在安全性方法中,任何非交互式 Petri 网 $N=(P,T,F,M_0)$都必须满足 $C(N)$约束,规定了在任何状态下每个位置内的令牌数都必须大于等于 0,所以,定理 1 的第 1 个条件天然满足.至于第 2 个条件,在安全性方法中,如果约束(1)有解(可能有多组解),即$\exists M'\in Set(M),\neg\varphi(M')$成立.我们可以通过定理 1 来判定 M'是否可达,即判定其对应的 X'是否合法.这样就可以严格地去验证非交互式 Petri 网是否覆盖某个错误状态,使得算法在理论上达到完备.

结合安全性方法与子网可标记验证,验证非交互式 Petri 网可覆盖性的方法可以表述如下.

从非交互式 Petri 网 N 和待验证性质φ中获取约束条件 $C(P,T,F,M_0)\wedge\neg\varphi$,作为 SMT 求解器的输入进行求解.

1) 无解,即约束条件不满足,说明该非交互式 Petri 网的所有状态都满足性质φ,N不会覆盖满足性质$\neg\varphi$的错误状态.

2) 有解,即$\exists M'\in Set(M),\neg\varphi(M')$成立.我们需要去验证 M'对应的 X'是否满足定理 1 的第 2 个条件.

a) 若满足,说明$\neg\varphi$会被一个可达状态满足,表明 N 不满足性质φ,N 会覆盖满足性质$\neg\varphi$的错误状态.

b) 若不满足,说明满足$\neg\varphi$的状态 M'不可达.由于满足约束 $C(P,T,F,M_0)\wedge\neg\varphi$的状态可能有多个,所以需要加上新的约束条件剔除状态 M',继续交给 SMT 求解器求解,进行下一次迭代(注:由于约束(1)中等式的数量可能少于变量的个数,且约束中存在不等式,所以约束(1)的解的数量可能是无限的,而且可能存在需要无数次迭代的极端情况.不过,从之后的测试发现,多次迭代的情况非常少.当然,约束(1)解的数量并不等于迭代的次数,因为每次迭代并不一定只排除一个解,也可能是一组甚至无数解,可以参考第 3.3 节中的实例,而排除解的数量在不同的模型中也互不相同,所以约束(1)解的数量和迭代次数之间的关系并不能用确切的表达式来表达.甚至可能存在约束(1)解的数量无限,但是只需要几次迭代即可成功验证的情况).

2.4 剪枝技巧

因为对于 SMT 求解器的每组解,我们都需要验证其是否符合定理 1 的条件 2),为了减少算法的迭代次数,我们可以增加两种剪枝加速的方法.

1) 在给定的非交互 Petri 网 $N=(P,T,F,M_0)$中,对于那些满足 $M_0(p)>0$ 的位置,它们的输出迁移必须至少有一个触发次数大于 0.约束化为 *Constraints* 1.

$$\left.\begin{aligned} InitPlace &= \{p \mid p\in P, M_0(p)>0\} \\ InitTransition &= \{t \mid t\in T, t\in InitPlace^{\bullet}\} \\ Constra\text{int}s\ 1 &= V_{t\in InitTransition}\, t>0 \end{aligned}\right\} \tag{2}$$

因为我们要验证每组解是否符合定理 1 的条件 2),也就是子网内的每个位置都要由 M_0 可标记.由引理 1 可知,该子网内至少有一个位置 p 满足 $M_0(p)>0$.因此,如果 *Constraint* 1 无法满足,也就是说,满足 $M_0(p)>0$ 的位置 p 根本没有路径“出去”,则子网内的其他位置 p'都无法由 M_0 可标记.

2) 在给定的非交互 Petri 网 $N=(P,T,F,M_0)$中,如果存在这样的位置,*InitPlace* 集合中的任意一个位置都不存在一条到它的路径,那么它的输入和输出迁移发生的次数都为 0.约束化为 *Constraints* 2.

$$\left.\begin{aligned} UnmarkPlace &= \{p \mid p\in P, \forall_{pp\in InitPlace}\neg path(pp,p)\} \\ UnmarkTransition &= \{t \mid t\in T, t\in {}^{\bullet}UnmarkPlace\cup UnmarkPlace^{\bullet}\} \\ Constraints2 &= \Lambda_{t\in UnmarkTransition}\, t=0 \end{aligned}\right\} \tag{3}$$

因为对于这些无法从 *InitTransition* 中的位置到达的位置,它们在任何状态子网上都是无法由 M_0 可标记的,如果它们的输入或者输出迁移发生的次数大于 0,那么子网就必须将这些位置包含进去,那么该子网就不可能满足定理 1 的条件 2).

使用上述两种剪枝技巧,可以有效地减少算法的迭代次数,使得验证更加高效、实用.

3 CFPCV 工具技术框架

我们采用基于约束的方法,实现了可以高效验证非交互式 Petri 网可覆盖性的工具 CFPCV,它在安全性方法的基础上,加上子网可标记验证,从而使得该算法在理论上完备.本节主要介绍 CFPCV 工具的技术架构以及使

用到的具体算法.

3.1 技术架构

本文的技术方案如图 5 所示.主要分为约束提取、约束求解、候选解验证、增加约束进行迭代这 4 个部分.具体内容如下.

1) 首先根据给定的非交互 Petri 网模型 $N=(P,T,F,M_0)$得到一些模型的基本约束,例如每个位置内的令牌数必须大于等于 0,每个迁移发生的次数必须大于等于 0,再根据需要验证的状态 M 得到状态约束,例如待验证性质为 $p_1=1\&p_2=1$,则约束 $p_1\geqslant 1\&p_2\geqslant 1$ 可以覆盖满足此性质的状态.

2) 将步骤 1)得到的约束条件和剪枝技巧合并为约束文件,作为输入交给 SMT 求解器求解.

3) 如果无解,则表明不存在满足这些约束条件的状态,也就是说,N 不能覆盖状态 M';如果有解并不能代表 N 覆盖状态 M,只能代表 M'满足这些约束条件,还必须验证 M'由 M_0 可达,即需要将状态 M'从状态方程的状态空间压缩到该非交互式 Petri 网的可达状态空间内.

4) 如果验证出 M'状态确实由 M_0 可达,则可以表明 N 覆盖 M;如果不可达,说明 SMT 求解器求出的这组解(基于约束条件可能有很多组解,求解器每次只给出一组解)不满足要求.需要将状态 M'代表的这一类状态剔除,即生成新的约束加入到约束文件中,重复步骤 3)和步骤 4),直到程序得到解退出为止.

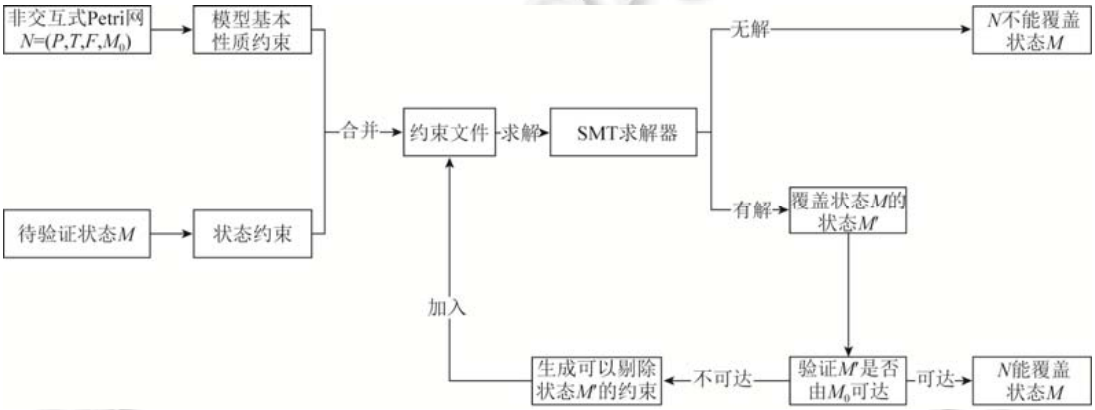

Fig.5 Technology architecture of CFPCV

图 5 CFPCV 工具技术架构

3.2 算法实现

CFPCV 工具使用到的核心算法伪代码如下.

```
1:  C:=C(N)
2:  while SAT(C∪{¬P}) then
3:      A:=Model(C∪{¬P})
4:      SN:=Subnet(C,A)
5:      If markable(SN) then
6:          return "The Communication-free Petri Net do not satisfy property p"
7:      else
8:          δ:=Constraint(A)
9:          C:=C∪δ
10:     end if
11: end while
12: return "The Communication-free Petri Net satisfy property p"
```

关于算法的逻辑,前文已经解释清楚,主要分为约束提取、约束求解、候选解验证、增加约束进行迭代这 4 个部分,这里不再赘述.其中,第 8 行的代码表示提取新约束来剔除子网 SN 所代表的一系列解.比如,$T=\{t_1,t_2,t_3,t_4\}$,$X=\{1,0,3,1\}$.那么,SN 就是根据 t_1,t_3,t_4 这 3 个迁移构成的子网,如果子网 SN 不满足定理 1 的条件 2),

则在下一次迭代中,必须增加约束将 SN 所代表的一系列解剔除,即新约束δ为 not(t_1>0 & t_2=0 & t_3>0 & t_4>0).

3.3 算法求解示例

我们可以通过一个实例再进一步更加直观、清晰地认识这一算法,对于图 8 给定的非交互式 Petri 网和带验证性质φ(p_1+p_3<3),安全性约束 $C(P,T,F,M_0)$∧¬φ 为

$p_1,p_2,p_3,p_4 \geqslant 0$

$t_1,t_2,t_3,t_4,t_5 \geqslant 0$

$p_1=0+t_5$

$p_2=1+t_3+t_4$

$p_3=0+t_3+t_4+t_5$

$p_4=0+t_1+t_2+t_3+t_4-t_5$

$p_1+p_3 \geqslant 3$

两种剪枝约束 *Constraint* $1^{(2)}$和 *Constraint* $2^{(3)}$.

$t_2>0$

将上述约束作为输入交给 SMT 求解器求解,有解,解 Model $A1$ 为

$p_1=0,p_2=3,p_3=3,p_4=4$

$t_1=0,t_2=1,t_3=0,t_4=3,t_5=0$

所以 X_1=(0,1,0,3,0),构成的子网 N_{X1} 如图 6 所示.

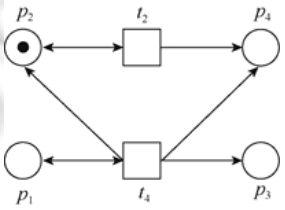

Fig.6 Subnet N_{X1}

图 6 子网 N_{X1}

显然,该子网中只有 p_2,p_4 由 M_0 可标记,因此需要增加新约束δ not(t_1=0 & t_2>0 & t_3=0 & t_4>0 & t_5=0),剔除该子网进行下一次迭代,下一次迭代的约束即为

$p_1,p_2,p_3,p_4 \geqslant 0$

$t_1,t_2,t_3,t_4,t_5 \geqslant 0$

$p_1=0+t_5$

$p_2=1+t_3+t_4$

$p_3=0+t_3+t_4+t_5$

$p_4=0+t_1+t_2+t_3+t_4-t_5$

$p_1+p_3 \geqslant 3$

$t_2>0$

not(t_1=0 & t_2>0 & t_3=0 & t_4>0 & t_5=0)

将新的约束文件作为输入交给 SMT 求解器求解,依然有解,解 Model $A2$ 为

$p_1=1$

$p_2,p_2,p_3=3$

$t_1,t_2,t_3,t_4,t_5=1$

所以 X_2=(1,1,1,1,1),而由 N_{X2} 构成的子网如图 7 所示.

我们可以发现,子网 N_{X2} 就是本来的 Petri 网,该子网内每个位置都由 M_0 可标记,所以满足定理 1 的条件.因

此存在一个迁移序列σ满足 $M_0 \xrightarrow{\sigma}$ 且 $Parikh(\sigma)=X_2$,所以$\neg\varphi$在 $R(M_0)$内满足,即该 Petri 网不满足性质φ.如图 8 所示.

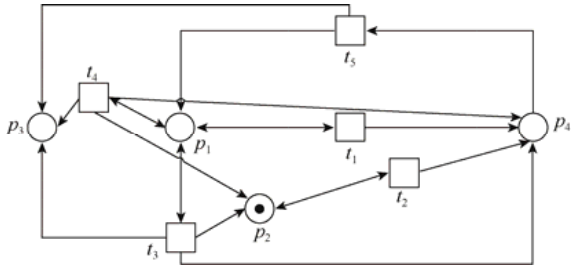

Fig.7 Subnet N_{X2}

图 7 子网 N_{X2}

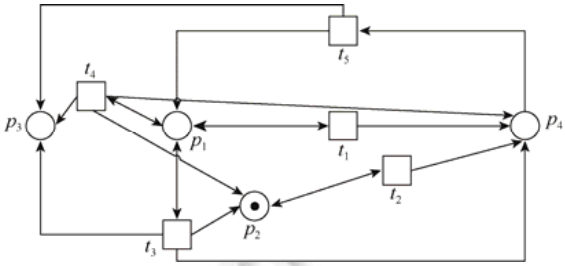

Fig.8 Communication-free Petri net instance

图 8 示例非交互式 Petri 网

4 实验及结果分析

由于现有的验证可覆盖性问题的工具都是针对传统 Petri 网的,它们所使用到的标准测试集也都是传统 Petri 网,所以这些测试集对于 CFPCV 工具的测试不再适用.因此,针对非交互式 Petri 网,我们随机生成了 3 组非交互式 Petri 网的测试用例,它们的位置和迁移数规模分别是 1~10、1~100、1~1000.我们主要从成功率、迭代次数、性能比这 3 个方面对 CFPCV 进行了评测.我们主要与 Petrinizer、MIST 这两个工具进行了比较,Petrinizer 工具是通过提取约束并求解的方法来验证传统 Petri 网的可覆盖性问题,与 CFPCV 一样,它也是采用安全性方法来获得约束.不过,正如前文所述,安全性方法在理论上是不完备的,其只能验证那些覆盖的用例,对于不覆盖的用例则无能为力.但从文献[7]中的实验结果来看,Petrinizer 在应对一些特定的测试集时仍然有不错的成功率.而 MIST 工具是采用状态空间探索的方法,其内部集成了多种验证算法,包括从待验证状态反向探索状态空间的 backward[14]算法,以及先对原 Petri 网模型进行抽象精化(abstraction refinement)来缩小模型规模,然后再结合前驱和反向探索状态空间来进行验证的 ic4pn[15]算法、tsi[15]算法和 eec[16]算法.尽管增加了抽象精化的过程来缩小模型的规模,但是对于随机模型这个过程的效果甚微,依然会有状态爆炸(state explosion)的问题存在.

4.1 成功率

我们将随机生成的 3 组测试用例作为输入交给 CFPCV、Petrinizer、MIST 工具求解,后两种工具都是验证传统 Petri 网可覆盖性的工具,所以它们肯定也可以验证非交互式 Petri 网的可覆盖性问题.我们分别比较了这 3 种工具在每组测试用例下的成功率,见表 1~表 3(注:Positive 表示满足性质φ,Negative 表示不满足,Timeout 表示超时,Don't know 表示无法判定,Success rate 表示成功率).

从 3 张表可以发现,Petrinizer 工具的成功率最低,因为 Petrinizer 使用到的方法也是安全性方法,但因其针对传统 Petri 网,并没有子网可标记验证来保证候选解是正确解,所以该工具使用到的算法在理论上不完备,可能存在其无法判定的情况,所以对于随机生成的测试用例,有大量的测试用例其无法判定,因此成功率在 3 种工具中最低,在第 3 组测试用例上只有 4.6%的成功率.而 MIST 工具是基于 Petri 网可达状态空间的搜索,由于 Petri 网的可达状态空间可能很大,所以该工具经常发生超时情况,对于规模较大的输入,超时情况更加严重.所以,对于随机生成的测试用例,MIST 工具超时最多,而且随着测试用例规模的扩大,其超时情况变得非常严重,成功率直接从 100%下降到了 46.9%.显然,CFPCV 工具的成功率最为优异,对于 3 组测试用例成功率都在 99%以上.

Table 1 Success rate of the 3 tools in test cases which scales of place and transition are between 1 to 10

表 1 测试用例位置迁移数规模 1~10 之间 3 种工具的成功率

1~10	Positive	Negative	Timeout	Don't know	Total	Success rate (%)
CFPCV	410	590	0	0	1 000	100
Petrinizer	410	0	0	590	1 000	41
MIST	410	590	0	0	1 000	100

Table 2 Success rate of the 3 tools in test cases which scales of place and transition are between 1 to 100

表 2 测试用例位置迁移数规模 1~100 之间 3 种工具的成功率

1~100	Positive	Negative	Timeout	Don't know	Total	Success rate (%)
CFPCV	181	818	1	0	1 000	99.9
Petrinizer	181	0	0	819	1 000	18.1
MIST	154	775	71	0	1 000	92.9

Table 3 Success rate of the 3 tools in test cases which scales of place and transition are between 1 to 1000

表 3 测试用例位置迁移数规模 1~1000 之间 3 种工具的成功率

1~1000	Positive	Negative	Timeout	Don't know	Total	Success rate (%)
CFPCV	46	945	9	0	1 000	99.1
Petrinizer	46	0	8	946	1 000	4.6
MIST	34	435	531	0	1 000	46.9

4.2 迭代次数

因为 CFPCV 使用到的算法是基于迭代的,需要在每次迭代中验证候选解是否是正确解,如果不是,则需要增加约束继续迭代求解,所以算法的迭代次数直接决定了算法的运行效率.如果迭代次数过多,就可能发生超时的情况.我们记录了每组测试用例算法的迭代次数,见表 4.

由表 4 可以发现,对于绝大多数(2587+397)测试用例,只需要 1~2 次迭代即可得解,只有极个别的测试用例需要 10 次以上的迭代(12 次 2 个,16 次 1 个,超时 10 个).因为只需要很少的迭代次数即可得解,所以 CFPCV 工具的运行效率理应很高.

Table 4 Iteration time of CFPCV

表 4 CFPCV 运行迭代次数

迭代次数	1	2	3	4	5	12	16	TO
测试用例个数	2 587	397	1	1	1	2	1	10

4.3 性能比

因为 CFPCV 和 Petrinizer 都用到了安全性方法,且 Petrinizer 的性能也非常高,只不过其成功率较低,所以我们比较了 3 组测试用例下这两种工具的性能(未比较 MIST 是因为 MIST 超时情况严重,所以其性能自然较低),性能比如图 9~图 11 所示,单位为 s.

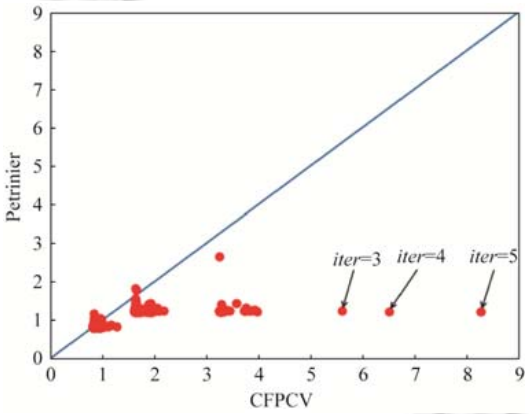

Fig.9 Performance ratio of Petrinizer and CFPCV in test cases which scales of place and transition are between 1 to 10

图 9 测试用例位置迁移数规模 1~10 之间 Petrinizer 和 CFPCV 的性能比

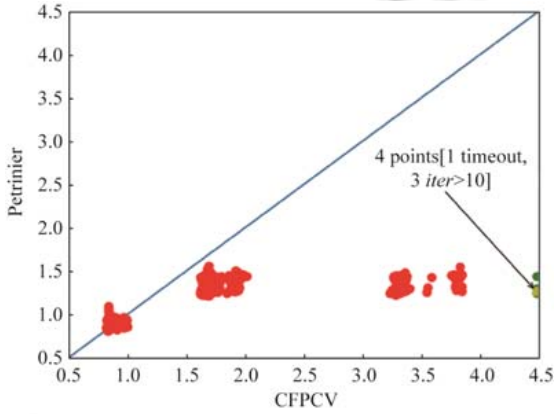

Fig.10 Performance ratio of Petrinizer and CFPCV in test cases which scales of place and transition are between 1 to 100

图 10 测试用例位置迁移数规模 1~100 之间 Petrinizer 和 CFPCV 的性能比

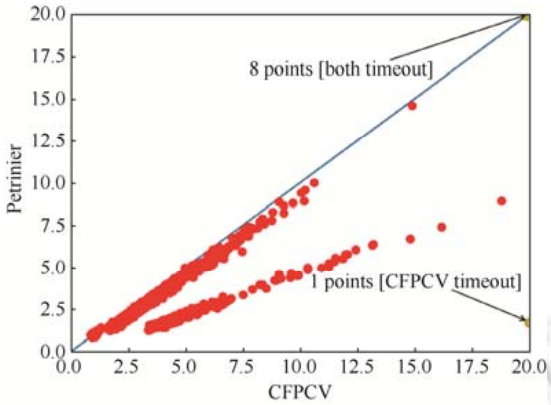

Fig.11 Performance ratio of Petrinizer and CFPCV in test cases which scales of place and transition are between 1 to 1000

图 11 测试用例位置迁移数规模 1~1000 之间 Petrinizer 和 CFPCV 的性能比

由图 9~图 11 可知,Petrinizer 和 CFPCV 性能相当,大部分测试用例两种工具在 20s 内得解.图中箭头标注的点分为两类,一类是 CFPCV 需要的较多的迭代才可得解,因此这些点代表的测试用例 CFPCV 运行较慢.另一类是两种工具都超时(图 11 右上角的 8 个点),原因是因为 SMT 求解器求解约束超时,这种情况是合理的,因为有些约束求解问题(比如 SAT 问题)就是 NP 完备问题,可能存在一些测试用例 SMT 求解器根本无法求解,因此两种工具都发生了超时的情况.因为 Petrinizer 没有验证候选解的迭代过程,所以 Petrinizer 的性能理应比 CFPCV 要好.但是由第 4.2 节迭代次数的记录来看,对于绝大多数测试用例,CFPCV 只需要一两次迭代即可得解,因此 CFPCV 的性能和 Petrinizerx 相差无几.考虑到第 4.1 节提到的 CFPCV 极高的成功率,这样的性能是非常令人欣慰的.

5 相关工作

Petri 网的可覆盖性问题虽然已被证明是 EXPSPACE 完备问题[17,18],BPP 的可达性以及可覆盖性问题也都被证明是 NP 完备问题[3],但是近年来学术界依然提出了很多解决该问题的算法,它们可以大致分为两类:第 1 类就是基于状态空间的探索.由 Karp 和 Miller(简称 KM)提出的 Karp and Miller procedure[17]是第一个可以验证 Petri 网可覆盖性的完备算法,它的主要思想是从 Petri 网的初始状态前驱探索(forward exploration)状态空间,不断加入可以覆盖前一状态的新状态来构造 Petri 网的覆盖树,然后判断待验证状态是否在该覆盖树上来进行验证.但是,由于 Petri 网的可达状态可以无限制地增长,导致覆盖树的规模可能非常之大,所以这个构造过程通常比较低效,它具有非原始递归最坏情况的复杂度.这项技术已在 TINA-KM[19]工具上实现,可想而知,该工具在处理状态空间较大的模型时效率很低.另外,这项技术的一种优化方法就是构造 Petri 网的最小覆盖集(minimal coverability sets)[20]而非覆盖树,最小覆盖集已被证明是存在且有限的[21],而且其规模要远小于覆盖树的规模,所以其构造效率有了较大的提升,这项技术已在 Pep[22]工具上实现.另外,还有反向探索(backward exploration)[23]状态空间的算法,就是从待验证状态反向探索状态空间,直到到达初始状态为止,这项技术已在工具 IST-BC 和 PETR-BC[24]上实现,不过,反向探索和前驱探索本质上没有太大的区别,所以算法效率并没有显著的提升.当然,还有将前驱探索和反向探索结合[25,26]的算法,分别从初始状态前驱探索状态空间和从待验证状态反向探索状态空间,直到两条探索路径触碰为止,这项技术已在工具 BFC 上实现,性能得到了较大的提升.文献[16]中提出的‘Expand, Enlarge and Check’方法通过并发地构造两个 Petri 网的近似序列来验证可覆盖性,第 1 个序列是系统的向下近似,它可以用来判定覆盖的实例,另外一个序列是系统的向上近似,用来判定那些不覆盖的实例.这项技术已在工具 MIST 上实现.MIST 内部集成了多种算法,不过,它们的思路大体一致,都是先对原模型进行一层抽象来缩小模型的规模,然后对抽象后的模型通过前驱探索和反向探索结合的方法来加以验证.另外一类是基

于约束的方法.其主要思想与本文的算法类似,都是用约束条件来表达 Petri 网的性质,通过求解约束来验证可覆盖性.然而,由于约束的表达能力有限,不可能准确表述出 Petri 网的可达状态空间,所以这些约束条件只能成为可覆盖性问题的必要条件而非充分条件.因此这类算法在理论上是不完备的,比如 Petrinizer[7]工具,它是通过安全性方法来提取约束并求解来进行验证,不过,由于安全性方法在理论上不完备,所以它只能验证那些不覆盖的测试用例,对于覆盖的测试用例则无法验证.

除此之外,并发程序的性质也可以通过基于 Petri 网的模型检测技术来分析,使用 TCTL(时间计算树逻辑)[27]来描述待验证性质,通过检测 Petri 网的迁移发生序列来验证模型是否满足这些性质.其具体思路是先构造包含待验证性质取反语义序列的 Büchi 自动机,然后再计算 Petri 网可达图和自动机的乘积图.若将乘积图看成一个有向图,则模型检测的问题等价于检测乘积图中是否包含一个从初始状态可达的最大强连通分量,且在该强连通分量中包含了一个接受状态.对于安全性一类的性质来说,等价于检测乘积图中是否存在一条从初始节点到接受节点的路径.对于活性一类的性质来说,等价于检测乘积图中是否存在由初始节点可达的包含接受节点的环.但是,基于 Petri 网的模型检测技术也会受到状态爆炸问题的困扰,因为 Petri 网模型的状态空间通常非常之大,甚至无界,所以 Petri 网的模型检测问题难度非常之大,其上的很多逻辑算子都不存在多项式时间算法.比如 EG/AF(即我们通常所说的活性)逻辑在 Petri 网和 BPP 上都是不可判定的,EF 逻辑在 Petri 网同样不可判定,在 BPP 上虽然可判定,但也拥有 PSPACE 完备的复杂度[28].所以,传统的基于 Petri 网的模型检测技术在验证并发程序的性质时存在较大的局限性.

6 总结及未来的工作

本文设计并实现了可以高效验证非交互式 Petri 网可覆盖性的工具 CFPCV,它在验证传统 Petri 网可覆盖性使用到的安全性方法的基础上,增加了只对非交互式 Petri 网适用的子网可标记验证,从而保证了其解的正确性,并且通过实验验证了该工具具有较高的成功率以及不错的性能.

由于业界缺乏针对非交互 Petri 网可覆盖性验证的标准测试集,所以本文测试所使用测试集都是随机产生的.未来会使用数量更多以及质量更高的测试集进行测试,以验证 CFPCV 的表现是否依然优秀.另外,其实除了本文所提的剪枝方法以外,还有一种叫作阱(trap)约束[29]的技巧可以进行加速,不过我们通过测试发现,阱约束的表现却不尽人意,可能和我们测试集的随机性有关.未来业界若有质量较高的测试集公开,我们会在算法上增加阱约束进行测试,如果性能得到提升,则将加以改进.

未来,我们也会在 BPP 上做模型检测,虽然第 5 节提到 BPP 上的模型检测问题难度很大,但是我们如果做有界模型检测,比如将某个性质在无限的转移序列上都要成立限定为在 k(k 为大于 0 的自然数)步转移内成立,同时也保证这样的性质具有一定的实际意义,那么问题的复杂度将大幅下降.同样通过提取约束,使用 SMT 求解器求解约束的方法,BPP 上的有界模型检测问题可能会有很高效的解决方案.

References:

[1] Bouajjani A, Emmi M. Bounded phase analysis of message-passing programs. Int'l Journal on Software Tools for Technology Transfer (STTT), 2014,16(2):127–146.

[2] D'Osualdo E, Kochems J, Ong CHL. Automatic verification of erlang-style concurrency. In: Proc. of the Int'l Static Analysis Symposium. Berlin: Springer-Verlag, 2013. 454–476.

[3] Ganty P, Majumdar R. Algorithmic verification of asynchronous programs. ACM Trans. on Programming Languages and Systems, 2012,34(1):1–48.

[4] Kaiser A, Kroening D, Wahl T. Efficient coverability analysis by proof minimization. In: Proc. of the Int'l Conf. on Concurrency Theory. Berlin: Springer-Verlag, 2012. 500–515.

[5] German SM, Sistla AP. Reasoning about systems with many processes. Journal of the ACM (JACM), 1992,39(3):675–735.

[6] Kloos J, Majumdar R, Niksic F, *et al*. Incremental, inductive coverability. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin: Springer-Verlag, 2013. 158–173.

[7] Esparza J, Ledesma-Garza R, Majumdar R, *et al*. An SMT-based approach to coverability analysis. In: Proc. of the Int'l Conf. on Computer Aided Verification. Cham: Springer-Verlag, 2014. 603–619.

[8] Esparza J. Petri nets, commutative context-free grammars, and basic parallel processes. Fundamenta Informaticae, 1997,31(1): 13–25.

[9] Barrett C, Tinelli C. Satisfiability modulo theories. Journal on Satisfiability Boolean Modeling and Computation, 2018,3(3): 141–224.

[10] Bofill M, Nieuwenhuis R, Oliveras A, *et al*. The barcelogic SMT solver. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin: Springer-Verlag, 2008. 294–298.

[11] Jha S, Limaye R, Seshia SA. Beaver: Engineering an efficient SMT solver for bit-vector arithmetic. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin: Springer-Verlag, 2009. 668–674.

[12] Dutertre B, De Moura L. The YICES SMT solver. 2006. http://yices.csl.sri.com/tool-paper.pdf

[13] Moura LD, Bjørner N. Z3: An efficient SMT solver. In: Proc. of the Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-Verlag, 2008. 337–340.

[14] Delzanno G, Raskin JF, Begin LV. Towards the automated verification of multithreaded Java programs. In: Proc. of the Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-Verlag, 2002. 173–187.

[15] Ganty P, Raskin JF, Begin LV. From many places to few: Automatic abstraction refinement for Petri nets. Fundamenta Informaticae, 2008,88(3):275–305.

[16] Geeraerts G, Raskin JF, Van Begin L. Expand, enlarge, and check: New algorithms for the coverability problem of WSTS. Journal of Computer and System Sciences, 2006,72(1):180–203.

[17] Karp RM, Miller RE. Parallel program schemata. Journal of Computer and System Sciences, 1969,3(2):147–195.

[18] Rackoff C. The covering and boundedness problems for vector addition systems. Theoretical Computer Science, 1978,6(2): 223–231.

[19] Berthomieu B, Ribet PO, Vernadat F. The tool TINA—Construction of abstract state spaces for Petri nets and time Petri nets. Int'l Journal of Production Research, 2004,42(14):2741–2756.

[20] Geeraerts G, Raskin JF, Begin LV. On the efficient computation of the minimal coverability set for Petri net. In: Proc. of the Int'l Symp. on Automated Technology for Verification and Analysis. Berlin: Springer-Verlag, 2007. 98–113.

[21] Finkel A. Reduction and covering of infinite reachability trees. Information and Computation, 1990,89(2):144–179.

[22] Grahlmann B. The PEP tool. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin: Springer-Verlag, 1997. 440–443.

[23] Abdulla PA, Cerans K, Jonsson B, *et al*. General decidability theorems for infinite-state systems. In: Proc. of the Logic in Computer Science (LICS'96). IEEE, 1996. 313–321.

[24] Meyer R, Strazny T. Petruchio: From dynamic networks to nets. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin: Springer-Verlag, 2010. 175–179.

[25] Finkel A. Monotonic extensions of Petri nets. Electronic Notes in Theoretical Computer Science, 2003,68(6):85–106.

[26] Ganty P, Raskin JF, Begin LV. A complete abstract interpretation framework for coverability properties of WSTS. In: Proc. of the Int'l Workshop on Verification, Model Checking, and Abstract Interpretation. Berlin: Springer-Verlag, 2006. 49–64.

[27] Gerth R, Peled D, Vardi MY, *et al*. Simple on-the-fly automatic verification of linear temporal logic. In: Protocol Specification, Testing and Verification. Boston: Springer-Verlag, 1995. 3–18.

[28] Esparza J. Decidability of model checking for infinite-state concurrent systems. Acta Informatica, 1997,34(2):85–107.

[29] Murata T. Petri nets: Properties, analysis and applications. Proc. of the IEEE, 1989,77(4):541–580.

丁如江(1994－),男,江苏盐城人,硕士,主要研究领域为形式化方法,知识表示与推理.

李国强(1979－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为形式化方法,程序语言理论,知识表示与推理.