基于ASP的CSP进程描述与组合研究

赵岭忠[1,2] 司徒凌云[1] 翟仲毅[1] 钱俊彦[1]
(桂林电子科技大学计算机科学与工程学院 桂林 541004)[1] (广西可信软件重点实验室 桂林 541004)[2]

摘 要 前期工作中,为解决CSP模型检测不支持一次运行验证多条性质的问题,构建了基于ASP的CSP模型检测框架,但其存在着可描述并发进程形态不完善与可验证并发系统规模受限的问题。构建了全新的并发系统ASP描述体系,其解决了前期工作中前缀描述不允许出现类环状结构的问题,可完整描述各种形态的CSP进程。研究了并发组合进程生成技术,它可使多个进程自动化并发组合,并生成一个满足所有行为特性、具有一致结构特性的新进程,保持了验证框架内进程描述的一致性,有利于并发进程的抽象与验证。实验表明了基于ASP的CSP进程描述与组合进程生成技术的有效性,以及基于该ASP描述体系的系统性质验证的可行性。

关键词 CSP,ASP,模型检测
中图法分类号 TP311 **文献标识码** A

Description and Combination of CSP Process Based on ASP

ZHAO Ling-zhong[1,2] SITU Ling-yun[1] ZHAI Zhong-yi[1] QIAN Jun-yan[1]
(School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)[1]
(Guangxi Key Laboratory of Trusted Software, Guilin 541004, China)[2]

Abstract In the previous work, an ASP based framework, which is used to verify the model described by CSP, was proposed to solve the problem of verifying multiple properties in one run of a model checker. However, some problems still exit, which include the inability to describe some forms of concurrent process and the scale limitation of the concurrent systems that can be described. In this paper, a new description system for concurrent systems was constructed, which allows loop structure in prefix process, and therefore can be used to describe various forms of concurrent processes. It allows several processes to be combined automatically to generate a new process, which not only fulfills all behavioral characteristics but also keeps the structural consistency with the original processes. In this way, the process description within the verification framework follows a uniform style, which is helpful for the abstraction and validation of concurrent processes. The effectiveness of the ASP description system and the combination process generation technique, and the feasibility of the verification based on the description system were illustrated by examples.

Keywords CSP, ASP, Model checking

1 引言

语言模型、语义模型和实现模型是模型检测[3,7]技术中典型的3个层次模型。语言模型(如CSP[1,2]等)具有表达能力强和方便用户使用的特征;语义模型(如有限状态迁移系统、PETRI网[15]等)以较为直观的方式表示高层语言模型的含义,既可直接作为模型检测算法的输入,也可作为高层语言模型和低层实现模型之间的中介。实现模型直接服务于模型检测算法,主要有显式模型和符号模型[6]两种。显式模型直接采用语义模型实现算法,符号模型则先把语义模型以某种方式进行表示,如OBDD、约束、表等,然后在此基础上实现对应的模型检测算法。

目前,主流的基于CSP的并发系统性质验证工具是FDR[9,10],其主要思想是将高层语言模型CSP,根据其操作语义转化成低层的迁移系统模型,进而利用状态测试机检验是否存在一个符合性质的精化迁移系统来判定性质的可满足性。然而,高层模型与低层模型的差异性,一方面要求验证系统研发人员同时考虑多层模型,增加了系统开发难度,且不利于系统的扩展;另一方面性质验证过程中高层模型与低层模型间多次转换的开销,制约着验证效率的提高,也增大了模型修改与调试的复杂度。与此同时,主流的并发系统性质验证系统[7](如基于Kripke[13]结构、Petri网等)的共同特点是不支持系统的一次运行验证多条性质。

为了解决以上问题,课题组前期工作,基于ASP(Answer

到稿日期:2013-05-21 返修日期:2013-07-08 本文受国家自然科学基金(61262008,61100186,61063002),广西自然基金(2011GXNSFA018166,2011GXNSFA018164),广西可信软件重点实验室基金项目(kx201113)资助。

赵岭忠(1977—),男,博士,教授,主要研究方向为逻辑程序、形式化验证,E-mail:zhaolingzhong163@163.com;司徒凌云(1988—),男,硕士生,主要研究方向为并发系统验证;翟仲毅(1986—),男,硕士生,主要研究方向为并发系统验证;钱俊彦(1973—),男,博士,教授,主要研究方向为模型检验。

set Programming)[4,14]，建立了一套基于 ASP 的 CSP 并发系统性质验证框架[5]。该框架下，待验证性质被统一转化为 ASP 规则，且每个性质都有与之对应的谓词。通过 ASP 求解器的谨慎推理与果敢推理，确定性质是否成立。该框架基于 ASP 及高效的 ASP 求解器，使建模模型与实现模型保持在同一层次，避免了高层模型与低层模型的转换开销，实现了系统的一次运行验证多条性质，大大提高了性质验证效率。然而，该框架依旧存在不足：第一，不完善性，只能描述部分形态的并发进程(如：不允许原子进程前缀中出现类环状结构，即：形如进程 $P=a\rightarrow b\rightarrow a\rightarrow b\rightarrow c\rightarrow$STOP，框架基于谓词 $pre(X,Y,P)$描述，该进程的描述需要两次出现 $pre(a,b)$，但是框架描述时不允许相同事实出现多次)；第二，并发进程组合方式的局限性，并发组合进程与进行并发的进程描述结构形式的不一致，使并发组合进程无法进行后续的再并发组合，致使该框架只能描述和验证两个进程并发组合构成的系统模型。

本文基于并发系统 CSP 描述体系，构建了全新的并发系统 ASP 描述体系；解决了前缀描述中不允许出现类环状结构的问题，从而可完整描述各种形态的 CSP 进程；研究了并发组合进程的生成技术，该技术可以使多个进程自动化并发，并生成一个满足所有行为特性、保持一致结构形式的新进程，新进程可以进行后续的再并发组合，从而保持了验证框架内进程描述的一致性，有利于并发进程的抽象与验证。构建的基于 ASP 描述体系的性质验证框架，可以描述和验证多个(≥3)进程并发组合成的系统模型，扩大了可验证并发系统规模。

2 基础知识

2.1 回答集编程

ASP[3]是一种声明式特征的非单调逻辑程序设计语言，由稳定模型语义(Stable Model Semantics)发展而来。设 A 是一个原子，一个文字可以是 A 也可以是 $\sim A$，其中 A 表示正文字，$\sim A$ 表示负文字。

一个 ASP 逻辑程序是一个 ASP 规则集，其中每条规则 r 形如：

$$L_1 \vee \cdots \vee L_k \text{ :- } L_{k+1},\cdots,L_m,\text{not } L_{m+1},\cdots\text{not } L_n$$

其中，$n>m>k$，L_i 是一个文字，not 表示失败即否定。$head(r)=\{L_1,\cdots,L_k\}$表示规则头部；$pos(r)=\{L_{k+1},\cdots,L_m\}$表示规则体的正文字，$neg(r)=\{L_{m+1},\cdots,L_n\}$表示规则体的负文字。没有规则头部的规则 r，称为约束。如果 $pos(r)$和 $neg(r)$均为空，则称规则 r 为事实。

假设 P 是一个扩展析取逻辑程序，$Lit(P)$是基文字的集合。

若 P 不包含 not，则 P 的回答集 S 是关于 $Lit(P)$的最小子集。S 满足以下条件：

(1)对 P 中任意的规则 L_0 :- $L_1,\cdots,L_m$，如果 $L_1,\cdots,L_m$ 属于 S，则 L_0 属于 S。

(2)如果在 S 中有一对互补的文字，则 $S=Lit(P)$。

若 P 不包含 not，则程序 P 关于集合 S 的约简 Ps 的计算过程如下：

(1)如果规则中含有 not L，且 $L\in S$，则将该规则删除。

(2)在剩余规则中，将含有“not”的文字删除。

2.2 ASP 求解器

典型的有 DLV[11]，Smodels[16]，Cmodels[12] 等。其中，DLV 和 Smodels 是基于回答集语义设计的最为常见的回答集求解器，Cmodels 是利用回答集的语义与逻辑程序的语义之间的等价性质来对逻辑程序求解的。

3 CSP 进程基本结构的 ASP 描述

在 CSP 体系中，核心是进程。进程是对现实客体的抽象，进程之间的组合构成更大的进程。进程的事件是对客体动作的抽象，进程的迹表示客体的具体行为。原子进程的基本结构有前缀、选择、递归，分别表示有限行为进程、分支进程、无限行为进程。3 种原子结构进程的组合可以构成庞大复杂多样的进程(系统)。

3.1 前缀

3.1.1 前缀的 CSP 定义

定义 1(前缀) 设 x 是一个事件，Q 是一个进程，且 $x\in\alpha Q$，$\alpha Q=\alpha(x\rightarrow P)$，则 $Q=x\rightarrow P$ 表示进程 Q 是先执行事件 x，然后按照 P 的行为进行动作的进程。其中 x 称为前缀，$x\rightarrow P$ 称为前缀表达式。

前缀表示的是行为动作有限的顺序进程，即 Q 是可终止的，由此可知 P 也是可终止的，$P=$SKIP 或 $P=$STOP(注：STOP 是一个特殊进程，即从不执行任何动作的进程，一般表示死锁；SKIP 区别于 STOP 表示成功终止)，表现为该类进程的迹是一种事件节点有限的有序线形结构，如图 1 所示。

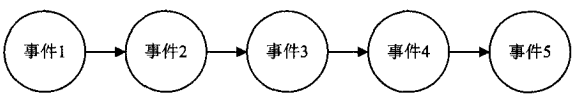


图 1 前缀进程迹结构

3.1.2 前缀的 ASP 描述

为了能够直观简便地描述前缀进程，定义下列谓词，如表 1 所列。

表 1 前缀进程谓词

谓词	含义
prefix(X,P,Q)	进程 Q 是以 X 为初始事件，后续按 P 进程动作的进程
event(X,P)	事件 X 是进程 P 的事件，即 X∈αP
process(P)	P 是一个进程
subsequent(P,Q)	进程 P 是进程 Q 的后续进程
startEvent(X,P)	X 是进程 P 的初始事件，即第一个执行的事件
endEvent(X,P)	X 是进程 P 的结束事件，即最后一个执行的事件
eventFlow(X,Y,P)	进程 P 的迹，X 事件在 Y 事件之前执行

上述所定义谓词以 ASP 规则形式给出相互之间的推导关系，如下：

规则 1 $event(X,Q)$:- $prefix(X,P,Q)$

规则 2 $event(X,P)$:- $event(X,Q)$, $subsequent(Q,P)$

由规则 1、规则 2 可知给定进程的前缀表示集合，可得进程的字母表。

规则 3 $process(P)$:- $prefix(X,P,Q)$

规则 4 $process(Q)$:- $prefix(X,P,Q)$

由规则 3、规则 4 可知给定进程的前缀表示集合，可得进程的集合。

规则 5 $subsequent(P,Q)$:- $prefix(X,P,Q)$

由规则 5 可知给定进程的前缀表示集合，可得所有进程的后继关系。

规则 6 $startEvent(X,Q)$:- $prefix(X,P,Q)$

由规则 6 可知给定进程的前缀表示集合，可得所有进程的初始事件。

规则 7 $endEvent(deadlock,P)$:- $prefix(deadlock,stop,P)$

规则 8　*endEvent*(*succStop*, *P*) :- *prefix*(*succStop*, *skip*, *P*)

规则 9　*endEvent*(*X*, *Q*) :- *endEvent*(*X*, *P*), *subsequent*(*P*, *Q*)

由规则 7—规则 9 可知给定进程的前缀表示集合，可得所有进程的终止事件。

规则 10　*eventFlow*(*X*, *Y*, *P*) :- *prefix*(*X*, *R*, *P*), *prefix*(*Y*, *Q*, *R*)

规则 11　*eventFlow*(*X*, *Y*, *P*) :- *eventFlow*(*X*, *Y*, *Q*), *subsequent*(*Q*, *P*)

由规则 10、规则 11 可知给定进程的前缀表示集合，可得所有进程的迹。

3.1.3　实例

例如：哲学家就餐问题中，一个哲学家连续两次拿起叉子、放下叉子之后再起立，起立之后就死锁。记该进程为 Ph。

用 CSP 可描述为：

Ph=pickFork→(downFork→(pickFork→(downFork→(up→STOP))))

因为"→"是右结合的，所以可以省略线形序列中的括号记为：Ph=pickFork→downFork→pickFork→downFork→up→STOP。

上述定义的前缀谓词 *prefix*(*X*, *P*, *Q*)与 CSP 的前缀定义 $Q=x\rightarrow P$ 保持高度一致性并且易于理解，参照图 2 可以更好地说明一个前缀进程如何用 ASP 定义的前缀谓词给出描述的。

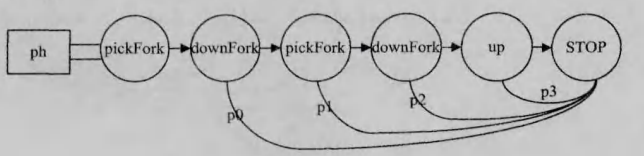


图 2　3.1.3-ph 进程行为

由图 2，ph 进程的 ASP 描述为下列前缀谓词集合，记为 D：{prefix(pickFork, p0, ph). prefix(downFork, p1, p0). prefix(pickFork, p2, p1). prefix(downFork, p3, p2). prefix(up, stop, p3). prefix(deadlock, stop, stop). }

将 D 与规则 1—规则 11 组合作为输入，调用 ASP 求解器可得回答集，如图 3 所示。

图 3　3.1.3-ph 进程回答集

注：在 ASP 描述体系中规定 STOP 进程为 prefix(deadlock, stop, stop)，SKIP 进程为 prefix(succStop, skip, skip)，其中 succStop 代表成功终止进程 SKIP 的特殊事件，类似于 CSP 中的符号"√"。deadlock 代表死锁进程 STOP 的特殊事件。

由上述可知，通过前缀谓词和其它辅助谓词，可以有效地描述前缀进程。实验结果(即生成的回答集)包含了所有的进程集合(包括子进程)、所有进程事件集合、初始事件集合，终止事件集合以及所有进程迹的集合，为模型的性质验证提供了条件。

3.2　递归

3.2.1　递归的 CSP 定义

定义 2(递归)　设 X 是一个进程变量，$A=\alpha X$，则递归进程的一种表示为进程的递归方程 $X=F(X)$，其中，$F(X)$为包含进程变量 X 的前缀表达式。该递归方程在事件集合 A 上的唯一解 $\mu X:A\cdot F(X)$是递归进程的另一种表示，其中，X 为局部变量，A 为进程 P 的字母表，$F(X)$为包含进程 P 的前缀表达式。

最简单的，例如进程描述 $P=x\rightarrow P$ 或 $P=\mu X:\{x\}\cdot(x\rightarrow X)$表示进程 P 是一个循环执行事件 x 的递归进程。

由定义 2 可知递归进程表示基于前缀表示的，递归进程描述的是该进程中事件的无限循环执行，为此递归进程是不可终止的。该类进程的迹是一种由事件节点组成的有序环形结构，如图 4 所示。

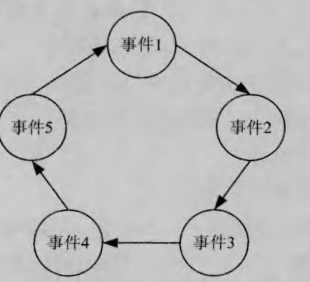


图 4　递归进程迹结构

3.2.2　递归的 ASP 描述

因为递归进程表示是基于前缀表示的，所以由 3.1.2 节中定义的 ASP 中的前缀谓词 *prefix*(*X*, *P*, *Q*)就可以描述一个递归进程。

最简单的，例如进程 $P=a\rightarrow P$，可以简单地描述为 *prefix*(*a*, *P*, *P*)。结合上述规则 1—规则 11 作为求解器的输入可以得出下列结果：{*event*(*a*, *p*), *subsequent*(*p*, *p*), *process*(*p*), *startEvent*(*a*, *p*), *eventFlow*(*a*, *a*, *p*)}，结果表明了 ASP 描述的正确性。

3.2.3　实例

例如：哲学家就餐问题中，一个哲学家循环依次进行坐下、拿起叉子、就餐、放下叉子、起立动作进行就餐行为，记该进程为 Ph。

用 CSP 可描述为：

Ph=sit→pickFork→eat→downFork→up→Ph

或

Ph=μX:{sit, pickFork, eat, downFork, up}·(sit→pickFork→eat→downFork→up→X)

同理，参照图 5，可以更好地说明如何用前缀谓词 *prefix*(*X*, *P*, *Q*)描述递归进程 *ph*。

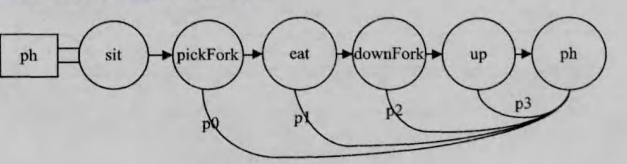


图 5　3.2.3-ph 进程行为

用 ASP 可描述为下列前缀谓词集合，记为 D：

{prefix(sit,p0,ph). prefix(pickFork,p1,p0). prefix(eat,p2,p1). prefix(downFork,p3,p2). prefix(up,ph,p3).}

将D与规则1－规则11组合作为输入，调用ASP求解器可得生成回答集中ph进程的迹信息如下：

{eventFlow(sit,pickFork,ph),eventFlow(pickFork,eat,ph),eventFlow(eat,downFork,ph),eventFlow(downFork,up,ph),eventFlow(up,sit,ph)}

由此可知，通过前缀谓词 *prefix*(X,P,Q)和其它辅助谓词，可以有效地描述递归进程。

3.3 选择

CSP体系中前缀与递归表示可以完全正确地描述单一线形特征顺序执行事件的客体，即单支结构的进程。然而有些客体本身存在多种行为执行可能，其在与外界环境的相互作用下，实时决定行为事件的执行轨迹。即该进程是一种本身具有分支结构的进程，由外界环境影响，影响的含义是指或由外界环境确定性的选择(即确定性选择，又称外部选择)决定执行其中一个分支行为，或由内部自身非确定的选择(即非确定性选择，又称内部选择)决定执行其中一个分支子进程。习惯上，称选择进程中由外部可以确定性选择的进程为确定性进程，由内部非确定性选择的进程为非确定性进程。

3.3.1 确定性选择

(a)确定性选择的定义

定义3(确定性选择) 设x和y是两个不同事件，P和Q是两个进程，则$R=(x\rightarrow P|y\rightarrow Q)$表示如果进程$R$第一个发生的事件是$x$，则客体的后续行为按$P$进行动作，如果发生的第一个事件是$y$，则客体的后续行为按$Q$进行动作。其中，$\alpha(x\rightarrow P|y\rightarrow Q)=\alpha P=\alpha Q$，"|"读作选择。

注：选择不限于二选一，多选一的情形可以用$R=(x:B\rightarrow P(x))$的简单形式表达，其中B是初始事件集合，$P(x)$为每个x对应的后续进程。

由选择的一般表达形式$R=(x:B\rightarrow P(x))$可知，选择进程的描述也是基于前缀描述，实质上是前缀描述和递归描述的组合，且其一般形式可以概括前缀、选择、递归的描述形式，如当B为空集时$R=$STOP；当B中只有一个元素时(如a)，$R=a\rightarrow P$；当B中有多个元素(如a,b)时，$R=(a\rightarrow P(a)|b\rightarrow p(b))$；同样的递归进程可以用上述形式描述为$(x:B\rightarrow F(x,uX.(x:B\rightarrow F(x,X))))$。

选择进程是一种具有分支结构的进程，外部环境决定了其具体的行为执行，表现为该进程的迹也是有多种可能(如图6所示)，外部环境的选择决定了执行哪一条迹。

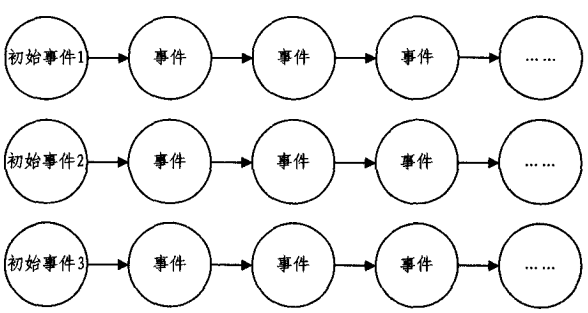


图6 选择进程迹结构

(b)确定性选择的ASP描述

因为选择进程实质是前缀进程或递归进程的组合，所以在一定意义上，选择既是进程的一种基本结构类型，也是一种进程组合方式，即由多条单支进程组合成一个分支进程。

使用3.1.2节中定义的ASP中的前缀谓词 *prefix*(X,P,Q)可以充分地描述出一个选择进程。一方面可以用ASP更精确地刻画选择进程，另一方面也为选择进程与环境相互作用提供接口。

引入选择谓词 *choice*(X,P,Q)，其含义为进程P是选择进程Q的待选后续进程，以X为初始事件(注：在确定性选择中，要求每一个待选子进程的初始事件不同，否则就会引入非确定性，褪变为一般选择，或非确定性选择。)。

(c)实例

例如：一台换钱机器同时提供两种方法破开十便士的硬币，或者是2个五便士组合，或者是2个二便士，1个五便士和1个一便士的组合，且机器上有两个按钮，供用户选择哪种组合。刻画该换钱机器的客体为进程CH10D。

用CSP描述为：

CH10D＝in10p→(out5p→out5p→CH10D | out2p→out1p→out5p→out1p→ CH10D)

同理，参照图7，可以更好地说明如何用前缀谓词 *prefix*(X,P,Q)和选择谓词 *choice*(X,P,Q)描述选择进程ch10D。

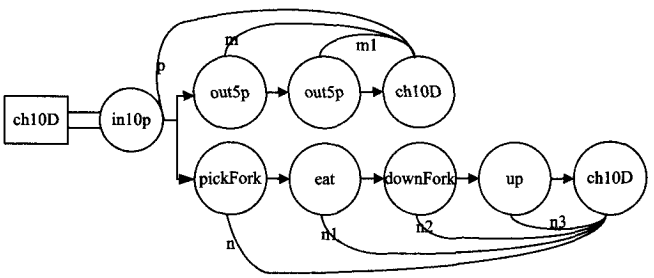


图7 CH10D进程行为

用ASP可描述为下列谓词集合，记为D：

{prefix(in10p, p, ch10D). prefix(out5p, m1, p). prefix(out5p,ch10D, m1). prefix(out2p, n1, p). prefix(out1p, n2, n1). prefix(out5p, n3, n2). prefix(out2p, ch10D, n3). choice(out5p,m,p). choice(out2p,n,p).}

将D与规则1－规则11组合作为输入，调用ASP求解器可得结果回答集迹中存在迹如下：

{choice(out5p,m,p),eventFlow(in10p,out5p,ch10D),eventFlow(out5p,out5p,ch10D),eventFlow(out5p,in10p,ch10D),choice(out2p,n,p),eventFlow(in10p,out2p,ch10D),eventFlow(out2p,out1p,ch10D),eventFlow(out1p,out5p,ch10D),eventFlow(out5p,out2p,ch10D),eventFlow(out2p,in10p,ch10D)}

由此可知，通过前缀谓词 *prefix* 和选择谓词 *choice*，可以有效地描述确定性选择进程。

3.3.2 非确定性选择

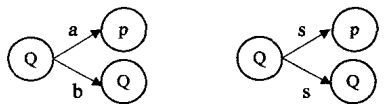
(a)非确定性选择的定义

定义4(非确定性选择) 设P和Q是进程，则$R=P\prod Q$(P或Q)表示一个进程R，或者按P动作，或者按Q动作，R在两者之间的选择，是外部环境不知道或不能控制情况下任意进行的，即由内部非确定性地任意选择。其中，$\alpha(P\prod Q)=\alpha P=\alpha Q$。

从两个不同的角度可以区分非确定性选择进程与确定性

选择进程，其一，确定性选择进程的待选子进程的初始事件完全不同，而非确定性选择进程的待选子进程初始事件相同；其二，外界环境作用时，确定性选择进程外部环境可以确定性地做出选择（表现为只保留选择后进程的迹），而非确定性进程外部环境不可控制，甚至不可观察（表现为保留多条可能执行的迹）。

图 8 可以更清晰地表明确定性选择进程和非确定性选择进程。



确定性选择（"|"）　　非确定性选择（"∏"）

图 8　确定性选择与非确定性选择

(b)非确定性选择的 ASP 描述

运用前文定义的前缀谓词 *prefix*(*X*,*P*,*Q*)和选择谓词 *choice*(*X*,*P*,*Q*)，在含义和形式上都可以充分描述出一个非确定性选择进程。其与确定性选择进程的不同在于描述的谓词 *choice*(*X*,*P*,*Q*)集合中初始事件 *X* 都相同。

(c)实例说明

简单的，例如 CSP 描述进程 $P=(a\rightarrow SKIP)\ \square\ (a\rightarrow b\rightarrow SKIP)$，可以用 ASP 简单描述为：

{prefix(a,skip,p). prefix(a,n1,p). prefix(b,skip,n1). prefix(succStop,skip,skip). choice(a,m,p). choice(a,n,p). }

结合规则 1—规则 16 通过求解器生成回答集存在进程 *p* 迹的信息如下：

{choice(a,n,p),eventFlow(a,b,p),eventFlow(b,succStop,p),eventFlow(succStop,succStop,skip),choice(a,m,p),eventFlow(a, succStop, p), eventFlow(succStop, succStop, skip)}

由此可知，通过前缀谓词 *prefix* 和选择谓词 *choice*，可以有效地描述非确定性选择进程。

3.3.3　一般选择

CSP 中引入的一般选择（"[]"）描述实质是对确定性选择和非确定性选择的概括表示。正如 CSP 中对一般选择 $P[]Q$ 的定义如下：

如果第一个动作不是 *P* 的可能动作，就选择 *Q*；

如果第一个动作不是 *Q* 的可能动作，就选择 *P*；

如果该动作对 *P* 和 *Q* 都是可能的，则它们之间的选择就是非确定性的。

用公式可以清晰地表示如下：

$$(c\rightarrow P[]d\rightarrow Q)=(c\rightarrow P|d\rightarrow Q),\quad \text{若 } c\neq d$$
$$=(c\rightarrow P)\ \square\ (d\rightarrow Q),\quad \text{若 } c=d$$

注：→的结合度高于[]。

即一般选择最终是根据待选子进程的初始事件是否相同而具体褪变成确定性选择和非确定性选择。CSP 之所以引入一般选择，是为了进一步提高描述的方便性和抽象性。

由上述 3.3.1 节与 3.3.2 节可知，定义的 ASP 中的前缀谓词和选择谓词可以有效地描述确定性选择与非确定性选择，二者的区分同样是看给出描述的谓词集合中，初始事件是否相同。换言之，上文定义的 ASP 描述，尤其是选择谓词 *choice*(*X*,*P*,*Q*)，足以起到一般选择描述的作用（即对确定性选择和非确定性选择的概括表示）。为此，可称 *choice*(*X*,*P*,*Q*)为一般选择谓词。

4　CSP 进程并发组合的 ASP 描述

进程是对客体行为的抽象描述，而进程与进程之间的组合可以构成对复杂系统客体行为的描述。外界环境同样可以描述为进程，所以组合方式可以理解为进程之间相互作用的抽象描述，不同的组合方式决定着进程可以通过不同的相互作用模式组合成更大的进程（或称系统）。CSP 体系提供了 3 种典型的组合方式，即并发组合、穿插组合、顺序组合。其中，并发组合要求共同事件共同参与执行；穿插组合允许各自的进程事件任意交替执行；而顺序组合规定前一进程成功终止，后一进程才能开始动作。并发组合与穿插组合可以归纳为同步组合，而顺序组合可以归纳为异步组合。本文集中研究并发组合。

4.1　并发的定义

定义 5(并发)　设 *P* 和 *Q* 是进程，$\alpha P\neq\alpha Q$，*P* 和 *Q* 并发组合成的进程对于 *P*，*Q* 共同拥有的事件共同执行，其他事件交替执行，记为：$P\parallel Q$。其中 $\alpha(P\parallel Q)=\alpha P\cup\alpha Q$，"$\parallel$"称为进程的并发组合算子。

注：当 $\alpha P=\alpha Q$ 时，并发演变为交互，交互是并发的一种特殊情况，即严格同步。

进程的并发组合有如下运算法则：

1. $P\parallel \mathrm{STOP}=\mathrm{STOP}$，设 $a\in(\alpha P-\alpha Q)$，$b\in(\alpha Q-\alpha P)$，$\{c,d\}\in(\alpha Q\cap\alpha P)$。

2. $(c\rightarrow P)\parallel(c\rightarrow Q)=(c\rightarrow(P\parallel Q))$

3. $(c\rightarrow P)\parallel(d\rightarrow Q)=\mathrm{STOP}$, if $c\neq d$

4. $(a\rightarrow P)\parallel(c\rightarrow Q)=(a\rightarrow(P\parallel(c\rightarrow Q))$

5. $(c\rightarrow P)\parallel(b\rightarrow Q)=(b\rightarrow((c\rightarrow P)\parallel Q))$

6. $(a\rightarrow P)\parallel(b\rightarrow Q)=(a\rightarrow(P\parallel(b\rightarrow Q))\mid b\rightarrow((a\rightarrow P)\parallel Q))$

运用这些法则可以把由并发算子定义的组合进程重新定义为基本结构形式构成的进程，使其中不出现并发算子。

4.2　并发的 ASP 描述

为了描述进程间的并发组合，引入并发谓词 *concurrent*(*P*,*Q*,*R*)，含义为进程 *P* 与进程 *Q* 并发组合成进程 *R*。由法则 1 可知，任何进程与 STOP 进程并发组合后的进程也是 STOP 进程，用 ASP 可以描述如下：

规则 12　*prefix*(*deadlock*,*stop*,*T*):-*prefix*(*deadlock*,*stop*,*R*),*concurrent*(*R*,*Q*,*T*),*process*(*Q*)

为了在 ASP 描述框架下同样实现由并发谓词定义的组合进程可以重新定义为基本结构形式构成的进程，本文采取并发状态作为中间过渡，多个进程并发组合时，先根据并发法则，依次生成并发进程的后续并发状态，再基于有序的并发状态集合，生成组合定义后的进程基本结构形式定义。

参照法则 2—法则 6，可知根据并发进程的初始事件，及其并发进程字母表信息，可以将具体并发情形细分为 4 种情况：

(1)并发进程的初始事件相同，且都在二者的字母表中。

(2)并发进程的初始事件不同，但都在二者的字母表中。

(3)并发进程的初始事件不同，一个初始事件独属于一个进程，另一初始事件二者共有。

(4)并发进程的初始事件不同，且两初始事件分别独属于各自的进程字母表。

对应上述 4 种情况，分别定义下列 ASP 规则，如表 5 所列。

表 5　并发组合 ASP 规则

ASP 规则	序号
concurrent(M,N,T1):-prefix(X,M,P),prefix(Y,N,Q),X=Y,P!=Q,event(X,P),event(X,Q),event(Y,P),event(Y,Q),concurrent(P,Q,T),not mod(T,10,0),#succ(T,T1).	规则 13
prefix(X,T1,T):-prefix(X,M,P),prefix(Y,N,Q),concurrent(P,Q,T),concurrent(M,N,T1).	规则 14
prefix(deadlock,stop,T):-prefix(X,M,P),prefix(Y,N,Q),X!=Y,P!=Q,event(X,P),event(X,Q),event(Y,P),event(Y,Q),concurrent(P,Q,T).	规则 15
concurrent(M,Q,T1):-prefix(X,M,P),prefix(Y,N,Q),X!=Y,P!=Q,event(X,P),not event(X,Q),event(Y,P),event(Y,Q),concurrent(P,Q,T),not mod(T,10,0),#succ(T,T1).	规则 16
prefix(X,T1,T):-prefix(X,M,P),prefix(Y,N,Q),concurrent(P,Q,T),concurrent(M,Q,T1).	规则 17
concurrent(P,N,T1):-prefix(X,M,P),prefix(Y,N,Q),X!=Y,P!=Q,event(X,P),event(X,Q),not event(Y,P),event(Y,Q),concurrent(P,Q,T),not mod(T,10,0),#succ(T,T1).	规则 18
prefix(Y,T1,T):-prefix(X,M,P),prefix(Y,N,Q),concurrent(P,Q,T),concurrent(P,N,T1).	规则 19
concurrent(M,Q,T1):-prefix(X,M,P),prefix(Y,N,Q),X!=Y,P!=Q,event(X,P). not event(X,Q),not event(Y,P),event(Y,Q),concurrent(P,Q,T),not mod(T,10,0),#succ(T,T1).	规则 20
prefix(X,T1,T):-prefix(X,M,P),prefix(Y,N,Q),concurrent(P,Q,T),concurrent(M,Q,T1).	规则 21
concurrent(P,N,T1):-prefix(X,M,P),prefix(Y,N,Q),X!=Y,P!=Q,event(X,P),not event(X,Q),not event(Y,P),event(Y,Q),concurrent(P,Q,T),not mod(T,10,0),#succ(T,T1).	规则 22
prefix(Y,T1,T):-prefix(X,M,P),prefix(Y,N,Q),concurrent(P,Q,T),concurrent(P,N,T1).	规则 23
div(X,Y,Z):-XD=Y*Z,X=XD+D,D<Y.	规则 24
mod(X,Y,Z):-div(X,Y,XY),XZ=XY*Y,X=XZ+Z.	规则 25

如同 CSP 给出的并发法则，由上述 ASP 并发规则 13－规则 23 为并发的每一种情况提供了 ASP 体系下的并发机制，由此可以自动化生成并发组合进程，并保持了前后描述结构的一致性，从而确保了多个(≥3)进程并发的实现，扩大了该 ASP 体系可描述的并发系统规模。

另外，考虑到对于两个递归进程并发状态，如果不进行限定，将会持续循环重复地并发下去，这不仅生成过多冗余重复信息，而且增加了计算复杂度，为此，定义规则 24 和规则 25 对两个进程并发次数进行限界[8](注：本文暂时限定为 10 次，可根据具体需要修改限定参数)。

4.3　实例

例如：并发系统研究中的经典模型——哲学家就餐模型，即几个哲学家在思考问题，感觉饿了就会到餐桌准备进餐。整个进餐行为如下：首先坐下来，然后依次拿起左边和右边餐叉，进而进餐，进餐完毕之后，依次放下左边和右边餐叉，最后站起离开座位继续去思考问题。图 9 为哲学家就餐模型示意图。

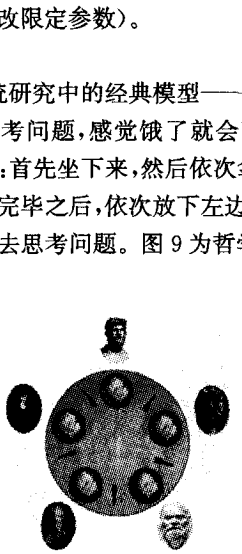


图 9　哲学家就餐模型图

每位哲学家的就餐行为刻画成进程如下：

Phi＝sit→pickFork$i$→pickFork$i$ ⊕ 1→eat→downFork$i$→downFork_i ⊕ 1→up→Phi。

注：其中⊕是模数为 3 的加法，因此 i ⊕ 1 就是第 i 位哲学家右侧的邻座。

简便起见，我们以 3 个哲学家进程(即 PHS)组成的并发模型作为研究对象。

用 CSP 描述如下：

Ph1＝sit→pickFork1→pickFork2→eat→downFork1→downFork2→up→Ph1

Ph2＝sit→pickFork2→pickFork3→eat→downFork2→downFork3→up→Ph2

Ph3＝sit→pickFork3→pickFork1→eat→downFork3→downFork1→up→Ph3

则 PHS 可以刻画为：PHS＝PH1 ‖ PH2 ‖ PH3。

通过 CSP 并发法则推导可得：

PHS ＝Ph1 ‖ Ph2 ‖ Ph3

＝uX.(sit→pickFork1→pickFork2→pickFork3→eat→downFork1→downFork2 → downFork3→up→X) ‖ Ph3

＝sit→STOP

用 ASP 描述可描述为下列前缀谓词与并发谓词集合，记为 D：

```
{%Ph1
prefix(sit,ph1_1,ph1). prefix(pickFork1,ph1_2,ph1_1).
prefix(pickFork2,ph1_3,ph1_2). prefix(eat,ph1_4,ph1_3).
prefix(downFork1,ph1_5,ph1_4). prefix(downFork2,ph1_6,ph1_5). prefix(up,ph1,ph1_6).
%Ph2
prefix(sit,ph2_1,ph2). prefix(pickFork2,ph2_2,ph2_1).
prefix(pickFork3,ph2_3,ph2_2). prefix(eat,ph2_4,ph2_3).
prefix(downFork2,ph2_5,ph2_4). prefix(downFork3,ph2_6,ph2_5). prefix(up,ph2,ph2_6).
%Ph3
prefix(sit,ph3_1,ph3). prefix(pickFork3,ph3_2,ph3_1).
prefix(pickFork1,ph3_3,ph3_2). prefix(eat,ph3_4,ph3_3).
prefix(downFork3,ph3_5,ph3_4). prefix(downFork1,ph3_6,ph3_5).
prefix(up,ph3,ph3_6).
%并发
concurrent(ph1,ph2,1). concurrent(1,ph3,11).
}
```

注：concurrent(ph1,ph2,1)中的“1”代表 ph1 ‖ ph2 并发组合成的进程名。

concurrent(1,ph3,11)中的“11”代表 ph1 ‖ ph2 ‖ ph3 并发组合成的进程名。

将 D 与规则 1－规则 29 组合作为输入，通过命令 dlv.mingw.exe 调用 ASP 求解器得到的回答集中有：

{prefix(sit,2,1),prefix(pickFork1,3,2),prefix(pickFork2,4,3),prefix(pickFork3,5,4). prefix(eat,6,5),prefix(downFork1,7,6),prefix(downFork2,8,7),prefix(downFork3,9,8),prefix(up,1,9),prefix(sit,12,11),prefix(deadlock,stop,12)}

由实验结果可知：基于上述定义的 ASP 并发规则 13－规则 23，在 ASP 描述框架体系中，可以成功地实现多个进程并发组合成一个保持一致结构形式、满足行为特性的新进程。从而新进程又可以与其他进程并发组合成更大的系统，达到了扩大并发系统规模的目的，表明了组合生成技术的有效性。

至此，较为完善的基于并发系统的 ASP 描述体系已经建立，为并发系统建模、并发系统性质验证奠定了基础。

5 基于 ASP 的模型检测

5.1 基于 ASP 的模型检测框架

第 3、4 节构建了完善的并发系统 ASP 描述体系，结合通用的性质描述规范——LTL 公式，建立了基于 ASP 的模型检测框架，如图 10 所示（注：组合方式中虚线表示本文暂不包括）。

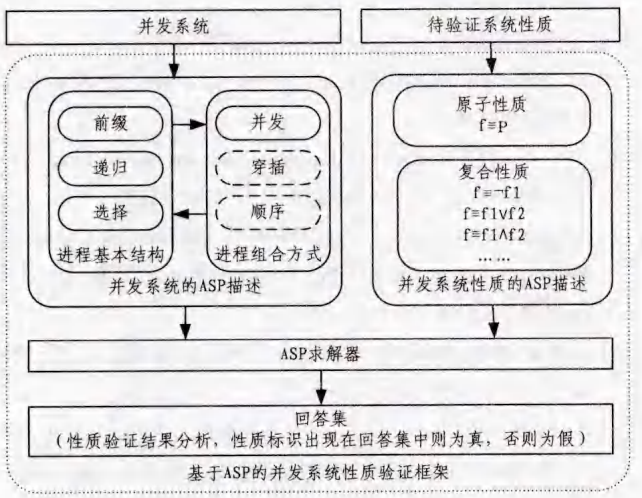


图 10 基于 ASP 的并发系统性质验证框架

为了简便起见，本文仅表示部分 LTL 性质公式定义的情形：

$\Phi ::= p \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi)$

下列给出 LTL 性质定义的 ASP 规则描述：

$f \equiv P$。表示原子性质 f 由原子命题 P 可以直接描述定义。用 ASP 规则表示为：f :- P。

$f \equiv \neg f1$。表示复杂性质 f 是在原子性质 $f1$ 为假时成立。用 ASP 规则表示为：f :- not $f1$。

$f \equiv f1 \vee f2$。表示若原子性质 $f1$ 为真，或者 $f2$ 为真，则复杂性质 f 为真。用 ASP 规则表示为：f :- $f1$.　f :- $f2$。

$f \equiv f1 \wedge f2$。表示若原子性质 $f1$ 为真且 $f2$ 也为真，则复杂性质 f 为真。用 ASP 规则表示为：f :- $f1$, $f2$。

注：P 代表描述中特定含义的谓词。

5.2 基于 ASP 的并发系统性质验证流程

至此，可以给出基于 ASP 的并发系统性质验证流程：

(1)将并发模型的 CSP 描述转化成 ASP 描述，记作 D。

(2)将待验证性质的 LTL 公式描述为 ASP 规则，性质标识(即规则头部)记作 f。

(3)将 $D \cup P$ 作为求解器的输入，求解回答集。根据回答集结果确定待验证性质是否成立，即如果性质标识出现在回答集中，则证明性质成立；否则不成立。

注：因为多个性质可以描述成多个 ASP 规则，且回答集中会出现所有成立性质的性质标识，所以该性质验证方法决定了本文建立的基于 ASP 的模型检测证框架一次可以验证多条性质。

5.3 实例

例 1　验证 3.3.1 节(c)中的换钱机器 CH10D 进程与顾客一次成功换钱进程 GUST＝in10p→out5p→out5p→SKIP 并发构成模型的性质，模型记做 CHOT＝(CH10D ‖ GUST)。易知该并发模型中顾客确定性地选择了投入 10 便士换两个 5 便士的组合。为此必然满足性质 a：两个 out5p 事件相继发生。

首先将系统模型 CHOT 用 ASP 描述，记为 3.3.1 节(c)中的集合 $D \cup E$，其中 E 如下：

{prefix(in10p, q, gust). prefix(out5p, q1, q). prefix(out5p,skip,q1).

concurrent(ch10D,gust,1).}(注：其中 1 代表 CHOT)

其次将要验证性质的 LTL 公式描述成 ASP 规则。

性质 a　LTL 描述：f1≡G(out5p→X out5p)

ASP 描述：P≡{f1 :-prefix(out5p,N,M),prefix(out5p,W,N),M!＝p,M!＝q.}

最后将 D∪E∪P3 以及规则 1－规则 25 作为 DLV 求解器的输入，可得回答集如图 11 所示。

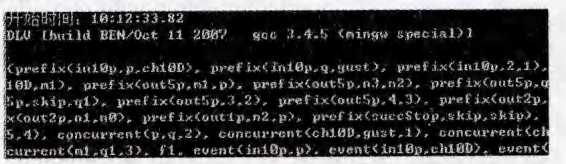


图 11　例 1 性质验证回答集

实验结果中性质标识 $f1$ 出现在回答集中，说明对于 CHOT 并发系统，性质 a 是可满足的，与事实一致。

例 2　验证 4.3 节中 3 个哲学家进程并发组成的系统模型 (PHS＝Ph1 ‖ Ph2 ‖ Ph3)的性质。

通常哲学家进餐模型满足如下性质：

性质 1　一个哲学家不能同一时刻拿起左右两边的叉子。

性质 2　一把叉子不能同时被两个哲学家拿起。

首先将系统模型 PHS 用 ASP 描述，即为 4.3 节中的集合 D。

其次将要验证性质的 LTL 公式描述成 ASP 规则：

性质 1：LTL 描述：f1≡G¬(pickForki∧pickForki⊕1)

ASP 描述：P1＝{f1 :- not t1. t1 :- prefix(pickForki,Q,P),prefix(pickForkii⊕1,Q,P).}

性质 2：LTL 描述：f2≡G¬(pickForki∧pickForkj∧i＝j)

ASP 描述：P2＝{f2 :- not t2. t2 :- prefix(pickForki,M,P),prefix(pickForki,N,P),M!＝N.}

最后将 $D \cup P1 \cup P2$ 以及规则 1－规则 25 作为 DLV 求解器的输入，可得回答集如图 12 所示。

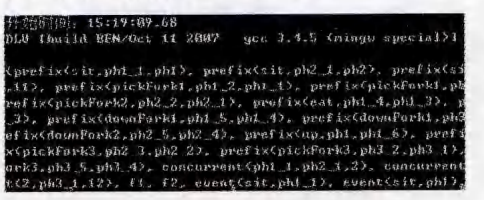


图 12　例 2 性质验证回答集

实验结果中性质标识 $f1$，$f2$ 出现在回答集中，说明对于PHS并发系统，性质1与性质2是可满足的，与事实一致。

以上实例说明本文建立的ASP的模型检测框架以及并发系统性质验证方法是可行的、有效的，能够验证多个(≥ 3)进程并发组合的模型，并一次验证多条性质。

结束语 本文研究了基于ASP的CSP进程描述与组合生成技术，构建了并发系统的ASP建模与验证框架，完善了可描述并发进程形态，扩大了可验证并发系统规模。实验结果表明了本文构建的ASP描述体系、并发组合进程生成技术以及基于该ASP描述体系的性质验证的有效性。然而随着系统规模的扩大，面对状态爆炸的难题，如何针对特定类型的待验证性质，对ASP构建的模型进行有效的抽象约简，提高性质验证效率，是下一阶段的研究重点，同时性质验证不满足时的反例生成技术也是研究点之一。

参 考 文 献

[1] Hoare C A R. Communicating Sequential Processes [M]. http://www.usingcsp.com/cspbooks,2004

[2] Roscoe A W. The Theory and Practice of Concurrency [M]. Prentice Hall,1998

[3] Clarke E M,Grumberg O,Peled D. Model Checking [M]. Cambridge:MIT Press,2001:35-49

[4] Baral C. Knowledge Representation,Reasoning,and Declarative Problem Solving [M]. Cambridge Press,2003

[5] 赵岭忠，张超，钱俊彦. 基于ASP的CSP并发系统验证研究[J]. 计算机科学，2012，39(12)：133-136

[6] Durobvin J. Efficient Symbolic Model Checking of Concurrent System [D]. Aalto University School of Science Department of Information of Computer Science,Doctoral Dissertation,2011

[7] D'Silva V,Kroening D,Weissenbacher G. A Survey of Automated Techniques for Formal Software Verification [J]. IEEE Trans. on CAD of Integrated Circuits and Systems,2008,27(7):1165-1178

[8] Heljanko K,Niemelä I. Bounded LTL model checking with stable models [J]. TPLP,2003(4/5):519-550

[9] Armstrong P J,Goldsmith M,Lowe G,et al. Recent Developments in FDR [J]. CAV,2012,7358:699-704

[10] Palikareva H,Ouaknine J,Roscoe B. Faster FDR Counterexample Generation Using SAT-Solving [C]// ECEASST. 2009

[11] Leone N,Pfeifer G,Faber W,et al. The DLV system for knowledge representation and reasoning [J]. ACM TOCL,2006,7(3):499-562

[12] Lierler Y,Maratea M. Cmodels-2:SAT-Based Answer Set Solver Enhanced to Non-tight Programs [J]. Proc. LPNMR,2004,2923:346-350

[13] Ilik D,Lee G,Herbelin H. Kripke models for classical logic [J]. Ann. Pure Appl. Logic,2010,161(11):1367-1378

[14] De Angelis E,Pettorossi A,Proietti M. Synthesizing Concurrent Programs Using Answer Set Programming [J]. Fundam. Inform,2012,120(3/4):205-229

[15] 蒋屹新，林闯，曲扬，等. 基于Petri网的模型检测研究[J]. 软件学报，2004，15(9)：1265-1276

[16] Niemelä I,Simons P,Syrjänen T. Smodels:A System for Answer Set Programming [C]// Proceeding of the 8th International Workshop on Non-Monotonic Reasoning. 2000

(上接第132页)

参 考 文 献

[1] Adleman L. Molecular computation of solutions to combinatorial problems[J]. Science,1994,266(11):1021-1024

[2] Lipton R J. DNA solution of computation problems [J]. Science,1995,268(4):542-545

[3] Quyang Q, Kaplan P D, Liu Shu-mo, et al. DNA solution of maximal clique problem[J]. Science,1997,278(17):446-449

[4] Head T,Rozenberg G,Bladergroen R R,et al. Computing with DNA by operating on plasmids[J]. Biosystem,2000,57:87-93

[5] Liu Q H,Wang L,Anthony G F,et al. DNA computing on surface[J]. Nature,2000,403(13):175-179

[6] 高琳，马润年，许进. 基于质粒DNA匹配问题的分子算法[J]. 生物化学与生物物理进展，2002，29(5)：820-823

[7] 刘文斌，高琳，王淑栋，等. 最大匹配问题的DNA表面计算模型[J]. 电子学报，2003，31(10)：1496-1499

[8] 谢飞舟，汤建钢. 最大匹配问题的DNA试管计算模型[J]. 甘肃联合大学学报，2012，26(6)：65-68

[9] Dong Ya-fei, Zheng Xue-dong. Molecule Algorithm for Perfect Matching Problem Based on Sticker Models[C]// Proceeding of International Conference on Intelligent Mechatronics and Automation. China,2004:249-253

[10] 周旭，李肯立，乐光学，等. 一种最大匹配问题DNA计算算法[J]. 计算机研究与发展，2011，48(11)：2147-2154

[11] Roweis S,Winfee E,Burgoyne R,et al. A sticker-based model for DNA computation[J]. Comput Biol,1998,5(4):615-629

[12] 许进，董亚非. 粘贴DNA计算机模型(I)：理论[J]. 计算机学报，2004，49(3)：205-212

[13] 许进，董亚非. 粘贴DNA计算机模型(II)：理论[J]. 计算机学报，2004，49(4)：299-307

[14] Bondy J A,Mum U S R. Graph Theory with Applications[M]. The Macmillan Press,LTD,1976

[15] Braich R S,Johnson C,Rothemund P W K,et al. Solution of a satisfiability problem on a gel-based DNA computer,2000 [C]// Proceedings of the 6th International Conference on DNA Computation in the Springer-Verlag Lecture Notes in Computer Science series. 2000,2054:27-42