ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.5, May 2008, pp.1149−1159
DOI: 10.3724/SP.J.1001.2008.01149

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

构件组合的抽象精化验证*

曾红卫+, 缪淮扣

(上海大学 计算机工程与科学学院,上海 200072)

Verification of Component Composition Based on Abstraction Refinement

ZENG Hong-Wei+, MIAO Huai-Kou

(School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

+ Corresponding author: E-mail: zenghongwei@shu.edu.cn, http://www.cs.shu.edu.cn

Zeng HW, Miao HK. Verification of component composition based on abstraction refinement. *Journal of Software*, 2008,19(5):1149−1159. http://www.jos.org.cn/1000-9825/19/1149.htm

Abstract: This paper addresses the state-space explosion problem in the context of verifying component composition. It adapts the counterexample guided abstraction refinement (CEGAR) scheme and proposes a compositional verification approach that the verification of component composition is transformed into local abstraction refinement for individual components participating in the composition in order to reduce analysis complexity. The existential quotient based on equivalence relation is employed to compute the abstraction of each component in component composition, and then the abstraction model for the component composition can be built by composing the existential quotients of components. A compositional validation theorem is proposed and proved, so validating if a generated counterexample is valid and refining the abstraction are all carried out component-wise. This approach does not require the construction of a complete state space of the concrete component composition under verification.

Key words: component composition; model checking; state explosion; equivalence relation; counterexample guided abstraction refinement

摘 要: 针对构件组合的状态爆炸问题,改进了反例引导的抽象精化框架,提出了组合式的抽象精化方法,使构件组合的模型检验转化为各成分构件的局部抽象精化,降低了分析的复杂度.提出了在构件组合情况下基于等价关系和存在商的构件抽象方法,用构件抽象的组合建立构件组合的抽象;提出了组合确认定理并给出证明,使反例确认分解为在各构件上对反例投影的确认;通过对单个构件的等价关系的精化实现构件组合的抽象模型的精化.在模型检验构件组合的过程中,不需要为构件组合建立全局的具体状态空间.

关键词: 构件组合;模型检验;状态爆炸;等价关系;反例引导的抽象精化

* Supported by the National Natural Science Foundation of China under Grant No.60673115 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z144 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant Nos.2007CB310800, 2002CB312001 (国家重点基础研究发展计划(973)); the Research Program of Shanghai Education Committee of China under Grant No.07ZZ06 (上海市教委科研项目); the Shanghai Leading Academic Discipline Project of China under Grant No.J50103 (上海市重点学科建设项目)

Received 2007-11-15; Accepted 2008-03-11

中图法分类号: TP311 文献标识码: A

构件的可重用性和可移植性以及面向服务的计算模式等新技术的发展使得构件式软件开发方法成为软件工程中的一种主流技术.目前,有越来越多的企业采用构件组合技术实现企业业务的应用系统,即用单个构件封装简单的业务功能,通过多个构件的集成逐步构造新的组合构件,实现较为复杂的业务功能.然而,构件在被集成后可能出错或工作不正常,一个主要原因是构件之间的交互可能引起构件的执行违背行为一致性或安全性规则.因此,构件组合时必须确保组合行为满足给定的性质.

模型检验(model checking)是一种验证系统性质的自动技术,在硬件和通信协议领域中得到了广泛的应用,目前已越来越多地用于软件系统的验证.然而,构件组合行为的状态空间呈指数级增长,导致状态空间爆炸(state explosion),从而可能难以在实际的构件组合模型上进行模型检验.抽象技术是应对状态爆炸问题的主要技术之一,其基本思想是构造模拟系统模型的抽象模型,使状态空间大为缩减,然后在抽象模型上对系统性质进行检验.抽象模型中的状态称为抽象状态.相应地,抽象前的系统模型称为具体模型,具体模型中的状态称为具体状态.

抽象一般要求具有性质保留性,也就是说,如果抽象模型满足某一性质,那么具体模型也满足该性质.但是,抽象技术的一个主要缺陷是抽象模型可能引入系统中不存在的附加行为,因此,当抽象模型违背某一性质时,检验产生的反例可能只是抽象模型中引入的一个附加行为,不能说明系统违背该性质,这样的反例称为伪反例(spurious counter-example).此时,需要精化抽象模型以便删除伪反例的行为并进一步对性质进行检验.为此,反例引导的抽象精化(counterexample guided abstraction refinement,简称 CEGAR)[1]框架被提出来并在多种模型检验工具如 SLAM[2,3]和 BLAST[4−6]中实现.框架将验证分成 3 个迭代执行步:构造抽象模型、模型检验以及反例确认和精化.然而,这些工具主要采用谓词抽象技术从顺序的 C 或 Java 程序中抽取系统的模型.谓词抽象本质上根据数据变量值而非系统事件的相似性来集聚系统的状态.在模型检验返回反例时,反例确认需要根据抽象状态对应的具体状态集来求解是否存在对应的具体反例,一旦是伪反例,则对抽象模型实施全局性的精化.

构件的源代码往往是不可见的,构件组合的行为由事件和数据共同确定,各构件之间通过事件进行通信.另一方面,由于状态爆炸问题,建立构件组合的全局状态空间非常困难甚至不可能.因此,一个挑战性的问题是,能否在不实际构造构件组合的全局状态空间的情况下,将构件组合的全局抽象精化转化为各成分构件的局部抽象精化.目前,相关的研究还很少见.

本文应用反例引导的抽象精化框架对构件组合的行为进行验证.针对构件组合的特性,结合组合推理的思想,对该框架进行改进,使抽象精化具有组合性(compositionality).除了需要对全局的抽象模型进行模型检验以外,抽象模型的生成、反例确认和精化都被转化为对成分构件的局部处理.改进后的抽象精化的主要任务是:

1) 生成初始的抽象模型.对构件组合的每个成分构件,根据要验证的性质建立状态等价关系,用等价关系对具体模型的状态集进行等价类划分产生抽象状态,构造抽象模型.用同步积(synchronous product)操作生成构件组合的抽象模型.
2) 模型检验.在抽象模型上检验性质,根据性质保留原理,如果抽象模型满足性质,则构件组合也满足性质,检验结束;否则,产生违背性质的抽象反例,转 3).
3) 反例确认与模型精化.这一步的首要任务是判别抽象反例的有效性.反例被投影到每个成分构件的抽象模型,如果每个成分构件的具体模型都包含反例投影对应的行为,则反例是有效反例,检验结束;否则(伪反例),对不包含反例投影行为的成分构件的抽象模型进行精化,从中排除伪反例投影对应的附加行为,生成新的组合抽象模型.转 2),再次对性质进行检查.

本文第 1 节介绍构件组合的形式化行为建模的方法.第 2 节描述存在性抽象的形式定义,给出构件抽象的定义以及构件组合的一些性质.第 3 节描述确认反例有效性的方法,提出组合确认定理并给出证明,讨论等价关系的精化.第 4 节介绍相关工作.最后是结论以及进一步的研究方向.

1 构件组合的形式建模

模型检验通常用 Kripke 结构表示系统模型,用时态逻辑描述系统性质.Kripke 结构是一个五元组$(S,Init,AP,L,T)$,其中 S 是一个有限状态集,$Init\subseteq S$ 是初始状态集;AP 为原子命题集;$L:S\to 2^{AP}$ 为状态标记函数,将每个状态标记为在该状态下为真值的原子命题集合;$T:S\times S$ 是状态迁移关系.

Kripke 结构的一条路径(path)是一个无穷状态序列$\pi=\langle s_1,s_2,s_3,\ldots\rangle$,其中,$\forall i\geq 1\cdot(s_i,s_{i+1})\in T$.当 $s_1\in Init$ 时,路径称为一次执行.在下面的描述中,用 $Path(M)$表示 Kripke 结构 M 的所有执行的集合.

本文使用线性时态逻辑 LTL 公式描述性质.LTL 公式由原子命题集 AP 中的原子命题、逻辑运算符以及时态运算符构成.LTL 公式定义为:

(1) 对于任意的 $p\in AP$,p 是一个 LTL 公式;

(2) 如果ϕ,φ是 LTL 公式,则$\neg\phi,\phi\wedge\varphi,\phi\vee\varphi$,X$\phi$,F$\phi$,G$\phi$和$\phi$U$\varphi$是 LTL 公式.

时态运算符 X,F,G 和 U 分别表示下一状态(next)、某一将来的状态(future)、所有将来的状态(globally)以及直到某个状态(until).如果 Kripke 结构 M 的每一条路径$\pi\in Path(M),\pi\models\phi$,那么 $M\models\phi$.

构件的行为主要体现在通过接口向外提供服务(输入动作)、调用其他构件的服务(输出动作)以及执行内部操作(内部动作).因此,在用 Kripke 结构描述构件的行为时,原子命题用于表示一个动作的发生.

定义 1(构件). 一个构件的行为模型是 Kripke 结构 $M=(S,Init,AP,L,T)$,其中 $AP=I\cup O\cup H$,I 为输入动作集合,O 为输出动作集合,H 为内部动作集合,I,O 和 H 两两不相交.状态 s 的标记 $L(s)$包含动作 a 表示在状态 s 发生了动作 a.

多个构件的组合可以构成更为复杂的组合构件.设 $M_1=(S_1,Init_1,AP_1,L_1,T_1)$和 $M_2=(S_2,Init_2,AP_2,L_2,T_2)$是两个构件,$AP_1=I_1\cup O_1\cup H_1$,$AP_2=I_2\cup O_2\cup H_2$.

定义 2(可组合). M_1 和 M_2 是可组合的(composable),当且仅当 $AP_1\cap AP_2=(I_1\cap O_2)\cup(I_2\cap O_1)$.

也就是说,两个构件的共享动作只能是那些构件之间的交互动作,即一个构件调用另一个构件的服务,这时,共享动作变为构件组合的内部动作.除此之外,$H_1\cap AP_2=H_2\cap AP_1=I_1\cap I_2=O_1\cap O_2=\varnothing$.如果出现重名情况,则需要对某一个构件的内部动作和输出动作重新命名.

定义 3(构件组合). 设 $M_1=(S_1,Init_1,AP_1,L_1,T_1)$和 $M_2=(S_2,Init_2,AP_2,L_2,T_2)$是两个可组合构件,其中 $AP_1=I_1\cup O_1\cup H_1$,$AP_2=I_2\cup O_2\cup H_2$,则它们的同步组合 $M=M_1||M_2=(S,Init,AP,L,T)$,满足:

(1) $S=\{(s_1,s_2)\mid L_1(s_1)\cap AP_2=L_2(s_2)\cap AP_1\}$;

(2) $Init=(Init_1\times Init_2)\cap S$;

(3) $AP=I\cup O\cup H=AP_1\cup AP_2$,其中 $I=(I_1\cup I_2)\backslash(AP_1\cap AP_2)$,$O=(O_1\cup O_2)\backslash(AP_1\cap AP_2)$,$H=(H_1\cup H_2)\cup(AP_1\cap AP_2)$;

(4) $L((s_1,s_2))=L_1(s_1)\cup L_2(s_2)$;

(5) $((s_1,s_2),(s_1',s_2'))\in T$,当且仅当 $(s_1,s_1')\in T_1$ 并且 $(s_2,s_2')\in T_2$.

构件组合的每个状态由来自两个构件的一对状态组合而成,如果状态$s=(s_1,s_2)\in S$,则称s_1,s_2为s的成分状态.构件组合的原子命题集为两个构件原子命题集的并,但共享原子命题的类型变成了内部动作.构件组合的状态迁移联合两个成分状态的迁移.两个以上构件的组合可以类似地进行定义.

2 抽 象

抽象的目的是使模型的状态空间减少但保留要验证的性质.本文研究构件组合的 LTL 性质的检验,因此根据路径来定义模型的抽象.

定义 4(抽象). 设 $M=(S,Init,AP,L,T)$,$M^a=(S^a,Init^a,AP^a,L^a,T^a)$是两个 Kripke 结构,称 M^a 是 M 的抽象,记为 $M\preceq M^a$,当且仅当 $AP^a\subseteq AP$ 并且对 M 的每个路径$\pi=\langle s_1,s_2,\ldots\rangle\in Path(M)$时,在 M^a 中存在一条对应的路径 $\pi^a=\langle s_1^a,s_2^a,\ldots\rangle\in Path(M^a)$,满足:$\forall i\geq 1\cdot L^a(s_i^a)=L(s_i)\cap AP^a$.

定理 1(性质保留). 设 M 和 M^a 是两个 Kripke 结构,$M\preceq M^a$,φ为由 AP^a 中的原子命题构成的 LTL 公式,如

果 $M^a \models \varphi$,那么 $M \models \varphi$.

证明:由定义 4,M 的每条路径 π 在 M^a 中都有一条对应的路径 π^a, $\forall i \geq 1$,π 中的状态 s_i 与 π^a 中的对应状态 s_i^a 满足 $L^a(s_i^a) = L(s_i) \cap AP^a$. 因此, $\forall p \in AP^a \cdot (p \in L^a(s_i^a) \rightarrow p \in L(s_i)) \wedge (p \notin L^a(s_i^a) \rightarrow p \notin L(s_i))$, 即关于 AP^a,π 和 π^a 在每对对应状态上是等价的.$M^a \models \varphi$ 意味着 M^a 中的每条路径都满足 φ,于是,$\forall \pi \in Path(M)$,M^a 中与 π 对应的路径 $\pi^a \models \varphi$,由于 φ 只涉及 AP^a 中的原子命题,因此 $\pi \models \varphi$,从而 $M \models \varphi$. □

存在性抽象是一种普遍应用的抽象方法,其思想是两个抽象状态之间存在迁移当且仅当在两个抽象状态中分别存在一个具体状态,这两个具体状态在具体模型中存在迁移.本文采用一种基于商(quotient)的存在性抽象(existential abstract).用等价关系对模型的状态集进行等价类划分,然后根据存在性抽象原则构造抽象模型.

定义 5(存在商). 设 $M=(S,Init,AP,L,T)$, $AP^a \subseteq AP$, R 是状态集 S 上关于 AP^a 的一个等价关系,满足 $\forall s,s' \in S$,如果 $s\ R\ s'$,那么 $L(s) \cap AP^a = L(s') \cap AP^a$.定义 M 关于 AP^a 和 R 的存在商 $M/(AP^a,R)=(S^a,Init^a,AP^a,L^a,T^a)$,其中:

(1) $S^a=(S/R)$,即 S^a 是 S 关于 R 的等价类划分;

(2) $Init^a=\{s^a \in S^a \mid \exists s' \in s^a \cdot s' \in Init\}$;

(3) $L^a(s^a)=L(s) \cap AP^a$,其中 $s \in s^a$;

(4) $\forall s_1^a, s_2^a \in S^a \cdot (s_1^a, s_2^a) \in T^a$ 当且仅当 $\exists s_1 \in s_1^a, \exists s_2 \in s_2^a \cdot (s_1, s_2) \in T$.

存在商的每个抽象状态 s^a 对应 M 中的一个状态集(等价类),为方便起见,下面的描述直接用抽象状态 s^a 表示其对应的具体状态集.

定理 2. 设 $M=(S,Init,AP,L,T)$,M 的任意一个存在性商 $M/(AP^a,R)$ 是 M 的一个抽象,即 $M \preceq M/(AP^a,R)$.

证明:根据定义 5,M 的每一条路径 $\langle s_1, s_2, ... \rangle$ 在 $M/(AP^a,R)$ 中都存在一条对应的路径 $\langle s_1^a, s_2^a, ... \rangle$, 并且 $\forall i \geq 1 \cdot (s_i \in s_i^a) \wedge (L^a(s_i^a) = L(s_i) \cap AP^a)$, 因此, $M \preceq M/(AP^a,R)$. □

在实际构造构件组合的初始抽象模型时,为了在不影响验证的同时提高抽象程度,通常根据要验证的 LTL 公式 φ 来构造抽象模型.设 $AP_\varphi = \{p \mid p$ 是 φ 中的一个原子命题$\}$,那么抽象模型 $M/(AP_\varphi,R)$ 就只包含与 φ 相关的原子命题.但在构件组合的情况下,每个构件的抽象还需要保留构件之间共享的原子命题,使抽象构件之间与具体构件之间具有相同的同步消息,即构件抽象要求不损失构件之间的交互.设 $M_i = (S_i, Init_i, AP_i, L_i, T_i)\ (1 \leq i \leq n)$ 是 n 个可组合构件,φ 是要验证的 LTL 性质,那么,构件 M_i 与其他构件共享的以及与性质 φ 有关的所有原子命题为

$$AP_i^a = \left(AP_\varphi \cup \bigcup_{1 \leq j < n, j \neq i} AP_j\right) \cap AP_i \tag{1}$$

定义 6(构件抽象). 设 $M_i = (S_i, Init_i, AP_i, L_i, T_i)\ (1 \leq i \leq n)$ 是 n 个可组合构件,φ 是要验证的 LTL 性质,R_i 是状态集 S_i 上关于 AP_i^a 的一个等价关系,定义 $M_i^a = M_i/(AP_i^a, R_i)$ 为 M_i 关于 $(M_1 \| M_2 \| ... \| M_n) \models \varphi$ 的抽象.

参照 Kripke 结构的模拟和组合的性质[7],构件的抽象和组合具有下面的重要性质.

定理 3. 下列命题成立:

(a) 构件组合具有结合律和交换律.

(b) $\preceq$ 是一个前序关系.

(c) 设 M_1, M_2 为两个可组合构件, $M_1 \preceq M_1^a$, 那么 $M_1 \| M_2 \preceq M_1^a \| M_2$.

推论 1. 设 M_1, M_2 为两个可组合构件, $M_1 \preceq M_1^a$, $M_2 \preceq M_2^a$,那么 $M_1 \| M_2 \preceq M_1^a \| M_2^a$.

证明:因为 $M_1 \preceq M_1^a$,根据定理 3(c), $M_1 \| M_2 \preceq M_1^a \| M_2$. 又因 $M_2 \preceq M_2^a$,则 $M_2 \| M_1^a \preceq M_2^a \| M_1^a$. 根据定理 3(a),组合具有交换律, $M_1^a \| M_2 \preceq M_1^a \| M_2^a$. 再由定理 3(b), $\preceq$ 是传递的,因此, $M_1 \| M_2 \preceq M_1^a \| M_2^a$. □

推论 2. 设 n 个两两可组合构件 $M_i\ (1 \leq i \leq n)$, $M_i \preceq M_i^a$,那么 $M_1 \| M_2 \| ... \| M_n \preceq M_1^a \| M_2^a \| ... \| M_n^a$.

推论 1 和推论 2 说明构件抽象的组合是构件组合的抽象.因此,给定 n 个可组合构件 $M_i\ (1 \leq i \leq n)$ 和 LTL 性质 φ,当检验 $(M_1 \| M_2 \| ... \| M_n) \models \varphi$ 时,首先构造每个 $M_i\ (1 \leq i \leq n)$ 的构件抽象 M_i^a, 然后检验 $(M_1^a \| M_2^a \| ... \| M_n^a) \models \varphi$, 而不必实际求解 $(M_1 \| M_2 \| ... \| M_n)$. 根据定理 1, 如果 $(M_1^a \| M_2^a \| ... \| M_n^a)$ 满足 φ, 那么构件组合 $(M_1 \| M_2 \| ... \| M_n)$ 也满足 φ.

3 反例确认和精化

当模型违背 LTL 性质时,模型检验返回的反例可能是一个有穷执行迹(trace)或者一个带有无穷循环的无穷执行迹(称为套索(lasso))[1].当抽象模型违背性质时,需要确认抽象反例的有效性,即抽象反例是否对应了具体模型中的具体行为.最直接的确认方法是检验具体模型中是否存在对应的具体执行迹,如果存在,则具体执行迹就是违背性质的具体反例,抽象反例是有效的;否则,认为抽象反例是一个伪反例.

3.1 抽象反例的有效性

设 $M=(S,Init,AP,L,T)$,$M^a=(S^a,Init^a,AP^a,L^a,T^a)$是 M 的存在商,φ是一个 LTL 公式,并假定 $M^a\not\models\varphi$并返回了一个抽象反例π^a.下面讨论判别π^a 是否有效的方法.

首先假定π^a 是有穷的,$\pi^a=\langle s_1^a,s_2^a,...,s_n^a\rangle$.由定义 5(存在商)可知,$\pi^a$ 的每个状态 $s_i^a(i\geq 1)$ 对应一个具体状态等价类,因此,π^a 对应 M 中的一个具体执行迹的集合,用 $CTrace(\pi^a)$ 表示.

$$CTrace(\pi^a)=\left\{\langle s_1,s_2,...,s_n\rangle \mid \bigwedge_{1\leq i\leq n}(s_i\in s_i^a)\wedge(s_1\in Init)\wedge\bigwedge_{1\leq i\leq n-1}((s_i,s_{i+1})\in T)\right\} \tag{2}$$

这里,直接用 $s_i^a(i\geq 1)$ 表示其对应的具体状态等价类.$CTrace(\pi^a)$ 中的每条具体执行迹起始于 $s_1^a\cap Init$ 中的某个状态,迹的每个状态 s_i 属于 s_i^a 并且相邻状态 s_i,s_{i+1} 之间存在迁移关系.显然,π^a 是有效的抽象反例当且仅当 $CTrace(\pi^a)\neq\varnothing$.

可以采用迭代方法来判别 $CTrace(\pi^a)\neq\varnothing$ 是否成立或者求解 $CTrace(\pi^a)$.用 RS_i 表示 s_i^a 中的可达具体状态集,即对任意状态 $s\in RS_i$,存在一条从 $s_1^a\cap Init$ 中的某一状态到 s 的路径,那么,$RS_1=s_1^a\cap Init$,并且

$$RS_{i+1}=s_{i+1}^a\cap\{s\mid\exists s'\in RS_i\cdot(s',s)\in T\} \tag{3}$$

定理 4. $CTrace(\pi^a)\neq\varnothing\Leftrightarrow\forall i\in[1,n]\cdot RS_i\neq\varnothing$.

证明:先证前向蕴涵.由于 $CTrace(\pi^a)\neq\varnothing$,因此存在一条具体执行迹 $\langle s_1,s_2,...,s_n\rangle$ 满足 $s_1\in Init$ 并且 $\forall i\in[1,n]\cdot s_i\in s_i^a$,因此 $s_1\in RS_1=s_1^a\cap Init$,$RS_1\neq\varnothing$.假定 $RS_i\neq\varnothing$ 并且 $s_i\in RS_i$,根据 $CTrace(\pi^a)$ 的定义,$s_{i+1}\in s_{i+1}^a$ 并且 $(s_i,s_{i+1})\in T$,因此 $s_{i+1}\in RS_{i+1}$,$RS_{i+1}\neq\varnothing$.

再证后向蕴涵.如果 $\forall i\in[1,n]\cdot RS_i\neq\varnothing$,可以从 RS_n 开始反向构造一条具体执行迹.从 RS_n 中任选一个状态 $s_n\in RS_n$,根据式(3),必定存在一个状态 $s_{n-1}\in RS_{n-1}\cdot(s_{n-1},s_n)\in T$.依此类推,最终可以构造一条具体执行迹 $\langle s_1,s_2,...,s_n\rangle$,其中 $\forall i\in[1,n]\cdot s_i\in s_i^a$ 并且 $\forall i\in[1,n-1]\cdot(s_i,s_{i+1})\in T$.同时,由于 $S_1=s_1^a\cap Init$,所以 $s_1\in Init$,因此 $\bigwedge_{1\leq i\leq n}(s_i\in s_i^a)\wedge(s_1\in Init)\wedge\bigwedge_{1\leq i\leq n-1}((s_i,s_{i+1})\in T)$ 成立,得到:$\langle s_1,s_2,...,s_n\rangle\in CTrace(\pi^a),CTrace(\pi^a)\neq\varnothing$. □

例如,图 1 中的具体模型 M 包含了 9 个状态(1,2,…,9),设 $Init$={1,2,6},抽象模型的 4 个状态Ⅰ,Ⅱ,Ⅲ,Ⅳ分别对应 4 个具体状态等价类:Ⅰ={1,2},Ⅱ={3,4,5},Ⅲ={6,7},Ⅳ={8,9}.状态迁移关系用箭头表示.如果抽象反例为〈Ⅰ,Ⅱ,Ⅲ,Ⅳ〉,容易求出:RS_1={1,2},RS_2={3,5},$RS_3=\varnothing$.因此,〈Ⅰ,Ⅱ,Ⅲ,Ⅳ〉是一个伪反例.但如果抽象反例为〈Ⅰ,Ⅲ,Ⅳ〉,可以求出:RS_1={1,2},RS_2={7},RS_3={9},〈2,7,9〉$\in CTrace$(〈Ⅰ,Ⅲ,Ⅳ〉),因此〈Ⅰ,Ⅲ,Ⅳ〉是一个有效反例.

套索反例可以表示为 $\pi^a=\langle s_1^a,s_2^a,...,s_i^a\rangle\langle s_{i+1}^a,...,s_n^a\rangle^\omega$.为了求套索对应的具体执行迹,需要对无穷循环部分进行展开,即

$$\pi^a=\langle s_1^a,s_2^a,...,s_i^a\rangle\langle s_{i+1}^a,...,s_n^a\rangle\langle s_{i+1}^a,...,s_n^a\rangle...=\langle s_1^a,s_2^a,...s_i^a,s_{i+1}^a,...,s_n^a,s_{i+1+(n-i)}^a,...,s_{n+(n-i)}^a,s_{i+1+2(n-i)}^a,...,s_{n+2(n-i)}^a,...\rangle,$$

其中,$\forall k\in[i+1,n],j\geq 0\cdot s_{k+j(n-i)}^a=s_k^a$.已经证明,在经过有穷次展开后,$\pi^a$ 对应的具体执行迹将呈周期性变化[1].设 $\min=\underset{i+1\leq j\leq n}{\mathrm{Min}}(|s_j^a|)$,$\pi_{\min}^a=\langle s_1^a,s_2^a,...,s_i^a\rangle\langle s_{i+1}^a,...,s_n^a\rangle^{\min+1}$,那么下面的定理成立.

定理 5. 套索 π^a 是有效的当且仅当 $CTrace(\pi_{\min}^a)\neq\varnothing$.

证明:设 $L=|\pi_{\min}^a|=n+\min\times(n-i)$.假定 π^a 是有效的,即存在对应的具体执行迹.由于 π^a 包含无穷循环,其对应的具体执行迹是一个无穷执行 $\langle s_1,s_2,...,s_L,...\rangle$.显然,$\langle s_1,s_2,...,s_L\rangle\in CTrace(\pi_{\min}^a)$,则 $CTrace(\pi_{\min}^a)\neq\varnothing$.

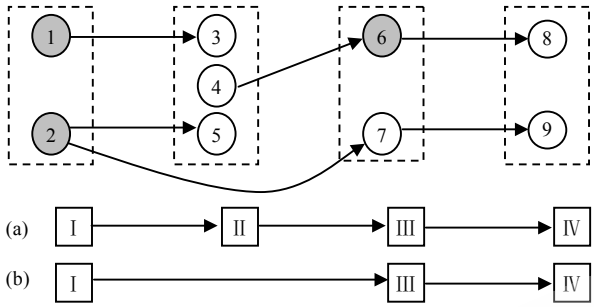

Fig.1 Abstraction counterexamples

图1 抽象反例

另一方面,如果 $CTrace(\pi^a_{\min})\neq\varnothing$, 则存在 $\pi=\langle s_1,s_2,...,s_L\rangle\in CTrace(\pi^a_{\min})$. 设抽象状态 s^a_k $(k\in[i+1,n])$ 包含 min 个具体状态,那么 π 可以表示为 $\langle s_1,s_2,...,s_i,...,s_k,...,s_{k+n-i},...,s_{k+\min\times(n-i)},...,s_L\rangle$, 其中, $\forall j\in[0,\min]\cdot\ s_{k+j\times(n-i)}\in s^a_k$. 因此,至少存在某个 j, $s_{k+j\times(n-i)}$ 会重复出现.设 $j<m$ 且 $s_{k+j\times(n-i)}=s_{k+m\times(n-i)}$,则必定有 $(s_{k+m\times(n-i)-1},s_{k+j\times(n-i)})\in T$. 这样,就可以构造 π^a 的一个具体执行迹 $\langle s_1,s_2,...,s_i,...,s_{k+j\times(n-i)-1}\rangle\langle s_{k+j\times(n-i)},...,s_{k+m\times(n-i)-1}\rangle^{\omega}$. □

因此,要判别套索是否有效只需先求出 $\pi^a_{\min}$,然后根据定理 4 再确认 $\pi^a_{\min}$ 的有效性.无论抽象反例是有穷的还是无穷的,反例确认都可以归结为有穷执行迹的有效性确认.在下面的讨论中,我们只考虑有穷反例.

3.2 组合式反例确认

当验证 n 个构件 $M_i(1\le i\le n)$ 的组合时,需要计算构件组合的抽象模型 $(M^a_1\|M^a_2\|...\|M^a_n)$,然后进行模型检验,但并没有计算具体模型 $M_1||M_2||...||M_n$,因此,定理 4 和定理 5 不能直接利用.本文利用组合推理原理,通过判别抽象反例在 $M^a_i(1\le i\le n)$ 上的投影的有效性来确认整个抽象反例的有效性.

定义 7(迹投影). 设 $M_k=(S_k,Init_k,AP_k,L_k,T_k)\ (k=1,2),\pi=\langle(s^1_1,s^2_1),(s^1_2,s^2_2),...,(s^1_n,s^2_n)\rangle$ 是 $M_1||M_2$ 的一条执行迹,定义 π 在 $M_k\ (k=1,2)$ 上的投影为 $\pi\downarrow M_k=\langle s^k_1,s^k_2,...,s^k_n\rangle$.

定理 6(组合确认). 设 M_1,M_2 是两个构件, M^a_1,M^a_2 分别是它们的构件抽象, $M_1\preceq M^a_1,M_2\preceq M^a_2,\pi^a$ 是 $M^a_1\|M^a_2$ 的一条有穷执行迹,则

$$CTrace(\pi^a)\neq\varnothing\Leftrightarrow(CTrace(\pi^a\downarrow M^a_1)\neq\varnothing)\wedge(CTrace(\pi^a\downarrow M^a_2)\neq\varnothing).$$

证明:设 $M_1=(S_1,Init_1,AP_1,L_1,T_1),M_2=(S_2,Init_2,AP_2,L_2,T_2),\ M_1\|M_2=(S,Init,AP,L,T),M^a_1=(S^a_1,Init^a_1,AP^a_1,L^a_1,T^a_1),M^a_2=(S^a_2,Init^a_2,AP^a_2,L^a_2,T^a_2),M^a_1\|M^a_2=(S^a,Init^a,AP^a,L^a,T^a)$. 由于 π^a 是 $M^a_1\|M^a_2$ 的一条有穷执行迹,设 $\pi^a=\langle(s^{a,1}_1,s^{a,2}_1),(s^{a,1}_2,s^{a,2}_2),...,(s^{a,1}_n,s^{a,2}_n)\rangle$.

先证前向蕴涵关系: $CTrace(\pi^a)\neq\varnothing\Rightarrow(CTrace(\pi^a\downarrow M^a_1)\neq\varnothing)\wedge(CTrace(\pi^a\downarrow M^a_2)\neq\varnothing)$.

$CTrace(\pi^a)\neq\varnothing$ 表明 $M_1\|M_2$ 中必定存在对应的具体执行迹 $\pi=\langle(s^1_1,s^2_1),(s^1_2,s^2_2),...,(s^1_n,s^2_n)\rangle$.

根据定义 3(构件组合),得:

$$s^1_1\in Init_1\wedge s^2_1\in Init_2\wedge\forall i\in[1,n]\cdot(s^1_i\in S_1\wedge s^2_i\in S_2)\wedge\forall i\in[1,n-1]\cdot((s^1_i,s^1_{i+1})\in T_1\wedge(s^2_i,s^2_{i+1})\in T_2)\tag{4}$$

根据抽象的定义(定义4、定义5), $\forall i\in[1,n]\cdot(s^1_i,s^2_i)\in(s^{a,1}_i,s^{a,2}_i)$,则

$$\forall i\in[1,n]\cdot(s^1_i\in s^{a,1}_i\wedge s^2_i\in s^{a,2}_i)\tag{5}$$

由式(4)和式(5)得:

$$s^1_1\in Init_1\wedge\forall i\in[1,n]\cdot(s^1_i\in S_1\wedge s^1_i\in s^{a,1}_i)\wedge\forall i\in[1,n-1]\cdot(s^1_i,s^1_{i+1})\in T_1\tag{6}$$

$$s^2_1\in Init_2\wedge\forall i\in[1,n]\cdot(s^2_i\in S_2\wedge s^2_i\in s^{a,2}_i)\wedge\forall i\in[1,n-1]\cdot(s^2_i,s^2_{i+1})\in T_2\tag{7}$$

因此, M_1 中存在对应 π^a 的具体执行迹 $\pi_1=\langle s^1_1,s^1_2,...,s^1_n\rangle$,由定义7(迹投影),得到:

$$\pi^a \downarrow M_1^a = \langle s_1^{a,1}, s_2^{a,1}, ..., s_n^{a,1} \rangle \text{ ,故 } \pi_1 \in CTrace(\pi^a \downarrow M_1^a).$$

同理,存在 $\pi_2 = \langle s_1^2, s_2^2, ..., s_n^2 \rangle \in CTrace(\pi^a \downarrow M_2^a)$.于是,

$$CTrace(\pi^a \downarrow M_1^a) \neq \varnothing, CTrace(\pi^a \downarrow M_2^a) \neq \varnothing.$$

再证后向蕴涵关系: $CTrace(\pi^a) \neq \varnothing \Leftarrow (CTrace(\pi^a \downarrow M_1^a) \neq \varnothing) \wedge (CTrace(\pi^a \downarrow M_2^a) \neq \varnothing).$

由于 $\pi^a = \langle (s_1^{a,1}, s_1^{a,2}), (s_2^{a,1}, s_2^{a,2}), ..., (s_n^{a,1}, s_n^{a,2}) \rangle$ 是 $M_1^a \| M_2^a$ 的一条有穷执行迹,根据定义 3(构件组合),可得

$$\forall i \in [1, n-1] \cdot ((s_i^{a,1}, s_i^{a,2}), (s_{i+1}^{a,1}, s_{i+1}^{a,2})) \in T^a \Leftrightarrow \forall i \in [1, n-1] \cdot ((s_i^{a,1}, s_{i+1}^{a,1}) \in T_1^a \wedge (s_i^{a,2}, s_{i+1}^{a,2}) \in T_2^a) \quad (8)$$

由式(8)可得, $\pi^a \downarrow M_1^a = \langle s_1^{a,1}, s_2^{a,1}, ..., s_n^{a,1} \rangle$ 是 M_1^a 中的有穷执行迹, $\pi^a \downarrow M_2^a = \langle s_1^{a,2}, s_2^{a,2}, ..., s_n^{a,2} \rangle$ 是 M_2^a 中的有穷执行迹.根据 $CTrace(\pi^a \downarrow M_1^a) \neq \varnothing$,可得:在 M_1 中存在一条具体的执行迹:

$$\pi_1 = \langle s_1^1, s_2^1, ..., s_n^1 \rangle \in CTrace(\pi^a \downarrow M_1^a), \quad s_1^1 \in Init_1 \wedge \forall i \in [1, n] \cdot s_i^1 \in s_i^{a,1}, \quad \forall i \in [1, n-1] \cdot (s_i^1, s_{i+1}^1) \in T_1 \quad (9)$$

同理,根据 $CTrace(\pi^a \downarrow M_2^a) \neq \varnothing$, M_2 中存在一条具体的执行迹:

$$\pi_2 = \langle s_1^2, s_2^2, ..., s_n^2 \rangle \in CTrace(\pi^a \downarrow M_2^a), \quad s_1^2 \in Init_2 \wedge \forall i \in [1, n] \cdot s_i^2 \in s_i^{a,2}, \quad \forall i \in [1, n-1] \cdot (s_i^2, s_{i+1}^2) \in T_2 \quad (10)$$

由式(9)和式(10)得:

$$((s_1^1, s_1^2) \in Init_1 \times Init_2) \wedge \forall i \in [1, n] \cdot ((s_i^1, s_i^2) \in (s_i^{a,1}, s_i^{a,2})) \quad (11)$$

下面证明 $\pi = ((s_1^1, s_1^2), (s_2^1, s_2^2), ..., (s_n^1, s_n^2))$ 是 $M_1 \| M_2$ 的一条执行迹.为此,需要证明 $\forall i \in [1, n] \cdot (s_i^1, s_i^2) \in S$. 即任意的 (s_i^1, s_i^2) 是 $M_1 \| M_2$ 的一个状态. 由于 π^a 是 $M_1^a \| M_2^a$ 的一条执行迹,故 $\forall i \in [1, n] \cdot (s_i^{a,1}, s_i^{a,2}) \in S^a$.

根据定义 3(构件组合),得:

$$L_1^a(s_i^{a,1}) \cap AP_2^a = L_2^a(s_i^{a,2}) \cap AP_1^a \quad (12)$$

根据定义 5(存在商),得:

$$L_1^a(s_i^{a,1}) = L_1(s_i^1) \cap AP_1^a, \quad L_2^a(s_i^{a,2}) = L_2(s_i^2) \cap AP_2^a \quad (13)$$

根据定义 6(构件抽象)和式(1),得:

$$AP_1^a = (AP_\varphi \cup AP_2) \cap AP_1, \quad AP_2^a = (AP_\varphi \cup AP_1) \cap AP_2 \quad (14)$$

由式(12)~式(14)得:

$$L_1^a(s_i^{a,1}) \cap AP_2^a = L_1(s_i^1) \cap AP_1^a \cap AP_2^a = L_1(s_i^1) \cap ((AP_\varphi \cup AP_2) \cap AP_1) \cap ((AP_\varphi \cup AP_1) \cap AP_2) \quad (15)$$

化简式(15)得:

$$L_1^a(s_i^{a,1}) \cap AP_2^a = L_1(s_i^1) \cap AP_2 \quad (16)$$

同理可得:

$$L_2^a(s_i^{a,2}) \cap AP_1^a = L_2(s_i^2) \cap AP_1 \quad (17)$$

联合式(16)和式(17)得:

$$L_1(s_i^1) \cap AP_2 = L_2(s_i^2) \cap AP_1 \quad (18)$$

再根据定义 3(构件组合)和式(11),可得:

$$\forall i \in [1, n] \cdot (s_i^1, s_i^2) \in S, \quad (s_1^1, s_1^2) \in (Init_1 \times Init_2) \cap S \quad (19)$$

由式(9)和式(10)可知: $\forall i \in [1, n-1] \cdot ((s_i^1, s_{i+1}^1) \in T_1 \wedge (s_i^2, s_{i+1}^2) \in T_2)$,那么由定义 3 可得:

$$\forall i \in [1, n-1] \cdot ((s_i^1, s_i^2), (s_{i+1}^1, s_{i+1}^2)) \in T \quad (20)$$

因此,根据式(19)和式(20), $\pi = ((s_1^1, s_1^2), (s_2^1, s_2^2), ..., (s_n^1, s_n^2))$ 是 $M_1 \| M_2$ 的一条执行迹.再根据式(11)和 *CTrace* 的定义(式(2)),得: $\pi = ((s_1^1, s_1^2), (s_2^1, s_2^2), ..., (s_n^1, s_n^2)) \in CTrace(\pi^a), CTrace(\pi^a) \neq \varnothing.$ □

定理 6 可以自然地扩展到 n 个构件的组合.利用抽象反例在构件上的投影来确认抽象反例的有效性不仅可以避免计算构件组合的具体模型所带来的状态爆炸问题,同时还可以标明哪个特定的构件造成抽象反例无效,即抽象反例是一个伪反例,这样就可以针对该构件的抽象模型进行精化以便去除伪反例的行为.

3.3 抽象精化

由于抽象基于状态空间的等价类划分,我们通过精化等价关系来实现抽象模型的精化,从而由抽象模型中

去除伪反例π^a的行为.

定义 8(等价精化). 设R_1和R_2是状态集S上的两个等价关系,称R_2是R_1的精化当且仅当

$$\forall s_1,s_2\in S\cdot((s_1,s_2)\in R_2\Rightarrow(s_1,s_2)\in R_1).$$

现在假定$M_i^a=M_i/(AP_i^a,R_i)(1\le i\le m)$为$m$个构件抽象,检验$(M_1^a\|M_2^a\|...\|M_m^a)\models\varphi$返回了抽象反例$\pi^a$并且反例的某个投影$\pi^a\downarrow M_k^a$是无效的,即$CTrace(\pi^a\downarrow M_k^a)=\varnothing$.设$\pi^a\downarrow M_k^a=\langle s_1^a,s_2^a,...,s_n^a\rangle$, 根据定理 4,必定存在一个下标$j(1\le j\le n)$满足$\forall l\le j\cdot RS_l\ne\varnothing\wedge RS_{j+1}=\varnothing$.称$s_j^a$为$\pi^a$的失败状态(failure state),而$RS_j$中的状态称为"死路"状态(dead-end state).可将s_j^a划分成 3 个两两不相交的子集:$s_j^a=RS_j\cup BS_j\cup IS_j$,其中:

(1) RS_j是死路状态集.如前所述,死路状态是可达的但不存在死路状态到s_{j+1}^a的状态迁移.

(2) $BS_j=\{s\mid s\in s_j^a\wedge\exists s'\in s_{j+1}^a\cdot(s,s')\in T\}$.$BS_j$中的状态称为坏状态(bad state).从坏状态可以迁移到s_{j+1}^a的状态但坏状态是从初始状态不可达的.

(3) $IS_j=s_j^a\setminus(RS_j\cup BS_j)$是无关状态(irrelevant state)集.

既然$(s_j^a,s_{j+1}^a)\in T^a$,那么至少有 1 个从s_j^a的具体状态到s_{j+1}^a的具体状态的迁移,因此$BS_j\ne\varnothing$.正是这种坏状态引起了伪反例π^a的产生.为了使死路状态和坏状态分离在不同的等价类,定义一个s_j^a上的等价关系R':

$$\forall s_1,s_2\in s_j^a\cdot((s_1,s_2)\in R'\Leftrightarrow(\exists s_1',s_2'\in s_{j+1}^a\cdot(s_1,s_1')\in T\wedge(s_2,s_2')\in T)\vee(\forall s'\in s_{j+1}^a\cdot(s_1,s)\notin T\wedge(s_2,s')\notin T).$$

也就是说,s_j^a中的两个状态s_1,s_2具有等价关系当且仅当s_1,s_2同时都有到s_{j+1}^a的具体状态的迁移或者都没有迁移,因此,R'将s_j^a划分为两个等价类$s_{j1}^a=BS_j$和$s_{j2}^a=(RS_j\cup IS_j)$.容易证明,复合关系$R_k;R'$是R_k的精化,并且除了对s_j^a进行划分以外,其他等价类保持不变.故$M_k\preceq M_k/(AP_k^a,R_k;R')\preceq M_k/(AP_k^a,R_k)$.因此,$M_k/(AP_k^a,R_k;R')$是$M_k^a$的精化.在下一次抽象精化迭代中,将用$M_k/(AP_k^a,R_k;R')$代替$M_k^a$,对$(M_1^a\|M_2^a\|...\|M_m^a)\models\varphi$再进行验证.

4 实例研究

考虑由 3 个构件组成的一个简单示例系统.构件*Machining*用于加工产品,*Testing*用于检测产品,*Buffer*管理大小为 1 的产品缓冲区.如图 2 所示,构件被表示为 Kripke 结构.$Machining=(S_M,Init_M,AP_M,L_M,T_M)$,其中$S_M=\{m_0,m_1,m_2,m_3\}$,$Init_M=\{m_0\}$,$AP_M=\{i_1,s_1,w_1,f_1,p\}$,$I_M=\{s_1\}$,$O_M=\{p\}$,$H_M=\{i_1,w_1,f_1\}$.$i_1,w_1,f_1$分别标记*Machining*的空闲、工作和完成状态,s_1表示开始加工产品,而p表示请求*Buffer*将加工好的产品加入缓冲区.$Testing=(S_T,Init_T,AP_T,L_T,T_T)$,其中,$S_T=\{t_0,t_1,t_2,t_3\}$,$Init_T=\{t_0\}$,$AP_T=\{i_2,s_2,w_2,f_2,g\}$,$I_T=\{s_2\}$,$O_T=\{g\}$,$H_T=\{i_2,w_2,f_2\}$.$i_2,w_2,f_2$分别标记*Testing*的空闲、工作和完成状态,s_2表示开始检测产品,而g请求 Buffer 从缓冲区中取出产品.$Buffer=(S_B,Init_B,AP_B,L_B,T_B)$,其中,$S_B=\{b_0,b_1,b_2,b_3,b_4,b_5,b_6\}$,$Init_B=\{b_0\}$,$AP_B=\{E,F,e,p,g\}$.$I_B=\{p,g\}$,$O_B=\{\}$,$H_M=\{E,F,e\}$.$E,F,e$分别表示缓冲区空、满和溢出状态,$p,g$表示加入产品和取出产品动作.不难看出,3 个构件是可组合的.

现在假定要验证系统是否满足 LTL 性质$\varphi=G(f_1\rightarrow X(\neg f_1Ug))$,即在*Machining*完成一个产品加工后,由于缓冲区的大小为 1,在该产品没有开始检测前不能完成另一个产品的加工.由构件抽象的定义可得:$AP_\varphi=\{f_1,g\}$, $AP_M^a=\{f_1,p\}$, $AP_T^a=\{s_2,g\}$, $AP_B^a=\{p,g\}$,从而可以分别求出 3 个构件的抽象 Kripke 结构.再根据构件组合定义可以求出它们的初始抽象组合模型,如图 3 所示.每个抽象状态对应具体状态的一个等价类,组合抽象状态则为一个抽象状态元组.

当在抽象组合模型$Machining^a\|Testing^a\|Buffer^a$上检验性质$\varphi$时,模型检验返回抽象反例$\langle c_0^a,c_1^a,c_0^a,c_1^a,c_0^a\rangle$.该反例在构件抽象$Machining^a$,$Testing^a$和$Buffer^a$上的投影分别为$\langle m_0^a,m_1^a,m_0^a,m_1^a,m_0^a\rangle$,$\langle t_0^a,t_0^a,t_0^a,t_0^a,t_0^a\rangle$和$\langle b_0^a,b_1^a,b_0^a,b_1^a,b_0^a\rangle$.由于$CTrace(\langle m_0^a,m_1^a,m_0^a,m_1^a,m_0^a\rangle)=\varnothing$,根据定理 6,$CTrace(\langle c_0^a,c_1^a,c_0^a,c_1^a,c_0^a\rangle)=\varnothing$.因此,$\langle c_0^a,c_1^a,c_0^a,c_1^a,c_0^a\rangle$是一个伪反例,需要精化$Machining^a$.精化将$m_0^a$划分为两个抽象状态$\{m_0,m_1\}$和$\{m_2\}$,精化后的$Machining^a$与其他两个抽象构件再次组合后重新检验性质$\varphi$,仍然得到一个伪反例并且需要对$Machining^a$再次精化.在经过第 2 次精化和检验后,最终得到一个违背性质φ的有效具体反例:

$$\langle(m_0,t_0,b_0),(m_1,t_0,b_0),(m_2,t_0,b_0),(m_3,t_0,b_1),(m_0,t_0,b_0),(m_1,t_0,b_0),(m_2,t_0,b_0),(m_3,t_0,b_1),(m_0,t_0,b_0)\rangle.$$

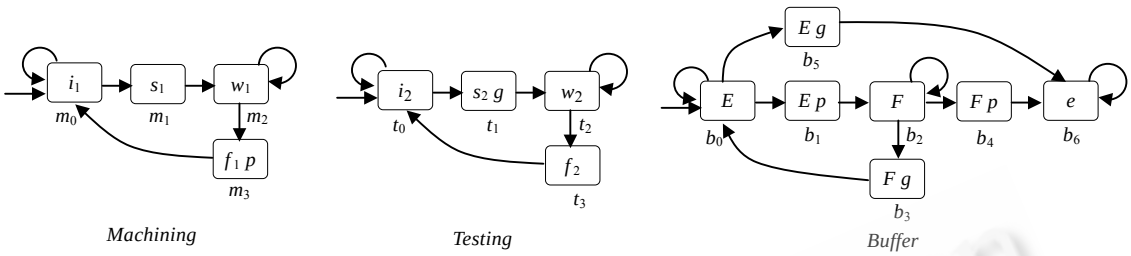

Fig.2 Kripke structures for the example

图 2 示例系统的 Kripke 结构

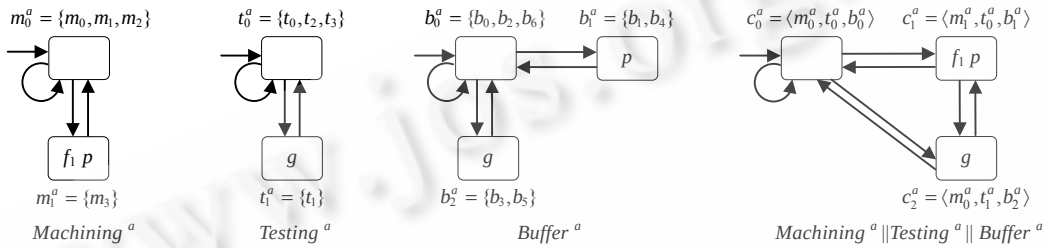

Fig.3 Component abstractions for the example

图 3 示例系统的构件抽象

表 1 列出了在检验性质 φ 的抽象精化过程中示例系统的每个构件抽象和构件组合抽象模型的状态数.可能状态数等于各构件状态规模的乘积,实际上这也是求构件组合的时间复杂度.由于构件组合时组合状态的各成分状态的标记必须相互兼容,因此构件组合模型的有效状态集只是可能状态构成的一个子集.直接组合可以通过一次验证就得到验证结果,但当构件数和构件规模增大时会面临状态爆炸问题.抽象精化验证无疑可以有效地缩小构件组合的状态空间,但其代价是可能需要多次进行组合、验证和精化操作.

Table 1 Size of models for the verification example

表 1 示例验证的模型大小

		Number of states for individual component			Component composition	
		Machining	Testing	Buffer	Possible states	Valid states
Direct composition		4	4	7	112	96
Abstraction refinement	Initial abstraction	2	2	3	12	3
	First refinement	3	2	3	18	5
	Second refinement	4	2	3	24	7

5 相关工作

反例引导的抽象精化已经普遍应用于程序,特别是 C 和 Java 程序的自动验证之中.SLAM 工具[2,3]在分析用 C 语言描述的驱动程序上非常成功.BLAST 工具[4−6]提供了 C 语言程序的 on-the-fly 可达性分析.Corbett 等人[8]给出了从 Java 程序抽取有限状态模型和基于抽象解释自动验证 Java 程序的方法,并实现了工具 BANDERA. Păsăreanu 等人[9]对两个 Java 程序的模型检验工具 BANDERA 和 Java PathFinder 进行集成并对 Java PathFinder 进行了扩展,用 BANDERA 对 Java 程序进行抽象和精化,用 Java PathFinder 进行检验和反例确认.He 等人[10]针对系统芯片(system-on-a-chip)设计的抽象精化验证中的状态划分问题提出了两种启发规则.本文的主要工作是在构件组合的验证过程中应用抽象精化技术,利用构件组合的结构,将复杂的、整体的抽象精化问题分解为比较简单的局部抽象精化问题,即验证具有组合性.然而,上述方法和工具都不支持构件组合验证的组合

性.MAGIC[11]是另一个C程序验证工具,其形式模型采用标记迁移系统(labeled transition system,简称LTS),主要目的是验证系统规约是否为系统实现的抽象.MAGIC的验证过程遵从反例引导的抽象精化框架并具有组合性,但它的组合性主要体现在:当要抽取某一模块的模型时,对它所调用的其他模块的抽象进行假设,通过谓词抽象和定理证明技术从C程序中抽取系统的实现模型.而我们结合要验证的性质,通过对每个构件的模型进行抽象来构造构件组合的抽象,反例的确认和抽象的精化都被转化到单个构件上进行,大幅度降低了分析的复杂度.Brandin等人[12]提出了一种基于子系统的系统验证方法,即将系统的验证转化为子系统的验证.该方法通过反例和一些启发规则来增量地发现合适的子系统.但验证的性质限于行为包含和可控制性,同时要求对系统的每个构件自动机进行扩展以使它们具有相同的动作集.

组合模型检验[13−15]是另一种应对状态爆炸问题的方法.它利用系统的组合结构对问题进行分解,从检验各子系统的性质综合推导出整个系统的性质,从而避免产生精确抽象的困难.这种分而治之的方法是降低模型检验复杂度的有效手段.文艳军等人[16]对组合模型检验和组合精化检验作了系统的分析和归纳,从模块检验的角度阐述了两种组合验证方法的原理及其相应的组合验证策略.尽管组合性是组合模型检验的固有特性,但组合模型检验的困难在于如何将系统的整体性质分解为各子系统的子性质以及如何构造环境模块.

另外,在构件组合的验证方面,许多研究者针对不同的验证问题提出了各种验证方法.胡军等人[17]使用接口自动机和接口自动机网络表示构件式系统的设计模型,使用UML顺序图表示基于场景的需求规约,并对此模型与规约之间的存在一致性和强制存在一致性等性质进行分析,给出了相关的验证算法.Lee等人[18]用接口自动机的组合运算对构件之间的交互兼容性进行检查,但没有对系统行为是否满足特定性质进行检验.Bultan等人[19]给出了一个对Web服务组合的行为进行规约和验证的形式化框架,用Mealy机为Web服务的行为建模,用Web服务之间的会话来表达规约,通过自动机的组合和包含关系进行验证.但是,这些研究都没有采用抽象精化技术.

6 结 论

由于组合状态爆炸问题,应用反例引导的抽象精化技术验证构件组合的主要困难在于很难甚至不能构造全局的构件组合模型.结合组合推理原理,本文提出了组合式的抽象精化方法.主要工作包括:用Kripke结构对构件建模,结合要验证的性质和构件的共享行为,用等价关系对各成分构件的状态空间进行等价类划分并产生各成分构件模型的存在商,在此基础上用同步积构造构件组合的抽象模型;提出组合确认定理使反例确认分解为在各构件上对反例投影的确认,通过判别各成分构件是否包含反例投影所对应的行为来确认反例的有效性,即判别是否为一个伪反例;提出通过精化等价关系实现抽象精化的方法.

尽管还需要对构件抽象进行组合计算,但组合推理的应用避免了构建构件组合的全局具体模型所带来的组合爆炸问题,同时使抽象精化框架中的主要分析任务都是基于单个构件的,大幅度降低了分析的复杂度.

基于本文的研究,下一步的主要工作是利用现有的模型检验工具(如SMV)实现抽象精化的抽象模型检验,集成模型抽象、反例确认和抽象精化算法实现组合式的抽象精化的验证工具.

References:

[1] Clarke E, Grumberg O, Jha S, Lu Y, Veith H. Counterexample-guided abstraction refinement. In: Emerson EA, Sistla AP, eds. Proc. of the Int'l Conf. on Computer Aided Verification (CAV 2000). LNCS 1855, Heidelberg: Springer-Verlag, 2000. 154−169.

[2] Ball T, Majumdar R, Millstein T, Rajamani SK. Automatic predicate abstraction of C programs. ACM SIGPLAN Notices, 2001, 36(5):203−213.

[3] Ball T, Rajamani SK. Automatically validating temporal safety properties of interfaces. In: Matthew BD, ed. Proc. of the 8th Int'l SPIN Workshop on Model Checking of Software. LNCS 2057, Heidelberg: Springer-Verlag, 2001. 103−122.

[4] Henzinger TA, Jhala R, Majumdar R, Sutre G. Lazy abstraction. ACM SIGPLAN Notices, 2002,37(1):58−70.

[5] Henzinger TA, Jhala R, Majumdar R, Qadeer S. Thread-Modular abstraction refinement. In: Jr Hunt WA, Somenzi F, eds. Proc. of the 15th Int'l Conf. on Computer Aided Verification (CAV 2003). LNCS 2725, Springer-Verlag, 2003. 262−274.

[6] Henzinger TA, Jhala R, Majumdar R, Sutre G. Software verification with BLAST. In: Ball T, Rajamani SK, eds. Proc. of the 10th Int'l SPIN Workshop. LNCS 2648, Heidelberg: Springer-Verlag, 2003. 235−239.

[7] Grumberg O, Long DE. Model checking and modular verification. ACM Trans. on Programming Languages and Systems, 1994, 16(3):843−871.

[8] Corbett JC, Dwyer MB, Hatcliff J, Laubach S, Păsăreanu CS, Robby, Zheng HJ. Bandera: Extracting finite-state models from Java source code. In: Proc. of the 22nd Int'l Conf. on Software Engineering (ICSE 2000). New York: ACM Press, 2000. 439−448.

[9] Păsăreanu CS, Dwyer MB, Visser W. Finding feasible counter-examples when model checking abstracted Java programs. In: Margaria T, Wang Y, eds, Proc. of the 7th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001). LNCS 2031, Heidelberg: Springer-Verlag, 2001. 284−298.

[10] He F, Song XY, Gu M, Sun JG. Effective heuristics for counterexample-guided abstraction refinement. In: Proc. of the 17th Great Lakes Symp. on Great Lakes Symp. on VLSI. 2007. 393−398.

[11] Chaki S, Clarke EM, Groce A, Jha S, Veith H. Modular verification of software components in C. IEEE Trans. on Software Engineering, 2004,30(6):388−402.

[12] Brandin BA, Malik R, Malik P. Incremental verification and synthesis of discrete-event systems guided by counter examples. IEEE Trans. on Control Systems Technology, 2004,12(3):387−401.

[13] Berezin S, Campos S, Clarke EM. Compositional reasoning in model checking. In: Langmaack H, Pnueli A, de Roever WP, eds. Proc of the Int'l Symp. on Compositionality (COMPOS'97). LNCS 1536, Heidelberg: Springer-Verlag, 1998. 81−102.

[14] Henzinger TA, Qadeer S, Rajamani SK. Decomposing refinement proofs using assume-guarantee reasoning. In: Sentovich E, ed. Proc. of the 2000 Int'l Conf. on Computer-Aided Design (ICCAD 2000). IEEE Computer Society Press, 2000. 245−252.

[15] Cobleigh JM, Giannakopoulou D, Păsăreanu CS. Learning assumptions for compositional verification. In: Garavel H, Hatcliff J, eds. Proc. of the 9th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003). LNCS 2619, Heidelberg: Springer-Verlag, 2003. 331−346.

[16] Wen YJ, Wang J, Qi ZC. Compositional model checking and compositional refinement checking of concurrent reactive systems. Journal of Software, 2007,18(6):1270−1281 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1270.htm

[17] Hu J, Yu XF, Zhang Y, Wang LZ, LI XD, Zheng GL. Checking component-based designs for scenario-based specifications. Chinese Journal of Computers, 2006,29(4):513−525 (in Chinese with English abstract).

[18] Lee EA, Xiong YH. System-Level types for component-based design. In: Henzinger TA, Kirsch CM, eds. Proc. of the EMSOFT 2001. LNCS 2211, Heidelberg: Springer-Verlag, 2001. 237−253.

[19] Bultan T, Fu X, Hull R, Su JW. Conversation specification: A new approach to design and analysis of e-service composition. In: WWW 2003, Proc. of the 12th Int'l Conf. on World Wide Web. New York: ACM, 2003. 403−410.

附中文参考文献:

[16] 文艳军,王戟,齐治昌.并发反应式系统的组合模型检验与组合精化检验.软件学报,2007,18(6):1270−1281. http://www.jos.org.cn/1000-9825/18/1270.htm

[17] 胡军,于笑丰,张岩,王林章,李宣东,郑国梁.基于场景规约的构件式系统设计分析与验证.计算机学报,2006,29(4):513−525.

曾红卫(1966－),男,湖南岳阳人,博士生,副教授,主要研究领域为软件形式方法,软件验证与测试.

缪淮扣(1953－),男,教授,博士生导师,CCF高级会员,主要研究领域为软件工程,软件形式方法.