language oriented toward software engineering [J]. Journal of Software, 1994, 5(12): 1-16 (in Chinese)
(唐稚松, 赵琛. 一种面向软件工程的时序逻辑语言[J]. 软件学报, 1994, 5(12): 1-16)

[36] Zhu Xueyang. The dual software architecture description framework XYZ/ADL [J]. Journal Computer Research and Development, 2007, 44(9): 1485-1494 (in Chinese)
(朱雪阳. 双重软件体系结构描述框架 XYZ/ADL [J]. 计算机研究与发展, 2007, 44(9): 1485-1494)

[37] Reynolds J C. Separation logic: A logic for shared mutable data structure [C] //Proc of the 17th Annual IEEE Symp on Logic in Computer Science. Piscataway, NJ: IEEE, 2002: 55-74

[38] Zhao J H, Li X D. Scope logic: An extension to hoare logic for pointers and recursive data structures [G] //LNCS 8049: Theoretical Aspects of Computing. Berlin: Springer, 2013: 409-426

[39] Balbiani P, Vakarelov D. PDL with intersection of programs: A complete axiomatization [J]. Journal of Applied Non-Classical Logics, 2003, 13(4): 231-276

[40] Chen Yiyun, Hua Baojian, Ge Lin, et al. A pointer logic for safety verification of pointer programs [J]. Chinese Journal of Computers, 2008, 31(3): 372-380 (in Chinese)
(陈意云, 华保健, 葛琳, 等. 一种用于指针程序安全性证明的指针逻辑[J]. 计算机学报, 2008, 31(3): 372-380)

[41] Li Z P, Zhang Y, Chen Y Y. A shape graph logic and a shape system [J]. Journal of Computer Science & Technology, 2003, 28(6): 1063-1084

[42] Li Guangyuan, Tang Zhisong. Representing and verifying reactvie systems with continuous-time temporal logic [J]. Chinese Journal of Computers, 2003, 26(11): 1424-1434 (in Chinese)
(李广元, 唐稚松. 反应系统的连续时序逻辑表示和验证 [J]. 计算机学报, 2003, 26(11): 1424-1434)

[43] Lin Chang, Qu Yang, Li Yajuan. Modeling, consistency and inference of extended interval temporal logic [J]. Chinese Journal of Computers, 2002, 25(12): 1338-1347 (in Chinese)

(林闯, 曲杨, 李雅娟. 扩展时段时序逻辑的模型、一致性和推理[J]. 计算机学报, 2002, 25(12): 1338-1347)

[44] Bresolin D, Monica D D, Montanari A, et al. The light side of interval temporal logic: The Bernays-Schonfinkel fragment of CDT [J]. Annals of Mathematics and Artificial Intelligence, 2014, 71(1): 11-39

[45] Moszkowski B. Reasoning about digital circuits, STAN-CS-83-970 [R]. Palo Alto, CA: Stanford University, 1983

[46] Halpern J, Shoham Y. A propositional modal logic of time intervals [J]. Journal of ACM, 1991, 38(4): 935-962

[47] Duan Zhenhua. An extended interval temporal logic and a framing technique for temporal logic programming [D]. Newcastle Upon Tyne, UK: Newcastle University, 1996

[48] Tian Cong, Duan Zhenhua. Model checking rate monotone scheduling algorithm based on propositional projection temporal logic [J]. Journal of Software, 2011, 22(2): 211-221 (in Chinese)
(田聪, 段振华. 基于命题投影时序逻辑的单调速率调度算法模型检测[J]. 软件学报, 2011, 22(2): 211-221)

Chen Donghuo, born in 1974. PhD, lecturer. Member of China Computer Federation. His main research interests include program verification, model checking, and automated reasoning.

Liu Quan, born in 1969. Post doctor, professor and PhD supervisor. Senior member of China Computer Federation. His main research interests include intelligence information processing, automated reasoning, and machine learning.

Jin Haidong, born in 1973. PhD candidate, lecturer. His main research interests include reinforcement learning, and human-computer interaction.

Zhu Fei, born in 1978. PhD, associate professor. Member of China Computer Federation. His main research interests include reinforcement learning, text mining, and pattern recognition.

Wang Hui, born in 1968. PhD candidate, lecturer. Member of China Computer Federation. His main research interests include machine learning, and human-computer interaction.