软报 SN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006240]

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

Petri 网的反向展开及其在程序数据竞争检测的应用

郝宗寅[1,2], 鲁法明[2]

[1](山东科技大学 计算机科学与工程学院,山东 青岛 266590)

[2](厦门大学 信息学院,福建 厦门 361000)

通讯作者: 鲁法明, E-mail: fm_lu@163.com, 联系电话 15615125509

摘 要: 展开技术借助分支进程可在一定程度上缓解Petri网性质分析中的状态爆炸问题.但展开网中仍然包含了系统的所有状态信息.某些应用问题仅需对系统特定状态的可覆盖性进行判定,以此为目标有望缩减网系统展开的规模.为此,本文针对安全 Petri 网的可覆盖性判定问题提出了一种目标导向的反向展开算法,结合启发式技术缩减展开的规模,以此提高目标标识可覆盖性判定的效率.进而,将反向展开算法应用于并发程序的形式化验证,将并发程序的数据竞争检测问题转换为Petri网特定标识的可覆盖性判定问题.实验对比了正向展开与反向展开在Petri网可覆盖性判定问题上的效率,结果表明,当 Petri 网展开的正向分支较多时,反向展开相比正向展开具有更高的可覆盖性判定效率.最后,本文对影响反向展开效率的关键因素做了分析与总结.

关键词: Petri 网;可覆盖性判定;反向展开;启发式优化;数据竞争检测

Reverse Unfolding of Petri Nets and its Application in Program Data Race Detection

HAO Zong-Yin[1,2], LU Fa-Ming[2]

[1](School of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China)

[2](School of Information, Xiamen University, Xiamen 361000, China)

Abstract: Unfolding technology can partially alleviate the state explosion problem in Petri net through branching processes. However, an unfolding still contains all states of a system. Some practical problems only need to determine the coverability of a specific state, which is expected to reduce the scale of unfolding net. In this paper, we propose a target-oriented reverse unfolding algorithm for the coverability problem of 1-safe Petri nets, which combines heuristic technology to reduce the scale of unfolding nets, thereby improving the efficiency of coverability determination. Furthermore, we apply the reverse unfolding technology to the formal verification of concurrent programs, and convert the data race detection problem into the coverability problem of a specific state in 1-safe Petri nets. The experiment compares the efficiency between forward unfolding and reverse unfolding in the coverability problem of Petri net. The results show that when the Petri net has more forward branches than backward branches, reverse unfolding is more efficient than forward unfodling. Finally, the key factors influencing the efficiency of reverse unfolding are analyzed.

Key words: Petri net; coverability determination; reverse unfolding; heuristic optimization; data race detection

1 绪论

Petri网作为一种分布式并发系统的建模和分析工具,在柔性制造系统[1, 2,],业务过程管理系统[3, 4],并发程序形式化验证[5, 6]等方面得到了广泛使用.然而,状态爆炸问题妨碍了 Petri 网在大规模并发系统分析中的应用.为

* 基金项目: 国家自然科学基金(61602279, 61472229);国家重点研发计划(2016YFC0801406);山东省泰山学者工程专项基金(ts20190936);山东省高等学校青创科技支持计划(2019KJN024);山东省博士后创新专项基金(201603056);国家海洋局海洋遥测工程技术研究中心开放基金(2018002);山东科技大学领军人才与优秀科研创新团队项目(2015TDJH102)

Foundation items: Project supported by the National Natural Science Foundation,China (61602279,61472229), National Key Research and Development Plan (2016YFC0801406); the Taishan Scholars Program of Shandong Province, China (ts20190936); the Excellent Youth Innovation Team Foundation of Shandong Higher School (No.2019KJN024); the Postdoctoral Innovation Foundation of Shandong Province, China (201603056); the Open Foundation of First Institute of Oceanography,China (2018002); the Shandong University of Science and Technology Research Fund, China (2015TDJH102).

收稿时间: 2020-08-31; 修改时间: 2020-10-26; 采用时间: 2020-12-19; jos 在线出版时间: 2021-02-07

此,McMillan[9]首次提出利用网的展开描述系统行为,并通过分支进程[10]与偏序技术构造展开网的有限完备前缀,有效缓解了 Petri 网性质分析中的状态爆炸问题,由此受到了研究者们的广泛关注.

Esparza 等人[11]指出 McMillan 定义的偏序关系在某些情况下会导致有限完备前缀的规模呈指数级增长,并针对安全 Petri 网提出了一种全序关系,最小化有限完备前缀的规模.Khomenko 等人[12]规范了展开的定义,并对展开进行参数化配置.Heljanko 等人[13]将展开技术并行化,提高了展开的效率.Benito 等人[14]将展开技术扩展到时间 Petri 网.Schwarick 等人[15]将展开技术扩展到于颜色 Petri 网.在 Petri 网展开技术的应用方面,Lu 等人[5]针对网系统的死锁检测提出一种无界 Petri 网的有限展开技术.Xiang 等人[6]基于展开技术对并发系统中的数据不一致问题进行检测.Dong 等人[7]基于 Petri 网的可达图对 CTL 进行验证.Liu 等人[8]借助分支进程技术检测工作流的健壮性,并在[32]中基于展开技术检测并发系统的健壮性、兼容性与死锁.在 Petri 网的性质分析方面,Chatain 等人[16]针对 Petri 网的可覆盖性问题设计了一种目标导向的展开技术,通过分析内部因果关系对冗余变迁进行剪枝.Bonet 等人[17] 结合启发式技术提出了一种 semi-adequate ordering,提升了展开技术在 Petri 网可覆盖性分析上的效率,并在后续的论文[18]中证明扩展顺序可以与截断事件的偏序分离,拓宽了启发式展开技术在 Petri 网性质分析中的应用.

网的展开通过分支进程与偏序技术可在一定程度上缓解 Petri 网性质分析中的状态爆炸问题.但展开网中仍然包含了系统的所有状态信息.某些应用问题仅需对系统特定状态标识的可覆盖性进行判定,以此为目标有望对网系统的展开进行化简.为此,本文针对安全 Petri 网的可覆盖性判定问题提出了一种目标导向的反向展开算法.反向展开从需要作出可覆盖性判定的目标标识出发,仅刻画与可覆盖性判定相关的系统状态,并结合启发式技术缩减展开的规模,以此提高目标标识可覆盖判定的效率.进而,将反向展开算法应用于并发程序的形式化验证,将并发程序的数据竞争检测问题转换为Petri网特定标识的可覆盖性判定问题. 实验对比了启发式反向展开与 directed unfolding[17](一种同样采用了启发式技术的正向展开)在 Petri 网可覆盖性问题上的效率,结果表明在 415 组测试数据中,反向展开的规模在 85 组数据上优于 directed unfolding,在 26 组数据上与 directed unfolding 持平.最后,本文对影响反向展开效率的关键因素做了分析与总结.

文章的组织结构如下:第 2 节围绕 Petri 网的可覆盖性判定问题，分别分析正向展开与反向展开的适用场景,并结合实例说明反向展开相比正向展开的优势;第 3 节介绍 Petri 网的反向展开算法,包括相关概念的形式化定义、算法流程以及启发式优化策略;第 4 节将反向展开应用于并发程序的数据竞争检测;第 5 节结合实验评估正向展开与反向展开在 Petri 网可覆盖性判定问题上的效率;第 6 节对文章所做的工作进行总结与展望.

2 实例与动机分析

本节首先在 Petri 网的可覆盖性判定问题上分别论述正向展开与反向展开的适用场景.之后将通过一个实例说明反向展开相比正向展开的优势.

先来看两个简单的例子.图 1(a)中 Petri 网的初始标识为{ ps },需验证目标标识{ pt }的可覆盖性.为了便于表述,将图 1(a)左侧的路径称为 $path1$,右侧的路径称为 $path2$.在这个例子中,正向展开从初始标识{ ps }出发,难以在 $path1$ 与 $path2$ 间做出选择,若不慎选择了 $path2$ 将会做很多冗余的扩展.与正向展开不同,反向展开从目标标识 $\{pt\}$ 出发,仅需要沿着 $path1$ 反向扩展,很容易找到与初始标识{ ps }间的可达路径.

图 1(b)可以看作图 1(a)的"倒置".在这个例子中,反向展开从目标标识 $\{pt\}$ 出发,难以在 $path1$ 与 $path2$ 间做出选择.而正向展开从初始标识{ ps }出发,很容易沿着 $path1$ 到达目标标识 $\{pt\}$.

(a) 反向展开适用场景 (b) 正向展开适用场景

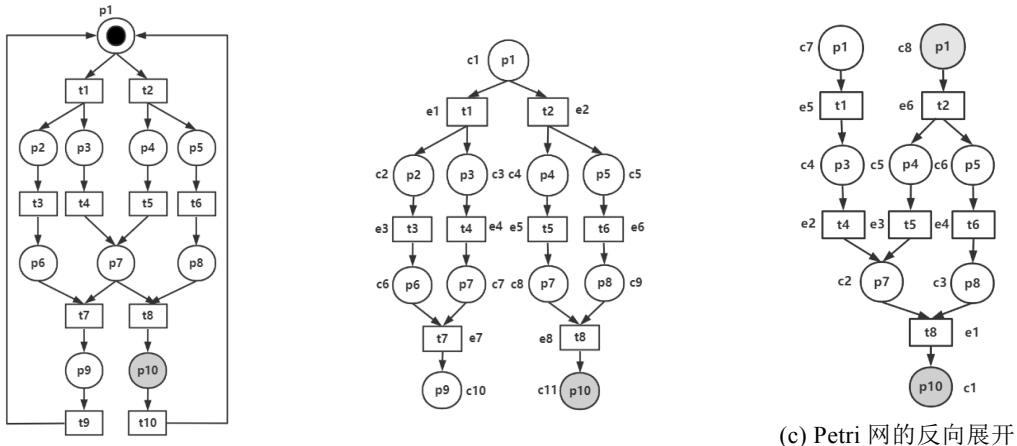
Fig.1 Suitable scenarios for forward unfolding and reverse unfolding

图 1 正向展开与反向展开的适用场景

从上述两个例子中可观察到当 Petri 网的正向分支较多时反向展开往往更加适用;当 Petri 网的反向分支较多时正向展开往往更加适用.具体而言, 正向展开从 Petri 网的初始标识出发刻画系统,隐含了系统完整行为.而

反向展开从需要作出可覆盖性判定的目标标识出发,仅刻画与可覆盖性判定相关的系统状态.本文正是从这个角度出发设计并实现了反向展开算法.

接下来将结合一个实例进一步说明反向展开相比正向展开的优势.图 2(a)中 Petri 网的初始标识为$\{p1\}$,需验证目标标识$\{p10\}$的可覆盖性.图 2(b)是 Petri 网的正向展开[9].图 2(c)是 Petri 网的反向展开.二者都采用基于广度优先策略的 adequate order 作为扩展顺序,一旦验证目标标识的可覆盖性就终止扩展.在这个例子中,正向展开共产生 19 个节点、20 条流关系,反向展开总共产生 14 个节点、14 条流关系,反向展开的规模优于正向展开.这是因为变迁$\{t1,t3,t4,t7\}$对于$\{p10\}$可覆盖性的贡献是冗余的(即"左侧"的路径是冗余的).正向展开隐含了系统的完整行为,难以避免对$\{t1,t3,t4,t7\}$的冗余行为分析.而反向展开仅刻画与可覆盖性判定相关的系统状态,避免了对$\{t3,t7\}$的冗余刻画.尽管反向展开中仍有冗余,但总体规模优于正向展开.



(a) Petri 网实例 (b) Petri 网的正向展开 (c) Petri 网的反向展开

Fig.2 An example where reverse unfolding is better than forward unfolding

图 2 反向展开优于正向展开的实例

本节论述了正向展开与反向展开各自的优势.值得一提的是,这种相互之间的优势是由算法的性质决定的,具备一定的一般性.像 goal-driven unfolding [16]、directed unfolding[17]等正向展开,尽管它们会使用内部因果关系进行剪枝、亦或是使用启发式技术优化算法效率,但在正向分支过多的情况下还是难以避免对系统冗余行为的分析.同样,即使反向展开配备了启发式技术,在反向分支过多时也难以保证算法的效率.在这个前提下,本节进而通过实例表明反向展开在某些情况下优于正向展开.接下来将详细介绍反向展开算法.

3 Petri 网的反向展开

3.1 反向展开的概念

3.1.1 Petri 网

一个网可以定义为一个三元组(P,T,F),P为库所集,T为变迁集,F为P与T间的流关系,$F\subset(P\times T)\cup(T\times P)$.节点$x$的前集定义为${}^{\bullet}x=\{y\in P\cup T\mid F(y,x)=1\}$,节点$x$的后集定义为$x^{\bullet}=\{y\in P\cup T\mid F(x,y)=1\}$.网$(P,T,F)$的标识是建立在$P$上的一个多重集.在图形化表示时,网的标识通过在各个库所中添加相应数量的 token 来表示.

一个网系统可以定义为一个四元组(P,T,F,M_0),其中M_0是网(P,T,F)的初始标识.如果$\forall p\in P:F(p,t)\le M(p)$,则标识$M$下变迁$t$使能,使能的变迁可以执行.$t$的执行使系统进入一个新的标识$M'$,记作$M\xrightarrow{t}M'$,即对每个库所$p$,$M'(p)=M(p)-F(p,t)+F(t,p)$.一个变迁序列$\sigma=t_1\,t_2\ldots t_n$称之为触发序列当且仅当存在标识$M_1,M_2,\ldots,M_{n-1},M_n$使得$M_0\xrightarrow{t_1}M_1\xrightarrow{t_2}\ldots\xrightarrow{t_{n-1}}M_{n-1}\xrightarrow{t_n}M_n$成立.上式也可以表示为$M_0\xrightarrow{\sigma}M_n$,称$M_n$是触发序列$\sigma$下的可达标识.

对于标识 M_f,若有标识 M, M' 和触发序列 σ,使得 $M \xrightarrow{\sigma} M' \wedge M_f \subseteq M'$,则称 M_f 可被 M 覆盖,记为 $M \mapsto M_f$.

若可达标识 M 满足 $\forall p: M(p) \le n$,则称 M 是 n-safe 的.一个网系统是 n-safe 的当且仅当其所有可达标识都是 n-safe 的.特别地,将 1-safe 网系统称为安全网系统.本文只考虑安全 Petri 网的可覆盖性问题,并记可覆盖性判定的目标标识为 M_f,即验证 $M_0 \mapsto M_f$ 是否成立.

3.1.2 反向出现网

定义 1 反向出现网 反向出现网是出现网的子类,用以判定一个 Petri 网中目标标识 M_f 的可覆盖性,对应一个四元组 $RON=(C,E,F',CM_f)$. 当中 C 为条件集,每个条件对应 Petri 网库所中的一个 token; E 为事件集,每个事件对应 Petri 网中某变迁的一次执行; F' 为 C 与 E 间的流关系,对应 Petri 网的流关系; CM_f 为 Petri 网目标标识 M_f 在 RON 中对应的条件集合,满足 $\forall c \in CM_f : c^{\bullet} = \varnothing$.

以图 3(a)中的 Petri 网与目标标识 $M_f=\{p4\}$ 为例,它对应的反向出现网见图 3(b),当中 $CM_f=\{c1\}$.

RON 上的两个节点 x_1, x_2 有以下三种关系:

(1) 反向因果:若从 x_1 出发的某条路径可到达 x_2,则记 $x_2 \le x_1$.特别的,对于任意节点 x,有 $x=x$.

图 3(b)中,有 $c1 \le c5$, $e2 \le e3$.

(2) 反向冲突:若存在两个不同的事件 $e_1, e_2 \in E$, $e_1^{\bullet} \cap e_2^{\bullet} \ne \varnothing \wedge e_1 \le x_1 \wedge e_2 \le x_2$,则称 x_1 与 x_2 反向冲突,记为 $x_1 \# x_2$.

图 3(b)中 $c2$ 与 $c3$ 反向冲突,这是因为 $e1 \le c2$, $e2 \le c3$,且 $e1^{\bullet} \cap e2^{\bullet} = \{c1\} \ne \varnothing$.同理 $c2$ 与 $c4$ 反向冲突、 $e1$ 与 $e2$ 反向冲突.

(3) 反向并发:若 $\neg(x_1 \le x_2 \vee x_2 \le x_1 \vee x_1 \# x_2)$,则称 x_1 与 x_2 反向并发,记为 $x_1 \parallel x_2$.

图 3(b)中 $c3$ 与 $c4$ 反向并发.

RON 满足以下三个性质:

(1) $\forall c \in C : |c^{\bullet}| \le 1$;

(2) RON 中没有反向自我冲突,即不存在事件 $e \in E$,使得 $e \# e$;

(3) F' 是无环的,即 F' 的反自反传递闭包是一个偏序.

定义 2 反向配置 RON 中的反向配置 Cfg 为若干个事件的集合,满足以下两个性质:

(1) 若事件 $e \in Cfg$,则 $\forall e' < e : e' \in Cfg$;

(2) Cfg 中不存在反向冲突的事件,即 $\forall e, e' \in Cfg : \neg(e \# e')$.

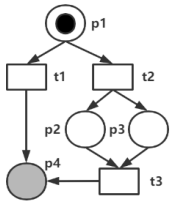
定义 3 反向局部配置 特别地,将 $\{e' \mid e' \in E \wedge e' < e\}$ 定义为事件 e 的反向局部配置,记作 $[e]$.

图 3(b)中 $[e1]=\{e1\}$, $[e3]=\{e2,e3\}$.

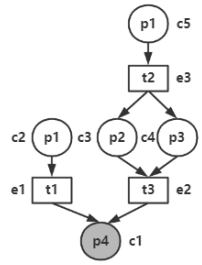
定义 4 反向切 对于配置 Cfg,定义其反向切为 $Cut(Cfg)=(CM_f \cup {}^{\bullet}Cfg) \setminus Cfg^{\bullet}$.另外,对于一个条件集合,若其各元素之间两两反向并发,则称之为一个 co-set.不难发现 $Cut(Cfg)$ 是一个 co-set.

图 3(b)中, $Cut([e2])=\{c3,c4\}$, $Cut([e3])=\{c5\}$.

反向配置、反向切会在 3.1.3 中用于建立反向出现网标识与 Petri 网标识间的映射关系.



(a) Petri 网实例



(b) 反向出现网实例

Fig.3 An example of Petri net(a) and its reverse occurrent net(b)
图 3 Petri 网实例(a)及其对应的反向出现网(b)

3.1.3 反向展开

对于给定的 Petri 网 $\Sigma=(P,T,F,M_0)$,需验证目标标识 M_f 的可覆盖性.定义 Σ 和 $RON=(C,E,F',CM_f)$ 各节点间的映射关系 $\mu: C\cup E\to P\cup T$ 如下:

(1) 若 $c\in C$ 则 $\mu(c)\in P$,若 $e\in E$ 则 $\mu(e)\in T$;

(2) $\forall e\in E$, ${}^\bullet e$ 到 ${}^\bullet\mu(e)$ 在 μ 的约束下满足双射关系, $e^\bullet$ 到 $\mu(e)^\bullet$ 在 μ 的约束下满足单射关系.这里与正向展开不同,在反向展开中 $e^\bullet$ 到 $\mu(e)^\bullet$ 可以不满足满射关系;

(3) CM_f 与 M_f 在 μ 的约束下满足双射关系.

定义 5 反向标识 定义 $Mark(Cfg)=\mu(Cut(Cfg))$ 为配置 Cfg 的反向标识. $Mark(Cfg)$ 可看做 RON 的一个中间标识, M_f 的可覆盖性判定可转化为 $Mark(Cfg)$ 的可覆盖性判定[19].

图 3(b)中, $Mark([e2])=\mu(\{c3,c4\})=\{p2,p3\}$, $Mark([e3])=\mu(\{c5\})=\{p1\}$.

定义 6 反向展开 在上述概念的基础上,Petri 网 Σ 中目标标识 M_f 的反向展开定义为一个二元组 $RUnf=(RON,\mu)$,满足以下性质:

(1) $RUnf$ 是完备的:记 Σ 的初始标识 M_0,若 $M_0\mapsto M_f$,则在 $RUnf$ 存在一个反向配置 Cfg,使得 $Mark(Cfg)\subseteq M_0$.这里只需要保证 M_f 的可覆盖性不被破坏即可,不必获取全部触发序列;

(2) $RUnf$ 是有限的,即 $RUnf$ 中包含有限的条件与事件.

3.2 反向展开算法

给定 Petri 网 $\Sigma=(P,T,F,M_0)$ 和需要进行可覆盖性判定的目标标识 M_f,反向展开算法的基本思想如下:首先,针对 M_f 中每个库所,在 $RUnf$ 中添加一个相应的条件,即创建 CM_f;之后,从 CM_f 出发逐步反向扩展,添加能生成这些条件的事件以及这些事件使能所需的条件,并对冗余的事件进行截断;如此重复,直到一个反向标识能被 Σ 的初始标识所覆盖,或者能证明目标标识不可覆盖.

接下来将结合图 4 所示的实例,给出反向扩展与反向截断事件的准确定义,并介绍目标标识可覆盖性的判定方法.假设图 4 中需进行可覆盖性判定的目标标识为 $\{p6,p7\}$.

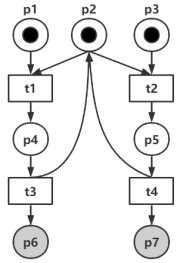


Fig.4 An example of Petri net
图 4 Petri 网实例

3.2.1 反向扩展

定义 7 反向扩展 反向扩展为一个二元组 $rext=(t,C)$,当中的 C 是一个 co-set, $\mu(C)\subseteq t^\bullet$,记反向扩展组成的集合为 $RExt$.

对每一个从 $RExt$ 中选中并用以扩充 $RUnf$ 的反向扩展 $rext=(t,C)$ 而言,一方面需要向 $RUnf$ 中添加一个事件 $e=(t,C)$;同时,需要为 ${}^\bullet t$ 中的每个库所添加一个条件至 $RUnf$ 中.之后重新计算 $RExt$,记此过程为 $NE(RUnf,e)$.

$NE(RUnf,e)$ 计算候选扩展集的过程如下: ${}^\bullet e$ 的加入会为 $RUnf$ 添加新的 co-set,对每个新增的 co-set C,若有变迁 t 满足 $\mu(C)\subseteq t^\bullet$,则添加一个新的候选扩展 $rext=(t,C)$ 到 $RExt$ 中.

按照上述定义, *RExt* 中会存在大量冗余扩展.例如,假设 $\{c1,c2,c3\}$ 是一个 co-set,那么按照 co-set 的定义 {c1,c2}、{c1,c3}、{c2,c3}、{c1}、{c2}、{c3}都是 co-set.若有反向扩展 $(t,\{c1,c2,c3\})$, 那么同样会有反向扩展 $(t,\{c1,c2\})$、$(t,\{c1,c3\})$、$(t,\{c2,c3\})$、$(t,\{c1\})$、$(t,\{c2\})$、$(t,\{c3\})$.可见不加额外限制的话 *RExt* 的规模将会非常大.因此,为 *RExt* 添加如下两个条件:

(1) 对于 *RUnf* 中任意事件 e, *RExt* 中不存在反向扩展 $rext=(t,C)$,使得 $\mu(e)=t\wedge e^{\bullet}=C$;

(2) 对于 *RExt* 内任意两不同的反向扩展 $rext_1=(t_1,C_1)$, $rext_2=(t_2,C_2)$, 若 $t_1=t_2\wedge C_1\subset C_2\wedge Mark([rext_1])\geq Mark([rext_2])$,则将扩展 $rext_1$ 从 *RExt* 中删除.这里假设 e 为 $rext$ 在 *RUnf* 中对应的事件,$[rext]$ 可视为 $[e]$.

条件(2)中 $Mark([rext_1])\geq Mark([rext_2])$ 看上去有些多余.实际上,Parosh 早在[19]中就提出过反向展开的概念,且仅用到了条件(2)中 $t_1=t_2\wedge C_1\subset C_2$ 这个限制.遗憾的是,这样做会破坏反向展开的完备性,致使 Parosh 的反向展开算法在某些情况下出错,附录 B 给出了相关反例的分析.为此,本文添加了 $Mark([rext_1])\geq Mark([rext_2])$ 这个限制,大量实践表明这个限制是有效的.

图 5 中,生成事件 $e2$ 时用到了反向扩展 $rext=(t1,\{c3\})$,同时生成了 $e2$ 的前驱条件 $\{c4,c5\}$.$\{c4,c5\}$ 的加入产生了新的 co-set, 并产生了相应的反向扩展 $(t3,\{c5\})$、$(t4,\{c5\})$、$(t4,\{c2,c5\})$. 其中, 对于 $rext_1=(t4,\{c5\}), rext_2=(t4,\{c2,c5\})$,有 $Mark([rext_1])=\{p1,p5,p7\}\geq\{p1,p7\}=Mark([rext_2])$,由 *RExt* 的条件(2)删除 $rext_1$.故最终 $NE(RUnf,e)=\{(t3,\{c5\}),(t4,\{c2,c5\})\}$.

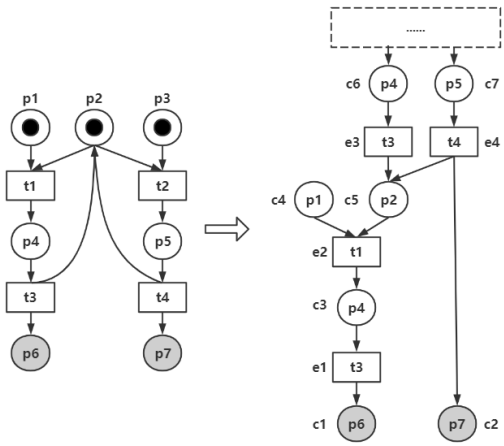


Fig.5 Reverse unfolding of the Petri net shown in Figure 4 (partial)

图 5.对图 4 所示的 Petri 网进行反向扩展(部分)

3.2.2 反向截断事件

前述反向扩展规则能确保 *RUnf* 的完备性,但无法保证其有限性.为此,下文将给出反向截断事件的识别规则,据此保证反向展开过程的可终止性.

定义 8 反向截断事件 事件 e 为反向截断事件当且仅当 *RUnf* 中存在事件 e' 满足以下两个条件:

(1) $Mark([e'])\leq Mark([e])$;

(2) $[e']\prec[e]$.

其中 $\prec$ 是定义在配置上的偏序,称为 adequate order,满足以下三个条件:

(1) $\prec$ 是良基的;

(2) $\prec$ 是 $\subset$ 的精炼, $Cfg_1\subset Cfg_2$ 意味着 $Cfg_1\prec Cfg_2$;

(3) 若 $Mark(Cfg_1)\leq Mark(Cfg_2)$, $Cfg_1\prec Cfg_2$, 则对 Cfg_2 的任意前缀 E_2 有 E_1 满足

$Mark(Cfg_1 \oplus E_1) \le Mark(Cfg_2 \oplus E_2)$,且 $Cfg_1 \oplus E_1 \prec Cfg_2 \oplus E_2$.其中,对于一个配置 Cfg, $Cfg \oplus E$ 表示存在事件集 E 使得 $Cfg \cap E = \varnothing$,且 $Cfg \cup E$ 也是一个配置. E 也叫作 Cfg 的前缀.

本文使用 $\prec_r$ 作为 adequate order,相关证明可参考附录 A 中的定理 1. $Cfg_1 \prec_r Cfg_2$ 定义为:

(1) $|Cfg_1| < |Cfg_2|$;

(2) $|Cfg_1| = |Cfg_2| \wedge Lex(\mu(Cfg_1)) < Lex(\mu(Cfg_2))$.其中 $\mu(Cfg)$ 是配置 Cfg 映射到 $\sum$ 上的变迁集合,这个变迁集合是一个多重集. $Lex(\mu(Cfg))$ 将 $\mu(Cfg)$ 中的变迁按照 ID 从小到大排序.可以理解为当两个配置大小相同时,比较其对应的变迁集合的字典序.

图 5 中的事件 $e3$ 将因事件 $e1$ 而截断,如图 6 所示.这是因为 $\|[e1]\| < \|[e3]\|$,即 $[e1] \prec_r [e3]$,且 $Mark([e1]) = \{p4, p7\} \le \{p1, p4, p7\} = Mark([e3])$.

反向扩展和反向截断事件保证了 $RUnf$ 的完备性与有限性,相关证明可参考附录 A 中的定理 2 与定理 3.

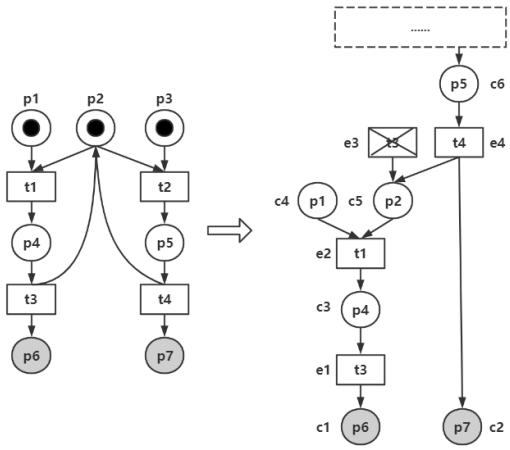


Fig.6 Event e3 is cut off by event e1

图 6. 事件 $e3$ 因事件 $e1$ 而截断

3.2.3 反向展开的可覆盖性判定

本文只考虑安全 Petri 网,可以在 Petri 网上添加源库所 ps,源变迁 ts,流关系 (ps, ts),并对 $\forall p \in M_0$ 添加流关系 (ts, p),从而将求解 $M_0 \mapsto M_f$ 转化为求解 $\{p_s\} \mapsto M_f$.这样仅需判断 $RUnf$ 中是否存在一个配置 Cfg 使得 $Mark(Cfg) = \{p_s\}$ 即可验证 M_f 的可覆盖性.以图 4 中的 Petri 网为例,添加源节点后的 Petri 网见图 7.

截至目前已介绍了反向扩展、反向截断事件以及反向展开的可覆盖性判定.接下来将对这些概念进行整合,结合实例介绍反向展开算法的流程.

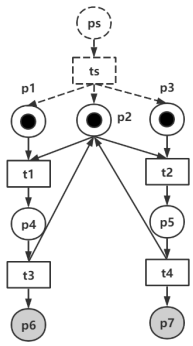


Fig.7 Petri net after adding source nodes

图 7 添加源节点后的 Petri 网

3.2.4 反向展开算法与可覆盖性判定实例

反向展开可覆盖性判定的流程可以总结为:初始时, $RUnf$ 中只有与 M_f 相对应的条件集合 CM_f ,算法会基于 CM_f 计算初始扩展集 $RExt$.随后,只要 $RExt$ 不为空则算法将持续进行反向扩展.每次扩展都会从 $RExt$ 中随机取出一个 $rext=(t,C)$,并在 $RUnf$ 中创建相应的事件 $e=(t,C)$.若 e 不为反向截断事件,则对 ${}^{\bullet}t$ 中的每个库所 p 在 $RUnf$ 中创建相关的条件 $c=(p,e)$. 之后通过计算 $NE(RUnf,e)$ 更新 $RExt$.若存在一个事件 e 使得 $Mark([e])=\{ps\}$,则说明 M_f 可覆盖,终止算法.若最终 $RExt$ 为空且没有事件 e 满足 $Mark([e])=\{ps\}$,则说明 M_f 不可覆盖.反向展开算法的伪代码如下:

Petri 网的反向展开与目标标识可覆盖性判定算法

Input: A net system $\Sigma=(P,T,F,\{p_s\})$, with target $M_f=\{p_1,p_2,...,p_n\}$

Output: The coverability of M_f

$RUnf=\{(p_1,\varnothing),(p_2,\varnothing),...,(p_n,\varnothing)\}$

$RExt=NE(RUnf,\varnothing)$

while $RExt\neq\varnothing$ **do**

poll an extension $rext=(t,C)$ from $RExt$ randomly

create an event $e=(t,C)$ in $RUnf$

if e is not a reverse cut-off event **then**

for $\forall p\in{}^{\bullet}t$, append a condition $c=(p,e)$ to $RUnf$

if $Mark([e])=\{ps\}$ **then**

return true

endif

$RExt=RExt\cup NE(RUnf,e)$

endif

endwhile

return false

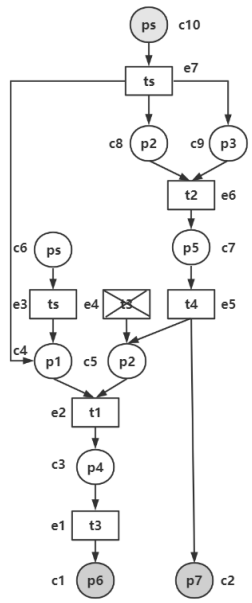
下面结合图 7 的 Petri 网实例展示反向展开判定目标标识{p6,p7}可覆盖性的完整流程,如表 1 所示.

Table 1 Example of reverse unfolding

表 1 反向展开实例分析

反向展开流程	RExt	RUnf

setp 0.初始 RUnf 中只有条件 c1、c2.	{rext1=(t3,{c1}),rext2=(t4,{c2})} {rext2=(t4,{c2}),rext3=(t1,{c3})}
step 1.选择 rext1 进行扩展,在 RUnf 中创建事件 e1=(t3, {c1})及其前驱条件{c3}.	
step 2.选择 rext3 进行扩展,在 RUnf 中创建事件 e2=(t1,{c3})及其前驱条件{c4, c5},生成新扩展(ts,{c4}),(t3,{c5}),(t4,{c5}),(t4,{c2,c5}).由 REXt 的限制条件删除扩展(t4,{c5}).	{rext2=(t4,{c2}),rext4=(ts,{c4}),rext5=(t3,{c5}), rext6=(t4,{c2, c5})}
step 3.选择 rext4 进行扩展,在 RUnf 中创建事件 e3=(ts, {c4})及其前驱条件{c6},此时已经没有新的扩展.	{rext2=(t4,{c2}),rext5=(t3,{c5}),rext6=(t4,{c2,c5})}
step 4.选择 rext5 进行扩展,在 RUnf 中创建事件 e4=(t3, {c5}),由反向截断事件的定义,e4 会因 e1 而截断,此次扩展中断.	{rext2=(t4,{c2}), rext6=(t4,{c2, c5})}
step 5.选择 rext6 进行扩展,在 RUnf 中创建事件 e5=(t4, {c2,c5})及其前驱条件{c7}.	{rext2=(t4,{c2}), rext7=(t2,{c7})}
step 6.选择 rext7 进行扩展,在 RUnf 中创建事件 e6=(t2,{c7})及其前驱条件{c8,c9},生成新扩展(ts,{c8}),(ts,{c9}),(ts,{c8,c9}),(ts,{c4,c8}),(ts,{c4,c9}),(ts,{c4,c8,c9}),(t3,{c8}),(t4,{c8}),由 REXt 的限制条件只保留(ts,{c4,c8,c9}),(t3,{c8}),(t4,{c8}).	{rext2=(t4,{c2}),rext8=(ts,{c4,c8,c9}), rext9=(t3,{c8}),rext10=(t4,{c8})}
step 7.选择 rext8 进行扩展,在 RUnf 中创建事件 e7=(ts,{c4,c8,c9})及其前驱条件{c10},此时目标标识可覆盖,算法结束.	



以上就是整个反向展开算法的流程,然而算法目前并没有规定扩展顺序,仅采用随机扩展.实际上扩展顺序对反向展开算法的效率有着决定性的影响.接下来将使用启发式技术对反向展开做优化.

3.3 反向展开算法的启发式优化

扩展顺序对目标标识可覆盖性验证的效率有着关键影响, 为此本文基于实践总结并参考文献[17]提出了三种启发式技术:

(1) *block* 策略.在反向扩展 $rext=(t,C)$ 中, C 到 ${}^{\bullet}t$ 可以不满足满射关系.然而实践表明 C 到 ${}^{\bullet}t$ 不满足满射时往往意味着 *rext* 产生得过早,相应的条件还没来得及产生,此时由其引导的扩展都将会是冗余扩展.为了尽可能避免这种情况发生,将不满足满射关系的 *rext* 加入阻塞队列,优先选择满足满射关系的反向扩展.仅当所有反向扩展都不满足满射关系时才会在阻塞队列中选择一个 *rext* 激活. 实践中 *block* 策略通常会配合其他启发式策略共同使用.

(2) *hmax* 策略. *hmax* 策略是 Bonet 在文献[17]中提出的一种基于距离的启发式策略.定义 $d(M,M')$ 为标识 M 到 M' 间的距离. *hmax* 将 M 到变迁 t 的距离定义为 $max_{p\in {}^{\bullet}t}d(M,\{p\})$,将 M 到库所 p 的距离定义为 $1+min_{t\in {}^{\bullet}p}d(M,{}^{\bullet}t)$,如此递推.计算 $d(M,M')$ 时只需求 M 到 M' 中每个库所的最大距离,即计算 $max_{p\in M'}d(M,\{p\})$. 上述过程可总结为:

$$d(M,M')=\begin{cases}0 & M'\subseteq M\\ 1+\min_{t\in {}^{\bullet}p}d(M,{}^{\bullet}t) & M'=\{p\}\\ \max_{p\in M'}d(M,\{p\}) & otherwise\end{cases}$$

(3) *hsum* 策略. *hsum* 与 *hmax* 十分相似.不同的是 *hsum* 将 M 到变迁 t 的距离定义为 $\sum_{p\in {}^{\bullet}t}d(M,\{p\})$,与之对应的计算 $d(M,M')$ 时统计 M 到 M' 中每个库所的距离总和,即计算 $\sum_{p\in M'}d(M,\{p\})$.上述过程可总结为:

$$d(M,M')=\begin{cases}0 & M'\subseteq M\\ 1+\min_{t\in {}^{\bullet}p}d(M,{}^{\bullet}t) & M'=\{p\}\\ \sum_{p\in {}^{\bullet}t}d(M,\{p\}) & otherwise\end{cases}$$

hmax 策略与 *hsum* 策略在使用时会结合配置大小.具体而言, 对于两个反向扩展 $rext_1=(t_1,C_1)$ 、

$rext_2=(t_2,C_2)$,若有 $\|[rext_1]\|+d(M_0,Mark([rext_1]))<\|[rext_2]\|+d(M_0,Mark([rext_2]))$,则优先选择 $rext_1$ 进行扩展.

4 基于反向展开的数据竞争检测

由于线程调度的不确定性,多线程程序往往伴随着数据竞争.数据竞争是指在非线程安全的情况下,多个线程对同一内存地址空间进行访问.数据竞争会影响程序的运行结果,甚至会导致系统崩溃.由于数据竞争通常只在一些特定的线程轨迹中出现,这给开发人员检测数据竞争带来了极大挑战.历史上几起严重的事故,如Therac-25 放射治疗设备事故[20],2003 年北美的大规模停电[21],以及纳斯达克的 FaceBook 故障[22]都与数据竞争有关.数据竞争检测主要分为静态检测[23-25]与动态检测[26-28]两类.本节将基于 Petri 网的反向展开对 Java 并发程序中的数据竞争进行静态检测.

4.1 程序Petri网模型的构建

Krishna[29]针对 C-Pthread 程序提出了其同步原语、流程控制语句的 Petri 网模型构建方法.本文将这些建模方法应用到 Java 并发程序上,主要考虑以下 4 种语句的模型构建:

(1) 线程启动与合并:Java 中线程 t 的启动与合并分别对应 t.start()、t.join().调用 t.start()后,线程 t 的状态变为 Runnable,在得到 CPU 调度后变为 Running 状态正式运行.调用 t.join()后,当前线程的状态变为 Blocked,直到线程 t 执行完毕后才会变为 Runnable 状态重新等待 CPU 调度.线程的启动与合并对应的 Petri 网模型如图 8(a)所示.

(2) 锁的申请与释放:Java 中使用 synchronized 语句实现线程的互斥,包括同步方法和同步代码块,二者本质上是相同的.当线程 t1 与线程 t2 同时访问同步代码块时,只有一个线程可获取访问权限.锁的申请与释放对应的 Petri 网模型如图 8(b)所示.

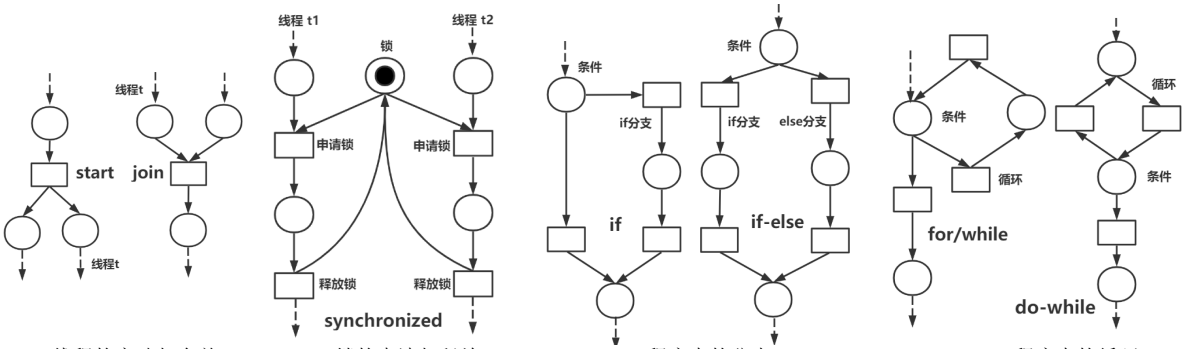
(3) 分支:Java 程序中的分支在逻辑上都等价于 if、if-else 型分支.对应的 Petri 网模型如图 8(c)所示.

(4) 循环:Java 程序中的循环分为 for/while 型循环和 do-while 型循环,二者的结构相似,都由一个控制条件和一个循环体构成.对应的 Petri 网模型如图 8(d)所示.

图 9 给出了一个 Java 多线程程序的实例以及由上述规则构造所得的程序 Petri 网模型.其中带星号的变迁对应程序语句,其余变迁仅用于表达程序结构.具体而言:

(1) 程序语句方面,$t1$ 代表创建线程 t1,$t6$ 代表创建线程 t2,$t14$ 代表将线程 t1 合并到主线程.$t7$、$t8$ 代表申请锁 lock,$t15$、$t16$ 代表释放锁 lock.$t9$ 对应语句 x=2,$t10$ 对应语句 x=1,$t11$ 对应语句 System.out.println(x).

(2) 程序结构方面,$t2$、$t4$ 代表进入 if 结构,$t13$、$t17$ 代表退出 if 结构.$t3$ 代表一轮循环的开始,$t5$ 代表一轮循环的结束,$t12$ 代表退出循环.



(a) 线程的启动与合并 (b) 锁的申请与释放 (c) 程序中的分支 (d) 程序中的循环

Fig.8 Petri net models of Java concurrent programs

图 8 Java 并发程序的 Petri 网模型

```
public static int x = 0;
public static Object lock = new Object();
public static void main(String [] args)
        throws InterruptedException {
    Thread t1 = new Thread() {
        @Override public void run() {
            long time = System.currentTimeMillis();
            if (time % 2 == 0) {
                synchronized(lock) {
                    x = 1;
                }
            } else {
                x = 2;
            }
        }
    };
    Thread t2 = new Thread() {
        @Override public void run() {
            while (true) {
                synchronized(lock) {
                    System.out.println(x);
                }
            }
        }
    };
    t1.start();
    t2.start();
    t1.join();
}
```

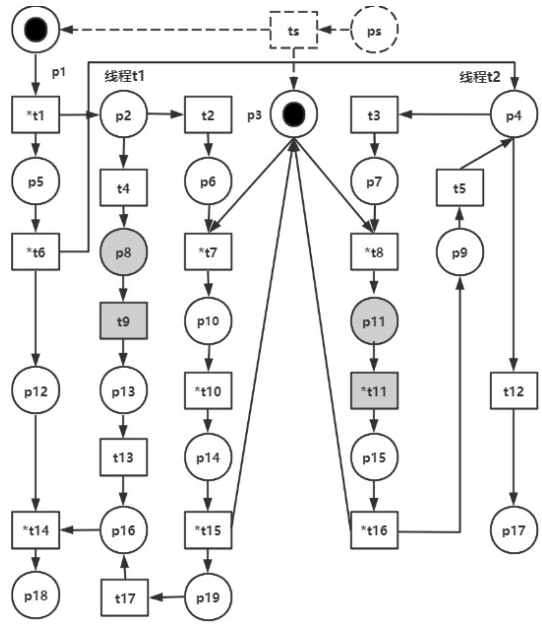


Fig.9 Java source code(left) and its Petri net model(right)
图 9 Java 程序源码(左)与其对应的 Petri 网模型(右)

4.2 基于可覆盖性判定的数据竞争检测

程序中关于同一个共享变量的两条读/写或者写/写操作,如果它们在执行顺序上存在先后依赖关系,则不会产生数据竞争;反之,若他们之间存在并发关系,则会导致数据竞争.由此以来,可以对程序的 Petri 网模型进行分析,判别两个操作对应的变迁是否在某些状态下具有并发关系.实际上,对于任意的两个变迁 t 和 t'，我们只需要判定是否存在一个可达标识 M 使得 $M \geq {}^{\bullet}t + {}^{\bullet}t'$,即判定标识 ${}^{\bullet}t + {}^{\bullet}t'$ 的可覆盖性.

考虑图 9 所示的例子,其中读共享变量 System.out.println(x)与写共享变量 x=2 之间未加同一把锁,会发生数据竞争.共享变量 x 的写操作 x=2 对应变迁 $t9$,读操作 System.out.println(x)对应变迁 $t11$,判定数据竞争只需验证目标标识 $\{p8, p11\}$ 的可覆盖性.

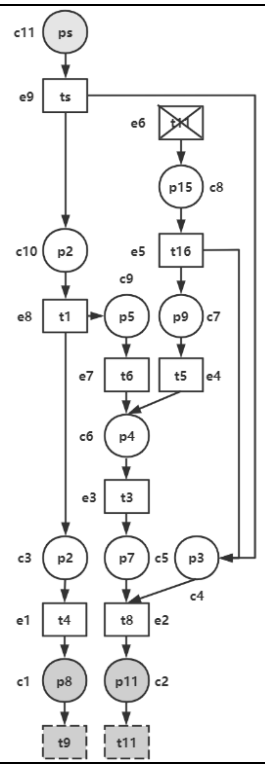
程序 Petri 网模型的反向展开流程如表 2 所示,不难发现, $Mark([e9]) = \{ps\}$,故目标标识 $\{p8, p11\}$ 是可覆盖的,两个读写操作对应的变迁可以并发,程序存在潜在的数据竞争.

本节构建了 Java 并发程序的 Petri 网模型,将数据竞争检测转化为验证相关变迁的可覆盖性,并结合实例展示了反向展开算法在数据竞争检测上的应用. 然而,本文仅给出了 Java 多线程程序的 Petri 网建模方法,并没有设计实际的建模工具.在实际应用中可以考虑使用 Soot[30]对 Java 并发程序自动化建模.Soot 是一款 Java 字节码分析优化框架,可以将 Java 源码转换为 Jimple 中间代码进行分析.比如可以借助 JInvokeStmt 分析线程的启动与合并,以及各种函数调用;借助 JEnterMonitorStmt 与 JExitMonitorStmt 分析锁的申请与释放;借助 JIfStmt 与 JGotoStmt 分析程序中的分支与循环.篇幅所限,我们将在后续工作对此展开研究.

Table 2 Example of concurrent program data race detection
表 2 并发程序数据竞争检测实例

反向展开流程	RExt	RUnf

step 0.初始 RUnf 中只有 c1,c2.	{**rext1**=(t4,{c1}),**rext2**=(t8,{c2})}
step 1.随机选择 rext1 进行扩展,创建 e1,c3.	{**rext2**=(t8,{c2}),rext3=(t1,{c3})}
step 3.选择 rext2 进行扩展,创建 e2,c4,c5.	{rext3=(t1,{c3}),rext4=(t15,{c4}), rext5=(t16,{c4}),rext6=(ts,{c4}), **rext7**=(t3,{c5})}
step 4.选择 rext7 进行扩展,创建 e3,c6.	{rext3=(t1,{c3}),rext4=(t15,{c4}), rext5=(t16,{c4}),rext6=(ts,{c4}), rext8=(t6,{c6}),**rext9**=(t5,{c6})}
step 5.选择 rext9 进行扩展,创建 e4,c7.新增扩展(t16,{c7}),(t16,{c4,c7}),由 RExt 的条件删除(t16,{c7})以及 rext5.	{rext3=(t1,{c3}),rext4=(t15,{c4}), rext6=(ts,{c4}),rext8=(t6,{c6}), **rext10**=(t16,{c4,c7})}
step 6.选择 rext10 进行扩展,创建 e5,c8.	{rext3=(t1,{c3}),rext4=(t15,{c4}), rext6=(ts,{c4}),rext8=(t6,{c6}), **rext11**=(t11,{c8})}
step 7.选择 rext11 进行扩展,创建 e6.由反向截断事件的定义,e6 会因 RUnf 的初始标识而截断,此次扩展中断.	{rext3=(t1,{c3}),rext4=(t15,{c4}), rext6=(ts,{c4}),**rext8**=(t6,{c6})}
step 8.选择 rext8 进行扩展,创建 e7,c9.新增扩展(t1,{c9}),(t1,{c3,c9}),由 RExt 的条件删除(t1,{c9}).	{rext3=(t1,{c3}),rext4=(t15,{c4}), rext6=(ts,{c4}),**rext12**=(t1,{c3,c9})}
step 9.选择 rext12 进行扩展,创建 e8,c10.新增扩展(ts,{c10}),(ts,{c4,c10}),由 RExt 的条件删除(ts,{c10})以及 rext6.	{rext3=(t1,{c3}),rext4=(t15,{c4}), **rext13**=(ts,{c4,c10})}
step 10.选择 rext13 进行扩展,创建 e9,c11.此时目标标识可覆盖,算法结束.	



5 实验评估

本节将通过实验对比启发式反向展开与 directed unfolding[17](一种同样采用了启发式技术的正向展开)在 Petri 网可覆盖性问题上的效率.本文将通过事件总数及运行时间评估算法效率.实验所用的基准库包含 415 组测试用例,包含 directed unfolding 的 358 组用例以及我们自己构造的 57 组用例,且所有用例均可覆盖.其中 Petri 网的变迁数量从 6 到 9462 不等,共计 261694 个变迁.实验环境为 AMD Ryzen 7 4800U、16GB RAM.源码及测试用例可通过[31]获取.

5.1 无启发式策略时反向展开与正向展开的对比

由于启发式策略对算法效率有很大影响,不利于直观地体现反向展开在性质上的优势.因此,本文首先采用基础的广度优先和深度优先策略对比两种算法(正向展开参考文献[18]).实验所用的用例集 randomtree 随机构建了 37 个树状结构的 Petri 网,树中每个节点都是类似图 8(d)的循环模型,并随机选择叶节点中的标识进行可覆盖性判定.规定算法的最大扩展次数为 10000 次,最大运行时间为 35 秒.实验结果如表 3 所示.

结果表明在 randomtree 上基于深度优先策略的反向展开效果最佳,在全部 37 组用例上|E|的规模均最优.同时,无论采用深度优先策略还是广度优先策略,反向展开的效率均优于正向展开.这是因为 randomtree 具备树状结构.正向展开隐含刻画了系统的完整行为,涵盖了树结构的每个分支.而反向展开仅刻画与可覆盖性判定相关的系统状态,只涵盖树的一条分支.

Table 3 Results of forward unfolding and reverse unfolding in randomtree under no heuristic strategy
表 3.无启发式策略下正向展开与反向展开在 randomtree 上的实验结果

		Unf-bfs		Unf-dfs		RUnf-bfs		RUnf-dfs	
测试用例	|T|	|E|	时间(毫秒)	|E|	时间(毫秒)	|E|	时间(毫秒)	|E|	时间(毫秒)
randomtree100	899	500	64	448	125	36	34	**29**	42
randomtree125	1194	318	20	1291	309	23	25	**18**	15
randomtree150	1387	656	98	428	21	38	15	**29**	18
randomtree175	1729	1165	241	1082	115	42	14	**34**	17
randomtree200	1854	886	60	1721	197	36	5	**27**	12
randomtree225	2130	134	3	4640	2575	15	3	**12**	2
randomtree250	2429	1278	106	1125	96	34	44	**26**	4
randomtree275	2575	1390	148	2309	415	43	8	**34**	5
randomtree300	2898	1690	248	1231	132	46	7	**37**	23

randomtree325	3132	1359	143	1238	129	33	22	**28**	4
randomtree350	3492	2128	330	383	20	40	12	**31**	8
randomtree375	3615	2134	404	2082	361	39	5	**31**	5
randomtree400	3712	1658	206	5942	7139	37	4	**31**	4
randomtree425	4076	1166	91	9642	20789	38	3	**29**	4
randomtree450	4304	2030	296	2600	611	45	10	**33**	17
randomtree475	4523	613	30	4801	2654	25	4	**20**	2
randomtree500	4732	1476	141	2887	887	34	24	**27**	5
randomtree525	5014	1828	204	5781	6730	37	4	**30**	4
randomtree550	5160	2059	308	159	4	38	3	**30**	18
randomtree575	5394	1008	77	8720	13284	36	5	**27**	14
randomtree600	5739	2610	483	-	-	42	7	**32**	18
randomtree625	5835	1314	123	891	70	29	4	**24**	7
randomtree650	6238	2760	651	9494	24708	43	4	**31**	4
randomtree675	6423	2829	630	6615	10874	41	4	**31**	5
randomtree700	6529	1427	133	6257	8508	34	4	**26**	5
randomtree725	7026	4759	3183	-	-	63	8	**51**	10
randomtree750	7133	2853	688	2061	280	52	9	**38**	7
randomtree775	7415	5005	5516	2340	525	63	12	**49**	9
randomtree800	7673	294	12	232	7	24	16	**20**	3
randomtree825	7689	2557	665	-	-	37	7	**29**	6
randomtree850	8030	267	10	5549	6056	21	3	**15**	5
randomtree875	8224	2182	373	-	-	34	4	**25**	6
randomtree900	8402	3507	1247	9323	20989	34	5	**28**	8
randomtree925	8762	5305	4789	2551	442	60	8	**47**	10
randomtree950	9135	6247	7353	-	-	75	10	**59**	12
randomtree975	9491	6274	8860	5583	6217	72	10	**52**	11
randomtree1000	9462	6025	8618	7241	12960	59	7	**43**	9

5.2 含启发式策略时反向展开与正向展开的对比

本节将对比启发式反向展开与 directed unfolding 在 Petri 网可覆盖性问题上的效率.在实现 directed unfolding 时,本文采用了文献[17]中的启发式策略 *hmax* 、 *hsum* ,以及文献[18]中扩展顺序可以与截断事件的偏序分离这一性质;在实现反向展开时,本文采用了 3.3 中的启发式策略 *block*、*hmax*、*hsum* ,并在实现细节上尽可能保证二者一致.为便于比较,每组实验仅选择 directed unfolding 与反向展开中最优的启发式策略进行对比.相关实验如下:

(1) randomtree

5.1 节中仅对比了无启发式策略下正向展开与反向展开的效率.这里为二者配备启发式技术进行比较.规定算法的最大扩展次数为 10000 次,最大运行时间为 35 秒.实验结果如表 4 与图 10(a)所示.

结果表明在 randomtree 上基于 *block* +深度优先策略的反向展开效果最佳,在全部 37 组用例上|E|的规模均最优.值得注意的是,在正向展开中 *hmax* 策略相比无启发式策略有明显的效率提升,这在一定程度上说明了启发式技术可以弥补算法在性质上的不足.

(2) threadlock

用例集 threadlock 按照如下规则模拟了一类线程-锁模型:假设有 x 个线程 y 把锁,x 个线程依次申请锁 1,锁 2,...,锁 y,接着依次释放锁 y,锁 y-1,...,锁 1.目标标识由每个线程的终止库所组成.规定算法的最大扩展次数为 10000 次,最大运行时间为 70 秒.实验结果如表 5 与图 10(b)所示.

Table 4 Results of directed unfolding and reverse unfolding in randomtree

表 4. directed unfolding 与反向展开在 randomtree 上的实验结果

		Unf-hmax		RUnf-blcok+dfs	
测试用例	|T|	|E|	时间(毫秒)	|E|	时间(毫秒)
randomtree100	899	30	55	**29**	40
randomtree125	1194	19	12	**18**	14
randomtree150	1387	32	21	**29**	11
randomtree175	1729	38	32	**34**	18
randomtree200	1854	31	38	**27**	6
randomtree225	2130	13	23	**12**	29
randomtree250	2429	28	47	**26**	5
randomtree275	2575	36	49	**34**	7
randomtree300	2898	39	32	**37**	5
randomtree325	3132	29	29	**28**	4
randomtree350	3492	34	45	**31**	5
randomtree375	3615	35	45	**31**	4
randomtree400	3712	33	28	**31**	4
randomtree425	4076	32	34	**29**	15

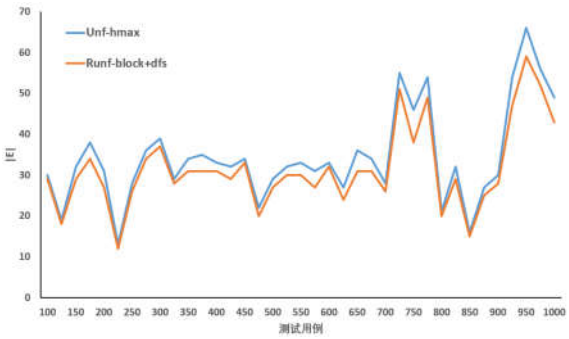
randomtree450	4304	34	55	**33**	44
randomtree475	4523	22	37	**20**	15
randomtree500	4732	29	85	**27**	6
randomtree525	5014	32	90	**30**	3
randomtree550	5160	33	133	**30**	4
randomtree575	5394	31	150	**27**	3
randomtree600	5739	33	183	**32**	4
randomtree625	5835	27	137	**24**	3
randomtree650	6238	36	83	**31**	5
randomtree675	6423	34	77	**31**	4
randomtree700	6529	28	57	**26**	4
randomtree725	7026	55	142	**51**	9
randomtree750	7133	46	112	**38**	26
randomtree775	7415	54	131	**49**	12
randomtree800	7673	21	59	**20**	4
randomtree825	7689	32	78	**29**	4
randomtree850	8030	16	41	**15**	3
randomtree875	8224	27	90	**25**	7
randomtree900	8402	30	71	**28**	19
randomtree925	8762	54	161	**47**	18
randomtree950	9135	66	229	**59**	12
randomtree975	9491	56	228	**52**	12
randomtree1000	9462	49	152	**43**	8

Table 4 Results of directed unfolding and reverse unfolding in threadlock

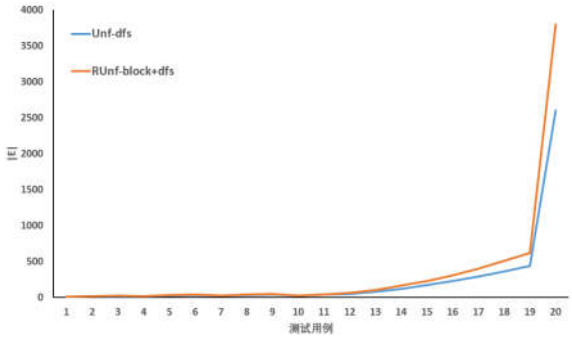
表 4. directed unfolding 与反向展开在 threadlock 上的实验结果

		Unf-dfs			RUnf-blcok+dfs		
测试用例	|T|	|E|	时间(毫秒)	|Ext|	|E|	事件(毫秒)	|RExt|
threadlock2_1	6	**8**	20	1	**8**	10	5
threadlock3_1	10	**12**	7	3	**12**	8	11
threadlock3_2	16	**18**	27	3	21	14	23
threadlock4_1	14	**16**	19	6	**16**	10	19
threadlock4_2	22	**24**	7	6	28	8	41
threadlock4_3	30	**32**	8	6	40	7	60
threadlock5_1	18	**20**	9	10	**20**	2	29
threadlock5_2	28	**30**	5	10	35	5	62
threadlock5_3	38	**40**	8	10	50	7	91
threadlock6_1	22	**24**	5	15	**24**	3	43
threadlock6_2	34	**36**	20	15	42	5	84
threadlock6_3	46	**48**	23	15	60	25	129
threadlock8_4	78	**80**	29	28	104	39	291
threadlock10_5	118	**120**	34	45	160	87	554
threadlock12_6	166	**168**	87	66	228	105	943
threadlock14_7	222	**224**	85	91	308	223	1474
threadlock16_8	286	**288**	72	120	400	161	2186
threadlock18_9	358	**360**	70	153	504	399	3088
threadlock20_10	438	**440**	131	190	620	497	4211
threadlock50_25	2598	**2600**	3746	1225	3800	49214	63785

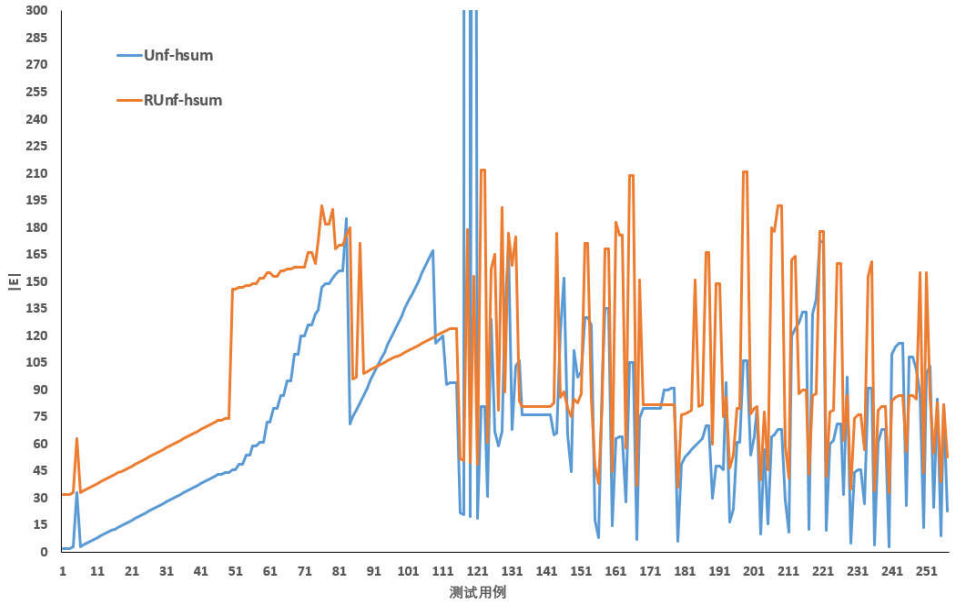
结果表明基于深度优先策略的正向展开在 threadlock 上效果最佳,其|E|的规模在 20 组用例中有 15 次最优,有 5 次同基于 *block* +深度优先策略的反向展开持平.且随着用例规模的增大,正向展开的运行时间明显优于反向展开,然而,threadlock 中 Petri 网的正向分支与反向分支数量相差不大,理论上正向展开与反向展开的效率也应该相近,这与实验结果相违背.分析可知,随着用例规模的增大 *RExt* 中的冗余扩展数量迅速增加,导致反向展开的运行速度缓慢,另外庞大的扩展集也不利于启发式技术做出正确的选择.可见如何有效减少冗余扩展的数量是提升反向展开效率的关键因素.



(a) randomtree



(b) threadlock



(c) dartes

Fig.10 Comparison of |E| in different test cases between directed unfolding and reverse unfolding

图 10. directed unfolding 与反向展开在不同测试用例上的事件总数对比

(3) directed unfolding 基准库

directed unfolding 使用了 dartes、random、airport、openstacks4 个用例集,本文分别与之进行比较.4 个用例集上的最大扩展次数均设为 10000 次,最大运行时间设为 35 秒.

dartes 用例集的实验结果如图 10(c)所示.正向展开中 *hsum* 策略效果最好,总共验证了 256/257 组用例的可覆盖性.反向展开中 *hmax* 策略效果最好,总共验证了 257/257 组用例的可覆盖性.进一步,反向展开|E|的规模在 48 组用例上最优,在 2 组用例上与正向展开持平.在这个用例集上正向展开与反向展开都发挥了各自的优势.

在 random 用例集上,正向展开中 *hsum* 策略效果最好,总共验证了 33/45 组用例的可覆盖性.反向展开中 *block*+*hsum* 策略效果最好,总共验证了 26/45 组用例的可覆盖性.进一步,正向展开|E|的规模在 33 组用例上最优,在 12 组用例上与反向展开持平.在接下来对 airport、openstacks 的测试中,正向展开结合 *hsum* 策略分别能验证 19/26、30/30 组用例的可覆盖性,而反向展开无法验证任何用例的可覆盖性.在这些用例上的低效甚至失效说明了反向展开并不能适用于所有场景,这也是其性质决定的.然而,启发式技术往往能够弥补算法在性质上的不足.因此,针对反向展开低效或失效的情况,应设计更有效的启发式技术加以优化.

本节对比了启发式反向展开与 directed unfolding 在 Petri 网可覆盖性上的效率.结果表明在 415 组测试数据中,反向展开的规模在 85 组数据上优于正向展开,在 26 组数据上与正向展开持平.实验验证了反向展开在部分用例上的有效性,在 Petri 网的正向分支较多时,反向展开可以从目标标识出发,仅刻画与可覆盖性判定相关的系统状态,从而提升目标标识可覆盖性判定的效率.然而,目前反向展开在多数用例中仍效率低下,主要原因分为以下两个方面:一是冗余扩展的数量.冗余扩展数量过多会导致算法运行缓慢,且不利于启发式技术做出正确的

选择;二是由算法的特性决定的.在Petri网的反向分支较多时,反向展开的效率往往低于正向展开.不过启发式技术往往能够弥补算法在性质上的不足.因此,针对反向展开低效或失效的情况,应考虑设计更有效的启发式技术加以优化.

6 总结与展望

本文针对安全 Petri 网的可覆盖性判定问题提出了一种目标导向的反向展开算法.反向展开从需要作出可覆盖性判定的目标标识出发,仅刻画与可覆盖性判定相关的系统状态,并结合 *block*、*hmax*、*hsum* 等启发式技术缩减展开的规模,以此提高目标标识可覆盖判定的效率.进而,将反向展开算法应用于并发程序的形式化验证,将并发程序的数据竞争检测问题转换为 Petri 网特定标识的可覆盖性判定问题. 实验对比了启发式反向展开与directed unfolding在Petri网可覆盖性上的效率,验证了反向展开在Petri网的正向分支较多时可提高可覆盖性判定效率.

然而,本文尚有很多不足.在算法效率上,一方面需要进一步减少冗余扩展的数量,并设计更加高效的启发式技术.另一方面可以尝试将正向展开与反向展开相结合,综合二者的优势.在数据竞争检测上,可以考虑使用其他静态检测算法初步确定可能发生数据竞争的语句位置,在此基础上再结合反向展开算法进行验证.

References:

[1] Han L, Xing K, Chen X, Xiong F. A Petri net-based particle swarm optimization approach for scheduling deadlock-prone flexible manufacturing systems. Journal of Intelligent Manufacturing, 2018,29(5):1083-1096. https://doi.org/10.1007/s10845-015-1161-2

[2] Hu L, Liu Z, Hu W, Wang Y, Tan J, Wu F. Petri-net-based dynamic scheduling of flexible manufacturing system via de ep reinforcement learning with graph convolutional network. Journal of Manufacturing Systems, 2020,55:1-4. https://doi.org /10.1016/j.jmsy.2020.02.004

[3] Fauzan AC, Sarno R, Yaqin MA. Performance measurement based on coloured Petri net simulation of scalable business p rocesses. In: 2017 4th International Conference on Electrical Engineering, Computer Science and Informatics(EECSI). IEE E, 2017. 1-6. https://doi.org/10.1109/EECSI.2017.8239121

[4] Liu C, Zeng Q, Duan H, Wang L, Tan J, Ren C, Yu W. Petri net based data-flow error detection and correction strateg y for business processes. IEEE Access, 2020,8:43265-43276. https://doi.org/10.1109/ACCESS.2020.2976124

[5] Lu F, Tao R, Du Y, Zeng Q, Bao Y. Deadlock detection-oriented unfolding of unbounded Petri nets. Information Science s, 2019,497:1-22. https://doi.org/10.1016/j.ins.2019.05.021

[6] Xiang D, Liu G, Yan C, Jiang C. Detecting data inconsistency based on the unfolding technique of petri nets. IEEE Tra nsactions on Industrial Informatics, 2017,13(6):2995-3005. https://doi.org/10.1109/TII.2017.2698640

[7] Dong L, Liu G, Xiang D. Verifying CTL with Unfoldings of Petri Nets. In: International Conference on Algorithms and Architectures for Parallel Processing. Springer, Cham, 2018. 47-61. https://doi.org/10.1007/978-3-030-05063-4_5

[8] Liu G, Reisig W, Jiang C, Zhou M. A branching-process-based method to check soundness of workflow systems. IEEE Access, 2016,4:4104-18. https://doi.org/10.1109/ACCESS.2016.2597061

[9] McMillan KL. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: Inte rnational Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 1992. 164-177.

[10] Engelfriet J. Branching processes of Petri nets. Acta Informatica, 1991.28(6):575-591. https://doi.org/10.1007/BF01463946

[11] Esparza J, Römer S, Vogler W. An improvement of McMillan's unfolding algorithm. Formal Methods in System Design, 2002.20(3):285-310. https://doi.org/10.1023/A:1014746130920

[12] Khomenko V, Koutny M, Vogler W. Canonical prefixes of Petri net unfoldings. Acta Informatica, 2003.40(2):95-118. http s://doi.org/10.1007/s00236-003-0122-y

[13] Heljanko K, Khomenko V, Koutny M. Parallelisation of the Petri net unfolding algorithm. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2002. 371-385. https:// doi.org/10.1007/3-540-46002-0_26

[14] Benito FC, Kunzle LA. Unfolding for Time Petri Net. IEEE Latin America Transactions, 2017.15(5):1001-1008. https://doi.org/10.1109/TLA.2017.7912599

[15] Schwarick M, Rohr C, Liu F, Assaf G, Chodak J, Heiner M. Efficient Unfolding of Coloured Petri Nets Using Interval Decision Diagrams. In: International Conference on Applications and Theory of Petri Nets and Concurrency. Springer, Cham, 2020. 324-344. https://doi.org/10.1007/978-3-030-51831-8_16

[16] Chatain T, Paulevé L. Goal-driven unfolding of Petri nets. arXiv preprint arXiv:1611.01296. 2016.

[17] Bonet B, Haslum P, Hickmott S, Thiébaux S. Directed unfolding of petri nets. In: Transactions on Petri Nets and Other Models of Concurrency I. Springer, Berlin, Heidelberg, 2008. 172-198. https://doi.org/10.1007/978-3-540-89287-8_11

[18] Bonet B, Haslum P, Khomenko V, Thiébaux S, Vogler W. Recent advances in unfolding technique. Theoretical Computer Science, 2014.551:84-101. https://doi.org/10.1016/j.tcs.2014.07.003

[19] Abdulla PA, Iyer SP, Nylén A. SAT-solving the coverability problem for Petri nets. Formal Methods in System Design, 2004.24(1):25-43. https://doi.org/10.1023/B:FORM.0000004786.30007.f8

[20] Leveson NG, Turner CS. An investigation of the Therac-25 accidents. Computer, 1993.26(7):18-41. https://doi.org/10.1109/MC.1993.274940

[21] Poulsen K (2004) Software bug contributed to blackout. Security Focus. http://www.securityfocus.com/news/8016

[22] Joab J (2012) Nasdaq's Facebook glitch came from 'race conditions'. http://www.computerworld.com/s/article/9227350

[23] Blackshear S, Gorogiannis N, O'Hearn PW, Sergey I. RacerD: compositional static race detection. Proceedings of the ACM on Programming Languages, 2018.2(OOPSLA):1-28. https://doi.org/10.1145/3276514

[24] Chatarasi P, Shirako J, Kong M, Sarkar V. An extended polyhedral model for SPMD programs and its use in static data race detection. In: International Workshop on Languages and Compilers for Parallel Computing. Springer, Cham, 2016. 106-120. https://doi.org/10.1007/978-3-319-52709-3_10

[25] Bora U, Das S, Kukreja P, Joshi S, Upadrasta R, Rajopadhye S. Llov: a fast static data-race checker for OpenMP programs. ACM Transactions on Architecture and Code Optimization (TACO), 2020.17(4):1-26. https://doi.org/10.1145/3418597

[26] Wilcox JR, Flanagan C, Freund SN. VerifiedFT: a verified, high-performance precise dynamic race detector. In: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2018. 354-367. https://doi.org/10.1145/3178487.3178514

[27] Gu Y, Mellor-Crummey J. Dynamic data race detection for OpenMP programs. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2018. 767-778. https://doi.org/10.1109/SC.2018.00064

[28] Lidbury C, Donaldson AF. Dynamic race detection for C++ 11. ACM SIGPLAN Notices, 2017.52(1):443-57. https://doi.org/10.1145/3093333.3009857

[29] Kavi KM, Moshtaghi A, Chen DJ. Modeling multithreaded applications using Petri nets. International Journal of Parallel Programming, 2002.30(5):353-71. https://doi.org/10.1023/A:1019917329895

[30] Vallée-Rai R, Co P, Gagnon E, Hendren L, Lam P, Sundaresan V. Soot: A Java bytecode optimization framework. In: CASCON First Decade High Impact Papers. 2010. 214-224. https://doi.org/10.1145/1925805.1925818

[31] https://github.com/Zongyin-Hao/Coverability

[32] 刘关俊 著. Petri 网的元展：一种并发系统模型检测方法. 北京：科学出版社. 2020.

附录 A

首先给出 adequate order 的定义.偏序关系 $\prec$ 是一个 adequate order, 如果它满足:

(1) $\prec$ 是良基的;

(2) $\prec$ 是 $\subset$ 的精炼, $Cfg_1 \subset Cfg_2$ 意味着 $Cfg_1 \prec Cfg_2$;

(3) 若 $Mark(Cfg_1) \le Mark(Cfg_2)$, $Cfg_1 \prec Cfg_2$, 则对 Cfg_2 的任意前缀 E_2 有 E_1 满足

$Mark(Cfg_1 \oplus E_1) \le Mark(Cfg_2 \oplus E_2)$,且 $Cfg_1 \oplus E_1 \prec Cfg_2 \oplus E_2$.

定理 1 $\prec_r$ 是 adequate order

证明:易得 $\prec_r$ 满足 adequate order 的条件(1)、(2).只需证明 $\prec_r$ 满足条件(3)即可.即证明当 $Mark(Cfg_1) \le Mark(Cfg_2) \wedge Cfg_1 \prec_r Cfg_2$ 时,对 Cfg_2 的任意前缀 E_2 有 E_1 满足 $Mark(Cfg_1 \oplus E_1) \le Mark(Cfg_2 \oplus E_2)$,且 $Cfg_1 \oplus E_1 \prec_r Cfg_2 \oplus E_2$.

定义 E^k 为配置 Cfg 的一个大小为 k 前缀,$Cfg^k = Cfg \oplus E^k$.接下来按 $|E_2|$ 的大小进行归纳证明.$|E_2|=0$ 时结论显然成立.$|E_2|=k$ 时由归纳假设有 $Mark(Cfg_1{}^k) \le Mark(Cfg_2{}^k)$,且 $Cfg_1{}^k \prec_r Cfg_2{}^k$.$|E_2|=k+1$ 时对任意变迁 t:

设 $e_2=(t,C_2)$,$\mu(C_2) \subseteq t^\bullet \wedge C_2 \subseteq Cut(Cfg_2{}^k)$,有 $Cfg_2{}^{k+1} = Cfg_2{}^k \cup \{e_2\}$;

设 $e_1=(t,C_1)$,$\mu(C_1) \subseteq t^\bullet \wedge C_1 \subseteq Cut(Cfg_1{}^k)$,有 $Cfg_1{}^{k+1} = Cfg_1{}^k \cup \{e_1\}$.要求 $\mu(C_1) \le \mu(C_2)$ 且 C_1 尽可能大(C_1 可以为 $\varnothing$).

接下来只需证明 $Mark(Cfg_1{}^{k+1}) \le Mark(Cfg_2{}^{k+1})$,且 $Cfg_1{}^{k+1} \prec_r Cfg_2{}^{k+1}$,有以下两种情况:

(1) $C_1=\varnothing$:$Cfg_1{}^k = Cfg_1{}^{k+1}$.由于要求了 C_1 尽可能大,可知 $\forall p \in \mu(C_2)$,$p \notin Mark(Cfg_1{}^k)$,易得 $Mark(Cfg_1{}^{k+1}) = Mark(Cfg_1{}^k) \le Mark(Cfg_2{}^k) - \mu(C_2) + {}^\bullet t = Mark(Cfg_2{}^{k+1})$.又因为 $|Cfg_2{}^k| < |Cfg_2{}^{k+1}|$,由 $\prec_r$ 的定义(1)有 $Cfg_2{}^k \prec Cfg_2{}^{k+1}$,进而有 $Cfg_1{}^{k+1} = Cfg_1{}^k \prec_r Cfg_2^k \prec_r Cfg_2^{k+1}$.$C_1=\varnothing$ 时结论成立.

(2) $C_1 \ne \varnothing$:由于要求了 C_1 尽可能大,且 $\mu(C_1) \le \mu(C_2)$,可知 $\forall p \in \mu(C_2)$,$p \in \mu(C_1) \vee p \notin Mark(Cfg_1{}^k)$,易得 $Mark(Cfg_1{}^{k+1}) = Mark(Cfg_1{}^k) - \mu(C_1) + {}^\bullet t \le Mark(Cfg_2{}^k) - \mu(C_2) + {}^\bullet t = Mark(Cfg_2{}^{k+1})$.当 $|Cfg_1{}^k| < |Cfg_2{}^k|$ 时,有 $|Cfg_1{}^{k+1}| = |Cfg_1{}^k| + 1 < |Cfg_2{}^k| + 1 = |Cfg_2{}^{k+1}|$,$Cfg_1{}^{k+1} \prec_r Cfg_2{}^{k+1}$.当 $|Cfg_1{}^k| = |Cfg_2{}^k|$,且 $Lex(\mu(Cfg_1{}^k)) < Lex(\mu(Cfg_2{}^k))$ 时,有 $|Cfg_1{}^{k+1}| = |Cfg_1{}^k| + 1 = |Cfg_2{}^k| + 1 = |Cfg_2{}^{k+1}|$.两个长度相同的有序串添加一个相同元素并重新排序后,二者间的字典序大小关系不会发生改变,因此 $Lex(\mu(Cfg_1{}^{k+1})) < Lex(\mu(Cfg_2{}^{k+1}))$,$Cfg_1{}^{k+1} \prec_r Cfg_2{}^{k+1}$.$C_1 \ne \varnothing$ 时结论成立.

综上,有 $Mark(Cfg_1 \oplus E_1) \le Mark(Cfg_2 \oplus E_2)$,且 $Cfg_1 \oplus E_1 \prec_r Cfg_2 \oplus E_2$.因此 $\prec_r$ 是 adequate order.

定理 2 反向截断事件不会破坏 $RUnf$ 的完备性.

证明:对于满足 $M \mapsto M_f$ 的任意可达标识 M,应存在配置 Cfg 满足 $Mark(Cfg) \subseteq M$.假设 Cfg 不存在于 $RUnf$ 中,则 Cfg 中一定包含反向截断事件 e,且 $RUnf$ 中存在事件 e',使得 $Mark([e']) \le Mark([e]) \wedge [e'] \prec [e]$.

由 adequate order 的条件(3)可知对于 $Cfg=[e]\oplus E$,存在 $Cfg'=[e']\oplus E'$,使得 $Mark(Cfg') \le Mark(Cfg) \le M \wedge Cfg' \prec Cfg$.由于 $\prec$ 是良基的,上述过程最终会找到一个存在于 $RUnf$ 中的极小配置,这与假设不符.故反向截断事件不会破坏 $RUnf$ 的完备性.

定理 3. $RUnf$ 是有限的.

证明:对于 $RUnf$ 内的任意事件 e,可以得到一条最长链 $e_1 < e_2 < \ldots < e$,设其长度为 $d(e)$,有以下三个结论:

(1) 对于任意条件 c,${}^\bullet c$ 与 $c^\bullet$ 是有限的.对于任意事件 e,${}^\bullet e$ 与 $e^\bullet$ 是有限的.

(2) 设安全 Petri 网的可达标识总数为 n,有 $d(e) \le n+1$.

长度为 $n+1$ 的链 $e_1 < e_2 < \ldots < e_{n+1}$ 必然存在两个事件 $e_i, e_j, i<j$,使得 $Mark([e_i]) \le Mark([e_j])$,又因为 $[e_i] \subset [e_j]$,由 adequate order 的条件(2)可知 $[e_i] \prec [e_j]$,e_j 为反向截断事件.由此可知链中不存在 e 使得 $e_j < e$,即链长不可能大于 $n+1$.

(3) 对于任意 k ($k \ge 0$),$RUnf$ 中仅包含有限多的事件 e,使得 $d(e) \le k$.

可以进行归纳证明:$k=0$ 时结论显然成立.设 E_k 为 $d(e) \le k$ 的事件集合,由归纳假设可知 E_k 是有限的.E_{k+1} 为 $d(e) \le k+1$ 的事件集合.由 $E_{k+1}{}^\bullet \subseteq {}^\bullet E_k \cup CM_f$ 以及(1)可知 E_{k+1} 是有限的.

由(2)、(3)可知 *RUnf* 包含有限多的事件,再由(1)可知 *RUnf* 包含有限多的条件,故 *RUnf* 是有限的.

附录 B

首先回顾一下 *RExt* 的两个条件:

(1) 对于 *RUnf* 中任意事件 e, *RExt* 中不存在反向扩展 $rext=(t,C)$,使得 $\mu(e)=t\wedge e^{\bullet}=C$;

(2) 对于 *RExt* 内任意两不同的反向扩展 $rext_1=(t_1,C_1)$, $rext_2=(t_2,C_2)$, 若 $t_1=t_2\wedge C_1\subset C_2\wedge Mark([rext_1])\geq Mark([rext_2])$,则将扩展 $rext_1$ 从 *RExt* 中删除.

Parosh 在文献[19]中首次提出反向展开的思想,但未在扩展规则中加入 $Mark([rext_1])\geq Mark([rext_2])$ 这个限制,由此导致其反向展开不满足完备性.这里分析其反例.

考虑图 11 所示的 Petri 网及其反向展开.左图是原 Petri 网,目标标识为 $\{p12\}$,不难发现目标标识是可覆盖的.右图是未加限制条件 $Mark([rext_1])\geq Mark([rext_2])$ 时的反向展开.反向展开算法在生成事件 $e8$ 时对应的反向扩展为 $rext1=(t8,\{c2,c9\})$,根据 Parosh 的规则会丢弃 $rext2=(t8,\{c2\})$.然而,分析原 Petri 网会发现 $rext1$ 对应的反向标识 $\{p4,p8\}$ 是不可覆盖的,而 $rext2$ 对应的反向标识 $\{p8,p11\}$ 是可覆盖的.进一步, $rext2$ 对应的反向标识对判断 $\{p12\}$ 的可覆盖性来说是必要的,此时无论后续如何扩展都无法验证目标标识的可覆盖性, *RUnf* 的完备性已被破坏.

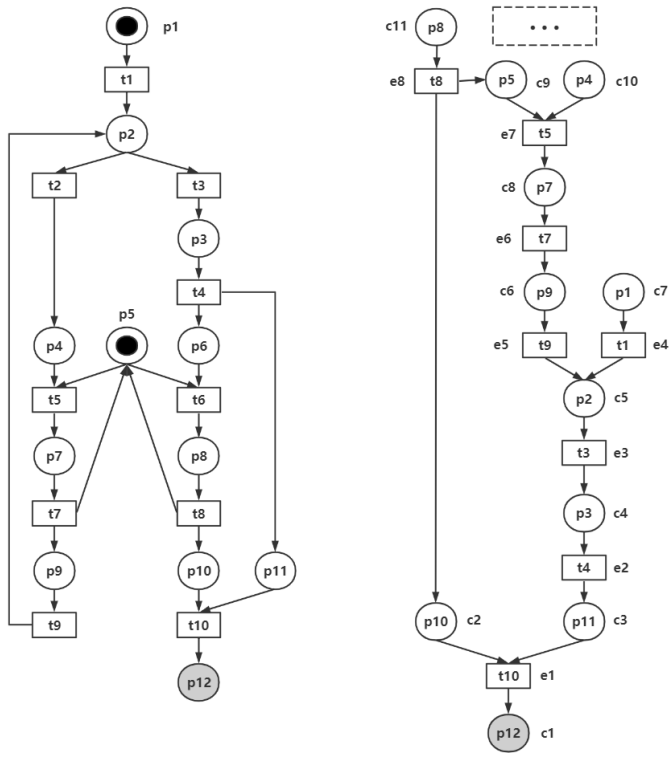


Fig.11 Example on completeness destruction of reverse unfolding

图 11 反向展开完备性被破坏的实例