基于进程迹的CSP模型验证框架

赵岭忠 翟仲毅 钱俊彦
(桂林电子科技大学计算机科学与工程学院 桂林 541004)

摘 要 CSP(Communicating Sequential Processes)是构建并发系统和网络安全协议的经典方法。当前主流的CSP模型验证方法需将进程转化为迁移系统,转化过程比较复杂;性质采用迹进行规范,不利于活性的描述。提出了一种基于进程迹的CSP模型验证框架,其性质采用通用的规范方法LTL进行描述。利用ASP(Answer Set Programming)技术实现了一个CSP验证系统。实验表明,与类似系统相比,该系统的描述能力更强,验证结果的准确性更高,在性质不满足时还可提供反例。

关键词 通信顺序进程(CSP),并发系统,迹模型,回答集编程(ASP)
中图法分类号 TP311 **文献标识码** A

Framework for Model Checking CSP with Traces of Processes

ZHAO Ling-zhong ZHAI Zhong-yi QIAN Jun-yan
(School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract Communicating Sequential Processes is a classic method for describing concurrent systems and network security protocols. The verification for CSP is a complex conversion process, which translates processes into a label transition system and describes the properties to be verified with traces. The main problem of the method is in the description of liveness properties. This paper proposed a framework for model checking CSP with traces, in which properties are specified by LTL, a universal property specification language. Finally, a verification system for CSP was implemented with answer set programming. It is shown that the ability of the system for describing CSP processes is more powerful and the verification accuracy of the system is higher than the similar system developed previously. When a property is not qualified, the system returns counterexamples for the property.

Keywords Communicating sequential processes, Concurrent system, Trace model, Answer set programming

1 引言

CSP是构建和验证并发系统的经典方法。CSP语言是一种高级语言,提供了高度抽象的描述方法,并且具有直观性[1]。CSP的验证主要包括程序正确性证明和模型检测两种方法。Hoare首次提出了一种公理化方法来证明程序的正确性,它通过一个逻辑系统和断言完成程序的证明;该方法主要在早期采用,并且自动化程度不高[2]。目前,CSP主要侧重于模型检测技术的研究[3-5]。FDR是主流的CSP模型检测工具,它的主要思想是:根据CSP的操作语义,将CSP模型转化为相应的迁移系统,然后对迁移系统进行性质验证,其中性质采用迹的表达式进行描述[3,4,6]。基于迁移系统的模型检测技术已经十分成熟,这为CSP的验证带来了很大便利。而CSP模型到迁移系统的转化是个十分复杂的过程,这将不利于系统的扩展和修改;此外,采用迹来规范性质不利于活性的描述。本课题组首次提出了一种基于LTL和CTL的验证方法[7]。LTL、CTL是通用的性质规范方式,并且能很好地对活性、安全性、公平性进行描述[8]。该方法的不足之处在于只提供了前缀进程和递归进程的验证方法,对于较为复杂的一般选择进程和非确定选择进程无法进行验证;此外,该方法通过并发状态来验证模型性质,相应的理论支撑比较薄弱。

鉴于此,本文提出了一种基于进程迹模型的CSP验证框架。迹模型属于数学模型,是在迹语义下对CSP进行解释的一种模型。它主要包括两方面的用途:一方面,为CSP语言的扩展提供正确性证明;另一方面,迹模型可以完成性质验证[9]。在一定条件下,CSP模型在迹语义下生成的迹模型与在操作语义下转化成的迁移系统是等价的;因此,基于迹模型的验证和基于迁移系统的验证在一定条件下是等价的[10]。该验证框架主要包括3部分:1. CSP进程到迹模型的转化;2. 并发操作下迹模型的计算;3. 基于迹模型的性质验证。其中,前两部分是将CSP模型转化为相应的迹模型;第3节根据定义的基于迹的LTL可满足语义对迹模型进行验证,当性质不满足时,系统可以提供反例。最后,本文利用ASP技术实现了一个CSP验证系统。ASP是一种纯声明性程序设计方法,

到稿日期:2013-01-23 返修日期:2013-06-17 本文受国家自然科学基金(61262008,61063002),广西自然基金(2011GXNSFA018166,2011GXNSFA018164),广西可信软件重点实验室基金(kx201113)资助。
赵岭忠(1977—),男,博士,教授,主要研究方向为形式化技术,E-mail:zhaolingzhong163@163.com;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。

它适用于复杂的、难以求解的以及知识密集型的问题[11]。此外，ASP存在高效的求解器，可对问题自动化求解[12,13]。CSP中的并发和验证是复杂度较高的求解问题，非常适合采用ASP方法进行求解；ASP纯声明性的特点也使该验证系统便于修改和扩展。为了更好地进行实验，本文利用CSP设计了几种不同的哲学家进程。实验表明，该系统在保证验证效率的同时，验证的准确性更高，同时在性质不满足时还可提供反例。

2 基础知识

2.1 ASP编程

ASP是一种声明式程序设计(Declarative Programming)方法，它是由稳定模型语义扩展而来的。若 A 是一个原子，则有 A 或者 $\sim A$ 都称为文字，其中 A 称为正文字，$\sim A$ 称为负文字；A 和 $\sim A$ 称为一对互补字。

一个扩展析取逻辑程序 P 是一个规则集；并且每条规则 r 满足如下形式：

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

式中，$n \geqslant m \geqslant k \geqslant 0$，$L_i$ 是一个文字，not表示失败及否定。这里用 $head(r)=\{L_1,\cdots,L_k\}$ 表示规则 r 的头部，$pos(r)=\{L_1,\cdots,L_m\}$ 表示规则 r 的体部的正文字集合，$neg(r)=\{L_{m+1},\cdots,L_n\}$ 表示规则 r 的体部的负文字集合。当头部为空时，称此规则 r 为约束[14]。

假设 Π 为扩展析取逻辑程序，Lit是基文字的集合。当 Π 不包含not时，Π 的回答集 S 是关于Lit的最小子集。S 集合满足以下条件：

1)对于任意规则 $r \in \Pi$，如果有 $pos(r) \subseteq S$，并且存在某个 $l \in head(r)$，则可得到 $l \in S$。

2)如果 S 包含互补的文字，那么 $S=Lit$。

若 Π 包含not，将程序 Π 关于集合 S 的约简记为 Π^s。其计算过程如下：

1)如果规则 r 的 $body^-(r)$ 不成立，则从程序 Π 中删除规则 r；

2)把 $body^-(r)$ 从规则 r 中删除。

性质1[15] 如果集合 S 是约简 Π^s 的答案集，那么 S 同时也是程序 Π 的答案集。

根据性质1，可以求得包含not的程序 Π 的回答集。

generate-and-test是ASP程序设计中的经典方法[16]。其中，generate模块由一组普通规则组成，该模块的回答集构造出了问题的所有可能的解。在此基础上，构造一个包含完整性约束规则、选择规则的规则集，即test模块；它的作用就是消除generate模块中无效的解，最终得到问题的正确结果。在模块化程序设计中，使用generate-and-test方法可以直观地体现出ASP程序设计中声明性特点。本文中ASP程序的书写规范采用DLV系统的相关规定(用:-来代替←；每条规则以英文句号结束)[17]。例如，程序 $a \leftarrow b,\text{not } c$ 可表示为 a :-b,not c.。

2.2 CSP基础理论

在CSP理论中，进程是描述系统中客体的基本单位；进程的基本结构包括：前缀、递归、一般选择、非确定选择[1]。下面分别对它们进行简要介绍。

设 x 是一个事件，P 是一个进程，αP 表示进程 P 的字母表；若 $x \in \alpha P$，则 $x \rightarrow P$ 表示一个进程，该进程首先执行事件 x，然后按照进程 P 的行为进行。

前缀：对于进程 $x \rightarrow P$，若 P 是可终止的或 $P=$STOP(STOP表示空进程，即什么也不做)，则 $(x \rightarrow P)$ 是一种前缀结构；它表示有限行为的进程。

递归：X 表示进程，$F(X)$ 表示进程 X 的卫式，即 $F(X)$ 服从 $(a \rightarrow X)$ 形式，若 X 是不可终止的，则 $X=F(X)$ 表示一个递归进程；它可以表示无穷行为的进程。

一般选择：P 和 Q 是两个进程，那么 $P \square Q$ 表示一个一般选择进程；该进程的行为可以选择 P 的行为，也可以选择 Q 的行为，它的选择由外界控制。假设进程 P 为：$x \rightarrow T$，进程 Q 为：$y \rightarrow S$，若外部进程提供事件 x，则选择进程 P；若外部进程提供事件 y，则选择进程 Q。

非确定选择：P 和 Q 是两个进程，那么 $P \sqcap Q$ 表示一个非确定进程；该进程的行为可以选择 P 的行为，也可以选择 Q 的行为，它的选择是任意的、不受外界控制的。该进程的字母表 $\alpha(P \sqcap Q)$ 符合条件：$\alpha(P \sqcap Q)=\alpha P=\alpha Q$。

进程的并发方式是构造并发系统的核心；它包括交互、并发、穿插3种方式。其中，并发操作须满足以下条件：1. 交互双方进程有部分字母表相同；2. 进程间的共同事件为通信事件，要求共同参与；其余事件则不需同时参与。假设有事件 a、b、c、d，进程 P、Q，且 $a \in (\alpha P-\alpha Q)$，$b \in (\alpha Q-\alpha P)$，$\{c,d\} \in (\alpha Q \cap \alpha P)$；那么可以得到如下CSP并发规则：

1. $(c \rightarrow P) \parallel (c \rightarrow Q)=(c \rightarrow (P \parallel Q))$
2. $(c \rightarrow P) \parallel (d \rightarrow Q)=$STOP 若 $c \neq d$
3. $(a \rightarrow P) \parallel (c \rightarrow Q)=(a \rightarrow (P \parallel (c \rightarrow Q))$
4. $(c \rightarrow P) \parallel (b \rightarrow Q)=(b \rightarrow ((c \rightarrow P) \parallel Q))$
5. $(a \rightarrow P) \parallel (b \rightarrow Q)=(a \rightarrow (P \parallel (b \rightarrow Q)) \mid b \rightarrow ((a \rightarrow P) \parallel Q))$

上面介绍了CSP进程以及进程并发的描述方法。下面将介绍CSP的一种数学模型——迹模型。CSP的迹模型是在迹语义下对CSP模型进行解释的一种模型，而迹语义属于一种指称语义。

定义1[10](CSP指称语义) 所谓的CSP指称语义，即为函数 $S[[.]]$，其中 S 是由CSP语言 L 构建的一个CSP模型到数学模型 M 的一个函数。

定义2[10](有穷迹语义) CSP的迹语义，即为函数 $S[[.]]$，其中 S 是由CSP语言 L 构建的一个CSP模型到有穷 *trace* 模型 M 的一个函数。

本文只在有穷迹语义下对进程进行解释，因此进程的迹模型只含有有限条迹。为了定义迹模型的概念，这里介绍一下迹的含义。CSP中，迹表示进程的一条行为序列，该序列由进程可执行的事件组成，并以事件发生的先后顺序进行排列。

定义3[10](迹模型) 对于一个进程 P，P 的有穷迹模型为 $traces(P)$，其中，$traces(P)$ 包含进程 P 所有可能的迹，且 $traces(P)$ 中只包含有穷条迹。

下面给出了几种基本结构进程的迹模型的计算，对于复杂的CSP进程可以递归调用这几个基本结构进行求解。

若进程 P 为STOP进程，则P的迹模型为：

$traces(\text{STOP})=\{t \mid t=\langle\rangle\}$

若进程 P 为：$P=a \rightarrow$STOP，则 P 的迹模型为：

$traces(P)=\{\langle\rangle\} \cup \{\langle a\rangle\}$

若进程 P 为：$P=a\rightarrow Q$(其中，进程 $Q=a\rightarrow\cdots\rightarrow$STOP 的形式)，则 P 的迹模型为：

$traces(P)=\{\langle\rangle\}\cup\{\langle a\rangle\frown t\mid t\in traces(Q)\}$

其中，“$\frown$”为迹之间的连接算子。

若进程 P 为：$P=a\rightarrow\cdots\rightarrow b\rightarrow P$，设进程 Q 为进程 P 的前缀进程 $Q=a\rightarrow\cdots\rightarrow b\rightarrow$STOP，进程 Q 的迹模型为 T，T 的所有子迹模型为 sub_T，则进程 P 的迹模型为：

$traces(P)=\{\langle\rangle\}\cup\{T^n\}\cup\{sub\_T^n\}$

若进程 P 为：$Q\square T$(Q 和 T 为前缀进程或递归进程)，则进程 P 的迹模型为：

$traces(P)=\{q\mid q\in traces(Q)\}\cup\{t\mid t\in traces(Q)\}$

若进程 P 为：$Q\prod T$(Q 和 T 为前缀进程或递归进程)，则进程 P 的迹模型为：

$traces(P)=\{q\mid q\in traces(Q)\}\cup\{t\mid t\in traces(Q)\}$

3 哲学家进程的 CSP 描述

本节将利用 CSP 对哲学家进程进行描述。下面分别介绍 3 种描述形式。

第 1 种形式采用递归方法进行描述，具体的描述内容如式(1)所示：

$$Phi=i.\ sit\rightarrow i.\ pick\_fork.\ i\rightarrow i.\ pick\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ up\rightarrow Phi \quad (1)$$

该方式对哲学家 Phi 的要求比较严格，他需满足：先坐下来，拿起左边的叉子，再拿右边的叉子，接着进餐进完，然后放下左边的叉子，再放右边的叉子，最后离开座位，如此重复。

第 2 种形式采用一般选择及其嵌套方式进行描述，具体的描述结果如式(2)所示：

$$\begin{aligned}Phi=i.\ sit\rightarrow(&i.\ pick\_fork.\ i\rightarrow i.\ pick\_fork.\ i\oplus 1\rightarrow(i.\ down\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ up\rightarrow Phi_c\\ &\square i.\ down\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ up\rightarrow Phi_c)\square i.\ pick\_fork.\ i\oplus 1\rightarrow i.\ pick\_fork.\ i\rightarrow(i.\ down\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ up\rightarrow Phi_c\\ &\square i.\ down\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ up\rightarrow Phi_c)\end{aligned} \quad (2)$$

与式(1)不同，哲学家进程开始时对于拿起和放下叉子的左右顺序不做限制，当选定某一方式后，哲学家按照已经选择的方式进行。每条分支最后调用的 Phi_c 表示从初始事件沿着所在路径循环进行的一个进程。例如，Phi 第一条分支所调用的 $Phi_c=i.\ sit\rightarrow i.\ pick\_fork.\ i\rightarrow i.\ pick\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ up\rightarrow Phi_c$。该哲学家进程分支的选择由与其交互的其他哲学家进程或者叉子进程提供。

第 3 种形式采用非确定选择及其嵌套方式进行描述，具体的描述结果如式(3)所示：

$$\begin{aligned}Phi=i.\ sit\rightarrow(&i.\ pick\_fork.\ i\rightarrow i.\ pick\_fork.\ i\oplus 1\rightarrow(i.\ down\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ up\rightarrow Phi_c\prod i.\ down\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ up\rightarrow Phi_c)\\ &\prod(i.\ pick\_fork.\ i\oplus 1\rightarrow i.\ pick\_fork.\ i\rightarrow(i.\ down\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ up\rightarrow Phi_c\prod i.\ down\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ up\rightarrow Phi_c)\end{aligned} \quad (3)$$

式(3)与式(2)的含义基本相同，Phi_c 的定义也与式(2)相同；不同点在于式(2)的选择由其他哲学家或叉子提供，式(3)的选择由自身提供。且进程式(3)通过规则 $x\rightarrow(P\prod Q)=(x\rightarrow P)\prod(x\rightarrow Q)$ 可以转化为如下形式：

$$\begin{aligned}Phi=(&i.\ sit\rightarrow i.\ pick\_fork.\ i\rightarrow i.\ pick\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ up\rightarrow Phi_c)\prod\\ &(i.\ sit\rightarrow i.\ pick\_fork.\ i\rightarrow i.\ pick\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ up\rightarrow Phi_c)\prod(i.\ sit\rightarrow i.\ pick\_fork.\ i\oplus 1\rightarrow i.\ pick\_fork.\ i\rightarrow i.\ down\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ up\rightarrow Phi_c)\prod(i.\ sit\rightarrow i.\ pick\_fork.\ i\oplus 1\rightarrow i.\ pick\_fork.\ i\rightarrow i.\ down\_fork.\ i\oplus 1\rightarrow i.\ down\_fork.\ i\rightarrow i.\ up\rightarrow Phi_c)\end{aligned}$$

4 基于进程迹的 CSP 模型验证框架

基于进程迹的 CSP 模型框架如图 1 所示，它主要包括 3 部分：1. 进程到相应迹模型的转化；2. 并发操作下迹模型的计算；3. 定义基于迹的 LTL 可满足语义(这里的性质采用 LTL 进行描述)，构建基于迹的性质验证方法。下面分别介绍每部分的具体内容。

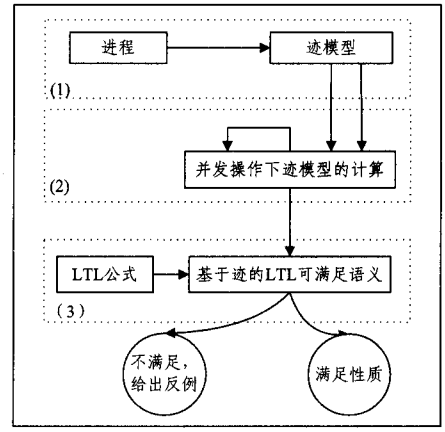

图 1 基于进程迹的 CSP 模型验证框架

第 1 部分的主要任务是将 CSP 语言描述的进程转化为相应的迹模型。其中，CSP 可以描述的进程形式主要包括：前缀进程、递归进程、一般选择进程和非确定选择进程。在实现的过程中，需要分别构造每种进程到迹模型的自动化转化方法；在 5.2 节中给出了一组进程到迹模型转化的自动化方法。

例如哲学家 ph1=1. sit→1. pick_fork. 1→i. pick_fork. 2→1. down_fork. 1→1. down_fork. 2→1. up→Ph1，经过前缀进程到迹模型的自动转化方法，可得到 $traces(P)$。其中，⟨1. sit⟩，⟨1. sit，1. pick_fork. 1，i. pick_fork. 2⟩和⟨1. sit，1. pick_fork. 1，i. pick_fork. 2，1. down_fork. 1，1. down_fork. 2，1. up，…⟩都为 $traces$(ph1)中的迹；由于 $traces$(ph1)中的迹比较多，不再一一列出。

第 2 部分需要完成并发操作下迹模型的计算。为了便于计算 CSP 的迹模型，统一采用进程的迹参与并发操作。其中，参与并发的迹可以是进程直接转化成的迹，也可以是并发后生成的中间进程的迹。两条进程的迹进行并发时，需依据基于迹的并发规则进行；当参与并发的迹不满足规则时，这组并发组合将被消除；最终生成满足并发规则的 CSP 的迹模型。基于迹的并发规则的设计是本部分的关键任务，它是根据迹的描述形式和并发思想设计的一组规则，与 2.2 节中的并发规则是等价的。在 5.2 节中，用 ASP 技术实现了一组基于迹的并发规则。下面举例说明此部分的作用。

哲学家 ph1 和 ph2 并发，即 ph1 ‖ ph2，ph*i* 都采用式(1)形式进行描述；则 ph1 ‖ ph2 可以转化为 *traces*(ph1)和 *traces*(ph2)的迹之间的并发。例如，*traces*(ph1)中的〈1. sit〉和 *traces*(ph2)中的〈2. sit〉并发时，不满足迹的并发规则，因此〈1. sit〉和〈2. sit〉的并发组合将被删除；而 *traces*(ph1)中的〈1. sit,1. pick_fork. 1,*i*. pick_fork. 2,1. down_fork. 1,1. down_fork. 2,1. up,…〉和 *traces*(ph2)中的〈2. sit,2. pick_fork. 2,2. pick_fork. 3,2. down_fork. 2,2. down_fork. 3,2. up,…〉并发时会生成一条迹 t=〈1. sit,1. pick_fork. 1,*i*. pick_fork. 2,1. down_fork. 1,1. down_fork. 2,1. up,2. sit,2. pick_fork. 2,2. pick_fork. 3,2. down_fork. 2,2. down_fork. 3,2. up,…〉，t 符合并发规则，所以 $t\in$*traces*(ph1 ‖ ph2)。

第 3 部分主要是对 CSP 进行性质验证。本部分的关键任务是设计基于迹模型的 CSP 验证方法和定义基于迹的 LTL 的可满足语义。经过前两部分的计算可以得到 CSP 的迹模型，它包含 CSP 的所有可能的迹。因此，基于迹模型的 CSP 的验证可以归结为迹模型中每条迹的验证。为了对每条迹进行验证，需要构造基于迹的 LTL 可满足语义。LTL 公式的语法结构如下：

$\psi ::= p\in A|\neg\psi|\psi_1\wedge\psi_2|\psi_1\vee\psi_2|G\psi|F\psi|X\psi$

简要起见，这里只引用 G、F、X 3 个时态算子；其中，G 算子表示“未来的所有状态”，F 算子表示“未来的某一状态”，X 算子表示“下一个状态”。用 π 来表示一条迹，则基于迹的 LTL 公式的可满足语义如下定义：

$\pi|=p$　当且仅当 $p\in L(\pi(i))$

$\pi|=\neg p$　当且仅当 $\pi|\neq p$

$\pi|=\psi_1\wedge\psi_2$　当且仅当 $\pi|=\psi_1$ 且 $\pi|=\psi_2$

$\pi|=\psi_1\vee\psi_2$　当且仅当 $\pi|=\psi_1$ 或 $\pi|=\psi_2$

$\pi|=X\psi$　当且仅当 $\pi_1|=\psi$

$\pi|=F\psi$　当且仅当对于某个 $i,\pi_i|=\psi$

$\pi|=G\psi$　当且仅当对于所有 $i,\pi_i|=\psi$

当给定一个待验证的 LTL 公式时，只要递归调用上述的可满足判定方法即可验证一条迹是否满足 LTL 公式。例如，验证 ph1 ‖ ph2 是否满足 P=F1. up∧F2. up，转化为验证每条迹 $s\in$*traces*(ph1 ‖ ph2)是否满足 P；调用上述可满足语义，即验证 s 是否满足 F1. up 和 F2. up。假设 $s=t=$〈1. sit,1. pick_fork. 1,*i*. pick_fork. 2,1. down_fork. 1,1. down_fork. 2,1. up,2. sit,2. pick_fork. 2,2. pick_fork. 3,2. down_fork. 2,2. down_fork. 3,2. up,…〉，t 中存在 $L(\pi(12n-6))$=1. up 并且 $L(\pi(12n))$=2. up，n 为自然数，即可判定 t 满足 P。

5 CSP 模型验证的 ASP 实现

5.1 进程的知识表示

在 ASP 框架下，基于进程迹的 CSP 模型的验证过程可以归结为回答集求解过程。由第 4 节的验证框架可知，回答集程序包括 CSP 进程的 ASP 表示、CSP 进程到迹模型的 ASP 转化规则、并发操作下迹模型计算的 ASP 规则和 LTL 公式的 ASP 描述。

为了对进程和迹进行描述，这里引入谓词 *flow*(X,Y,P)，其表示在进程 P 的事件 X 比事件 Y 先发生，谓词 *preset*(P,Q)表示 P 进程是 Q 进程前缀式中所有事件组成的进程(也就是说 P 是 Q 的子进程)，*invoking*(P,Q)表示进程 P 的构建过程中将调用进程 Q；*repeat*(P,Q)表示进程 Q 的行为是进程 P 自身调用形成的。

例如，进程 Q 的一条 *trace* 为：〈u,v,w,x,y,z〉；可用 ASP 表示为：

flow(u,v,q). *flow*(v,w,q). *flow*(w,x,q). *flow*(x,y,q). *flow*(y,z,q).

前缀进程 P 的 CSP 形式为：$P=w\rightarrow x\rightarrow y\rightarrow z\rightarrow$STOP，可用 ASP 表示为：

flow(w,x,q). *flow*(x,y,q). *flow*(y,z,q). *preset*(q,p).(这里用 q 表示前缀式中的子进程，p 表示进程 P)。

递归进程 P 为：$P=w\rightarrow x\rightarrow y\rightarrow z\rightarrow P$；可用 ASP 表示为：

flow(w,x,q). *flow*(x,y,q). *flow*(y,z,q). *preset*(q,p). *repeat*(p,q).

一般选择进程和非确定选择进程是由前缀结构和递归结构组成的分支结构进程，因此利用前缀进程和递归进程的表示方法很容易地对选择进程进行表示。

5.2 迹模型的计算

为了计算迹模型，需构造一组 ASP 规则。前缀进程和递归进程的转化过程比较简单，不再详述。下面对一般选择进程的转化过程进行分析。一般选择进程是一种分支结构的进程；进程执行哪个分支的行为，可由环境提供选择。引入谓词 *choice*(X,N)、*choice_p*(N)表示提供的选择；其中，*choice*(X,N)表示选择 N 进程中的 X 事件，*choice_p*(N)表示进程 N 将被选择。其转化过程需要构造以下几条 ASP 规则：

(1)根据选择机制提供的选择，可直接完成进程中分支事件选择，表示为：

flow(X,Y,N):-*pre*(X,Y,N),*choice*(Y,N).

(2)已被选择的分支，其后续线形路径也将被选择，可表示为：

flow(Y,Z,N):-*flow*(X,Y,N),*pre*(Y,Z,N).

(3)当在选择结构中嵌套了选择结构时，需构造如下规则来完成整个进程的选择：

choice_p(P):-*choice*(X,P).

flow(Y,Z,N):-*flow*(Y,Z,M),*isset*(M,N),*choice_p*(N).

其中，*pre*(X,Y,N)和 *isset*(M,N)是引入的辅助谓词；*pre*(X,Y,N)表示在进程 P 中先执行事件 X，然后执行事件 Y，*isset*(M,N)表示进程 M 是进程 N 的分子结构中的一个子进程。

一般选择进程的每条分支要么终止，要么递归调用某一进程。为了实现进程的递归调用，引入谓词 *aux_ repeat*(P,Q)表示 *repeat*(P,Q)是预备选择的。其递归调用过程可以表示为：

repeat(P,Q):-*pre_last*(X,P),*aux_repeat*(P,Q).

经过上述规则，一般选择进程可以转化为相应的迹模型。非确定选择进程到迹模型的转化方法和一般选择的基本相同；与之不同的是，非确定进程的选择的提供是由自身完成的。因此，可使用 *choice*(X,P)∨*choice*(Y,P)规则为其提供选择。

为了生成完整的迹模型，还需构造进程迹之间进行并发的 ASP 规则。在 2.2 节已经给出了进程并发的所有规则。根据这些规则可以计算并发操作下 CSP 的迹模型。例如，进程 P 和 Q 进行并发，当前参与并发的迹形式为 *flow*(X,Y,

P)和 $flow(X,Y,Q)$,其中 X,Y 是进程 P、Q 的共有事件。其并发规则可用 ASP 表示为:

$flow(X,Y,T)$:-initial_flow(X,Y,P),initial_flow(X,Y,Q),notfail_flow(X,Y,P),not fail_flow(X,Y,Q),event(X,P),event(Y,P),event(X,Q),event(Y,Q),concurrent(P,Q,T)

其中,谓词 initial_flow(X,Y,P)表示当前参与并发的事件流为 flow(X,Y,P),而 fail_flow(X,Y,P)表示 flow(X,Y,P)是已经失效的,event(X,P)表示事件 X 是进程 P 的事件,concurrent(P,Q,T)表示进程 P 和 Q 并发生成进程 T。

迹的并发操作可分为 10 种不同的形式,如图 2 所示。由此 10 种形式即可构造出完整的迹的并发规则。其中,每种形式的 ASP 描述与上述表示基本类似,不再一一阐述。

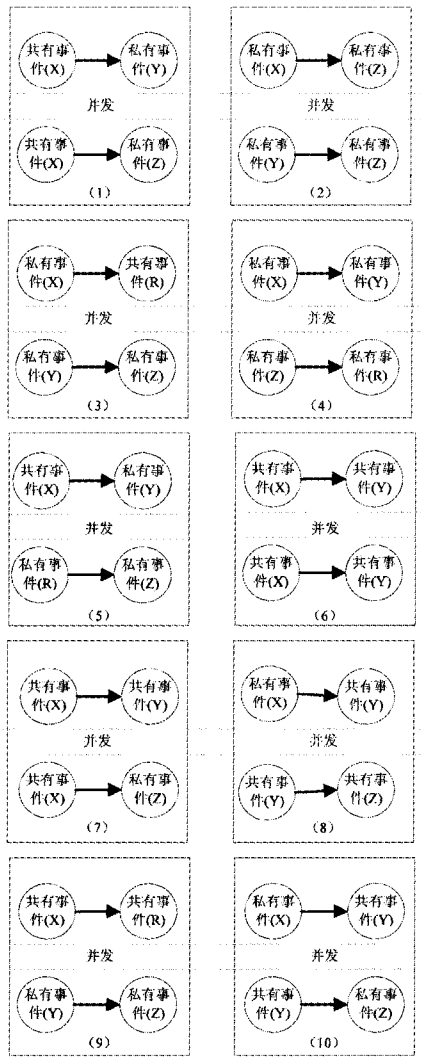

图 2 迹的并发的 10 种形式

5.3 性质验证

利用 ASP 进行性质验证的基本思想是:当要验证一个 CSP 是否满足性质 ψ 时,先构造 $\neg\psi$(即 ASP 程序 $\Pi_{ans}(\neg\psi)$);然后验证该 CSP 模型是否满足 $\neg\psi$。若没有回答集,则说明该模型满足性质 ψ;若有回答集,则说明该模型不满足性质 ψ,这些回答集即为不满足的反例[16]。

性质验证中的另一项重要工作是:将性质公式进行 ASP 表示。例如,对于性质 $p_1 \vee p_2 \vee p_3$,可以表示为:

$pl(p_1,q).\ pl(p_2,q).\ pl(p_3,q).\ r_1(X,Y,T)$:-$flow(X,Y,T),pl(X,T).\ r_2(X,Y,T)$:-$flow(X,Y,T),pl(X,T).$

$r_3(X,Y,T)$:-$flow(X,Y,T),pl(X,T).\ f_1$:-$r_1(X,Y,T).\ f_2$:-$r_3(X,Y,T).\ f_3$:-$r_4(X,Y,T).\ f$:-$f_1.\ f$:-$f_2.\ f$:-f_3.:-not f.

其中,谓词 $pl(X,Q)$表示原子命题 X 在进程 Q 中成立,$r_i(X,Y,T)$为其中的辅助谓词。

根据上述基本思想,当要验证一个性质时,首先对性质公式取反,并进行 ASP 描述;然后,把描述性质的 ASP 程序与构造模型的 ASP 程序进行组合,即可进行相应的验证。

6 实验

本文利用 ASP 技术实现了一个基于进程迹的 CSP 模型验证系统,它的输入为:模型的 CSP 描述和 LTL 公式;当模型不满足性质时,系统将给出反例。为了进一步说明该系统的验证效果,这里以哲学家就餐模型为例进行相关实验,并同文献[7]的方法进行比较。实验分别采用了 DLV、Smodels 和 Cmodels 3 种 ASP 求解器,主要从两方面进行比较:1. 待验证进程的描述能力;2. 验证结果的准确性。为简化起见,称文献[7]中的方法为 CS_ASP,本文的方法为 T_ASP。

为了比较系统的待验证进程的描述能力,分别在 $M1$、$M2$、$M3$、$M4$ 和 $M5$ 上对性质 $P1$ 进行验证。其中,$M1$=ph1 ‖ ph2,表示哲学家 ph1 和 ph2 进行并发,ph$i$ 采用式(1)形式进行描述;$M2$=ph1 ‖ ph2 ‖ ph3,表示哲学家 ph1、ph2 和 ph3 进行并发,ph$i$ 也采用式(1)形式进行描述;$M3$=ph1 ‖ ph2,ph$i$ 采用式(2)形式进行描述;$M4$=ph1 ‖ ph2,ph$i$ 采用式(3)形式进行描述;$M5$=ph1 ‖ ph2 ‖ ph3,ph$i$ 采用式(2)形式进行描述;待验证性质 $P1$= F1. up ∧ F2. up,即哲学家 ph1 和 ph2 都可以就餐。

表 1 给出了相应的实验结果(CS_ASP 的并发步骤限制在 25 步以内)。结果表明,T_ASP 调用 Cmodels 时验证效率最高,Smodels 次之,DLV 最低;CS_ASP 和 T_ASP 都可以完成对 $M1$ 的验证,在同种求解器下验证效率基本相当;CS_ASP 不能对 $M2$、$M3$、$M4$ 和 $M5$ 进行验证,这是由于 CS_ASP 中不具备分支进程的转化机制和并发的递归调用机制。CS_ASP 是在有穷步骤内对模型进行验证的,T_ASP 则不限定验证的步数。

表 1 CS_ASP 和 T_ASP 的验证能力的比较

验证方法		CSP模型	M1	M2	M3	M4	M5
CS_ASP	时间(s)	DLV	1.30	ND	ND	ND	ND
		Smodels	1.27	ND	ND	ND	ND
		Cmodels	1.30	ND	ND	ND	ND
	是否满足 P1		是	NV	NV	NV	NV
T_ASP	时间(s)	DLV	1.47	2.91	4.96	4.38	6.72
		Smodels	1.29	2.13	3.82	3.61	5.09
		Cmodels	1.09	1.96	3.76	3.07	4.27
	是否满足 P1		是	是	是	是	是

注:ND 表示相应的模型无法进行描述,且不能调用求解器进行计算;NV 表示不可验证。

为了比较这两种验证方法的效率和准确性,在 $M1$ 上对性质 $P1$、$P2$ 和 $P3$ 进行验证。其中,$P2$=G(1sit→F1up),即 ph1 必然可以吃到东西;$P3$=G(1. up→X2. up),即 ph1 起身后离开,接着 ph2 起身离开。

表 2　CS_ASP 和 T_ASP 验证准确性的比较

验证方法 / 性质	CS_ASP 验证结果	CS_ASP 验证时间(s) DLV	CS_ASP 验证时间(s) Smodels	CS_ASP 验证时间(s) Cmodels	CS_ASP 是否误判	T_ASP 验证结果	T_ASP 验证时间(s) DLV	T_ASP 验证时间(s) Smodels	T_ASP 验证时间(s) Cmodels	T_ASP 是否误判
P1	满足 P1	1.30	1.27	1.30	否	满足 P1	1.47	1.29	1.09	否
P2	不满足 P2(并发步数小于 12)	1.44	1.31	1.21	是	满足 P2	1.61	1.20	1.11	否
P3	不满足 P3	1.87	1.66	1.62	否	不满足 P3(提供反例集 CE)	1.71	1.68	1.56	否

由表 2 的实验结果可见，CS_ASP 和 T_ASP 在同种求解器下的验证效率相当；且调用 Cmodels 和 Smodels 的验证效率高于 DLV；此外，通过 CS_ASP 得到验证结果不满足 $P2$，T_ASP 输出满足 $P2$，这是由于 CS_ASP 的并发步骤限定较少时产生了误判；当验证 $P3$ 时，两种方法的验证结果都是不满足 $P3$，T_ASP 输出反例集 CE。

7　相关工作

当前，CSP 可以在公理化语义和操作语义下完成性质的验证。程序正确性证明方法是公理语义下进行性质验证的一种方法；它通过一组逻辑规则和断言来证明 CSP 的正确性；这组逻辑规则即为 CSP 的公理化语义。程序正确性证明方法主要偏向于公理语义的可靠性和完备性的证明，且自动化程度不高[2,18]。FDR 是 CSP 模型检测的主流工具，它是在 CSP 的操作语义下将 CSP 转化为相应的迁移系统，并通过迁移系统完成 CSP 的验证[3,4,6]。FDR 中性质采用迹进行规范，这不利于活性的描述；同时 CSP 到迁移系统的转化比较复杂。本文提出了一种基于进程迹的 CSP 验证框架，它在迹语义下计算出 CSP 的迹模型，并通过迹模型完成性质的验证。与 FDR 不同的是，该框架中性质采用 LTL 进行规范，它的通用性更广，描述能力更强。

Hoare 首次提出了 CSP 的迹模型的概念，并说明了迹模型的主要用途：1. 可以为 CSP 语言提供清晰的、一致的定义，进而提高 CSP 语言数学理论基础；2. 可以通过迹模型来证明程序的正确性[9]。通过文献[10]可知，在一定条件下，CSP 在迹语义下的迹模型与在操作语义下转化成的迁移系统是等价的。文献[20]提出了 CSP 的有界迹语义；在一定条件下，基于有界迹语义得到的有界迹模型与操作语义下转化生成的迁移系统的有界模型是等价的。此外，文献[21,22]指出，在特定条件下，基于状态的模型检测与基于迹的模型检测是等价的。以上的研究成果为基于进程迹的 CSP 验证提供了理论基础。

与本文工作类似，文献[7]提出了一种基于 ASP 的 CSP 验证方法；该方法是通过并发状态来对 CSP 进行验证，它的不足之处是理论支撑薄弱。另外，Y. Howard 和 S. Gruner 提出了一种面向迹的验证方法；该方法首先从待验证的程序或系统中提取相应的迹，然后将迹模型转化为 SPIN 工具的标准输入形式进行验证；在此验证体系统中，性质采用迹进行描述[23]。与之不同的是，本文的方法可以直接在迹模型上进行验证，不需要模型间的转化；性质采用 LTL 进行规范。

本文提出了一种基于进程迹的 CSP 模型验证方法，并利用 ASP 技术实现了一个 CSP 验证系统。与文献[7]不同的是，该系统进程的描述可采用一般选择形式和非确定选择形式，并且系统是通过可满足判定进行性质验证的，当性质不满足时，系统还可提供反例。本文在构建基于 ASP 的性质验证方法时，借鉴了文献[16]的基本思想。但与之不同的是，本文构造的方法可以对无穷模型进行验证，而文献[16]中只是完成对有界模型的验证。

结束语　本文设计了一种基于进程迹的 CSP 模型验证框架，并用 ASP 实现了此验证方法。该方法具有如下特点：1. 待验证的 CSP 可采用前缀、递归、一般选择、非确定选择进程进行描述，性质采用 LTL 进行规范；2. 性质的验证采用可满足判定方法，便于实现自动化；3. 当不满足性质时，系统将给出反例；4. 采用 ASP 方法，有利于系统进行扩展和修改。基于进程迹的方法为 CSP 的验证提供了一条很好的途径。但是进程的迹模型通常比较庞大，这也为并发和验证带来了瓶颈。此外，本文是在有穷迹语义下进行模型的转化，所以 CSP 的描述方式受到一定限制。因此，下一阶段的主要工作将集中在以下两方面：1. 状态爆炸问题的解决；2. 进一步扩展有穷迹语义，并在此语义下完成 CSP 到迹模型的自动转化。

参 考 文 献

[1] Hoare C A R. Communicating Sequential Processes [OL]. http://www.usingcsp.com/cspbooks, 2012

[2] Art K R. Ten years of Hoare's logic, A survey—Part I[J]. ACM TOPLAS, 1981, 3(4): 431-483

[3] Roscoe A W, Gardiner P H B, Goldsmith M H, et al. Hierarchical compression for model-checking CSP or how to check10 dining philosophers for deadlock[C]// Proceedings of the 1st TACAS, BRICS Notes Series NS-95-2. Department of Computer Science, University of Aarhus, 1995

[4] Roscoe A W. Model Checking CSP[M]// A classical mind: essays in honour of C. A. R. Hoare. Prentice Hall, 1994

[5] Armstrpng P, Lowe G, Ouaknine J, et al. Model checking timed CSP[C]// To appear in proceedings of HOWARD, Easychair, 2012

[6] Armstrpng P, Goldsmith M H, Lowe G, et al. Recent developments in FDR2[C]// Proceedings of CAV. 2012: 699-704

[7] 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

[8] Baier C, Katoen J-P. Principles of model checking[M]. Cambridge, Massachusetts: The MIT Press, 2007

[9] Hoare C A R. A model for communicating sequential processes [M]// McKeag, MacNaughten, eds. On the construction of programs. Cambridge University Press, 1980

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

[11] Lifschitz V. What is answer set programming? [C]// Proceedings of the AAAI Conference on Artificial Intelligence. 2008: 1594-1597

(下转第 221 页)

cisco:Morgan Kaufmann Publishers Inc. ,1996:544-555
[6] Hayes B. Cloud computing [J]. Communications of the ACM, 2008,51(7):9-11
[7] Armbrust M,Stoica I,Zaharia M,et al. A view of cloud computing [J]. Communications of the ACM,2010,53(4):50-58
[8] Mohammed J Z,Ho Ching-tien,Rakesh A. Parallel classification for data mining on shared-memory multiprocessors [C]// ICDE: IEEE International Conference on Data Engineering. Sydney: IEEE Computer Society,1999:198-205
[9] Mohammed J Z. Parallel and distributed association mining: a survey [J]. IEEE Concurrency,1999,7(4):14-25
[10] Ruo-ming J,Ge Yang,Gagan A. Shared memory parallelizetion of data mining algorithms: techniques, programming interface, and performance[J]. Transactions on Knowledge and Data Engineering,2005,17(1):71-89
[11] Peter S P. Parallel Programming with MPI [M]. San Francisco: Morgan Kaufmann Publishers Inc. ,1997
[12] Gropp W,Lusk E,Skjellum A. Using MPI portable parallel programming with the message-passing interface [M]. Cambridge: MIT Press,1999
[13] Nuno A,João G,Fernando S. Exploiting Parallelism in Decision Tree Induction[C]// ECML/PKDD Workshop on Parallel and Distributed computing for Machine Learning. 2003:13-22
[14] Wu Rong,Liang Shuai,Liao Hua-ming. Cyclic workflow execution mechanism on top of MapReduce framework[C]// Seventh International Conference on Semantics, Knowledge and Grids. Washington,DC: IEEE Computer Society,2011:28-35
[15] Jeffrey D,Sanjay G. MapReduce: Simplified Data Processing on Large Clusters [J]. Communications of the ACM,2008,51(1): 107-113
[16] Sanjay G,Howard G,Leung S T. The Google file system [C]// the Nineteenth ACM Symposium on Operating Systems Principles(SOSP' 03). Bolton Landing,NY,USA,2003,37(5):29-43
[17] Kang U,Tsourakakis C,Appel A P,et al. HADI: fast diameter estimation and mining in massive graphs with Hadoop[R]. Pittsburgh:Carnegie Mellon University,2008
[18] Lin J, Dyer C. Data-intensive text processing with MapReduce [M]. Morgan and Claypool Publishers,2010
[19] Ene A,Im S,Moseley B. Fast clustering using MapReduce[C]// Proceeding of the 17th ACM SIGKDD Inernational Conference on Knowledge Discovery and Data Ming. New York: ACM, 2011:681-689
[20] 向小军,高阳,商琳,等. 基于 Hadoop 平台的海量文本分类的并行化[J]. 计算机科学,2011,38(10):184-188
[21] 赵卫中,马慧芳,傅燕翔,等. 基于云计算平台 Hadoop 的并行 k-means 聚类算法设计研究[J]. 计算机科学,2011,38(10):166-168
[22] Biswanath P,Joshua S H,Sugato B,et al. PLANET: Massi-vely parallel learning of tree ensembles with MapReduce[C]// 35th International Conference on Very Large Data Bases (VLDB-2009). 2009:1426-1437
[23] Lu Qiu,Cheng Xiao-hui. The Research of Decision Tree Mining Based on Hadoop[C]// 9th International Conference on Fuzzy Systems and Knowledge Discovery(FSKD 2012). 2012:798-801
[24] Apache. Hadoop[EB/OL]. http://hadoop. apache. org/,2012-10-20
[25] White T. Hadoop:权威指南 [M]. 曾大聃,周傲英,译. 北京:清华大学出版社,2010
[26] Mohammed J Z. Parallel and Distributed Data Mining: An Introduction[C]// Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems. London,UK: SIGKDD,2000:1-23
[27] Ruoming J, Gagan A. Communication and memory efficient parallel decision tree construction[C]// the third SIAM International Conference on Data Mining. San Fra ncisco: Society for Industrial & Applied,2003:119-129

(上接第 186 页)

[12] Niemelä I,Simons P,Syrjänen T. Smodels: a system for answer set programming [C] // Proceedings of the 8th International Workshop on Non-Monotonic Reasoning. Brenkenridge, Colorado,USA,2000
[13] Lin Fang-zhen,Zhao Yu-ting. ASSAT: computing answer sets of a logic program by SAT slover[J]. Artificial Intelligence,2004, 157(1/2):115-137
[14] Lifschitz V, Turner H. Splitting a logic program [C]// Proceedings of the Eleventh International Conference on Logic Programming. 2008:23-37
[15] Baral C. Knowledge Representation, Reasoning, and Declarative Problem Solving [M]. Cambridge Press,2003
[16] Heljanko K, Niemelä I. Bounded LTL model checking with stable models [J]. Theory and Practice of Logic Programming, 2003,4(3):519-550
[17] Leone N,Pfeifer G,Faber W,et al. The DLV system for knowledge representation and reasoning [J]. ACM Transactions on Conputational Logic,2006,3(7):499-562
[18] 陆汝钤. 计算机语言的形式语义[M]. 北京:科学出版社,1992
[19] Brookes S D,Hoare C A R,Roscoe A W. A theory of communicating sequential processes [J]. Journal of the ACM, 1984, 31 (3):560-599
[20] Palikareva H, Ouaknine J, Roscoe A W. SAT-solving in CSP trace refinement [J]. Science of Computer Programming. Special issue on Automated Verification of Critical Systems, 2012, 77 (10/11):1178-1197
[21] Ranzato F. On the completeness of model checking[C]// Proc. ESOP'01,LCNS 2028. Springer,1999:137-154
[22] Giacobazzi R, Ranzato F. States vs. traces in model checking [C]// Proc. 9th International Static Analysis Symposium (SAS' 02). LNCS 2477,2002:461-476
[23] Howard Y,Gruner S,Gravell A,et al. Model-based trace checking [C]// SoftTest: UK Software Testing Research Workshop II. 2003