软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(3):580−592 [doi: 10.13328/j.cnki.jos.004982]

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

基于模拟关系的精化检测方法*

王 婷[1], 陈铁明[1], 刘 杨[2]

[1](浙江工业大学 计算机科学与技术学院,浙江 杭州 310023)

[2](School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore)

通讯作者: 陈铁明, E-mail: tmchen@zjut.edu.cn

摘 要: 精化检测是一种重要的形式化验证方法,将系统实现和性质规约用相同的形式化语言进行建模,如能证明两者间存在某种精化关系,且该关系能够维持性质,可得出系统实现满足性质规约.为验证不同类型的系统性质,traces,stable failures 和 failures-divergence 精化检测方法已被提出.精化检测算法依赖于子集构造,因而其面临状态空间爆炸问题.近年来,已有学者针对 NFA 语言包含问题提出了基于模拟关系的状态空间消减方法,极大地提高了算法的性能,且该方法能够直接用于 traces 精化检测.在此基础上,提出了基于模拟关系的 stable failures 和 failures-divergence 精化检测方法.此外,还将精化检测扩展到了时间系统的验证中,提出了基于模拟关系的时间自动机 traces 精化检测方法.实验结果表明,基于模拟关系的算法效率有很大提高.

关键词: 精化检测;模拟;failures;divergence;时间自动机

中图法分类号: TP301

中文引用格式: 王婷,陈铁明,刘杨.基于模拟关系的精化检测方法.软件学报,2016,27(3):580−592. http://www.jos.org.cn/1000-9825/4982.htm

英文引用格式: Wang T, Chen TM, Liu Y. Refinement checking based on simulation relations. Ruan Jian Xue Bao/Journal of Software, 2016,27(3):580−592 (in Chinese). http://www.jos.org.cn/1000-9825/4982.htm

Refinement Checking Based on Simulation Relations

WANG Ting[1], CHEN Tie-Ming[1], LIU Yang[2]

[1](College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

[2](School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore)

Abstract: Refinement checking is an important method in formal verification to convey a refinement relationship between an implementation model and a specification model in the same language. If the specification satisfies certain property and the refinement relationship is strong enough to preserve the property, then implementation satisfies the property. Refinement checking was developed in order to verify different kinds of properties, traces, stables failures and failures/divergence. Refinement checking often relies on the subset construction, thus suffers from state space explosion. Recently, some researchers proposed a simulation based approach for solving the language inclusion problem of NFA, which outperforms the previous methods significantly and can be directly used in traces refinement checking. Base on this advancement, this work further proposes stable failures and failures-divergence refinement checking algorithms based on simulation relations. In addition, this work also extends the idea of trace refinement checking to timed systems, and proposes timed automata traces refinement checking based on simulation relations. Experimental results confirm the efficiency of the presented approaches.

Key words: refinement checking; simulation; failure; divergence; timed automata

* 基金项目: 国家自然科学基金(61103044, U1509214); 浙江省自然科学基金(LQ15E050006, LY16F020035)

Foundation item: National Natural Science Foundation of China (61103044, U1509214); Natural Science Foundation of Zhejiang Province of China (LQ15E050006, LY16F020035)

收稿时间: 2015-07-15; 修改时间: 2015-10-20; 采用时间: 2015-11-27; jos 在线出版时间: 2016-01-05

CNKI 网络优先出版: 2016-01-05 16:39:55, http://www.cnki.net/kcms/detail/11.2560.TP.20160105.1639.008.html

精化(refinement)方法已在理论研究和工程实践中被广泛应用,一般过程是通过对抽象模型的逐步求精得到具体的系统实现,系统实现和抽象模型间即存在精化关系.两者均能满足某种性质,使得系统实现在代替抽象模型时能够保持一致性和正确性.在形式化验证方面,利用精化关系的验证(以下称精化检测)是一种十分重要的手段,代表性的精化检测工具 FDR[1]成功应用于学术界和工业界.精化检测方法可实现完全自动化:首先,对系统实现和规约用相同形式化语言进行建模,其中,规约代表系统需要满足的某种性质,只要证明系统实现和规约这两个模型间存在精化关系,并且该精化关系足以维持该性质,则系统实现一定满足规约.

为了验证不同类型的性质,如安全性(safety)和活性(liveness),已提出一系列不同的精化关系,包括 traces 精化、stable failures 精化和 failures-divergences 精化关系等[2].其中,安全性代表系统始终应保持安全状态,其验证通过 traces 精化关系实现;活性代表系统最终会达到的状态,其验证通过 stable failures 精化或 failures-divergences 精化关系来实现[3].FDR[1],PAT[4]等工具支持上述精化检测,其主要过程为:将两个CSP进程转化成带标签的转移系统(labeled transition systems,简称 LTS);然后,根据指定的语义模型(traces,failures,divergences 等)证明一个进程与另一个进程是否存在精化关系.

标签转移系统 LTS 可以被认为是非确定型有穷自动机 NFA.精化检测依赖于经典的子集构造方法,即:将 NFA 确定化,构造成一个确定型有穷自动机 DFA.精化检测根据由系统模型转化的 NFA 与规约模型转化的 DFA 做同步求精操作,生成乘积自动机,并将问题转换为乘积自动机中的可达性问题.如果在乘积自动机中访问到了一个问题状态,则视为找到了一个反例,亦即精化关系不成立.在最差情况下,与 NFA 相比,DFA 的状态数量呈指数级增长,因此,精化检测面临着状态空间爆炸问题.近年来,有关学者针对 NFA 语言包含问题提出了基于模拟关系的方法[5,6].该方法的主要思想为:不存储全部访问过的状态,只存储最大状态集合,并且满足状态空间中任意状态都能被这个最大状态集合中的某个状态模拟.因此,利用该方法能够避免完全的子集构造,从而减少不必要的搜索,极大地缩减状态空间.由于上述所有精化检测算法都是基于子集构造,很自然地,我们可以将基于模拟关系的方法应用到精化检测算法中.

针对精化检测常见的 3 种算法,即 traces 精化检测、stable failures 精化检测和 failures-divergence 精化检测算法,本文在研究基于模拟关系的 traces 精化检测[6]基础上,提出了基于模拟关系的 stable failures 和 failures-divergence 精化检测,提高了检测性能,并且证明了运用模拟关系后精化检测算法的正确性.即:在只访问部分状态的情况下,反例不会丢失.我们将基于模拟关系的精化检测算法集成到了 PAT 模型检测工具中,并以 on-the-fly 的可达性算法验证精化关系.实验结果表明,基于模拟关系的精化检测算法效率大大提高.

另一方面,已有学者将 traces 精化检测扩展到了概率系统中,称为概率精化检测[7].其主要思想为:给定一个带概率的系统模型(用 Markov decision process 表示)和一种非概率的规约模型(用 LTS 表示),概率精化检测可以计算系统实现能够正确执行规约的概率(规约代表系统应该执行的行为).本文则将精化检测扩展到了时间系统中,即:给定带有时间约束的系统模型(用时间自动机 timed automata 表示,以下简称 TA)和规约模型 LTS,时间精化检测验证在有时间限制的情况下,系统能否正确执行规约规定的行为.该方法为文献[8]关于时间自动机语言包含问题的扩充,其差别为规约模型的不同使得验证的系统性质不同.

本文第 1 节给出精化检测相关定义,并描述基于模拟关系的 traces 精化检测算法.第 2 节详细描述基于模拟关系的 stable failures 精化检测和 failures-divergence 精化检测算法.第 3 节给出基于模拟关系的时间自动机精化检测算法.第 4 节基于 PAT 工具和基准系统进行对比实验,说明利用模拟关系能够极大地提高精化检测算法性能.第 5 节为相关工作.最后总结并展望本文工作.

1 精化检测和模拟关系

1.1 定 义

首先给出LTS及其相关定义[9,10].令Σ为可见事件的集合,τ为内部事件(或称隐藏事件),Σ_τ为可见事件和内部事件的集合.

定义 1. LTS L 是四元组$(S,init,Act,T)$,其中,S 是状态集合,$init\in S$ 是 S 中的初始状态, $Act\subseteq\Sigma_\tau$是事件集

合,$T\subseteq S\times Act\times S$ 是所有带标签的转移关系的集合.

T 中的转移(s,e,s')被记为 $s\xrightarrow{e}s'$.LTS 是确定化的,当且仅当对所有的 $s,e(s\in S$ 且 $e\subseteq\Sigma_\tau)$,如果 T 中存在 $s\xrightarrow{e}u$ 和 $s\xrightarrow{e}v$,则 $u=v$.表达式 $enable(s)$表示集合 $\{e\mid\exists s'.s\xrightarrow{e}s'\}$,即,从状态 s 出发的转移上所有事件的集合.给定由状态组成的有穷序列$\langle s_0,s_1,\ldots,s_n\rangle$满足对所有整数 i, $s_i\xrightarrow{\tau}s_{i+1}$ 成立,并且令 $u=s_0,v=s_n$,则该序列可以用 $u\mapsto v$ 来简单表示.假设在 L 中,$u\mapsto u',u'\mapsto v'$以及 $v'\mapsto v$ 成立,则记为 $u\overset{e}{\mapsto}v$.一系列事件的有穷序列被记为$\langle e_0,e_1,\ldots,e_n\rangle$,当且仅当 L 中存在由状态和事件组成的有穷序列$\langle s_0,e_0,s_1,e_1,\ldots,e_n,s_{n+1}\rangle$,使得对所有整数 i, $s_i\xrightarrow{e_i}s_{i+1}$ 成立,并且 $s_0=init$,则称$\langle e_0,e_1,\ldots,e_n\rangle$是 L 的一条 trace.L 中所有 trace 的集合用符号 $traces(L)$来表示.

接下来给出模拟关系的基本定义.令 $F\subseteq S$ 为目标状态集合.给定 S 中的两个状态 s_0 和 s_1,关于 F,s_0 被 s_1 模拟(记为 $s_0\prec s_1$),需满足:如果 $s_0\in F$,则 $s_1\in F$;对任意 $e\in\Sigma_\tau$,如果 $(s_0,e,s_0')\in T$,则一定存在 $(s_1,e,s_1')\in T$,使得 s_0' 被 s_1' 模拟.

定义 2. 设 $L_1=(S_1,init_1,Act_1,T_1)$和 $L_2=(S_2,init_2,Act_2,T_2)$为两个 LTS,满足 $\tau\notin Act_2$.L_1 和 L_2 的乘积自动机被记为 $L_1\times L_2$,也是一个 LTS $L=(S,init,Act,T)$,其中,$S=S_1\times S_2$;$init=init_1\times init_2$;$Act=Act_1\cup Act_2$;T 是一个带标签的转移关系,满足以下两个条件:

(1) 给定 S 中的乘积状态(s_1,s_2),如果 $(s_1,\tau,s_1')\in T_1$,则 $((s_1,s_2),\tau,(s_1',s_2))\in T$;

(2) 给定 S 中的乘积状态(s_1,s_2),如果 $(s_1,e,s_1')\in T_1,(s_2,e,s_2')\in T_2$ 并且 $e\neq\tau$,则 $((s_1,s_2),e,(s_1',s_2))\in T$.

1.2 Traces精化检测

定义 3. 设 L_1 和 L_2 为两个 LTS.L_1 和 L_2 之间存在 traces 精化关系,当且仅当 $traces(L_1)\subseteq traces(L_2)$[9,10].

前文已经提到,traces 精化检测的标准算法基于子集构造.因此需要将 L_2 转化成一个确定化的 LTS,其与确定化前的 LTS 具有完全相同的 traces 集合,并且该系统中消去了 τ 事件.确定化过程定义如下:

定义 4. 设 $L=(S,init,Act,T)$为一个 LTS.L 的确定化后的 LTS 是四元组 $\det(L)=(S',init',Act',T')$,其中,$S'\subseteq 2^S$;$init'=\{s\mid init\mapsto s\}$;$Act'=Act\backslash\{\tau\}$;$T'$是转移关系,满足:$(N,e,N')\in T'$当且仅当 $N'=\{s'\mid\exists s:N.\ s\overset{e}{\mapsto}s'\}$.

由以上定义可知:在 $\det(L)$中,同一个 trace 所到达的所有状态被合并成了集合.因此,$\det(L)$中从某个状态集合出发,经过一个事件只能到达唯一一个确定的状态集合.给定 L_1 和 L_2,标准的 traces 精化检测算法过程为:构建(通常是 on-the-fly 的方式)乘积自动机 $L_1\times\det(L_2)$,然后,在 $L_1\times\det(L_2)$中查找状态(s_1,s_2)(其中,s_1 是 L_1 的一个状态,s_2 是 L_2 的一个状态集合),如果 s_2 是空集,则 traces 精化关系不成立.这样的目标状态被称为 TR-目标状态.在最差情况下,这个方法的复杂度随着 L_2 的状态数量呈指数级增长.

1.3 基于模拟关系的traces精化检测

给定 L_1 和 L_2,基于模拟关系的 traces 精化检测实际上是在 $L_1\times\det(L_2)$中检查状态间是否存在模拟关系.如果某个状态的行为能够被另一个状态模拟,则该状态可以用模拟状态替代.给定乘积自动机 $L_1\times\det(L_2)$中任意两个状态(s_1,s_2)和 (s_1',s_2'),如果满足 $s_1=s_1'$,并且对任意 $s\in s_2$(s_2 为一个状态集合)都存在 $s'\in s_2'$,使得 $s\prec s'$(记为 $s_2'\leqslant s_2$),则用 $(s_1',s_2')\prec(s_1,s_2)$ 来表示对于 TR-目标状态集合,(s_1,s_2)模拟 (s_1',s_2').

引理 1. 如果 $L_1\times\det(L_2)$中的两个状态(s_1,s_2)和 (s_1',s_2') 满足 $(s_1',s_2')\prec(s_1,s_2)$,并且存在 $(s_1,s_2)\xrightarrow{e}(u,v)$,则必然存在 $(s_1',s_2')\xrightarrow{e}(u',v')$,并且 $u=u',v'\leqslant v$.

基于以上引理,可得出以下结论:如果从 (s_1',s_2') 出发,一个 TR-目标状态是可达的,则从(s_1,s_2)出发,TR-目标状态一定也是可达的.因此在搜索时,如果已经访问过(s_1,s_2),当访问到 (s_1',s_2') 时,可以停止搜索它之后的状态.同样,已存储(s_1,s_2)时,对于任意 (s_1',s_2'),只要它被(s_1,s_2)模拟,都可以将 (s_1',s_2') 丢弃.(s_1,s_2)可看成是最大的状态.相反,如果已存储(s_1,s_2),之后又访问到一个状态满足 $(s_1,s_2)\prec(s_1'',s_2'')$,则存储 (s_1'',s_2'') 并将(s_1,s_2)丢弃.可知:算法搜索 $L_1\times\det(L_2)$中时,存储的最大状态集合中任意两个状态间不存在模拟关系.

算法 1 为基于模拟关系的 traces 精化检测算法.该算法首先初始化两个集合 *working* 和 *stored*,*working* 包含初始乘积状态,*stored* 为空.然后,算法进入 while 循环.循环中,算法从 *working* 中取出状态(*impl*,*spec*),将其添加到 *stored* 中并移除所有非最大状态(第 5 行);然后,根据转移上的事件是否为 τ,生成该状态相应的所有后续

状态.对任意后续状态(*impl*′,*spec*′),检查是否存在 $(s_1',s_2')\in stored$ 使得 $(impl',spec')\prec(s_1',s_2')$ 成立:如果不成立,则将(*impl*′,*spec*′)放入 *working*;反之,则舍弃该状态,不进行任何处理.算法在两种情况下结束,即,*working* 为空或者找到了一个 TR-目标状态(第 9 行).算法中的第 5 行和第 10 行保证了 *stored* 中状态是两两不存在模拟关系的.

算法 1. 基于模拟关系的 traces 精化检测算法.

1: **Let** $working=\{(init_1,\{s|init_2\mapsto s\})\}$;$stored=\varnothing$;
2: **While** {$working\neq\varnothing$}
3: { pop (*impl*,*spec*) from *working*;
4: $stored:=stored\cup(impl,spec)$;
5: remove all (s_1,s_2) satisfying $(s_1,s_2)\prec(impl,spec)$ from *stored*;
6: **Forall** $\{(impl,e,impl')\in T_1\}$
7: { **If** $\{e=\tau\}$ $spec:=spec'$;
8: **Else** $spec':=\{s'\mid\exists s\in spec.\ s\overset{e}{\mapsto}s'\}$;
9: **If** {$spec'=\varnothing$} **return** false;
10: **If** {there is no $(s_1',s_2')\in stored$ such that $(impl',spec')\prec(s_1',s_2')$ }
11: push (*impl*′,*spec*′) into *working*; }
12: }

定理 1. 当且仅当 $traces(L_1)\subseteq traces(L_2)$时,算法 1 返回 true[5,6].

为了说明算法 1 如何工作,图 1 演示了具体例子.图 1 右侧显示了 $L_1\times\det(L_2)$的乘积自动机.由于 $s_1'\prec s_2'$,使得 $(s_1,\{s_1'\})$ 和 $(s_1,\{s_2'\})$ 间满足 $(s_1,\{s_2'\})\prec(s_1,\{s_1'\})$,因此,$(s_1,\{s_2'\})$ 的后续状态不需要再进行搜索.

同理,由于 $(s_1,\{s_1',s_2'\})\prec(s_1,\{s_1'\})$,$(s_2,\{s_1',s_2'\})\prec(s_2,\{s_2'\})$(因为一个状态总是可以模拟自己本身),$(s_1,\{s_1',s_2'\})$ 和 $(s_2,\{s_1',s_2'\})$ 的后续状态也不需要再搜索.如图所示,原来基于子集构造的经典算法会产生 13 个状态,而算法 1 只需要搜索 6 个状态(用灰色表示).

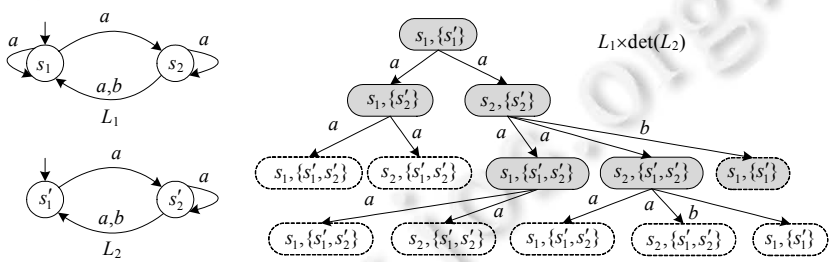

Fig.1 An example of traces refinement checking based on simulation relations

图 1 基于模拟关系的 traces 精化检测举例

2 基于模拟关系的 Failures/Divergence 精化检测

与 traces 精化检测相比,failure/divergence 精化检测引入了 failures 和 divergence 语义.接下来将利用模拟关系给出 failure/divergence 精化检测算法.

2.1 基于模拟关系的stable failures精化检测

设 $L=(S,init,Act,T)$为一个 LTS.给定 *S* 中的状态 *s*,如果$\tau\notin enable(s)$,则称 *s* 是稳定的.在定义 failures 之前,需要先定义 refusals 子集.直观地说,如果从稳定状态 *s* 出发的所有事件集合中不包含某个事件 *e*,则 *s* 拒绝事件 *e*.对于任意一个状态 *s*,*s* 的 refusals 子集用 *refusals*(*s*)表示.*refusals*(*s*)被定义为集合:

$$\{X\mid\exists s'.s\mapsto s'\wedge\tau\notin enable(s')\wedge X\subseteq\Sigma\backslash enable(s')\}.$$

L 的 failures 被记为 $failures(L)$,其被定义成集合 $\{(tr,X):\Sigma^*\times 2^{\Sigma}\mid \exists s.\ init\overset{tr}{\mapsto}s\wedge X\in refusals(s)\}$,其中,$init\overset{tr}{\mapsto}s$ 表示存在一条由状态和事件组成的有穷序列$\langle s_0,e_0,s_1,e_1,\ldots,e_n,s_{n+1}\rangle$,使得 $s_0=init,s_{n+1}=s$,且 $tr=\langle e_0,e_1,\ldots,e_n\rangle$.

定义 5. 给定两个 LTS L_1 和 L_2,当 $traces(L_1)\subseteq traces(L_2)$成立时,$L_1$ 和 L_2 之间存在 stable failures 精化关系,当且仅当 $failures(L_1)\subseteq failures(L_2)$.

stable failures 精化检测算法与 traces 精化检测算法类似,也是一个可达性算法.当 traces 精化关系成立时,在 $L_1\times\det(L_2)$中,stable failures 精化检测算法需要搜索状态(x,y),满足 $refusals(x)$不是 $refusals(y)$的子集.这样的状态称为 SFR-目标状态.给定一个状态集合 X,其拒绝子集 $refusals(X)$为集合$\{r\mid\exists s\in X.r\in refusals(s)\}$.因此, stable failures 精化检测算法见算法 2.于状态$(impl,spec)$,检测 $refusals(impl)\subseteq refusals(spec)$是否成立(第 6 行).

算法 2. 基于模拟关系的 stable failures 精化检测算法.

1: **Let** $working=\{(init_1,\{s\mid init_2\mapsto s\})\}$;$stored=\varnothing$;
2: **While** $\{working\neq\varnothing\}$
3: { pop $(impl,spec)$ from $working$;
4: $stored:=stored\cup(impl,spec)$;
5: remove all (s_1,s_2) satisfying $(s_1,s_2)\lhd(impl,spec)$ from $stored$;
6: **If** $\{refusals(impl)$ is not a subset of $refusals(spec)\}$ **return** false;
7: **Forall** $\{(impl,e,impl')\in T_1\}$
8: { **If** $\{e=\tau\}$ $spec:=spec'$;
9: **Else** $spec':=\{s'\mid\exists s\in spec.s\overset{e}{\mapsto}s'\}$;
10: **If** {there is no $(s_1',s_2')\in stored$ such that $(impl',spec')\lhd(s_1',s_2')$ }
11: push $(impl',spec')$ into $working$; }
12: }

由于 stable failures 精化检测关注系统不能执行的事件,因此并不能直接利用第 1 节的模拟关系.然而容易发现:一个状态必然能够模拟它本身,包括 failures 语义.因此,给定任意两个乘积自动机 $L_1\times\det(L_2)$中的状态(s_1,s_2)和(s_1',s_2'), 当模拟关系为相等时, $(s_1,s_2)\prec(s_1',s_2')$ 事实上满足 $s_1=s_1'$,并且 $s_2'\subseteq s_2$,记为 $(s_1,s_2)\lhd(s_1',s_2')$.

引理 2. 给定 $L_1\times\det(L_2)$中的状态(s_1,s_2),对所有 (s_1,s_2'),如果 $(s_1,s_2)\lhd(s_1,s_2')$,且从(s_1,s_2)出发,SFR-目标状态是可达的,则从 (s_1,s_2') 出发,SFR-目标状态也是可达的.

证明:该引理可以用归纳法来证明.

(1) 初始情况为:令(s_1,s_2)为 SFR-目标状态,根据定义可以得出 $refusals(s_1)$不是 $refusals(s_2)$的子集.由于引理中假定 $s_2'\subseteq s_2$, 则 $refusals(s_2')$ 是 $refusals(s_2)$的子集.如果 $refusals(s_1)$不是 $refusals(s_2)$的子集成立,则 $refusals(s_1)$不是 $refusals(s_2')$ 的子集也成立,即, (s_1,s_2') 也是 SFR-目标状态.

(2) 假定(s_1,s_2)满足引理,即:对所有 (s_1,s_2'),如果 $s_2'\subseteq s_2$,且从(s_1,s_2)出发 SFR-目标状态是可达的,则从 (s_1,s_2') 出发 SFR-目标状态也是可达的.令(x,y)为一个状态,从它出发存在一个到(s_1,s_2)的转移 $(x,y)\xrightarrow{e}(s_1,s_2)$. 对所有状态$(x,y')$,如果 $y'\subseteq y$,则从(x,y')出发,一定存在一个转移 $(x,y')\xrightarrow{e}(s_1,s_2')$, 并且 $s_2'\subseteq s_2$.因此,(x,y)也满足引理,即:对所有(x,y'),如果 $y'\subseteq y$,则从(x,y)出发,SFR-目标状态是可达的.可以推出:从(x,y')出发,SFR-目标状态也是可达的.综合情形(1)、情形(2),该引理成立. □

定理 2. 当且仅当 $failures(L_1)\subseteq failures(L_2)$时,算法 2 返回 true.

证明:给定 $L_1\times\det(L_2)$中的状态 S,将 $Dist(S)\in N\cup\{\infty\}$定义为从 S 出发到达 SFR-目标状态的最短 trace 长度(如果 SFR-目标状态从 S 出发是不可达的,则 $Dist(S)=\infty$).对于一个状态集合 $States$ 来说,如果 $States=\varnothing$,则 $Dist(States)=\infty$;否则,$Dist(States)=\min_{S\in States}Dist(S)$.当且仅当 $States$ 中的所有状态都不是 SFR-目标状态时,断言 $SFR(States)$为真.算法 2 的正确性可以用如下两个表达式来证明:

(1) $\neg SFR(working\cup stored)\Rightarrow\neg SFR(\{(i,\{s\mid init_2\mapsto s\})\mid i\in init_1\})$;

(2) $\neg SFR(\{(i,\{s|init_2 \mapsto s\})|i \in init_1\}) \Rightarrow Dist(stored) > Dist(working)$.

表达式(1)说明:如果集合 *working* 和 *stored* 中存在 SFR-目标状态,则从初始状态出发,SFR-目标状态是可达的.表达式(2)说明:如果从初始状态出发,SFR-目标状态是可达的,则 *Dist*(*Stored*)一定大于 *Dist*(*working*).

首先,由于 LTS 状态数量是有穷的,乘积自动机的状态数量也是有穷的,且算法中每个状态只可能访问 1 次,因此算法 2 最终能够停止.然后,只有当(*impl*,*spec*)满足条件 *refusals*(*impl*)不是 *refusals*(*spec*)的子集(第 6 行)时,算法 2 才会返回 false.当 *refusals*(*impl*)不是 *refusals*(*spec*)的子集时,(*impl*,*spec*)是一个 SFR-目标状态,因此,断言 *SFR*(*working*∪*stored*)是假的.

由表达式(1)可知,断言 *SFR*(*working*∪*stored*)为假可推出 *SFR*({(*i*,{*s*|*init*$_2$↦*s*})|*i*∈*init*$_1$})为假.即:从初始状态出发,SFR-目标状态是可达的,因此 L_1 与 L_2 的 stable failures 精化关系不能维持.此外,只有当 *working* 为空时,算法 2 返回 true,可推出 *Dist*(*Stored*)>*Dist*(*working*)是错误的,因为此时 *Dist*(*Stored*)和 *Dist*(*working*)都为∞.由于表达式(2)的逆反命题同样成立,可以得出:从初始状态出发,SFR-目标状态是不可达的.因此,L_1 与 L_2 存在 stable failures 精化关系. □

图 2 的例子说明了算法 2 对状态空间的消减不会导致反例丢失.

令图中的 $X_0=(s_1,\{s_1'\}),X_1=(s_1,\{s_1',s_2'\}),X_2=(s_2,\{s_2'\}),X_3=(s_2,\{s_1',s_2'\}),X_4=(s_3,\{s_2'\}),X_5=(s_3,\{s_1',s_2'\})$,从图 2 左侧的 LTS 可看出:$s_3$ 拒绝事件 *b*,而 s_1' 和 s_2' 不拒绝发生任何事件.因此,X_4 和 X_5 为 SFR-目标状态.由于 $X_1 \lhd X_0$,X_1 以后的状态不再搜索,包括 X_5.然而从图上可以看到:即使状态空间被消减,无法访问到 SFR-目标状态 X_5,从初始状态 X_0 出发,还是能从另外一条 trace 访问到 SFR-目标状态 X_4.因此,反例并不会丢失.此外,由图 2 可知:搜索顺序虽然具有任意性,但基于模拟关系的算法有可能更快找到较短的反例.

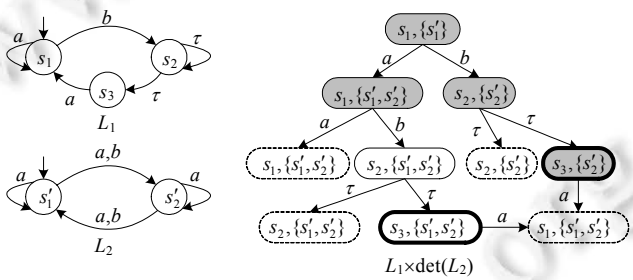

Fig.2 An example of stable failures refinement checking based on simulation relations

图 2 基于模拟关系的 stable failures 精化检测举例

2.2 基于模拟关系的failures-Divergence精化检测

本节首先定义 failures-divergence 精化关系.设 *L*=(*S*,*init*,*Act*,*T*)为 LTS.对于任意一个 *S* 中的状态 *s* 来说,如果 *s* 能够无休止地执行 *τ*转移,则称 *s* 是 diverge 的.给定一条 trace *tr*,当且仅当存在该 trace 的前缀 *pre* 或者 *tr* 本身,使得 $init \overset{pre}{\mapsto} s$ 并且 *s* 是 diverge 的,则 *tr* 是 diverge 的,用符号 *div*(*tr*)表示.所有 *L* 中发生 diverge 的 traces 为集合{*tr*|*div*(*tr*)},用 *divergences*(*L*)表示.

定义 6. 给定两个 LTS L_1 和 L_2,当 *traces*(L_1)⊆*traces*(L_2)和 *failures*(L_1)⊆*failures*(L_2)成立时,L_1 和 L_2 之间存在 failures-divergence 精化关系,当且仅当 *divergences*(L_1)⊆*divergences*(L_2).

Failures-divergence 精化检测算法也是一个可达性算法,见算法 3.

算法 3. 基于模拟关系的 failures-divergence 精化检测算法.

```
1:    Let working={(init1,{s|init2↦s})};stored=∅;
2:    While {working≠∅}
3:    {   pop (impl,spec) from working;
4:        stored:=stored∪(impl,spec);
```

5: remove all (s_1,s_2) satisfying $(s_1,s_2)\prec(impl,spec)$ from *stored*;
6: **If** {*impl* diverges but *spec* does not diverge} **return** false;
7: **Forall** $\{(impl,e,impl')\in T_1\}$
8: { **If** $\{e=\tau\}$ $spec:=spec'$;
9: **Else** $spec':=\{s'\mid \exists s\in spec.s\overset{e}{\mapsto}s'\}$;
10: **If** {there is no $(s_1',s_2')\in stored$ such that $(impl',spec')\prec(s_1',s_2')$ }
11: push $(impl',spec')$ into *working*; }
12: }

给定一个状态集合 X,如果存在 $s\in X$ 并且 s 是 diverge 的,则 X 也是 diverge 的.

在 $L_1\times\det(L_2)$中,当 $traces(L_1)\subseteq traces(L_2)$和 $failures(L_1)\subseteq failures(L_2)$成立时,failures-divergence 精化检测算法需要搜索乘积状态(x,y),满足 x 是 diverge 的但 y 不是 diverge 的.这样的乘积状态称为 FDR-目标状态.算法 3 中,对于状态$(impl,spec)$,第 6 行检测 *impl* 和 *spec* 是否 diverge.

引理 3. 给定 $L_1\times\det(L_2)$中的状态(s_1,s_2),对所有 (s_1,s_2'),如果 $(s_1,s_2)\prec(s_1,s_2')$,且从(s_1,s_2)出发,FDR-目标状态是可达的,则从 (s_1,s_2') 出发,FDR-目标状态也是可达的.

证明:该引理可以用归纳法来证明.

(1) 初始情况为:令(s_1,s_2)为 FDR-目标状态,根据定义可以得出,s_1 是 diverge 的但 s_2 不是.由于引理中假定 $(s_1,s_2)\prec(s_1,s_2')$,则对于任意 $s'\in s_2'$,存在 $s\in s_2$ 使得 $s'\prec s$,如果 s'是 diverge 的,则 s 也是 diverge 的.由此可得出:如果 s_2'是 diverge 的,则 s_2 也是 diverge 的.因此,s_2 不是 diverge 可推出 s_2'不是 diverge,即,(s_1,s_2') 也是 FDR-目标状态;

(2) 假定(s_1,s_2)满足引理,即:对所有状态 (s_1,s_2'),如果 $(s_1,s_2)\prec(s_1,s_2')$,且从(s_1,s_2)出发,FDR-目标状态是可达的,则从 (s_1,s_2') 出发,FDR-目标状态也是可达的.令(x,y)为一个状态,存在转移 $(x,y)\xrightarrow{e}(s_1,s_2)$.对所有状态$(x,y')$,如果$(x,y)\prec(x,y')$,则一定存在一个转移 $(x,y')\xrightarrow{e}(s_1,s_2')$,并且 $(s_1,s_2)\prec(s_1,s_2')$ 成立.因此,(x,y)也满足引理.

综合情形(1)、情形(2),该引理成立. □

定理 3. 当且仅当 $divergences(L_1)\subseteq divergences(L_2)$时,算法 3 返回 true.

定理 3 的证明与定理 2 类似,这里不再赘述.我们使用图 3 的例子来演示算法 3 如何工作.

令图中的 $X_0=(s_1,\{s_1'\}),X_1=(s_2,\{s_1',s_2'\}),X_2=(s_2,\{s_2'\}),X_3=(s_2,\{s_1'\})$,由于 $X_1\prec X_2$ 和 $X_3\prec X_2$,X_1 和 X_3 以后的状态不再搜索,包括 X_3.从图 3 的 LTS 中可以看到:状态 s_2'不是 diverge 的,其他状态都是 diverge 的.因此可以得到,X_2 为 FDR-目标状态.同样地,从图上可以看到:即使状态空间被消减,FDR-目标状态 X_2 还是能被访问到.

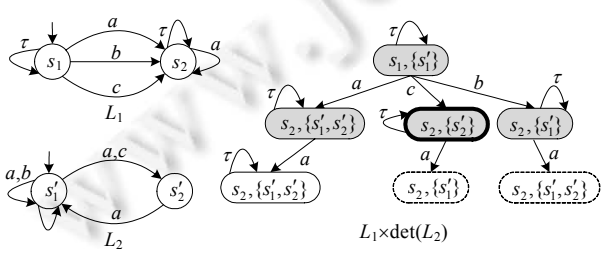

Fig.3 An example of failures-divergence refinement checking based on simulation relations

图 3 基于模拟关系的 failures-divergence 精化检测举例

3 基于模拟关系的时间自动机精化检测

3.1 时间自动机定义

首先给出时间自动机相关定义[8,21].设 C 是时间自动机中的时钟集合,$\Phi(C)$表示时间约束集合.每一个时间

约束定义如下:δ:=true|$x\sim n|\delta_1\wedge\delta_2|\neg\delta_1$,其中,$\sim\in\{=,\leqslant,\geqslant,<,>\}$,$x$ 是时钟集合 C 中的一个时钟,n 是一个非负整数.由符号$\sim\in\{\leqslant,<\}$可以获得下行的时间约束,用$\Phi_{\leqslant,<}(C)$表示.C 上的一个时钟赋值 v 是指为每个时钟赋予一个实数值.如果 v 的时钟赋值使得时间约束δ的布尔值为真,则称 v 满足 C 上的一个时间约束δ.对任意 $d\in\Re^+$,令 $v+d$ 代表时钟赋值 v',使得对所有 $c\in C$ 都满足 $v'(c)=v(c)+d$.令 X 代表被重置的时钟集合,$[X\mapsto 0]v$ 表示对所有 $t\in X$ 都赋值为零,对于所有满足 $t\in C$ 且 $t\notin X$ 的时钟仍与 v 相同.

定义 7. 一个时间自动机是六元组$A=(S,Init,\Sigma,C,L,T)$,其中,S是有穷状态集合;$Init\subseteq S$是初始状态集合;Σ是一个事件集合,满足$\tau\notin\Sigma$;C 是有穷时钟集合;$L:S\to\Phi_{\leqslant,<}(C)$是函数,在每个状态上添加一个状态不变式;$T\subseteq S\times\Sigma\times\Phi(C)\times 2^C\times S$ 是带时间约束的转移关系.

时间自动机 A 的具体语义为具有无穷状态的状态转移系统,表示为 $C(A)=(S_c,Init_c,\Re^+\times\Sigma,T_c)$,其中:$S_c$ 为 A 的具体配置集合,具体配置为(s,v),满足 $s\in S$ 并且 v 是一个时钟值;$Init_c=\{(s,C=0)|s\in Init_c\}$为初始具体配置集合;$T_c$ 为具体转移集合,每个具体转移表示为$((s,v),(d,e),(s',v'))$,满足:存在一个转移$(s,e,\delta,X,s')\in T$;$v+d\in\delta$;$v+d\in L(s)$;$[X\mapsto 0](v+d)=v'$;$v'\in L(s')$.直观地看,系统在 s 等待 d 个时间周期,然后执行转移到达状态 s'.

给定 $C(A)$中的一个运行$\langle(s_0,v_0),(d_1,e_1),(s_1,v_1),(d_2,e_2),\ldots,(s_n,v_n)\rangle$,可获得时间-事件序列$\langle(D_1,e_1),(D_2,e_2),\ldots,(D_n,e_n)\rangle$,满足对所有 $1\leqslant i\leqslant n$, $D_i=\sum_{j=1}^{i}d_j$.$C(A,(s,v))$表示从所有(s,v)出发的运行得到的时间-事件序列集合.A 表示的语言用 $L(A)$表示,其为从 A 中任何初始运行得到的时间-事件序列总和.当两个时间自动机定义相同的语言,则它们是等价的.给定上述 $C(A)$的一个运行,也可以得到一个不带时间的事件序列$\langle e_0,e_1,\ldots,e_n\rangle$.所有这些不带时间的事件序列用 $traces_{TA}(A)$表示.

由于时间点的无穷性,使得 $C(A)$中状态数量也是无穷的.因此,需要将其转化为有穷状态系统才能进行相关验证.时钟域抽象(zone abstraction)是目前最常用的抽象技术.时间自动机经过时钟域抽象后,得到有穷状态的时钟域图.时钟域是时钟赋值的集合,里面的所有时钟值都满足某个的时间约束.一个时钟域是由定义在 C 上的简单线性不等式或线性不等式的合取式结合组成,例如 $x-y\leqslant 5,y>3\wedge x<7$ 等,其中,$x,y\in C$.时钟域和时间约束是一致的.给定一个时钟域,用$\delta^{\uparrow}$表示从δ经过任意时间后得到的时钟域.

定义 8. 设 $A=(S,Init,\Sigma,C,L,T)$为时间自动机,其时钟域图 $ZG(A)$是四元组$(S_z,Init_z,\Sigma_z,T_z)$,其中,

- S_z 表示节点集合,每个节点用(s,δ)表示,且 $s\in S$ 代表一个状态,δ代表一个时间约束(时钟域);
- $Init_z=\{(init,(\wedge_{c\in C}c=0)^{\uparrow}\wedge L(init))|init\in Init\}$是初始节点集合;
- $T_z:S_z\times\Sigma\times S_z$ 代表转移关系,其中,$((s_1,\delta_1),e,(s_2,\delta_2))\in T_z$ 当且仅当 $(s_1,e,\delta,X,s_2)\in T,\delta_1\wedge\delta\neq$false,$[X\mapsto 0](\delta_1\wedge\delta)\wedge L(s_2)\neq$false,并且$\delta_2=(N([X\mapsto 0](\delta_1\wedge\delta)\wedge L(s_2)))^{\uparrow}$.

上述定义中的 N 是时钟域标准化(zone normalization)函数.为了保证时钟域图中的节点数量是有限的,标准化过程[18]是必要的.其基本原理为:时间自动机中的时间约束中,对于每个时钟都存在一个最大数值,当一个时钟域中的时钟值大于该数值时,不需要关心其具体数值,只需要将这个值设为该时钟的最大数值即可.

$ZG(A)$中的运行是无穷序列$\pi_z=\langle(s_0,\delta_0),e_0,(s_1,\delta_1),e_1,(s_2,\delta_2),e_2,\ldots\rangle$,其中,$(s_0,\delta_0)\in Init_z$,并且对所有 $i\geqslant 0$,$((s_i,\delta_i),e_i,(s_{i+1},\delta_{i+1}))\in T_z$.$C(A)$的一个运行$\langle(s_0,v_0),(d_0,e_0),(s_1,v_1),(d_1,e_1),\ldots\rangle$是$\pi_z$ 的一个实例,当且仅当对所有 $i\geqslant 0$,$v_i\triangleright\delta_i$ 成立.π_z 被称为该 $C(A)$运行的一个抽象.从 π_z 可以得到不带时间的有穷事件序列$\langle e_1,e_2,\ldots\rangle$,所有这些不带时间的事件序列用 $traces_{ZG}(A)$表示.容易看出,$traces_{ZG}(A)$和 $traces_{TA}(A)$是等价的.

3.2 时间自动机的traces精化检测

给定时间自动机 A 和 LTS L,时间自动机 traces 精化检测目标是验证 $traces_{TA}(A)\subseteq traces(L)$是否成立.直观地看,检测目的在于验证 A 在有时间约束的情况下是否只能执行 L 中的 traces 而不存在反例.由从前文可知,$traces_{ZG}(A)$和 $traces_{TA}(A)$是相同的,因此有如下时间自动机 traces 精化检测的定义.

定义 9. 设 A 为时间自动机,L 为一个 LTS.当且仅当 $traces_{ZG}(A)\subseteq traces(L)$,$A$ 和 L 存在 traces 精化关系.

定义 10. 设$A=(S_a,Init_a,\Sigma_a,C_a,L_a,T_a)$为时间自动机,$L=(S_l,Init_l,Act_l,T_l)$为 LTS,并且 $\det(L)=(S_l',init_l',\ Act_l',T_l')$ 为 L 的确定化后的 LTS.A 和 L 的乘积自动机,即 A 和 $\det(L)$的乘积自动机,是一个时钟域图 $ZG_{AL}=(S,Init,$

Σ,T),满足如下条件:

(1) S 中的一个元素是抽象配置(s,δ,Y),其中 s 是 S_a 中的状态,δ是 C_a 上的时钟域,T 是 S_l' 中的一个状态;

(2) $Init=\{(init_a(C_a=0)^{\uparrow}\wedge L_a(init_a)),Init_l|init_a\in Init_a\}$是初始抽象配置集合;

(3) Σ等于Σ_a;

(4) $T:S\times T\times S$ 是转移关系,其中,$((s_1,\delta_1,Y_1),e,(s_2,\delta_2,Y_2))\in T$,当且仅当以下条件被满足:1) $(s_1,e,\delta,X,s_2)\in T_a,\delta_1\wedge\delta\neq$false,$[X\mapsto 0](\delta_1\wedge\delta)\wedge L(s_2)\neq$false,并且$\delta_2=(N([X\mapsto 0](\delta_1\wedge\delta)\wedge L(s_2)))^{\uparrow}$;2) $(Y_1,e,Y_2)\in T_l'$.

对于 ZG_{AL} 中的抽象配置 ps,其所有的后续抽象配置表示为 $post(ps,ZG_{AL})$.

定理 4. 设 A 为时间自动机,L 为 LTS.当且仅当在 ZG_{AL} 中,$(s,\delta,\varnothing)$是不可达的,$traces_{ZG}(A)\subseteq traces(L)$成立.

证明:首先证明命题当 ZG_{AL} 中$(s,\delta,\varnothing)$不可达,$traces_{ZG}(A)\subseteq traces(L)$成立.如果 $traces_{ZG}(A)\subseteq traces(L)$不成立,则在 $traces_{ZG}(A)$中一定存在一个运行π_a,使得π_a不在 $traces(L)$中.令$|\pi_a|$为π_a的长度.从π_a必然可以得到前缀π_l,使得$|\pi_l|=|\pi_a|-1$,并且π_l既在 $traces_{ZG}(A)$中又在 $traces(L)$中.令(s,δ,Y)为抽象配置,其中是 $ZG(A)$中的抽象配置,并且$ZG(A)$的运行完全执行π_l且该运行最后的状态是(s,δ);Y是 det(L)的状态,L 的运行也完全执行π_l且该运行的最后状态为 Y.此时,Y 并不是空集.接下来,(s,δ)能够执行某个事件 e,并且转移到抽象配置(s',δ'),但是对于任何一个 Y 中的元素均不能执行同样的事件.因此,在这种情况下,$(s',\delta',\varnothing)$是可达的,命题得证.对于命题当$traces_{ZG}(A)\subseteq traces(L)$时,$ZG_{AL}$ 中$(s,\delta,\varnothing)$是不可达的,其证明方式类似,这里不再赘述. □

抽象配置$(s,\delta,\varnothing)$被称为 TA-目标状态.

3.3 基于模拟关系的时间自动机traces精化检测

两个时钟域之间是可比较的.给定两个在同一个时钟集合上的时钟域δ和δ',当且仅当对任意 $v\in\delta,v\in\delta'$成立,则$\delta\subseteq\delta'$.$(s,\delta,Y)\propto(s,\delta',Y')$表示$\delta\subseteq\delta'$,并且对任意$y'\in Y'$都存在$y\in Y$,使得$y'\prec y$(即 $Y'\leqslant Y$).下面的引理说明了如何在算法中利用该模拟关系:

引理 4. 设(s,δ,Y)和(s,δ',Y')为 ZG_{AL} 中的两个抽象配置.如果$(s,\delta,Y)\propto(s,\delta',Y')$,且从$(s,\delta,Y)$出发 TA-目标状态是可达的,则从$(s,\delta',Y')$出发 TA-目标状态也是可达的.

证明:该引理可以用归纳法来证明.

(1) 初始情况为:令(s,δ,Y)为 TA-目标状态,即 $Y=\varnothing$.由于 $Y'\leqslant Y$,因此 $Y'=\varnothing$,因此,(s,δ',Y')也是 TA-目标状态;

(2) 假定两个抽象配置满足引理,即$(s,\delta,Y)\propto(s,\delta',Y')$,且如果从$(s,\delta,Y)$出发,TA-目标状态是可达的,则从$(s,\delta',Y')$出发,TA-目标状态也是可达的.设$((s_0,\delta_0,Y_0),e,(s,\delta,Y))$为 ZG_{AL} 的一个转移,并且存在 (s_0,δ_0',Y_0') 满足 $\delta_0\subseteq\delta_0'$ 以及 $Y_0'\leqslant Y_0$ (即 $(s_0,\delta_0,Y_0)\propto(s_0,\delta_0',Y_0')$).由 $Y_0'\leqslant Y_0$ 推出 $Y'\leqslant Y$; $\delta_0\subseteq\delta_0'$ 推出$\delta\subseteq\delta'$,其原因为,从 s_0 出发的两个转移其实为同一个转移.因此我们得到: $((s_0,\delta_0',Y_0'),e,(s,\delta',Y'))$ 是 ZG_{AL} 中的一个转移,并且$\delta\subseteq\delta'$,$Y'\leqslant Y$.

综合情形(1)、情形(2),该引理成立. □

基于模拟关系的时间自动机 traces 精化检测算法见算法 4.算法 4 为一个可达性算法,其采用 on-the-fly 的方式构建时钟域图 ZG_{AL}.

算法 4. 基于模拟关系的时间自动机 traces 精化检测算法.

```
1:    Let working=Init;stored=∅;
2:    While {working≠∅}
3:    {   pop ps:=(s,δ,Y) from working;
4:        stored:=stored∪ps;
5:        remove all ps′ satisfying ps′∝ps from stored;
6:        Forall {(s′,δ′,Y′)∈post(ps,ZG_AL)}
7:        {  If {Y′=∅}   return false;
8:           If {there is no ps′∈stored such that (s′,δ′,Y′)∝ps′}
9:                push (s′,δ′,Y′) into working;      }
```

10: }

下面的定理说明了算法 4 的正确性.

定理 5. 当且仅当 $traces_{ZG}(A)\subseteq traces(L)$时,算法 4 返回 true.

由于时钟域标准化函数,时钟域的数量是有穷的,因此抽象配置的数量也是有穷的且算法是可以终止的.算法 4 正确性的证明与算法 1~算法 3 的证明过程类似,在此不予赘述.

在图 4 中,我们展示了一个例子来说明算法 4 如何工作.在图中,时间自动机 A 的状态 s_2 上有一个状态不变式 $x<5$,其含义为:A 在 s_2 上不能一直停留,当时钟值大于或等于 5 时,A 必须进入下一个状态.图中的 ZG_{AL} 为时钟域图.令 $ps_0=(s_1,\{s_1'\},x\geqslant 0)$, $ps_1=(s_1,\{s_1',s_2'\},x>2)$ 并且 $ps_2=(s_1,\{s_1',s_2'\},x>3)$.根据模拟关系,$ps_1\propto ps_0$ 和 $ps_2\propto ps_0$ 成立,因此,ps_1 和 ps_2 以后的状态不需要再搜索.由图 4 可见,状态搜索数量从 10 个减少为 4 个.

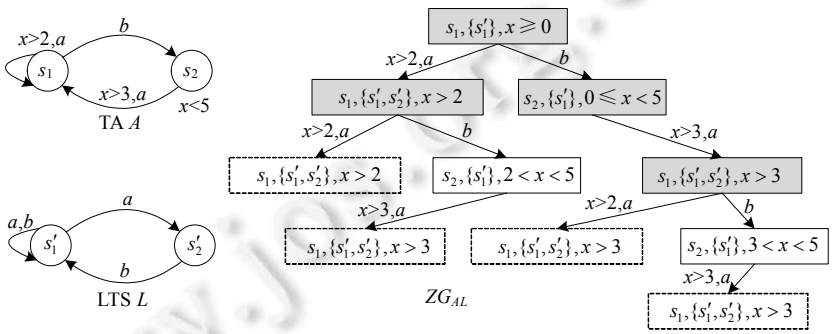

Fig.4 An example of timed automata traces refinement checking based on simulation relations

图 4 基于模拟关系的时间自动机 traces 精化检测举例

4 实验结果

本文提出的算法都已在形式化建模和验证工具 PAT(process analysis toolkit)[4]中实现.PAT 工具的设计目标是,使用当前最先进的精化检测和模型检测技术对分布式/并发系统进行系统化验证.为了衡量算法性能,实验使用一系列实际系统作为基准系统,所有系统都内嵌在 PAT 中.实验的运行环境为 PC 机,其处理器为 Intel(R) Core(TM) i5-4460 CPU@3.20GHz,内存为 8GB.

第 1 部分实验对基于模拟关系和不使用模拟关系的 traces,stable failures 和 failures-divergence 精化检测算法性能进行对比.本实验使用的基准系统[19]包括:邮箱问题(MB),主要为邮差和收信人之间的同步问题;基于单个或多个读者的多值注册模拟系统(MRO 和 MRM),核心为分布式计算中的读写问题;基于线性化点或非线性化点的并发堆栈系统(CSWL 和 CSOL),面向多处理器系统并行编程问题;可扩展非零指示器(SNZI),用于提高事物内存并行程序的性能.基准系统一般具有输入参数(如进程数量、队列长度等).

完整的实验结果见表 1.不带#的基准系统实验结果为 true,即精化关系成立;带#的基准系统结果为 false,即精化关系不成立,存在反例.基准系统中的数字代表系统并发进程数量.表中的倍数为不使用模拟关系算法的检测时间与基于模拟算法的检测时间之间的比值.从表格可知:基准系统不同,则使用基于模拟关系的算法性能提高程度也不同.在大多数情况下,基于模拟关系的算法性能远远好于不使用模拟关系的算法.例如在 CSOL*3 中,基于模拟关系的 stable failures 精化检测算法性能提高了 8.6 倍.然而表格中也存在某些基准系统,基于模拟关系的算法不能发挥明显作用.例如 SNZI*11,尽管基于模拟关系的算法消减了一小部分状态,但由于算法中检查状态间是否存在模拟关系的相关操作存在额外开销,算法从模拟关系获得的状态空间的缩减无法抵消模拟操作的额外开销,因此性能提升并不明显.另外,从#MRO*12 和#MRM*9 的实验结果看出:对于结果为 false 的验证,基于模拟关系的算法有时能够极大地提高反例的查找速度.

Table 1 Experimental results of the refinement checking algorithms based on simulation relations

表 1 基于模拟关系的精化检测算法实验结果

基准系统	是否使用模拟关系	状态数	Traces 精化		Stable failures 精化		Failures-divergence 精化	
			时间(s)	倍数	时间(s)	倍数	时间(s)	倍数
MB*3	是	155K	2.5	1.1	2.4	1.1	186.2	1.3
	否	202K	2.8		2.6		246.7	
MRO*6	是	260K	5.3	4.2	5.4	4.1	103.3	6.5
	否	2930K	22.0		21.9		674.9	
MRM*9	是	125K	14.3	2.9	14.5	3.4	34.4	3.0
	否	3630K	41.4		49.5		104.2	
CSWL*2	是	445K	6.2	1.6	6.3	1.7	45.8	2.8
	否	1366K	10.0		10.8		127.0	
CSOL*3	是	0.1K	0.7	9.6	0.8	8.6	2.6	3.3
	否	3K	6.7		6.9		8.6	
SNZI*11	是	51K	1.3	0.9	1.2	1.1	56.8	1.0
	否	67K	1.2		1.3		57.1	
#MRO*12	是	2K	0.1	1279.0	0.1	1367.0	1.4	98.7
	否	15K	127.9		136.7		138.2	
#MRM*9	是	2K	0.1	51.0	0.1	63.0	0.8	15.1
	否	31K	5.1		6.3		12.1	

第 2 部分实验对基于模拟关系和不使用模拟关系的时间自动机 traces 精化检测算法性能进行对比.本实验使用的基准时间系统[8]包括 Fischer 互斥协议(FISCH)、Lynch-Shavit 互斥协议(LYNCH)、火车控制系统(RWCS)以及 CSMA/CD 协议(CSMACD,针对 broadcast 网络的协议).

实验结果见表 2.表中的基准系统包含了时间自动机和 LTS 的进程数量,例如,FISCH*6(2)表示时间自动机具有 6 个进程,LTS 包含 2 个进程.从算法运行时间倍数和状态数倍数来看:在大部分的例子中,基于模拟关系的算法要极大地优于不使用模拟关系的算法.这种性能提升包含两个原因:其一为 LTS 往往是非确定化的,模拟关系的使用能够避免完全的子集构造;其二为时钟域之间包含关系也是一种模拟关系,使得同样的时间值不需要重复搜索.在某些例子中,如 FISCH*8(1),模拟关系的使用并不能减少状态数量,反而使得验证时间变长.这是由于 LTS 是确定化的并且状态间也不存在模拟关系.另外,时钟域之间的包含关系也没有起作用.

Table 2 Experimental results of timed automata traces refinement checking based on simulation relations

表 2 基于模拟关系的时间自动机 traces 精化检测算法实验结果

基准系统	算法运行时间(s)			状态数		
	无模拟	有模拟	倍数	无模拟(K)	有模拟(K)	倍数
FISCH*6(2)	12.1	8.6	1.4	261	173	1.5
FISCH*6(6)	22.7	6.4	3.5	497	100	5.0
FISCH*8(1)	4.3	5.0	0.9	88	88	1.0
RWCS*6(3)	1.6	0.7	2.3	69	23	3.0
RWCS*8(3)	123.9	42.6	2.9	4 300	1 100	3.9
LYNCH*7(2)	4.4	3.4	1.3	165	130	1.3
LYNCH*8(2)	25.8	19.1	1.4	659	519	1.3
CSMACD*7(1)	15.7	16.8	0.9	146	146	1.0

5 相关工作

抽象和模拟是形式化方法中十分重要的概念.状态转移系统可以在不同的抽象级别上对软硬件系统进行建模.我们往往使用两个系统状态之间的二元关系来比较不同的转移系统,这种二元关系可以定义在在不同的抽象层.最先提出的二元关系为互模拟等价(bisimulation equivalence)关系[11],并且这种互模拟等价关系已被应用到了 NFA 等价问题的检测中[12].然而互模拟等价关系较为严格,并不常常能使状态空间得到明显的缩减,因此引入了模拟(simulation)关系[13],用来验证一个系统是否正确执行了另一个更抽象的系统行为.互模拟等价和模拟关系相关概念已经被扩展到很多不同的方向,如公理化、精化检测、领域方法等.有很多学者研究了基于

模拟关系的抽象方法,并提出了很多算法来计算模拟关系并且得到相应的抽象系统,最新的相关技术包括基于反例的抽象精化[14]、面向无穷状态的抽象和学习方法等[15].

文献[5]提出了基于反链这种模拟关系的非确定型有穷自动机语言通用性和语言包含检测方法,文中的结论表明:基于该模拟关系的方法在某些情况下,其性能可能会高出标准方法几个数量级.他们接下来又研究了基于反链的线性时序逻辑 LTL 模型检测算法,得出了类似的结论[16,17].文献[19]首次将反链关系引入到了 failures/divergence 精化检测算法中.然而上述论文中的反链关系只是一种子集包含关系,其包含在本文定义的模拟关系中,并且本文的模拟关系能够更好地提高 failures/divergence 精化检测和时间自动机 traces 精化检测算法的性能.另外,文献[6,20]中关于树自动机等利用模拟关系的语言通用性和语言包含问题也与本文的工作密切相关.

6 结束语

本文提出了基于模拟关系的 stable failures 和 failures-divergence 精化检测方法,还将精化检测扩展到了时间系统的验证中,提出了基于模拟关系的时间自动机 traces 精化检测方法.利用模拟相关原理,本文提出的算法均可有效减少不必要的状态搜索;同时,利用归纳法证明了算法正确性,并说明了在不搜索全部状态空间的情况下,反例并不会丢失.实验结果表明,基于模拟关系的算法效率有很大提高.

关于未来工作,一方面将研究利用更多的模拟关系进一步优化算法,例如在时间自动机精化检测中利用 lower/upper bounds 的方法或研究时间自动机 robustness 问题来减少状态空间;另一方面可不断完善时间系统和概率系统的精化检测,例如研究解决时间自动机的 non-Zenoness 问题、两个概率系统精化关系如何定义等问题.

References:

[1] Gibson-Robinson T, Armstrong P, Boulgakov A, Roscoe AW. FDR3: A modern refinement checker for CSP. In: Proc. of the 20th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. 2014. 187−201. [doi: 10.1007/978-3-642-54862-8_13]

[2] Roscoe AW. Model-Checking CSP. Prentice-Hall, 1994.

[3] Roscoe AW. On the expressive power of CSP refinement. Formal Aspects of Computing, 2005,17(2):93−112. [doi: 10.1007/s00165-005-0065-x]

[4] Sun J, Liu Y, Dong JS, Pang J. PAT: Towards flexible verification under fairness. In: Proc. of the 21st Int'l Conf. on Computer Aided Verification. 2009. 709−714. [doi: 10.1007/978-3-642-02658-4_59]

[5] Wulf MD, Doyen L, Henzinger TA, Raskin JF. Antichains: A new algorithm for checking universality of finite automata. In: Proc. of the 18th Int'l Conf. on Computer Aided Verification. 2006. 17−30. [doi: 10.1007/11817963_5]

[6] Abdulla PA, Chen YF, Holik L, Mayr R, Vojnar T. When simulation meets antichains. In: Proc. of the 16th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. 2010. 158−174. [doi: 10.1007/978-3-642-12002-2_14]

[7] Song SZ. Model checking stochastic systems in PAT [Ph.D. Thesis]. National University of Singapore, 2013.

[8] Wang T, Sun J, Liu Y, Wang X, Li S. Are timed automata bad for a specification language? Language inclusion checking for timed automata. In: Proc. of the 20th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. 2014. 310−325. [doi: 10.1007/978-3-642-54862-8_21]

[9] Roscoe AW. The Theory and Practice of Concurrency. Prentice-Hall, 1999.

[10] Sun J, Liu Y, Dong JS. Model checking CSP revisited: Introducing a process analysis toolkit. In: Proc. of the 3rd Int'l Symp. on Leveraging Applications of Formal Methods, Verification and Validation. 2008. 307−322. [doi: 10.1007/978-3-540-88479-8_22]

[11] Glabbeek RJV, Weijland WP. Branching time and abstraction in bisimulation semantics. Journal of the ACM, 1996,43(3):555−600. [doi: 10.1145/233551.233556]

[12] Bonchi F, Pous D. Checking NFA equivalence with bisimulations up to congruence. In: Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. 2013. 457−468. [doi: 10.1145/2429069.2429124]

[13] Griffioen WD, Vaandrager F. A theory of normed simulations. ACM Trans. on Computational Logic, 2004,5(4):577−610. [doi: 10.1145/1024922.1024923]

[14] Seipp J, Helmert M. Counterexample-Guided cartesian abstraction refinement. In: Proc. of the 23rd Int'l Conf. on Automated Planning and Scheduling. 2013. 347−351.

[15] Giannakopoulou D, Corina SP. Special issue on learning techniques for compositional reasoning. Formal Methods in System Design, 2008,32(3):173−174. [doi: 10.1007/s10703-008-0054-9]

[16] Doyen L, Raskin JF. Antichains for the automata-based approach to model checking. Logical Methods in Computer Science, 2009, 5(1):1−20. [doi: 10.2168/LMCS-5(1:5)2009]

[17] Wulf MD, Doyen L, Maquet N, Raskin JF. Antichains: Alternative algorithms for LTL satisfiability and model-checking. In: Proc. of the 14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. 2008. 63−77. [doi: 10.1007/978-3-540-78800-3_6]

[18] Bengtsson J, Wang Y. Timed Automata: Semantics, Algorithms and Tools. Lectures on Concurrency and Petri Nets, 2004. 87−124. [doi: 10.1007/978-3-540-27755-2_3]

[19] Wang T, Song SZ, Sun J, Liu Y, Dong JS, Wang X, Li S. More anti-chain based refinement checking. In: Proc. of the 14th Int'l Conf. on Formal Engineering Methods. 2012. 364−380. [doi: 10.1007/978-3-642-34281-3_26]

[20] Bouajjani A, Habermehl P, Holik L, Touili T, Vojnar T. Antichain-Based universality and inclusion testing over nondeterministic finite tree automata. In: Proc. of the 13th Int'l Conf. on Implementation and Applications of Automata. 2008. 57−67. [doi: 10.1007/978-3-540-70844-5_7]

[21] Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science, 1994,126(2):183−235. [doi: 10.1016/0304-3975(94)90010-8]

王婷(1985－),女,浙江慈溪人,博士,讲师,CCF 会员,主要研究领域为形式化方法,软件工程.

陈铁明(1978－),男,博士,副教授,CCF 会员,主要研究领域为形式化方法,软件安全.

刘杨(1981－),男,博士,讲师,主要研究领域为形式化方法,软件安全.