基于 T³BDD 的动态模型检查*

倪 彬 冯玉琳 黄 涛

(中国科学院软件研究所计算机科学开放研究实验室 北京 100080)

E-mail: tao@ox.jos.ac.cn

摘要 Java Beans 是一种组件标准.该文定义了 JBDL(Java Beans description language)语言,用于描述组件语义约束规范.为了检测 Java Beans 组件语义约束与其实现之间的一致性,文章给出了一种基于 JBDL 公式的三值语义和模型的抽象化动态模型检查方法.文章重点介绍了利用 T³BDD (3-terminal binary decision diagram) 的符号化动态模型检查方法.

关 键 词 组件,Java Beans,形式规范,符号化,模型检查,二叉判定图.

中图法分类号 TP311

组合软件开发方法和对象技术正在成为支持网络计算模型的一种主流方法和技术.可复用的组件是组合软件开发方法的基本单元和核心.组件的规范通常依据一定的标准,提供支持组件构造的系统结构和 API.但是,这些标准目前都采用非形式的方法来刻画组件的语义.高质量的组件是支持组件复用和分布式计算的关键.为了开发高质量的组件,我们希望对组件的语义形式化地进行描述和验证.在很多情况下,这样的描述和验证是必要的.在组件的制造过程中,往往需要进行多层次的分析、设计和实现.组件语义精确和严格的形式描述,使各层次开发人员对组件语义不会产生理解上的歧义.对组件的验证有利于保证组件的正确性.这样的描述和验证也是在开发过程中选择组件、评价组件、调整组件、组合构造以及系统进化的重要依据和正确性检验手段.

我们选择 Java Beans 组件标准作为研究对象,设计了一个基于多类一阶时序逻辑的规范语言——JBDL(Java Beans description language)来刻画 Bean 的语义约束.为了检测 Bean 的 JBDL 规范与其实现之间的一致性,我们提出了一种基于抽象和符号化的动态模型检查方法.在动态检测过程中,由于模型是动态建立的,并且随着检测的进行,模型还在不断演化.针对这种情况,本文给出了时序公式在动态演变模型下的三值语义,以及利用 T³BDD(3-terminal binary decision diagram)有效地进行模型检查的方法.

模型检查是验证系统行为约束的一种方式.其核心问题是模型状态数目无限膨胀的问题.目前的模型检查主要是从技术上尽可能地避免状态爆炸.下面是一些代表性的技术.

(1) 使用有效的搜索算法来搜索状态空间.代表性的工作有 CMU 的 E.M.Clarke 和 R.E.Bryant,TEXAS 的 E.A.Emerson 等人使用 BDD、划分和对称性等方法对逻辑部件进行验证[1~3];P.Godefroid 等人利用偏序特性和状态无关的搜索方法检查并发系统的性质[4].

(2) 组合模型检查(compositional model checking).代表性的工作有 Pnueli 的假设-保证法;Winskel 对命题μ演算的验证;Graf 和 Steffen 利用约简全空间进行验证的方法等.

(3) 符号模型检查(symbolic model checking).代表性的工作有 UPPSALA 的 Wang Yi 和 K.G.Larson 等人对实时系统进行的正确性验证;Denmark 的 H.R.Anderson,Sussex 的 M.Henessy,IBM 的 D.Kozen,Edinberge

---

* 本文研究得到国家自然科学基金和国家 863 高科技项目基金资助.作者倪彬,1969 年生,博士,主要研究领域为面向对象的理论和技术,模型检查.冯玉琳,1942 年生,博士,研究员,博士生导师,主要研究领域为对象技术,分布计算技术,移动计算技术,软件工程.黄涛,1965 年生,博士,研究员,主要研究领域为软件工程,对象技术,分布计算技术,程序设计语言及环境.

本文通讯联系人:黄涛,北京 100080,中国科学院软件研究所计算机科学开放研究实验室

本文 1998-07-07 收到原稿,1998-10-19 收到修改稿

的 C.Stirling 和中国科学院软件研究所的林惠民等人对用 CCS 表达的并发系统进行的正确性验证.

(4) 利用抽象进行模型检查.代表性的工作有:CMU 的 D.E.Long 等人对每个变量进行抽象划分,并对 ACTL*公式表达的性质进行验证;Wolper 对数据无关的系统进行模型检查;Kurshan 在有限状态图方面的工作;Burch 和 Bensalem 等人基于 trace 理论的工作.

与上述几类方法相比,我们工作的不同点在于:(1) 我们要检查的模型是在运行时动态产生的,同时,模型在不断地演化,使得时序公式的语义是三值的;(2) 我们采用了一种特殊的抽象方法,这种方法使得抽象可以自动进行,而且不必在抽象空间上进行操作和谓词的抽象计算;(3) 我们扩充了 BDD,采用 T^3BDD 来表示模型和公式,并进行模型检查.

下面我们首先简要介绍 JBDL 规范,然后给出检测模型和抽象模型的形式定义以及基于 T^3BDD 的模型检查算法,最后给出实例研究的结果.

1 JBDL 规范

语义约束刻画了 Bean 的静态结构和动态行为应该满足的条件,包括 4 种约束.静态约束刻画了 Bean 的状态空间.行为约束刻画了 Bean 的各个操作应该满足的前后条件.生命期约束刻画了 Bean 在创建之后,由事件驱动的整个生命期过程中,自身状态和行为应该满足的时序性质.协作约束刻画了由低层 Bean 协作完成高层 Bean 的操作时应该满足的时序性质.下面是 JBDL 的范式:

⟨JBDL Specification⟩ ::=Spec ⟨Bean Name⟩ Inheritance ⟨Bean Name⟩ ⟨Static Constraints⟩;
⟨Behavior Constraints⟩; ⟨Life Cycle Constraints⟩;
⟨Cooperation Constraints⟩;
End Spec

⟨Static Constraints⟩ ::=Static Constraints: ⟨Static Formulas⟩

⟨Behavior Constraints⟩ ::=Behavior Constraints: (⟨Behavior Name⟩:
Pre Conditions ⟨Static Formulas⟩ Post Conditions ⟨Static Formulas⟩)*

⟨Life Cycle Constraints⟩ ::=Life Cycle Constraints: ⟨Temporal Formulas⟩

⟨Cooperation Constraints⟩ ::=Cooperation Constrains: (⟨Behavior Name⟩: ⟨Temporal Formulas⟩)*

JBDL 是基于多类一阶谓词分支时序逻辑的描述语言.我们在逻辑语言中扩充对面向对象特性的支持,以便表达 Java Beans 的各种对象特性.我们还对逻辑语言的语义进行了扩充,以便统一地、混合地表达 Bean 状态和操作的时序关系.关于 JBDL 规范的细节可以参考文献[5~7].

2 模 型

要进行动态检测,就要求待检测 Bean 在运行过程中需要检测的时候激发检测点事件,并等待检测结果.正在侦听该事件的检测 Bean 接收到检测点事件后,根据待检测 Bean 的当前状态建立和演化检测模型,并根据该模型来进行一致性检测,然后把检测结果返回给待检测的 Bean.如果检测未发现不一致,则待检测 Bean 继续运行;否则,产生不一致例外,并提示检测人员进行干预.由于篇幅的限制,我们直接给出检测模型和抽象模型的形式定义,关于支持检测点事件机制和抽象方法的细节可以参考文献[5~9].

定义 1. 检测模型是如下定义的三元组 $M=\{S,R,I\}$:

(1) S 是 Bean 状态的集合,状态 $s\in S$ 当且仅当 s 是待检测 Bean 在检测过程中处于某个检测点事件时的状态.

(2) R 是 $S\times S$ 上称为状态转换的一个二元关系,$(s,t)\in R$ 当且仅当确定 t 的检测点事件在检测过程中发生在确定 s 的检测点的事件之后,并且它们之间没有其他检测点事件.

(3) $I\subseteq R, s\in I$ 当且仅当 s 是待检测 Bean 在检测过程中处于初始状态检测点事件时的状态.

由于检测模型是不断演化扩充的,在一个检测点,某些时序公式的值按照 JBDL 的语义虽然为假,但随着检测的继续,在进一步演化后的检测模型上,它们的值可能为真.这样,如果我们检测到一个时序约束不满足,

我们并不知道它是暂时不满足还是永久不满足.因此,我们需要区分两种情况:(1) 公式目前不满足,但随着检测模型的演化,公式可能在将来满足;(2) 公式在从现在开始的检测模型上不可能满足.我们把第 1 种情况下公式的值定义为不确定(unknown).在这种情况下,时序公式在检测模型下的语义就有 3 种取值.JBDL 时序公式及其三值语义定义如下.

定义 2. 在检测模型 M 和论域空间 C 上,$M,s_0 \models f$,含义为公式 f 在检测模型 M 下的状态 s_0 满足,$M,s_0 \not\models f$,含义为公式 f 在检测模型 M 下的状态 s_0 不满足,$M,s_0 \models ?f$,含义为公式 f 在检测模型 M 下的状态 s_0 不确定.关系 $\models$, $\not\models$ 和 $\models ?$ 递归定义如下:

(1) $M,s_0 \models p$　当且仅当 p 是静态公式,p 在状态 s_0 满足,

$M,s_0 \not\models p$　当且仅当 p 是静态公式,p 在状态 s_0 不满足;

(2) $M,s_0 \models f_1 \wedge f_2$　当且仅当 $M,s_0 \models f_1$ 并且 $M,s_0 \models f_2$,

$M,s_0 \not\models f_1 \wedge f_2$　当且仅当 $M,s_0 \not\models f_1$ 或者 $M,s_0 \not\models f_2$,

$M,s_0 \models ?f_1 \wedge f_2$　否则;

(3) $M,s_0 \models f_1 \vee f_2$　当且仅当 $M,s_0 \models f_1$ 或者 $M,s_0 \models f_2$,

$M,s_0 \not\models f_1 \vee f_2$　当且仅当 $M,s_0 \not\models f_1$ 并且 $M,s_0 \not\models f_2$,

$M,s_0 \models ?f_1 \vee f_2$　否则;

(4) $M,s_0 \models AX(f_1)$　当且仅当对满足 $(s_0,t) \in R$ 的所有状态 t,有 $M,t \models f_1$,

$M,s_0 \not\models AX(f_1)$　当且仅当对满足 $(s_0,t) \in R$ 的某个状态 t,有 $M,t \not\models f_1$,

$M,s_0 \models ?AX(f_1)$　否则;

(5) $M,s_0 \models EX(f_1)$　当且仅当对满足 $(s_0,t) \in R$ 的某个状态 t,有 $M,t \models f_1$,

$M,s_0 \models ?EX(f_1)$　否则;

(6) $M,s_0 \models A[f_1 U f_2]$　当且仅当对所有路径 $(s_0,s_1,\ldots)$,$\exists i(i \geq 0)[M,s_i \models f_2 \wedge \forall j(i>j \geq 0)[M,s_j \models f_1]]$,

$M,s_0 \not\models A[f_1 U f_2]$　当且仅当有某个路径 $(s_0,s_1,\ldots)$,$\exists i(i \geq 0)[M,s_i \not\models (f_2 \vee f_1) \wedge \forall j(i>j \geq 0)[M,s_j \not\models f_2]]$,

$M,s_0 \models ?A[f_1 U f_2]$　否则;

(7) $M,s_0 \models E[f_1 U f_2]$　当且仅当有某个路径 $(s_0,s_1,\ldots)$,$\exists i(i \geq 0)[M,s_i \models f_2 \wedge \forall j(i>j \geq 0)[M,s_j \models f_1]]$,

$M,s_0 \not\models E[f_1 U f_2]$　当且仅当 $M,s_0 \not\models (f_2 \vee f_1)$,

$M,s_0 \models ?E[f_1 U f_2]$　否则;

(8) $M,s_0 \models AG(f)$　当且仅当对所有路径 $(s_0,s_1,\ldots)$,$\forall i(i \geq 0)[M,s_i \models f]$,

$M,s_0 \not\models AG(f)$　当且仅当有某个路径 $(s_0,s_1,\ldots,t)$,$M,t \not\models f$,

$M,s_0 \models ?AG(f)$　否则;

(9) $M,s_0 \models EG(f)$　当且仅当有路径 $(s_0,s_1,\ldots)$,$\forall i(i \geq 0)[M,s_i \models f]$,

$M,s_0 \not\models EG(f)$　当且仅当 $M,s_0 \not\models f$,

$M,s_0 \models ?EG(f)$　否则;

(10) $M,s_0 \models AF(f)$　当且仅当对所有路径 $(s_0,s_1,\ldots)$,$\exists i(i \geq 0)[M,s_i \models f]$,

$M,s_0 \models ?AF(f)$　否则;

(11) $M,s_0 \models EF(f)$　当且仅当有路径 $(s_0,s_1,\ldots)$,$\exists i(i \geq 0)[M,s_i \models f]$,

$M,s_0 \models ?EF(f)$　否则.

3　模型检查方法

我们通过模型检查来检测 Bean 的时序性质.首先,我们选择一种抽象检测模型的有效的表示方法,这种方法可以减少模型检查时要搜索的状态空间,进一步提高检查的效率和可以检查的状态空间规模.然后,我们给出在这样的抽象检测模型上进行模型检查的算法.

3.1　符号化表示

符号化表示是用符号来表示抽象状态和状态转换,这种表示方式有助于设计和实现状态空间检测的有

效算法.二叉判定图 BDD(binary decision diagram)[2]方法是近年来应用较为广泛的一种符号化表示方法.我们准备采用 BDD 来表示抽象检测模型.但是由于 JBDL 的时序公式在检测过程中的三值语义,而 BDD 只能表示真假两种结果,所以无法直接用 BDD 来表示 JBDL 的时序公式.为此,我们对 BDD 进行扩充,使 BDD 的终结顶增加一个,成为三终结顶的 BDD,即 T³BDD.

定义 3. 令 $f:boolean^n \to \{true,false,unknown\}$是一个把 n 维布尔向量映射到集合$\{true,false,unknown\}$的函数.函数 f 把布尔向量空间划分为 3 个集合$\{S_{true},S_{false},S_{unknown}\}$,使得 $S_{true}=\{\langle x_1,x_2,\ldots,x_n\rangle | f(x_1,x_2,\ldots,x_n)=true\}$, $S_{false}=\{\langle x_1,x_2,\ldots,x_n\rangle | f(x_1,x_2,\ldots,x_n)=false\}$,$S_{unknown}=\{\langle x_1,x_2,\ldots,x_n\rangle | f(x_1,x_2,\ldots,x_n)=unknown\}$.令 f_{true},f_{false},$f_{unknown}$ 分别是集合 S_{true},S_{false} 和 $S_{unknown}$ 的特征函数,则 f 的表示形式 $f_{true}*true+f_{false}*false+f_{unknown}*unknown$ 是 f 的范式.这个和的形式可以用有 3 个终结顶 *true*,*false*,*unknown* 的 BDD 来表示.我们把这样的 BDD 称为三终结顶 BDD,即 T³BDD.

定义 4. 不含有左右子图都相同的两个顶的 T³BDD 称为约简的 T³BDD.

定理 1. 任何函数 $f:Boolean^n \to \{true,false,unknown\}$有唯一(同态意义下)的约简 T³BDD 与之对应,且该 T³BDD 是顶点最少的 T³BDD.

根据定理 1,JBDL 的时序公式都可以用 T³BDD 的特征函数 f 来表示,满足公式的状态集合是 S_{true},不满足公式的状态集合是 S_{false},不确定的状态集合是 $S_{unknown}$.抽象检测模型也可以用 T³BDD 来表示.抽象状态是由一组表征 Bean 约束中静态子公式的布尔变量值确定的.每个抽象状态对应一组具体的布尔变量值.这样,若布尔变量的个数为 n,则每个抽象状态可以看成是一个 n 元布尔向量.我们就可以用 n 元布尔关系来表示抽象状态的集合和初始状态的集合,而 n 元布尔关系可以用布尔公式来表示,也就可以用 T³BDD 来表示.同时,用表征 Bean 约束中静态子公式的布尔变量组 V 构造一个布尔变量组的偶对(V_{from},V_{to}),把在抽象状态转换中表示起始状态和到达状态的布尔变量组分别对应于偶对中的 V_{from} 变量组和 V_{to} 变量组,用 V_{from} 变量组和 V_{to} 变量组作为 T³BDD 对应的布尔公式的变量,用抽象状态转换中起始状态和到达状态对应的布尔向量作为元素构造一个布尔关系.这个布尔公式对应的 T³BDD 就表示包含抽象状态转换的集合.

3.2 模型检查算法

下面我们给出在用 T³BDD 表示的抽象检测模型上,对用 T³BDD 表示的 JBDL 正时序公式进行模型检查的算法.

进行模型检查的过程,主要是根据抽象检测模型和公式的形式递归构造 T³BDD.设 $T^3BDD_S(s)$表示状态集合 S 对应的 T³BDD,s 是表示抽象状态的布尔变量的向量.$T^3BDD_R(s,t)$表示状态转换集合对应的 T³BDD,s 为布尔变量的向量 V_{from},t 为布尔变量的向量 V_{to}.$T^3BDD_I(s)$表示初始状态集合 I 对应的 T³BBD,s 是表示抽象状态的布尔变量的向量.我们用 $T^3BDD_f(s)$表示公式 f 对应的 T³BDD,s 为该 T³BDD 中的布尔变量向量.下面说明如何按照 JBDL 时序公式的形式递归地构造公式的 T³BDD.为了表达方便,下面的公式中有一部分并不是要检测的 JBDL 的时序公式,这些公式标注为辅助公式.

(1) 公式 p 是静态子公式

根据抽象的方法,p 在模型中有相应的布尔变量与之对应.设模型中 p 对应的布尔变量为 v_i,v_i 是 s 的分量,则 $T^3BDD_p(s)$是仅含变量为 v_i 的非终结顶的 T³BDD.

(2) 公式 $\neg f$(辅助公式)

把 $T^3BDD_f(s)$中的终结顶点 *true* 换为 *false*,*false* 换为 *true*,*unknown* 不变,得到 $T^3BDD_{\neg f}(s)$.

(3) 公式 *u2f(f)*(辅助公式)

把 *T3BDDf*(s)中的终结顶点 *unknown* 换为 *false*,得到 $T3BDD_{u2f(f)}(s)$.

(4) 公式 *f 2u(f)*(辅助公式)

把 $T^3BDD_f(s)$中的终结顶点 *false* 换为 *unknown*,得到 $T^3BDD_{f2u(f)}(s)$.

(5) 公式 f_1*f_2,其中*为∨或者∧

计算 $T^3BDD_{f_1*f_2}(s)=T^3BDD_{f_1}(s)*T^3BDD_{f_2}(s)$的有效算法与 BDD 的类似算法一致,其不同之处在于∨和∧对于 3 种值$\{true,false,unknown\}$的计算方法:

∨	*true*	*false*	*unknown*
true	*true*	*true*	*true*
false	*true*	*false*	*unknown*
unknown	*true*	*unknown*	*unknown*

∧	*true*	*false*	*unknown*
true	*true*	*false*	*unknown*
false	*false*	*false*	*false*
unknown	*unknown*	*false*	*unknown*

(6) 公式 $f(s)|v_i=b$,其中 $s=\langle v_1,v_2,\ldots,v_n\rangle$,$b$ 是 *true* 或 *false*(辅助公式)

这个公式表示将 v_i 限制到 b.计算限制的算法与 BDD 计算限制的算法相似,具体如下:

(a) 遍历整个 T³BDD 的顶点,用 v_i 顶点的对应子树(若 b 为 *true*,则用 v_i 为 *true* 对应的子树;若 b 为 *false*,则用 v_i 为 *false* 对应的子树)替代该顶点.

(b) 约简 T³BDD.

(7) 公式 $\exists tf(s,t)$(辅助公式)

设 $\langle v_1,v_2,\ldots,v_n\rangle$ 为 t 对应的布尔变量向量,$\exists tf(s,t)=\exists v_1\exists v_2\ldots\exists v_n f(s,t)$,而 $\exists v_i f(s,t)=f(s,t)|v_i=true\vee f(s,t)|v_i=false$.

(8) 公式 $EX(f)$

计算公式 $f\,2u(\exists t(f(t)\wedge R(s,t)))$对应的 T³BDD.

(9) 公式 $AX(f)$

计算公式 $\neg(\exists t(\neg f(t)\wedge R(s,t)))$对应的 T³BDD.

(10) 公式 $\mu Q(g(Q))$,其中 $g(Q)=f(s)\vee\exists t(Q(t)\wedge R(s,t))$(辅助公式).

令 $g(Q)_{true}$,$g(Q)_{false}$,$g(Q)_{unknown}$ 分别是 $T^3BDD_{g(Q)}(s)$确定的集合 $S_{true}(g(Q))$,$S_{false}(g(Q))$和 $S_{unknown}(g(Q))$的特征函数,由于 Q 在 g 中的出现不含¬,g_{true} 和 g_{false} 均是 $2^{|S|}\rightarrow 2^{|S|}$的按照$\subseteq$的单调函数,根据不动点理论,$g_{true}$ 和 g_{false} 均存在最大不动点和最小不动点.$T^3BDD_{\mu Q(g(Q))}(s)$确定的集合 S_{true} 表示 g_{true} 确定的最小不动点,S_{false} 表示 g_{false} 确定的最大不动点,$S_{unknown}$ 表示其他状态.

根据文献[1]的论述,通过计算 $f(s)\vee\exists t(f(t)\wedge T(s,t))$来计算 $\mu Q(g(Q))$比直接计算 $\mu Q(g(Q))$效率要高,其中 $T(s,t)=\mu P(R(s,t)\vee\exists w(P(s,w)\wedge P(w,t)))$.令 $h(P)=R(s,t)\vee\exists w(P(s,w)\wedge P(w,t))$,计算 $\mu P(h(P))$采用叠代的方法.首先令 $P_0=false$ T^3BDD,然后计算 $P_{i+1}=g(P_i)$,直到 P_{i+1} 和 P_i 相同.

(11) 公式 $\nu Q(g(Q))$,其中 $g(Q)=f(s)\vee\exists t(Q(t)\wedge R(s,t))$(辅助公式)

令 $g(Q)_{true}$,$g(Q)_{false}$,$g(Q)_{unknown}$ 分别是 $T^3BDD_{g(Q)}(s)$确定的集合 $S_{true}(g(Q))$,$S_{false}(g(Q))$和 $S_{unknown}(g(Q))$的特征函数,由于 Q 在 g 中的出现不含¬,则 g_{true} 和 g_{false} 均是 $2^{|S|}\rightarrow 2^{|S|}$的按照$\subseteq$的单调函数,根据不动点理论,$g_{true}$ 和 g_{false} 均存在最大不动点和最小不动点.$T^3BDD_{\nu Q(g(Q))}(s)$确定的集合 S_{true} 表示 g_{true} 确定的最大不动点,S_{false} 表示 g_{false} 确定的最小不动点,$S_{unknown}$ 表示其他状态.

根据文献[1]的论述,我们计算 $\nu Q(g(Q))=\mu Q(g(Q))\vee(\exists t(T(t,t)\wedge T(s,t)))$,其中 $T(s,t)=\mu P(R(s,t)\vee\exists w(P(s,w)\wedge P(w,t)))$.

(12) 公式 $E(f_1Uf_2)$

计算公式 $(f_1\vee f_2)\wedge f\,2u(\mu Q(f_2(s)\vee\exists t(Q(t)\wedge(f_1(s)\wedge R(s,t)))))$.

(13) 公式 $EG(f)$

计算公式 $f\wedge f\,2u(\nu Q((\neg\exists tR(s,t)\wedge f(s))\vee\exists t(Q(t)\wedge(f(s)\wedge R(s,t)))))$.

(14) 公式 $A(f_1Uf_2)$

计算公式 $\neg(u2f(F_1))\wedge f\,2u(\neg F_2)\wedge f\,2u(\neg F_3)\wedge f\,2u(\neg F_4)$.其中 $F_1=\mu Q((\neg f_1(s)\wedge\neg f_2(s))\vee\exists t(Q(t)\wedge(\neg f_2(s)\wedge R(s,t))))$,$F_2=\mu Q((\neg(u2f(f_1(s)))\wedge\neg(u2f(f_2(s))))\vee\exists t(Q(t)\wedge(\neg f_2(s)\wedge R(s,t)))))$,$F_3=\mu Q((\neg\exists tR(s,t)\wedge\neg f_2(s))\vee\exists t(Q(t)\wedge(\neg f_2(s)\wedge R(s,t)))))$,$F_4=\exists t(T(t,t)\wedge T(s,t))$,$T(s,t)=\mu P((R(s,t)\wedge f_1(s))\vee\exists w(P(s,w)\wedge P(w,t)))$.

(15) 公式 $EF(f)$

计算公式 $f\,2u(\mu Q((f(s)\vee(\exists t(Q(t)\wedge R(s,t))))))$.

(16) 公式 $AG(f)$

计算公式 $\neg\mu Q((\neg f(s)\vee(\exists t(Q(t)\wedge R(s,t)))))$.

(17) 公式 $AF(f)$

计算公式 $f\ 2u(\neg(\mu Q((\neg\exists tR(s,t)\wedge\neg\ f\ (s))\vee\exists t(Q(t)\wedge(\neg\ f\ (s)\wedge R(s,t))))))\wedge\neg(\exists t(T(t,t)\wedge T(s,t))))$,其中 $T(s,t)=\mu P((R(s,t)\wedge\neg\ f(s))\vee\exists w(P(s,w)\wedge P(w,t)))$.

构造了 JBDL 时序公式对应的 T³BDD 之后,我们通过对每个公式 f 计算 $\forall s(\neg I(s)\vee f(s))$来获得检查结果,其中 $\forall sf(s)=\forall v_1\forall v_2\ldots\forall v_n(f(s))$,其中 $\langle v_1,v_2,\ldots,v_n\rangle$为 s 对应的布尔向量,$\forall v_i(f(s))=(f(s)|v_i=true\wedge f(s)|v_i=false)$, $f(s)|v_i=b$ 是把 *false* 中的变量 *false* 限制到 *false* 得到的 T³BDD.$T^3BDD_{\forall s(\neg I(s)\vee f(s))}$为 *true*,表示 f 在所有初始状态满足;为 *false*,表示 f 在某个初始状态不满足;为 *unknown*,表示 f 在某个初始状态不确定.

算法的空间复杂度与使用 BDD 相同,即与变量的顺序密切相关.算法的时间复杂度按公式的形式归纳如下,其中$|T^3BDD_f|$为 T^3BDD_f 的顶点数,$|t|$为向量 t 的长度,$|d|$为抽象检测模型状态转换图的直径,$|d|$最坏情况为 $2^{|t|}$.

公式	复杂度	公式	复杂度
静态公式	$O(1)$	$\neg f$	$O(1)$
u2f(f)	$O(1)$	$f\ 2u(f)$	$O(1)$
$f(s)\|v_i=b$	$O(\|T^3BDD_f\|*\log(\|T^3BDD_f\|))$	f_1*f_2	$O(\|T^3BDD_{f_1}\|*\|T^3BDD_{f_2}\|)$
$AF(f)$	$O(\|T^3BDD_f\|^2*\|t\|*\log(\|d\|)))$	$EX(f)$	$O(\|T^3BDD_f\|^2*\|t\|))$
$E(f_1Uf_2)$	$O(\|T^3BDD_f\|^2*\|t\|*\log(\|d\|)))$	$AX(f)$	$O(\|T^3BDD_f\|^2*\|t\|))$
$EG(f)$	$O(\|T^3BDD_f\|^2*\|t\|*\log(\|d\|)))$	$A(f_1Uf_2)$	$O(\|T^3BDD_f\|^2*\|t\|*\log(\|d\|)))$
$EF(f)$	$O(\|T^3BDD_f\|^2*\|t\|*\log(\|d\|)))$	$AG(f)$	$O(\|T^3BDD_f\|^2*\|t\|*\log(\|d\|)))$
$\exists tf(s,t)$		$O(\|T^3BDD_f\|^2*\|t\|))$	
$\mu Q(f(s)\vee\exists t(Q(t)\wedge R(s,t)))$		$O(\|T^3BDD_f\|^2*\|t\|*\log(\|d\|)))$	
$\nu Q(f(s)\vee\exists t(Q(t)\wedge R(s,t)))$		$O(\|T^3BDD_f\|^2*\|t\|\ *\log(\|d\|)))$	

由于 Bean 每个状态的占用空间较多,同时在动态检测中状态数目不断增长,直接在检测模型上进行模型检查很容易导致空间耗尽.为此,我们提出了一种抽象的模型表示方法,不仅能减少检测模型的状态数目,而且能保证总的状态数目有界.同时,我们还证明了这种抽象方法对 JBDL 正时序公式的保守性.事实上,检测是在抽象模型上进行的.由于篇幅的限制,本文着重论述基于 T³BDD 的模型检查方法.关于抽象方法的论述详见文献[5,6,8].

4 实例研究

根据 JBDL 规范和动态模型检查方法,我们设计和实现了检测的支撑框架和工具——MChecker.利用 MChecker,我们对 JavaSoft 提供的分布对象计算实例——股票报价监测器和动画实例——杂耍者进行了研究.实验结果与理论结果完全一致,表明了动态检测在时间和空间上都是有效的,算法的实际时空复杂度要远小于最坏情况.关于工具系统以及实例研究的细节可参见文献[5,6,9].

5 总 结

开发高质量和高可复用性的组件要求对组件的语义进行形式化描述和验证.我们选择 Java Beans 组件标准作为研究对象,设计了一个基于多类一阶时序逻辑的规范语言——JBDL 来刻画 Bean 的语义约束.为了检测规范与其实现的一致性,我们提出了一种基于抽象和符号化的动态模型检查方法.本文给出了动态模型检查方法中,根据 JBDL 公式的三值语义和模型的抽象化,利用 T³BDD 在抽象检测模型上进行符号化模型检查的方法.实验结果表明,该方法在时间和空间上都是有效的.

参考文献

1 Burch J R, Clarke E M, McMillan K L *et al*. Symbolic model checking: 1020 states and beyond. Information and Computation, 1992,98(2):142~170

2 Bryant R E. Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, 1986,C-35(8),677~

691

3 Clarke E M, Grumberg O, Long D E. Model checking and abstraction. In: SIGACT & SIGPALN ed. Proceedings of the 19th Annual ACM Symposium on Principles of Programming Languages. New York: ACM Press, 1992. 343~354

4 Godefroid P. Model checking for programming languages using VeriSoft. In: SIGACT & SIGPALN ed. Proceedings of the 24th ACM Symposium on Principles of Programming Languages. New York: ACM Press, 1997. 174~186

5 倪彬.组件语义约束及动态模型检测方法和技术研究[博士学位论文].中国科学院软件研究所,1998
(Ni Bin. An approach to dynamic model checking for component semantic constraints [Ph.D. Thesis]. Institute of Software, The Chinese Academy of Sciences, 1998)

6 Ni Bin, Feng Yu-lin. Specifying and checking for semantic constraints on Java Beans. In: Keijro Araki, Feng Yu-lin eds. Proceedings of the International Symposium on Future Software Technology'97. Tokyo: SEA, 1997. 79~84

7 Ni Bin, Feng Yu-lin. Exploiting abstraction to check semantic constraints for Java Beans. In: Jose Cuena ed. IT&KNOWS Proceedings of the XV IFIP World Computer Congress. Kluwer Academic Publishers Group, 1998. 187~200

8 Ni Bin, Zhou Ze-hua. Mchecker——an automatically dynamic checking tool for Java Beans semantic constraints. In: Chen Jian, Li Ming-shu *et al* eds. Proceedings of the 27th Technology of Object Oriented Languages and Systems. Beijing, 1998. 164~172

T³BDD Based Dynamic Model Checking

NI Bin FENG Yu-lin HUANG Tao

(*Laboratory of Computer Science Institute of Software The Chinese Academy of Sciences Beijing* 100080)

Abstract Java Beans is a standard for software components. For checking the consistency of the Java Beans semantic constraints with its implementation, a formal Java Beans description language (JBDL) and a dynamic model checking method are proposed in this paper. The authors contribute an efficient symbolic model checking approach using T³BDD. The approach is based on three valued semantics for JBDL formulas and a kind of abstract model which is dynamically established and evolved during Bean's execution.

Key words Component, Java Beans, formal specification, symbolic, model checking, binary decision diagram (BDD).