高技术通讯 2015 年 第25 卷 第10-11 期:895 ~ 904

doi:10.3772/j.issn.1002-0470.2015.10-11.005

# 基于嵌套树的对等博弈应用研究①

郭 婧② 徐中伟

(同济大学电子与信息工程学院 上海 201804)

**摘 要** 进行了软件系统性质验证研究。首先提出了用嵌套树表示软件程序的方法,以解决软件系统的抽象表示问题,该方法能在表示程序顺序结构的同时,更好地表示调用返回关系。然后定义了嵌套树上的 $\mu$-演算,以便将要验证的需求性质用公式表示,并把公式转化成非确定对等嵌套树自动机。最后将自动机与嵌套树结合,转化形成博弈图,并用对等博弈条件来判断博弈的输赢,这等同于检验验证公式是否在嵌套树上成立。相比直接验证,这种判定方法表达更为直观,且更有利于整个过程的自动化。研究表明,将嵌套树中的调用关系展开可形成概要标签树,嵌套树的对等博弈理论也可以应用到概要标签树中。

**关键词** 对等博弈,嵌套树,交替树自动机,$\mu$-演算,概要

## 0 引 言

近年来,模型检测技术[1]越来越广泛地被应用到实践中。它是一种用数学方法表示并在此基础上证明系统性质的验证技术,相比软件测试具有精确度高、自动化率高等优点。传统的模型检测方法将要验证的系统用变迁系统来表示,然后通过验证来检验这系统是否满足需求。而软件系统的模型检测主要针对程序的执行[2],因而,需要构造一个概念框架去说明程序的逻辑结构,在这个框架下判定性质是否满足[3]。传统的模型检测方法无法在软件系统上被直接应用,因此,为了突破此限制,必须对传统方法加以改进以适应不同系统的需求。人们以软件系统为对象开展了大量研究。80 年代初,程序逻辑与相关的判定过程相融合,并使用基本算法在无限状态空间上验证性质,初步形成了软件系统的模型检测方法理论[4,5]。文献[6]针对 JAVA 程序的数据连续性,转换成 EML 程序,同时支持单个位置、多个位置的数据点。文献[7]提出了一种产生主类型的程序分析方法,只针对程序语法说明进行前向、后向的执行,这种分析方法还可以用于模型检测工具 SLAM 和 BLAST。文献[8]提出利用标志记录信息、以 SPIN 为工具的检测方法,但建立新标志时较为费时、易出错。

现有的研究都存在不同的局限性,没有构成一个完整的理论体系,且只针对特定的语言,无法广泛地被应用。软件系统的模型检测主要需要解决三个方面的问题,分别为软件系统的模型构建问题、需求的逻辑表示问题及需求在构建的模型上的验证问题。针对第一个问题,以嵌套树作为模型来表达软件程序,先根据程序的顺序结构,用树的结构进行表示[9]。对于程序中的调用返回关系,将存在此关系的结点用边相连,构成跳跃边。跳跃边不存在交叉关系,是以嵌套的形式存在。针对第二个问题,形式上的 $\mu$-演算是用来描述需求的一种基础逻辑,其数学内涵十分丰富,可以与 LTL[10]、CTL[11]、CTL* 等价。嵌套树下的 $\mu$-演算[12]可以表示软件丰富的需

① 国家自然科学基金(61075002),国家科技支撑计划重大项目(2011BAG01B03)和 863 计划(2012AA112801)资助项目。
② 女,1987 年生,博士生;研究方向:形式化验证,软件测试;联系人,E-mail:guo_jing163@163.com
(收稿日期:2015-04-14)

求,并且将形式的 $\mu$ -演算转化成交替树自动机[13],可以将简明地表示随意性分支[14],满足下一步验证的需要。针对第三个问题,验证中性质的存在性和性质的可满足性可以分别转化成交替树自动机上的接收性和非空性[15],下一步,交替自动机上的接收性和非空性与博弈上的接收性和非空性等价。用博弈表达验证过程更为直观,验证性质是否得到满足的过程同时又是判定选手是否赢的过程[16]。

本文在上述理论的基础上,进一步对其进行改进。首先,改进表示的基本单位,不是以结点为单位,而是定义一种新的单位—概要。它不仅考虑从当前结点到调用结点的顺序过程,还同时考虑所有可能执行的其他过程。其次,在嵌套树上以概要为单位对形式的 $\mu$ -演算详细说明,来表达不同的性质需求。再次,在系统的验证上,不采用简单的博弈理论,而采用博弈的分支—对等博弈,它能简化判断博弈输赢的条件,用对等条件能提高判断的效率。最后,研究另一种嵌套树的转化形式—概要标签树。它通过增加树的分支来去除嵌套,来表示调用返回关系,使结构更加明了。通过研究发现概要标签树公式的拆分方法与嵌套树类似,因此嵌套树的验证理论也可直接被应用到其中。

## 1 基本概念及相关定义

**定义 1**(嵌套树):$T=(S,r,\rightarrow,\mapsto)$,其中 $S$ 是结点集合,$r$ 是根节点,$\rightarrow\subseteq S\times S$ 是边的变迁关系集合,$\mapsto\subseteq S\times(S\cup\{\infty\})$ 是跳跃边的集合。对于结点 $s$,用 $s\rightarrow t$ 表示 $s$ 和 $t$ 之间存在直接变迁关系,$s$ 为源结点,$t$ 为目标结点;$s\mapsto t$ 表示调用关系,$s$ 是调用结点,$t$ 是返回结点;$s\mapsto\infty$ 是特殊的调用关系,表示 $s$ 是调用结点且调用没有返回。嵌套树中的道路表示为 $\pi=s_1s_2\cdots s_n\cdots$,且对所有 $i\geqslant 1$,$s_i\rightarrow s_{i+1}$。子嵌套树结构为 $(T,s_I)$,其中 $T=(S,r,\rightarrow,\mapsto)$ 为子嵌套树所在的主嵌套树,$s_I$ 为子嵌套树的根状态。$E$ 为边的集合,被分为三类:调用边(*call*)、返回边(*return*)和局部边(*local*)。调用边的源结点是调用结点,返回边的目标结点是返回结点,其余的边则是局部边。

嵌套标签树 $T=(S,r,\rightarrow,\mapsto,\lambda)$ 在嵌套树的基础上,增加结点与标签之间的映射关系 $\lambda:S\rightarrow\sum$,$S$ 是结点集合,$\sum$ 是嵌套树的标签集合。

概要是嵌套树下定义的一个基本语法结构,针对功能的执行,首先定义 $ME(s)$,为结点 $s$ 的匹配出口结点集合。如果 $t$ 为 $s$ 的匹配出口结点,必须满足两个条件:

(a)如果 $s\xrightarrow{+}t$,并且存在结点 $s'\xrightarrow{+}s$ 和 $s'\mapsto t$。该条件表示 $s'$ 是 $s$ 的前驱结点,且 $s'$、$s$、$t$ 之间有道路相连。

(b)不存在结点 $v$、$w$,使 $s'\xrightarrow{+}v\xrightarrow{+}s\xrightarrow{+}w$ 和 $v\mapsto w$。该条件表示 $s'$、$s$ 之间不存在结点 $v$,$v$ 结点是调用结点,且相应的返回结点其他结点都能到达。

总之,求解结点 $s$ 的匹配出口结点集合 $ME(s)$ 的过程有两个步骤,第一,记录它临近的前驱调用结点对应的返回结点集合,第二,将该集合中与结点 $s$ 无道路相连的结点去除,得到的结点集合即为 $ME(s)$。

**定义 2**(概要):对于一个非负整数 $k$,$k$ 种颜色的概要是一个数组 $\langle s,U_1,U_2,\cdots,U_k\rangle$,$s$ 是根结点,$U_1,U_2,\cdots,U_k$ 是 $ME(s)$ 的子集,集合中下标标签与集合中含有的结点数量相同。

## 2 嵌套树的固定点计算

### 2.1 语法规则

$AP$ 为原子命题的集合,$Var$ 为变量的有限集合,$\{M_1,M_2,\cdots\}$ 为可数的、有序的标签集合。对于 $p\in AP$,$x$,$X\in Var$,$k\geqslant 0$,公式的基本定义如下:

$$\Phi,\Phi'=p\mid\neg p\mid x\mid tt\mid ff\mid\Phi\vee\Phi'\mid\Phi\wedge\Phi'\mid\mu x.\Psi\mid\nu x.\Psi\mid\langle call\rangle\Phi'\{\varphi_1,\varphi_2,\cdots,\varphi_k\} \quad (1)$$

$$\mid[call]\Phi'\{\varphi_1,\varphi_2,\cdots,\varphi_k\}\mid\langle loc\rangle\Phi'\mid[loc]\Phi'\mid\langle ret\rangle M_i\mid[ret]M_i\mid\overline{\langle loc\rangle}\Phi'\mid\overline{[loc]}\Phi' \quad (2)$$

$$\mid\overline{[call]}\Phi'\{\varphi_1,\varphi_2,\cdots,\varphi_n\}\mid\overline{[call]}\Phi'\{\varphi_1,\varphi_2,\cdots,\varphi_n\}\mid\overline{\langle ret\rangle}\Phi'(R_i)\mid\overline{[ret]}\Phi'(R_i) \quad (3)$$

其中 $tt$ 表示所有结点都满足的公式,即所有原子公式的全集,$ff$ 为 $tt$ 的补集,即空公式集。设 $\alpha$ 有三种形式:*call*、*loc*、*ret*,分别为调用边(*call*)、局部边(*local*)和返回边(*return*)的缩写。〈 〉、[ ] 表示

前向变迁关系,$\overline{\langle\rangle}$、$\overline{[\,]}$则为后向变迁关系。

### 2.2　嵌套树环境下公式的语法意义

**定义 3**(环境变量):环境变量 $\varepsilon$:$free(\Phi)\to 2^S$,表示把 $\Phi$ 中的自由变量集合映射到嵌套树中的概要集合。$[\![\Phi]\!]_\varepsilon^T$ 表示在环境 $\varepsilon$ 满足 $\Phi$ 的 $T$ 中的概要集合。一般 $T$ 默认为当前嵌套树,所以可简写成 $[\![\Phi]\!]_\varepsilon$。

**定义 4**(在环境下公式的语法意义):对于一个概要 $s=\langle s,U_1,U_2,\cdots,U_k\rangle$, $s\in[\![\Phi]\!]_\varepsilon$, 当且仅当 $NT-\mu$ 公式在环境下满足以下条件:

(1) 如果 $\Phi=p\in AP$, 则 $p\in\lambda(s)$。

(2) 如果 $\Phi=\neg p$, $p\in AP$, 则 $p\notin\lambda(s)$。

(3) 如果 $\Phi=\Phi_1\vee\Phi_2$, 则, $s\in[\![\Phi_1]\!]_\varepsilon$ 或 $s\in[\![\Phi_2]\!]_\varepsilon$。

(4) 如果 $\Phi=\Phi_1\wedge\Phi_2$, 则, $s\in[\![\Phi_1]\!]_\varepsilon$ 和 $s\in[\![\Phi_2]\!]_\varepsilon$。

(5)如果结点 $s$ 存在调用关系,它的一个调用后驱结点 $t(s\xrightarrow{call}t)$所在概要$\langle t,V_1,V_2,\cdots,V_n\rangle$, 对于所有所有 $1\leqslant i\leqslant n$,$Vi=ME(t)\cap\{s'\colon <s',U_1\cap ME(s'),\cdots,U_k\cap ME(s')>\in[\![\varphi_i]\!]_\varepsilon\}$ 成立,那么 $\Phi=\langle call\rangle\Phi'\{\varphi_1,\varphi_2,\cdots,\varphi_k\}$ 在该概要上成立;如果结点 $s$ 存在调用关系,它的所有调用后驱结点 $t(s\xrightarrow{call}t)$所在概要 $\langle t,V_1,V_2,\cdots,V_n\rangle$, 对于所有所有 $1\leqslant i\leqslant n$,$Vi=ME(t)\cap\{s'\colon <s',U_1\cap ME(s'),\cdots,U_k\cap ME(s')>\in[\![\varphi_i]\!]_\varepsilon\}$ 成立,那么 $\Phi=[call]\Phi'\{\varphi_1,\varphi_2,\cdots,\varphi_k\}$ 在该概要上成立。

(6)如果结点 $s$ 存在局部关系,它的一个局部后驱结点 $t(s\xrightarrow{loc}t)$所在概要 $\langle t,V_1,V_2,\cdots,V_n\rangle$,$Vi=ME(t)\cap U_i$ 成立且该概要满足 $[\![\Phi']\!]_\varepsilon$, 那么 $\Phi=\langle loc\rangle\Phi'$ 在该概要上成立;如果结点 $s$ 存在局部关系,它的所有后驱结点 $t(s\xrightarrow{loc}t)$ 所在概要 $\langle t,V_1,V_2,\cdots,V_n\rangle$,$Vi=ME(t)\cap U_i$ 成立且该概要满足 $[\![\Phi']\!]_\varepsilon$, 那么 $\Phi=[loc]\Phi'$ 在该概要上成立。

(7)如果结点 $s$ 存在返回关系,它的一个返回后驱结点 $t(s\xrightarrow{ret}t)$ 满足 $t\in U_i$, 那么式子 $\Phi=\langle ret\rangle M_i$ 在该概要上成立;如果结点 $s$ 存在返回关系,它的所有后驱结点 $t(s\xrightarrow{ret}t)$ 满足 $t\in U_i$, 那么式子 $\Phi=[ret]M_i$ 在该概要上成立。

(8)如果结点 $s$ 存在局部关系,它的一个局部前驱结点 $t(t\xrightarrow{loc}s)$ 所在概要 $\langle t,V_1,V_2,\cdots,V_k\rangle$, 所有 $1\leqslant i\leqslant k$ 满足 $V_i\supseteq U_i$ 那么式子 $\Phi=\overline{<loc>}\Phi'$ 在该概要上成立;如果结点 $s$ 存在局部关系,它的所有局部前驱结点 $t(t\xrightarrow{loc}s)$ 所在概要 $\langle t,V_1,V_2,\cdots,V_k\rangle$, 所有 $1\leqslant i\leqslant k$ 满足 $V_i\supseteq U_i$ 那么式子 $\Phi=\overline{[loc]}\Phi'$ 在该概要上成立。

(9)如果结点 $s$ 存在调用关系,它的一个调用前驱结点 $t(t\xrightarrow{call}s)$ 所在概要满足 $\langle t,V_1,V_2,\cdots,V_n\rangle\in[\![\Phi']\!]_\varepsilon$, 且所有 $1\leqslant i\leqslant n$,$\{s'\colon <s',V_1\cap ME(s'),\cdots,V_n\cap ME(s')>\in[\![\varphi_i]\!]_\varepsilon\}\supseteq U_i$ 成立,那么式子 $\Phi=\overline{<call>}\Phi'\{\varphi_1,\varphi_2,\cdots,\varphi_k\}$ 在该概要上成立;如果结点 $s$ 存在调用关系,它的所有调用前驱结点 $t(t\xrightarrow{call}s)$ 所在概要都满足 $\langle t,V_1,V_2,\cdots,V_n\rangle\in[\![\Phi']\!]_\varepsilon$, 且所有 $1\leqslant i\leqslant n$,$\{s'\colon <s',V_1\cap ME(s'),\cdots,V_n\cap ME(s')>\in[\![\varphi_i]\!]_\varepsilon\}\supseteq U_i$ 成立,那么式子 $\Phi=\overline{[call]}\Phi'\{\varphi_1,\varphi_2,\cdots,\varphi_k\}$ 在该概要上成立。

(10)如果结点 $s$ 存在返回关系,它的一个前驱结点 $t(t\xrightarrow{ret}s)$ 所在的概要 $\langle t,V_1,V_2,\cdots V_k\rangle$ 满足 $[\![\Phi']\!]_\varepsilon$, 且 $s$ 所在概要满足 $\langle s,U_1,U_2,\cdots,U_k\rangle=\langle s\rangle$ 与 $s\in U_i$, 那么式子 $\Phi=\overline{\langle ret\rangle}\Phi'(M_i)$ 在该概要上成立;如果结点 $s$ 存在返回关系,它的所有前驱结点 $t(t\xrightarrow{ret}s)$ 所在的概要 $\langle t,V_1,V_2,\cdots,V_k\rangle$ 都满足 $[\![\Phi']\!]_\varepsilon$, 且 $s$ 所在概要满足 $\langle s,U_1,U_2,\cdots,U_k\rangle=\langle s\rangle$ 与 $s\in U_i$, 那么式子 $\Phi=\overline{[ret]}\Phi'(M_i)$ 在该概要上成立。

(11)如果概要集合 $S$ 满足 $[\![\Phi']\!]_{\varepsilon[x:=S]}\subseteq S$, 且该集合至少包含一个最小不动点,那么式子 $\Phi=\mu x.\Phi'$ 在该集合上成立。

(12)如果概要集合 $S$ 满足 $S\subseteq[\![\Phi']\!]_{\varepsilon[x:=S]}$, 该集合至少包含一个最大不动点,那么式子 $\Phi=\nu x.\Phi'$ 在该集合上成立。

## 3　嵌套树上的对等博弈

### 3.1　博弈、对等博弈

**定义 5**(博弈、博弈图):一个博弈 $G=(A,W_{in})$

包含博弈图和博弈赢的条件,其中 $W_{in} \subseteq V^w$,$w$ 为无限整数。博弈图是一个数组 $A = (V_0, V_1, E)$,$V_0$ 是 0 结点集合,$V_1$ 是 1 结点集合,$E \subseteq (V_0 \cup V_1) \times (V_0 \cup V_1)$ 是边关系即变迁关系集合。结点 $v \in V$ 的后续点集合为 $vE = \{v' \in V \mid (v,v') \in E\}$。选手 $\sigma(\sigma \in \{0,1\})$ 的对手为选手 $\overline{\sigma}(\overline{\sigma} = 1 - \sigma)$。

**定义 6**(博弈赢的条件):选手 $\sigma$ 在博弈 $G$ 中为博弈路线 $\pi$ 的赢家,当满足条件

(a) $\pi = v_0 v_1 \cdots v_l \in V^+$ 是有限博弈路线,$v_l$ 是一个 $\overline{\sigma}$ 结点且选手 $\overline{\sigma}$ 没法移动。

(b) $\pi$ 是无限博弈路线且 $\pi \in W_{in}$。对等博弈只与选手选择的策略有关,选手之间是平等的。需要扩展优先级功能,为每个结点制定优先级 $\Omega: V \to C$,$C$ 为优先级集合,为整数集合 $\mathbb{N}$ 的子集。最大对等赢的条件:如果无限出现结点集合的最大优先级为偶数,则 $\pi \in W_{in}$。最小对等赢的条件:如果无限出现的最小优先级为偶数,则 $\pi \in W_{in}$。

选手 0 的策略是一个功能 $f: V * V_0 \to V$,道路 $\pi$ 的任何前缀 $\pi = v_0 v_1 \cdots v_n, v_n \in V_0$ 则决定下一步移动点 $v_n E v_{n+1}$。选手 0 的策略是决定 $\pi$ 中结点集合 $V_\pi$ 与 $V_0$ 的交集中的结点下一步的移动。无记忆的策略只与当前结点相关 $f: V_0 \to V$。如果它从结点 $v_0$ 开始的每条道路遵从这个策略都能赢,则称选手 0 从结点 $v_0$ 出发赢得博弈。对于选手 0,能存在赢的博弈策略的开始结点集合称为选手 0 赢的区域。

### 3.2 基于嵌套树的自动机

嵌套树自动机结点的状态只与结点的前驱结点、结点的跳跃前驱有关。构建自动机必须要考虑非确定、交互性质。自动机的接收条件用对等条件,采用对等条件能更好地考虑无限路线的接收。

#### 3.2.1 嵌套树自动机的定义

**定义 7** 非确定对等嵌套树自动机(nondeterministic parity nested tree automaton,NPNTA):基于语言集合 $\Sigma$ 的非确定对等嵌套树自动机 $A = (Q, q_0, \Delta, \Omega)$,其中 $Q$ 是有限状态集合,$q_0 \in Q$ 是起始状态,变迁关系 $\Delta \subseteq Q \times \Sigma \times (TT \times TT)$ 且 $TT = Q \cup (Q \times Q) \cup \{\perp\}$,$\Omega$ 为对等接收条件映射每个状态到优先级,为 $\Omega: Q \to \{0,1\cdots,n\}$。

NPNTA 接收基于有序嵌套树上的语言,必须在嵌套树结点与自动机状态之间建立映射关系为 $\rho: S \to Q$。在自动机上的执行路线为

(a) $\rho(s_1) = q_0$,嵌套树的根部结点映射为自动机的起始状态。

(b) $\rho(s) = q, \lambda(s) = \sigma$。如果结点 s 有左右子结点非空,自动机变迁关系 $(q,\sigma,(\tau_1,\tau_2)) \in \Delta$。若子结点为调用或局部结点,则 $\tau_i = \rho(s_i)$;若子结点为返回结点且调用结点为 t,则 $\tau_i = (\rho(t), \rho(s_i))$。

(c) $\rho(s) = q, \lambda(s) = \sigma$。如果结点 s 有左右子结点其中一边为空,空用 $\perp$ 表示。$(q,\sigma,(\tau', \perp)) \in \Delta$ 或 $(q,\sigma,(\perp,\tau')) \in \Delta$。若子结点 $s'$ 为调用或局部结点,则 $\tau' = \rho(s')$;若子结点 $s'$ 为返回结点且调用结点为 t,则 $\tau' = (\rho(t), \rho(s'))$。

**定义 8** 交替对等嵌套树自动机(alternate parity nested tree automaton,APNTA):交替对等嵌套树自动机 $A = (Q, q_0, \Delta, \Omega, \sum)$,它考虑嵌套树的语言集合,其中 $Q$ 是有限状态集合,$q_0 \in Q$ 是起始状态,变迁功能 $\Delta: Q \times \Sigma \to TT(Q)$,其中变迁条件 $TT(Q)$ 包含变迁形式 $f: = tt \mid ff \mid p \mid \neg p \mid X \mid f \vee f \mid f \wedge f \mid \mu x.q \mid \nu x.q \mid [loc]q \mid \langle loc\rangle q \mid [call]q \mid \langle call\rangle q \mid [ret,q']q \mid \langle ret,q'\rangle q, q, q' \in Q$,$\Omega$ 为在等接收条件下,每个状态的优先级映射关系 $\Omega: Q \to \{0,1\cdots, n\}$。

在 APNTA 中,从状态 $q$ 到状态 $q'$ 存在变迁关系,当 $q'$ 出现在 $\Delta(q,\sigma)$ 中,$\Delta(q,\sigma)$ 描述了要验证的性质表达式的拆分变迁关系,即 APNTA 是验证性质公式 $\Phi$ 的子公式的等价拆分表达。$q$ 为状态机中状态描述了相应公式所表示的状态,$\sigma$ 为状态 $q$ 对应嵌套树中的标签,且公式所含自由变量集合 $free(\Phi)$。各种形式的 $NT-\mu$ 公式的变迁关系为($q$ 的下标为变迁的形式)

$$\Delta(q_{tt},\sigma) = 1,\ \Delta(q_{ff},\sigma) = 0 \tag{4}$$

当 $(p = \sigma$ 时 $\Delta(q_p,\sigma) = 1$,否则 $\Delta(q_p,\sigma) = 0$;$p = \sigma$ 时 $\Delta(q_{\neg p},\sigma) = 0$,否则 $\Delta(q_{\neg p},\sigma) = 1)$

$$\Delta(q_X) = \begin{cases} \{q_X,\ X \in free(\Phi) \\ q_{\Phi'},\ X \notin free(\Phi), \\ \Phi' \text{ 为包含 } X \text{ 且 } X \text{ 在其中不为自由变量} \end{cases} \tag{5}$$

$$\Delta(q_{\Phi_1\vee\Phi_2},\sigma) = \Delta(q_{\Phi_1},\sigma) \vee \Delta(q_{\Phi_2},\sigma) \quad (6)$$

$$\Delta(q_{\Phi_1\wedge\Phi_2},\sigma) = \Delta(q_{\Phi_1},\sigma) \wedge \Delta(q_{\Phi_2},\sigma) \quad (7)$$

$$\Delta(q_{\mu x.\Phi'},\sigma) = \Delta(q_{\Phi'},\sigma) \quad (8)$$

$$\Delta(q_{\nu x.\Phi'},\sigma) = \Delta(q_{\Phi'},\sigma) \quad (9)$$

$$\Delta(q_{\langle call\rangle\Phi'|\varphi_1,\varphi_2,\cdots,\varphi_k\}},\sigma) = \Delta([call]q_{\Phi'},\sigma') \quad (10)$$

$$\Delta(q_{[call]\Phi'|\varphi_1,\varphi_2,\cdots,\varphi_k\}},\sigma) = \Delta([call]q_{\Phi'},\sigma') \quad (11)$$

$$\Delta(q_{\langle loc\rangle\Phi'},\sigma) = \Delta(\langle loc\rangle q_{\Phi'},\sigma') \quad (12)$$

$$\Delta(q_{[loc]\Phi'},\sigma) = \Delta([loc]q_{\Phi'},\sigma') \quad (13)$$

$$\Delta(q_{[ret]R_i},\sigma) = \bigvee_{\Phi',\varphi_1\leqslant i\leqslant j\leqslant k}(\Delta(\langle ret,q_{\langle call\rangle\Phi'|\varphi_1,\cdots,\varphi_j\}}\rangle q_{\varphi_i},\sigma') \vee \Delta(\langle ret,q_{[call]\Phi'|\varphi_1,\cdots,\varphi_j\}}\rangle q_{\varphi_i},\sigma'')) \quad (14)$$

$$\Delta(q_{[ret]R_i},\sigma) = \bigvee_{\Phi',\varphi_1\leqslant i\leqslant j\leqslant k}\Delta([ret,q_{\langle call\rangle\Phi'|\varphi_1,\cdots,\varphi_j\}}]q_{\varphi_i},\sigma') \vee \Delta([ret,q_{[call]\Phi'|\varphi_1,\cdots,\varphi_j\}}]q_{\varphi_i},\sigma'') \quad (15)$$

$\Delta(q,\sigma)$ 当存在 $\langle\beta\rangle$、$[\beta]$ 形式时,对应嵌套树存在变迁关系,因此对应的结点和标签发生变化,标签变为 $\sigma'$。当前要验证的公式为变量时,该变量为自由变量(判断的自由变量集合为要验证的总公式的自由变量集合)时为本身的状态变量且变迁截止,不为自由变量时为某一公式的限定变量,则转到该公式表示的状态。因为根据不动点的定义限定变量与包含限定变量的公式、限定变量限定的子公式三者相等,在子公式的拆分变迁关系上,包含限定变量的公式的状态为源状态,限定变量限定的子公式为目标状态,同时限定变量的状态为源状态,包含限定变量的公式的状态为目标状态。调用公式 $q_{\langle call\rangle\Phi'|\varphi_1,\varphi_2,\cdots,\varphi_k\}}$ 或 $q_{[call]\Phi'|\varphi_1,\varphi_2,\cdots,\varphi_k\}}$ 的状态时,状态下一步调用的子公式状态 $\langle call\rangle q_{\Phi'}$ 或 $[call]q_{\Phi'}$。返回公式时,调用结点状态需要列举有可能的调用公式状态。变迁过程是公式不断拆分的过程,变迁截止时状态表示的公式为不可拆分的原子公式为自由变量公式。

### 3.2.2　APNTA 的接收条件

基于要验证的公式 $\Phi_A$ 建立交替对等嵌套树自动机(APNTA),APNTA 中状态的数量取决于 $\Phi_A$ 子公式的不停拆分的次数。考虑 $\Phi_A$ 不停拆分的过程,被成功接收当且仅当最后产生的所有状态被成功接收。不同形式的 $NT-\mu$ 公式拆分为

$$CF_{q,q'}(tt) = tt \quad (16)$$

$$CF_{q,q'}(ff) = ff \quad (17)$$

$$CF_{q,q'}(f_1 \vee f_2) = CF_{q,q'}(f_1) \vee CF_{q,q'}(f_2) \quad (18)$$

$$CF_{q,q'}(f_1 \wedge f_2) = CF_{q,q'}(f_1) \wedge CF_{q,q'}(f_2) \quad (19)$$

$$CF_{q,q'}(\langle call\rangle q_1) = \langle call\rangle X_{q_1,q}\{X_{q_1,q'},\cdots,X_{q_n,q'}\} \quad (20)$$

$$CF_{q,q'}([call]q_1) = [call]X_{q_1,q}\{X_{q_1,q'},\cdots,X_{q_n,q'}\} \quad (21)$$

$$CF_{q,q'}(\langle loc\rangle q_1) = \langle loc\rangle X_{q_1,q} \quad (22)$$

$$CF_{q,q'}([loc]q_1) = [loc]X_{q_1,q} \quad (23)$$

$$CF_{q,q'}(\langle ret,q\rangle q_1) = \langle ret\rangle R_{q_1} \quad (24)$$

最大最小不动点的情况属于无限道路的情况,因此暂时不考虑。$CF_{q,q'}$ 表示公式的拆分,下标为状态的源状态与目标状态,公式被自动机接收拆分的每个部分成功,不停递归直至拆分成原子公式。下标 $q$ 表示当前状态,$q'$ 表示向前遍历经过的第一个调用结点的状态。$X_{q,q'}$ 下标的表示同 $CF_{q,q'}$,表示所需要公式集合。例如 $CF_{q,q'}(\langle call\rangle q_1) = \langle call\rangle X_{q_1,q}\{X_{q_1,q'},\cdots,X_{q_n,q'}\}$,$X_{q_1,q}$ 表示调用的公式集合,$\langle call\rangle$ 要求调用的一个公式成功,$[call]$ 要求所有调用的公式成功,$\{X_{q_1,q'},\cdots,X_{q_n,q'}\}$ 要求调用后返回执行的公式成功。

### 3.3　嵌套树的对等博弈图的形成

设 $B=(Q,q_0,\Delta,\Omega)$ 为一个交替嵌套树自动机,当选手 0 在相应的博弈 $G=(A,W_{in})$ 中存在赢的策略,即该自动机能接收嵌套树结构 $(T,s_I)$。接收策略 $W_{in}$ 使用对等博弈的接收条件,具体为无限博弈路线用最大对等接收条件,即对手 0 若在道路中无限出现的最大优先级为偶数则赢得博弈。博弈图 $A=(V_0,V_1,E)$ 中结点形式为 $MV=T\times Q\times Q^*\times TT$,设博弈图中结点 $v\in MV$ 的形式为 $(s,q,q',f)$,其中 $s\in T$ 为嵌套树中的一个结点,对应的 APNTA 中状态结点 $\rho(s)=q$,$q'$ 为向后变迁第一次遇到的匹配调用结点的状态,$f$ 为当前状态 $q$ 对应

的 $NT-\mu$ 公式。博弈 $G=(A,W_{in})$ 以嵌套树和要验证的性质公式的 APNTA 为基础，用对等条件来判断选手的输赢，以此来判断性质在嵌套树中是否被满足。在博弈图 $G(A)$ 中移动的变迁关系如下：

(1) 对形式如 $(s,q,q',f_1 \vee f_2)$ 或 $(s,q,q',f_1 \wedge f_2)$ 的结点，下一步的移动变迁到 $(s,q,q',f_1)$ 和 $(s,q,q',f_2)$。

(2) 对形式如 $(s,q,q',\langle loc\rangle q_1)$ 或 $(s,q,q',[loc]q_1)$ 的结点且结点 $s$ 与结点 $t$ 存在局部关系，下一步的移动变迁到所有 $(t,q^t,q',\Delta(q^t,\lambda(t)))$。

(3) 对形式如 $(s,q,q',\langle call\rangle q_1)$ 或 $(s,q,q',[call]q_1)$ 的结点且结点 $s$ 与结点 $t$ 存在调用关系，下一步的移动变迁到所有 $(t,q^t,q,\Delta(q^t,\lambda(t)))$。

(4) 对形式如 $(s,q,q',\langle ret,q'\rangle q_1)$ 或 $(s,q,q',[ret,q']q_1)$ 的结点且结点 $s$ 与结点 $t$ 存在返回关系，下一步的移动变迁到所有 $(t,q_1,q',\Delta(q_1,\lambda(t)))$。

可以看出，$f$ 中存在变迁时，$s$、$q$、$q'$ 相应地发生变化。博弈图的变迁关系是结合了 APNTA 的变迁关系和嵌套树的变迁关系。将所有可能的变迁关系列出，在道路中判断语言接受性。

### 3.4 对等博弈图中结点的分类

博弈图中结点集合的关系为 $V_0 \cup V_1 = V$、$V_0 \cap V_1 = \varnothing$，$V_0$ 为选手 0 的移动结点集合，$V_1$ 为选手 1 的移动结点集合。对于博弈图中结点 $(s,q,q',f)$ 的分类，不考虑 $s$、$q$、$q'$ 只通过 $f$ 来区分结点是属于结点集合 $V_0$ 还是集合 $V_1$，具体分类如下：

结点 $(s,q,q',f) \in V_0$ 当 $f$ 满足以下形式：$f=ff \mid f=p$ 且 $p \neq \sigma \mid f=\neg p$ 且 $p=\sigma \mid f=X$ 且 $X \notin free(\Phi) \mid f=f_1 \vee f_2 \mid f=\mu x.q_1 \mid f=\nu x.q_1 \mid \langle loc\rangle q_1 \mid \langle call\rangle q_1 \mid \langle ret,q'\rangle q_1$。

结点 $(s,q,q',f) \in V_1$ 当 $f$ 满足以下形式：$f=tt \mid f=p$ 且 $p=\sigma \mid f=\neg p$ 且 $p \neq \sigma \mid f=X$ 且 $X \in free(\Phi) \mid f=q_1 \wedge q_2 \mid [loc]q_1 \mid [call]q_1 \mid [ret,q']q_1$。

博弈的输赢要观察道路的最终结点，例如如果最终结点属于 $V_1$ 则判断选手 0 的输赢，反之亦然。$f=f_1 \vee f_2$ 在博弈图中是拆分成两个子公式，判断时选择任意一条子公式道路 $f$ 都为真，同理 $f=f_1 \wedge f_2$ 选择任意一条子公式道路 $f$ 都为假。包含 $\langle\beta\rangle$、$[\beta]$ 形式的公式与上述原理相同，$\langle\beta\rangle$ 选择任一条道路公式都为真，$[\beta]$ 选择任一条道路公式都为假。

### 3.5 博弈图中结点优先级的确定

#### 3.5.1 公式的交替深度

判断结点的优先级，先研究公式中包含最大最小不动点公式的交替性，即最大不动点与最小不动点交替的个数，用 $\alpha(\Phi)$ 来表示公式的交替深度，具体定义为：

$$\alpha(tt)=\alpha(ff)=\alpha(X)=\alpha(p)=\alpha(\neg p)=0 \tag{25}$$

$$\alpha(\Phi_1 \vee \Phi_2)=\alpha(\Phi_1 \wedge \Phi_2)=\max(\alpha(\Phi_1),\alpha(\Phi_2)) \tag{26}$$

$$\alpha(\langle . \rangle \Phi)=\alpha([.]\Phi)=\alpha(\Phi) \tag{27}$$

$$\alpha(\mu x.\Phi)=\max\{1,\Phi\} \cup \{\alpha(\nu x'.\Phi')+1 \mid \nu x'.\ \Phi' \text{ 是 } \Phi \text{ 的子公式}, x \in free(\nu x'.\Phi')\} \tag{28}$$

$$\alpha(\nu x.\Phi)=\max\{1,\Phi\} \cup \{\alpha(\mu x'.\Phi')+1 \mid \mu x'.\ \Phi' \text{ 是 } \Phi \text{ 的子公式}, x \in free(\mu x'.\Phi')\} \tag{29}$$

最小最大不动点公式的交替深度计算符号 $\cup$ 表示将情况分类，当公式中不含不同不动点的子公式或包含但公式中的限制变量不是该子公式中的自由变量，则交替深度取 1 和将不动点符号去除的子公式的交替深度两者之间较大的那个值，否则则计算同不动点的子公式的交替深度并加 1。例如公式 $\Phi=\nu x.(\mu x.(x \vee x') \wedge [loc]x)$，公式中包含不动点子公式为 $\Phi'=\mu x.(x \vee x')$，两者的限制变量相同，即公式中限制变量不是子公式的自由变量，因此 $\alpha(\Phi)=\max\{1,\alpha\{\mu x.(x \vee x') \wedge [loc]x\}\}=1$。从例子可以看出交替深度计算不同不动点嵌套的情况，且限制自由变量要不相同。

#### 3.5.2 公式优先级的规定

优先级决定博弈接收的条件，考虑优先级映射 $\Omega: Q \to \{0,1,\cdots,n\}$，优先级与当前状态的公式相关，在博弈图中结点 $v=(s,q,q',f)$ 对各种公式的优先级进行规定，具体情况如下：

(1) $f=X, X \in free(\Phi)$，则 $\Omega(v)=1,(v,v)$

$\in E$。

(2) $f = X, X \notin free(\Phi)$, 则 $\Omega(v) = 1, (v,v) \in E$。

(3) $f = tt$ 则 $\Omega(v) = 2$。

(4) $f = ff$ 则 $\Omega(v) = 1$。

(5) $f = p$ 且 $p = \sigma$ 或 $f = \neg p$ 且 $p \neq \sigma$, 则 $\Omega(v) = 2$, $(v,v) \in E$。

(6) $f = p$ 且 $p \neq \sigma$ 或 $f = \neg p$ 且 $p = \sigma$, 则 $\Omega(v) = 1$, $(v,v) \in E$。

(7) $f = f_1 \vee f_2$ 或 $f = f_1 \wedge f_2$, 则 $\Omega(v) = 1$。

(8) $f = \langle loc \rangle q_1 \mid \langle call \rangle q_1 \mid \langle ret, q' \rangle q_1$ 且 $\{t \mid s \xrightarrow{\beta} t\} = \varnothing$( $\beta$ 为对应的 loc、call、ret), 则 $\Omega(v) = 1$, $(v,v) \in E$。

(9) $f = \langle loc \rangle q_1 \mid \langle call \rangle q_1 \mid \langle ret, q' \rangle q_1$ 且 $\{t \mid s \xrightarrow{\beta} t\} \neq \varnothing$( $\beta$ 为对应的 loc、call、ret), 则 $\Omega(v) = 1$。

(10) $f = [loc] q_1 \mid [call] q_1 \mid [ret, q'] q_1$ 且 $\{t \mid s \xrightarrow{\beta} t\} = \varnothing$( $\beta$ 为对应的 loc、call、ret), 则 $\Omega(v) = 2$, $(v,v) \in E$。

(11) $f = [loc] q_1 \mid [call] q_1 \mid [ret, q'] q_1$ 且 $\{t \mid s \xrightarrow{\beta} t\} \neq \varnothing$( $\beta$ 为对应的 loc、call、ret), 则 $\Omega(v) = 1$。

(12) $f = \mu x.\, q_1$, 则 $\Omega(v) = \begin{cases} \alpha(f) + 2, & \alpha(f) \text{ 为奇数} \\ \alpha(f) + 1, & \alpha(f) \text{ 为偶数} \end{cases}$

(13) $f = \nu x.\, q_1$, 则 $\Omega(v) = \begin{cases} \alpha(f) + 2, & \alpha(f) \text{ 为偶数} \\ \alpha(f) + 1, & \alpha(f) \text{ 为奇数} \end{cases}$

图 1 和图 2 分别给出了最小不动点的和最大不动点的性质表示。不动点公式的情况从图 1 可以看出,最小不动点一般表示可达性,在于验证道路中某一结点是否满足某一子性质 P。最大不动点表示性质的持续性,将优先级设置成无限偶数,希望性质能

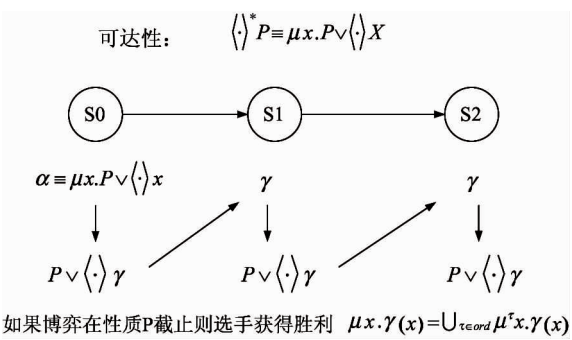


**图 1　最小不动点的性质表示**

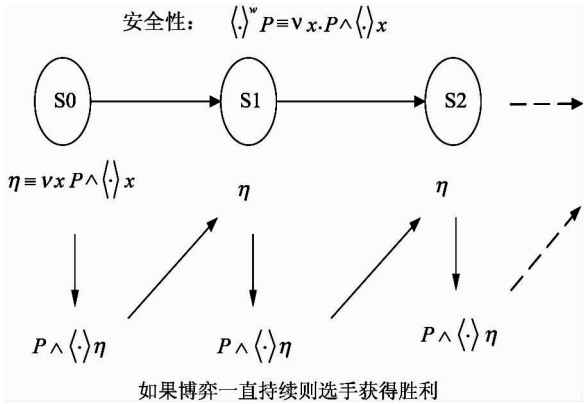


**图 2　最大不动点的性质表示**

一直得到满足。如果在某一结点上无法继续拆分,判断被中止,在该结点上增加边 $(v, v) \in E$ 使之变成自循环无限道路,使其在对等条件下更好地判断。

### 3.6　博弈赢的策略

博弈图也可以扩展成 $G = (V = V_0 \cup V_1, E, \Omega: V \to \{1, \cdots, n\})$ 表示。$G$ 转换成 Kripke 结构为 $G^K = (V, E, \{P_0^G, 1^G, \cdots, n^G\})$, 其中 $V$ 是结点集合与博弈图中结点相对应, $E$ 为边集合, $P_0^G$ 为选手 0 的结点集合, $1^G, \cdots, n^G$ 表示具有不同优先级的结点集合 ($i^G = \{v: \Omega(v) = i\}$)。如果从结点 $v_0$ 开始的每条道路某条策略都能赢,则称选手 0 从结点 $v_0$ 出发可以赢得博弈。选手 0 出发能存在赢的博弈策略的开始结点的集合称为选手 0 赢的区域。研究选手 0 赢的区域,设最大优先级 $n$ 为偶数,则有下列公式:

$$\varphi_0(Z_1, \cdots, Z_n) = (P_0 \Rightarrow \bigwedge_{i \in \{1, \cdots, n\}} (i \Rightarrow \langle\rangle Z_i)) \wedge (\neg P_0 \Rightarrow \bigwedge_{i \in \{1, \cdots, n\}} (i \Rightarrow [\,] Z_i)) \tag{30}$$

$$W_0 = \| \mu Z_1 . \nu Z_2 . \cdots \mu Z_{n-1} . \nu Z_n . \varphi_0(Z_1, \cdots, Z_n) \|^G \tag{31}$$

如果当前结点属于选手 0,则有一个后续结点的优先级与之相同,如果当前结点属于选手 1,则所有后续结点的优先级与之相同。奇数优先级被 $\mu$ 所限定表示有限循环,偶数优先级被 $\nu$ 所限定表示无限循环。

## 4　对概要标签树的思考

### 4.1　嵌套树转换概要标签树

嵌套树以结点为单位,如果程序为递归程序,则

嵌套树是无限的。考虑以概要为基本单位的标签树,并将除嵌套树中的跳跃边去除,转换过程写作:$NT \times \mathbb{N} \to T^L$。嵌套树与整数 k 映射成一个 k 种颜色概要的标签树,这里嵌套树定义为 $T = (S, r, \to, \mapsto, \lambda)$,其中结点标签集合为 $2^{AP}$,对应的 $k$ 种颜色概要的标签树定义为 $T^L = Summarize(T, k)$。对 $T^L$ 来说,概要的颜色数是固定的,结点的标签集合为 $AP^L = AP \cup \{leaf_i : 1 \leqslant i \leqslant k\}$,边的标签集合为 $L = \{call, ret, loc\} \cup \{cp, in\} \cup \{i : 1 \leqslant i \leqslant k\}$,结点集合为 $N = Summ_k^s \cup \{s' \mid s \xrightarrow{call} s'\} \cup \{(V_1, \cdots, V_k), 1 \leqslant j \leqslant k, V_j \subseteq ME(s')\}$($Summ_k^s$ 为在嵌套树中结点集合为 $s$ 下 $K$ 种颜色的概要集合)。标签树的根节点为 $n_0 = \langle s_0, \varnothing, \cdots, \varnothing \rangle$,边标签映射为 $\eta_T : E \to L$,结点标签映射为 $\lambda_T : N \to AP^L$。根据变迁关系的不同,嵌套树转换成 $k$ 种颜色概要的标签树过程如下:

(1) 如果 $s \xrightarrow{loc} s'$ 且 $\lambda_T(S) \neq leaf_i$,在 $T^L$ 上增加边 $S \xrightarrow{loc} S'$,其中 $S = \langle s, U_1, \cdots, U_k \rangle$,$S' = \langle s', U'_1, \cdots, U'_k \rangle$ 中 $U'_i = U_j \cap ME(s')$。

(2) 如果 $s \xrightarrow{call} s'$ 且 $\lambda_T(S) \neq leaf_i$,则对 $T^L$ 处理步骤如下:

(a) 在 $T^L$ 上增加边 $S \xrightarrow{call} s'$,结点的标签为 $\varnothing$。

(b) 在 $T^L$ 上增加边 $s' \xrightarrow{cp} \langle V_1, \cdots, V_k \rangle$ 且 $V_1, \cdots, V_k \subseteq ME(s')$,结点的标签为 $\varnothing$。

(c) 在 $T^L$ 上增加边 $\langle V_1, \cdots, V_k \rangle \xrightarrow{in} \langle s', V_1, \cdots, V_k \rangle$,对所有结点 $t \in V_j$,增加边 $\langle V_1, \cdots, V_k \rangle \xrightarrow{j} \langle t, U'_1, \cdots, U'_k \rangle$,其中 $U'_i = U_i \cap ME(t)$。

(3) 如果 $s \xrightarrow{ret} s'$ 且 $\lambda_T(S) \neq leaf_i$,在 $T^L$ 上增加边 $S \xrightarrow{ret} S'$,其中 $S = \langle s, U_1, \cdots, U_k \rangle$、$S' = \langle s', U_1, \cdots, U_k \rangle$,$S'$ 为 *leaf* 结点。

分析情况(2)总的过程为 $S \xrightarrow{call} s' \xrightarrow{cp} \langle V_1, \cdots, V_k \rangle \begin{cases} \xrightarrow{in} \langle s', V_1, \cdots, V_k \rangle \\ \xrightarrow{1} \langle t_1 \in V_1, U'_1, \cdots, U'_k \rangle \\ \cdots \\ \xrightarrow{j} \langle t_j \in V_j, U'_1, \cdots, U'_k \rangle \end{cases}$,大括号前面为调用,大括号后面为返回要执行的内容。$S \xrightarrow{call} s' \xrightarrow{cp} \langle V_1, \cdots, V_k \rangle$ 和 $\langle V_1, \cdots, V_k \rangle \xrightarrow{in} \langle s', V_1, \cdots, V_k \rangle$ 表示调用内容就必须在 $T^L$ 中表示出来,结点 $s'$ 作为调用内容的开始引出调用后面的内容。返回结点的 *leaf* 结点表示当前调用内容的结束,$leaf_i$ 是返回内容的开始对应于边 $\xrightarrow{j}$。$T^L$ 的表示相比嵌套树 $T$ 来说,采用概要作为结点的单位,以增加边的数量为代价来去除跳跃边。

### 4.2 概要标签树下公式的表示与拆分

概要标签树 $T^L$ 的公式有如下形式:

$$\begin{aligned} \phi :: = & \ tt \mid ff \mid p \mid \neg p \mid X \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mu x.\phi \mid \nu x.\phi \mid \langle loc \rangle \phi \mid [loc] \phi \\ & \left| \begin{array}{l} \langle call \rangle (cp) \langle \{in\} \varphi \wedge \{1\} \varphi_1 \wedge \cdots \wedge \{k\} \varphi_k \rangle \mid \\ [call] (cp) \langle \{in\} \varphi \wedge \{1\} \varphi_1 \wedge \cdots \wedge \{k\} \varphi_k \rangle \\ \mid \langle ret \rangle leaf_i \mid [ret] leaf_i \end{array} \right. \end{aligned}$$

相对地,$T^L$ 不同子公式的拆分形式为($CF^L$ 表示在概要标签树 $T^L$ 中对其中表示的公式进行拆分)

$$CF^L(p) : CF^L(p) = p \tag{32}$$

$$CF^L(\neg p) = \neg p \tag{33}$$

$$CF^L(\langle loc \rangle \phi) = \langle loc \rangle CF^L(\phi) \tag{34}$$

$$CF^L([loc] \phi) = [loc] CF^L(\phi) \tag{35}$$

$$CF^L(X) = \begin{cases} X, & X \in free(\phi) \\ \phi', & X \notin free(\phi), \\ & \phi' \text{ 为包含 } X \text{ 且 } X \text{ 在其中不为自由变量} \end{cases} \tag{36}$$

$$CF^L(\langle call \rangle (cp) \langle \{in\} \varphi \wedge \{1\} \varphi_1 \wedge \cdots \wedge \{k\} \varphi_k \rangle) = \langle call \rangle (CF^L(\varphi)) \{CF^L(\varphi_1), \cdots, CF^L(\varphi_k)\} \tag{37}$$

$$CF^L([call] (cp) \langle \{in\} \varphi \wedge \{1\} \varphi_1 \wedge \cdots \wedge \{k\} \varphi_k \rangle) = [call] (CF^L(\varphi)) \{CF^L(\varphi_1), \cdots, CF^L(\varphi_k)\} \tag{38}$$

$$CF^L(\langle ret \rangle leaf_i) = \langle ret \rangle R_i \tag{39}$$

$$CF^L([ret] leaf_i) = [ret] R_i \tag{40}$$

$$CF^L(\mu x.\phi) = \mu x.\, CF^L(\phi) \tag{41}$$

$$CF^L(\nu x.\phi) = \nu x.\, CF^L(\phi) \tag{42}$$

可以看出,基于 $T^L$ 的公式拆分类似于前文的公式拆分方法。

### 4.3　概要标签树上的博弈

概要的标签树的结点映射 $\lambda_T: N \rightarrow AP^L$,需要将 $T^L$ 中结点映射为标签。如果该结点为嵌套树 $T$ 中结点,则它的结点标签与嵌套树中该结点标签相同即 $\lambda_T(s) = \lambda(s)$;如果结点为概要结点,则它的标签与概要的根结点的标签相同即 $\lambda_T(S) = \lambda(s)$;如果结点为 $\langle V_1, \cdots, V_k \rangle$ 形式,则它的标签为空集即 $\lambda_T(\langle V_1, \cdots, V_k \rangle) = \varnothing$;如果结点为 leaf 结点 $S' = \langle s', U_1, \cdots, U_k \rangle$,它的标签定义为 *leaf* 结点的标签即 $\lambda_T(S') = \{leaf_i : s' \in U_j\}$。

对于概要标签树的交替对等嵌套树自动机(APNTA) $A = (Q, q_0, \Delta, \Omega)$,自动机变迁功能 $\Delta: Q \times \Sigma \rightarrow TT^L(Q)$ 与嵌套树的 APNTA 形式相似,但是变迁条件 $TT^L(Q)$ 改为 $p \mid \neg p \mid \langle loc \rangle q \mid [loc] q \mid \langle call \rangle (cp) \langle \{in\} q \wedge \{1\} q_1 \wedge \cdots \wedge \{k\} q_k \rangle \mid [call] (cp) \langle \{in\} q \wedge \{1\} q_1 \wedge \cdots \wedge \{k\} q_k \rangle \mid \langle ret \rangle leaf_i \mid [ret] leaf_i$。概要标签树博弈图的形成、博弈图中结点的分类、优先级的定义等与嵌套树相似。

## 5　结　论

本文将软件系统用嵌套树来表示,为了精化逻辑表示及更好地掌握程序的调用返回关系,定义了一种基于嵌套树的基本单位—概要。接着,以概要为基本单位定义了基于嵌套树的 $\mu$-演算($NT$-$\mu$),使之能根据程序执行的功能特征来更好地描述各种需求。为了表示程序验证的过程,将 $NT$-$\mu$ 逻辑公式直接转化为交替对等嵌套树自动机 APNTA,再根据公式的不同形式定义自动机的接收条件。APNTA 与嵌套树相结合,可以判断该需求是否被满足。但此验证过程过于繁琐,且主要依赖人工判断,效率相对低下,因此,在上述理论基础上,引进对等博弈来自动化判断的过程。在嵌套树与 APNTA 上形成对嵌套树博弈图及在其中定义各要素,用对等条件来判断输赢,判断的过程与验证的过程等价。最后,研究嵌套树的另一种更为直观的形式—概要标签树,它以增加边为代价去除了跳跃边,使树的表示方式更为简洁。通过分析证明嵌套树的相关理论同样适用于概要标签树。下一步的研究继续挖掘概要标签树与嵌套树,考虑不同程序以两种结构为基础而形成的博弈图的联系与区别,并试图对整个验证过程进一步地自动化。

**参考文献**


[1] Clarke E, Grumberg O, Peled D. Model checking. Cambridge: MIT Press, 1999. 1-20

[2] Sharir M, Pnueli A. Program Flow Analysis: Theory and Applications. Upper Saddle River /USA: Prentice Hall Professional Technical Reference, 1981. 15-40

[3] Jensen T, Metayer L, Thorn T. Verification of control flow based security properties. In: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, USA, 1999. 89-103

[4] Nelson G, Oppen D. Fast decision procedures based on congruence closure. *Journal of the ACM*, 1980, 27(2): 356-364

[5] Shostak R. Deciding combinations of theories. *Journal of the ACM*, 1984, 31(1): 1-12

[6] Nicholas K, Peter L, Tayssir T. A decision procedure for detecting atomicity violations for communicating processes with locks. *Lecture Notes in Computer Science*, 2009, 5578: 125-142

[7] Lim J, Lal A, Reps T. Symbolic analysis via semantic reinterpretation. In: Proceedings of 16th International SPIN Workshop, Model Checking Software, Grenoble, France, 2009. 61-87

[8] Schmer S, Vogel M, Konig H. Using model checking to identify errors in intrusion detection signatures. *International Journal on Software Tools for Technology Transfer*, 2011, 13(1): 89-106

[9] Alur R, Chaudhuri S, Madhusudan P. Languages of nested trees. In: Proceedings of the Symposium on Computer-Aided Verification, USA, 2006. 329-342

[10] Esparza J, Kucera A, Schwoon S. Model-checking LIL with regular valuations for pushdown systems. *Information and Computation*, 2003, 186(2): 355-376

[11] Kupferman O, Vardi M, Wolper P. An automata-theoretic approach to branching-time model checking. *Journal of The ACM*, 2000, 47(2): 312-360

[12] Bradfield J. The modal μ-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 1998, 195: 133-153

[13] David E, Paul E. Alternating automata on infinite trees.

*Theoretical Computer Science*, 1987, 54(2-3):267-276

[14] Igor W. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 2002, 275(1-2): 311-346

[15] Kairong Q, Albert N. Language-emptiness checking of alternating tree automata using symbolic reachability analysis. *Electronic Notes in Theoretical Computer Science*, 2006, 149(2):33-49

[16] Thomas W. Alternating tree automata, parity games, and modal $\mu$-Calculus. *Bulletin of the Belgian Math*, 2001, 8(2):359-391

# Study of the parity games application based on nested tree

Guo Jing, Xu Zhongwei

(School of Electronics & Information Engineering, Tongji University, Shanghai 201804)

**Abstract**

The nature verification of software systems was studied. Firstly, a method for program representation based on a nested tree was proposed to tackle the abstract representation problem of a software system. The method can indicate the program sequence structure, and can better represent the relationship between call and return. Then, the $\mu$-calculus of the nested tree was defined to formulize the requirement nature to be verified and to transfer the formula into a nondeterministic parity nested tree automation. Finally, the automation was combined with the nested tree to form a game graph, and the parity game conditions were used to determine the game's winners and losers, which is equivalent to testing whether the validation formula is satisfied in the nested tree. Compared with the direct verification, the above-stated method is more audio-visual and more beneficial to process automation. The study shows that the call-return relation can be expanded to form a summary tag tree, and the parity game theory can be well applied to the summary tag tree.

**Key words**: parity games, nested tree, alternating tree automata, $\mu$-calculus, summary