文章编号:1000-2367(2016)05-0140-06 DOI:10.16366/j.cnki.1000-2367.2016.05.024

基于 CTL 的并发系统 CSP 模型验证

王亚丽[1a,b],杨育捷[1a,b],赵岭忠[2],翟仲毅[2]

(1.河南师范大学 a.计算机与信息工程学院;b."智慧商务与物联网技术"河南省工程实验室,河南 新乡 453007;2.桂林电子科技大学 广西可信软件重点实验室,广西 桂林 541004)

摘 要:主要通过指称语义和回答集程序(Answer Set Programming,简称 ASP)完成迹模型的生成,并构建了一套基于计算树逻辑(computing tree logic,简称 CTL)的 CSP 模型验证方法.实验表明,该方法对于分支类型的性质具有较好的描述能力,且保证了验证的正确性.

关键词:模型检测;迹模型;计算树逻辑;回答集程序设计

中图分类号:TP311 **文献标志码**:A

模型检测作为保证软件可靠性的主要方法,已被应用到了许多软件系统,如 Web 服务组合[1]、数据库[2]和协议验证[3].CSP 是最为流行的模型检测语言之一,其验证工具主要通过操作语义和指称语义两种途径[4-5].其中,FDR[6],ARC[7]和 PAT[8]基于操作语义计算出 CSP 模型对应的迁移系统,进而对迁移系统进行性质验证.操作语义是一种底层的模型语义,处理能力强大,可解释模型变化的细节,但其理论复杂,不易扩展[4].为了更好地把 CSP 模型检测器应用到领域软件中,我们前期构建了一种扩展性较强的模型检测工具——T_ASP[9].T_ASP 通过指称语义计算出 CSP 的迹模型,然后基于迹模型的线性时态逻辑(Linear Temporal Logic,简称 LTL)的可满足语义对性质进行验证.指称语义是一种较高层的模型解释方式,可直观地显示模型与语义的映射关系,方便了用户的扩展和理解.T_ASP 中,语义的计算细节都交给回答集程序求解器完成,扩展者只需规范 CSP 模型的指称语义.性质规范方面,T_ASP 采用时态逻辑进行刻画,其优点在于可以简洁、直观地表示性质,且描述能力强.目前,T_ASP 只能对 LTL 公式进行验证.为了加强 T_ASP 的验证能力,本文构建了基于 CTL 的 CSP 模型验证框架.该框架提出了迹模型下 CTL 的可满足语义,并将 CTL 可满足判定转化为回答集求解.基于此框架,分别在 DLV[10],Smodels[11],Cmodels[12]3 种 ASP 求解器上进行了相关实验,结果显示了该框架的正确性和高效性.

1 CSP 迹模型的生成方法

1.1 CSP 的迹语义

CSP 理论中,进程是描述客体的基本单位,其语法为:

STOP | $x \to P_1 | .\mu P_2.\ F(P_2)$ | $P_3 \square P_4$ | $P_3 \prod P_4$ | $P_3 \| P_4$,其中 P_1 是一个可终止的进程.

CSP 提供了一系列指称语义模型,如 trace 模型,failures/divergences 模型.CSP 的 trace 模型包含一组迹,其一条迹是进程的一个可能的行为,每条迹由进程的离散事件形成,并以事件发生的次序排列.trace 模型显式地提供了进程的所可能发生的一切行为序列.

定义 1[4] (CSP 迹语义)给定一个 P 进程,其迹语义是从抽象语言模型 $CSP_M(P)$ 到 trace 模型的一个函数.

收稿日期:2016-03-03;**修回日期**:2016-06-25.

基金项目:国家自然科学基金(61262008);广西可信软件重点实验室开放课题(kx201415);河南省科技厅基础与前沿技术研究项目(132300410430).

第 1 作者简介(通信作者):王亚丽(1979-),女,河南三门峡人,河南师范大学讲师,北京邮电大学博士研究生,研究方向为智慧商务中间件技术、算法设计与分析,E-mail:yaliwan_g@163.com.

定义 2[4]　(CSP 的 trace 模型)给定一个 P 进程，其 trace 模型可写为 traces(P)，形式地表示 P 的所有可能的行为.

通过以下若干种基本进程的 trace 模型，复合进程的 trace 模型可以通过调用基本进程的 trace 模型进行求解.

若 P 进程为终止(STOP)进程，则其 trace 模型可表示为：traces(P)=$\{p|p=<>\}$.

若 P 进程为：$P=a\rightarrow$STOP；则其 trace 模型可表示为：traces(P)=$\{<>\}\cup\{<a>\}$.

若 P 进程为：$P=a\rightarrow Q$，且 $Q=x\rightarrow\cdots\rightarrow$STOP 的形式)；则其 trace 模型可表示为：

traces(P)=$\{<>\}\cup\{<a>\frown p|p\in$ traces(Q)$\}$，“$\frown$”为迹的连接算子.

若 P 进程为：$P=a\rightarrow\cdots\rightarrow b\rightarrow P$，设 Q 进程为 P 的前缀，即 $Q=a\rightarrow\cdots\rightarrow b\rightarrow$STOP，设 Q 的迹模型表示为 T^Q，T 的所有子迹为 sub_T^T；则其 trace 模型可表示为：

$$\text{traces(P)}=\{<>\}\cup\{T^Q\}\cup\{\text{sub\_}T^T\}.$$

若 P 进程为：$Q\square T$(Q 和 T 是前缀进程或递归进程)，则其 trace 模型可表示为：

$$\text{traces}(P)=\{q|q\in\text{traces}(Q)\}\cup\{t|t\in\text{traces}(Q)\}.$$

若 P 进程为：$Q\amalg T$(Q 和 T 是前缀进程或递归进程)；则其 trace 模型可表示为：

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

1.2　迹模型的生成

本文采用文献[9]的计算方式，其基本思想是：将迹模型的计算过程归结为回答集求解过程. 该计算机制包括进程的知识表示，迹模型的生成，并发操作下迹的计算方法 3 部分，如图 1 所示.

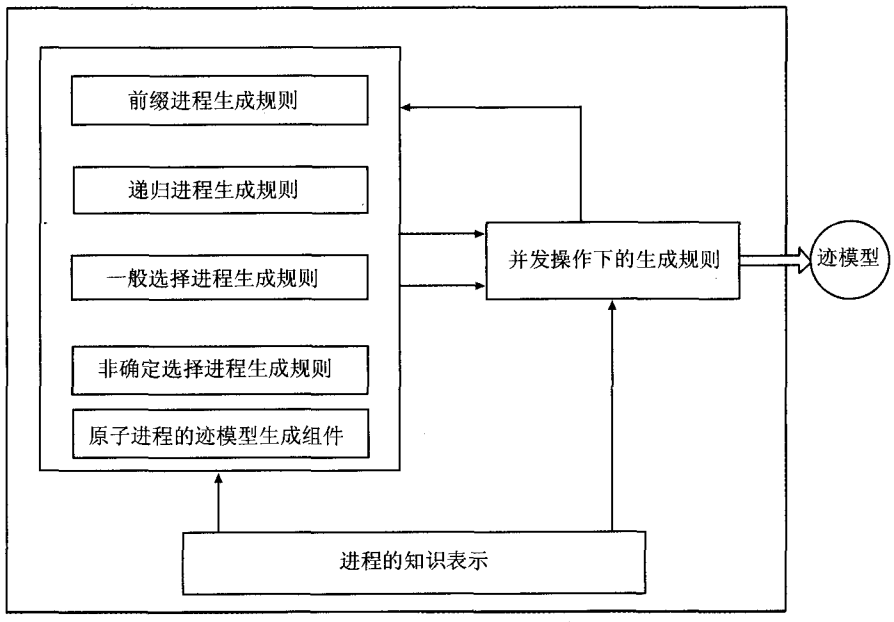

图1 CSP迹模型的生成机制

进程的知识表示：为迹模型的归约计算提供基础支持. 其主要内容是：1)根据 ASP 的基础知识，提供进程描述的基础谓词；2)利用基础谓词和逻辑规律，刻画基础规则.

原子进程到迹模型生成组件：提供原子进程迹模型的计算规则. 其主要内容包括：1)分析原子进程的基本结构，制定谓词和约束规则；2)刻画约束规则.

并发操作下迹模型的计算规则：生成并发操作下 CSP 模型所有可能的迹. 其主要内容包括：1)给定迹并发的基本结构，并分析约束规则；2)刻画约束规则.

为更好地说明上述 3 部分的具体工作，表 1 给出了每部分内容的实例说明，详细的实现过程见文献[9].

表 1 迹模型生成机制知识表示示例

组件	任务类别		描述形式
基础谓词与规则集合	基础谓词		flow(X,Y,P)表示进程 P 的 X 事件比 Y 先发生;event(X,P)表示事件 X 属于 P 进程;first(X,P)表示事件 X 是 P 前缀式的首事件;not_first(X,P)表示 X 不是进程 P 前缀式的首事件.
	基础规则的描述		event(X,P):-flow(X,Y,P). event(Y,P):-flow(X,Y,P). not_first(X,P):-flow(Y,X,P). first(X,P):-not_first(X,P), event(X,P).
原子进程的生成规则	递归进程	谓词解释	pre(P,Q)表示 P 进程是 Q 进程的前缀式形成的进程;invoke (P,Q)表示 P 进程可以调用 Q,完成后续任务;repeat(P,Q)表示 Q 由 P 自身调用形成.
		约束规则	flow(X,Y,Q):-flow(X,Y,P),pre(P,Q). flow(X,Y,Q):-pre_last(X,P),first(Y,P),notflow(X,Y,P),repeat(P,Q). flow(X,Y,Q):-flow(X,Y,P),invoke(Q,P). flow(X,Y,Q):-invoke(Q,P),pre(R,P),preset(T,Q), pre_last(X,T), first(Y,R), not flow(X,Y,P).
并发操作的生成规则	谓词解释		init_flow (X,Y,P)表示当前参与并发的事件流为 flow(X,Y,P); concur(P,Q,T)表示进程 P 和 Q 并发生成进程 T;f_flow(X, Y,P)表示事件流 flow (X,Y,P)已经失效
	约束规则		flow(X,Y,T):-init_flow(X, Y,P), init_flow(X,Y,Q), not f_flow(X,Y,P), not f_flow(X,Y,Q), event(X,P),event(Y,P), event(X,Q), event(Y,Q), concur(P,Q,T). :-init_flow(X,Y,P), init_flow(X, Z,Q), not f_flow(X, Y,P), not f_flow(X,Z,Q), event(X,P), event(Y,P),event(X,Q), event(Y,Q),event(Z,P), concur(P,Q,T). :-init_flow(X,_,P), init_flow(Y, _,Q), not f_flow(X,_,P), not f_flow(Y,_,Q), event(X,P), event(Y,P),event(X,Q),event(Y,Q), concur (P,Q,T).

2 基于 CTL 的 CSP 模型验证框架

2.1 CSP 模型验证框架

面向 CTL 的通信顺序进程(CSP)模型验证框架如图 2 所示,包括 3 部分:1)基本进程的迹模型的计算方法;2)并发操作下迹模型的计算方法;3)基于迹的 CTL 性质的可满足语义,并构建验证程序. $\Pi(\psi)$.

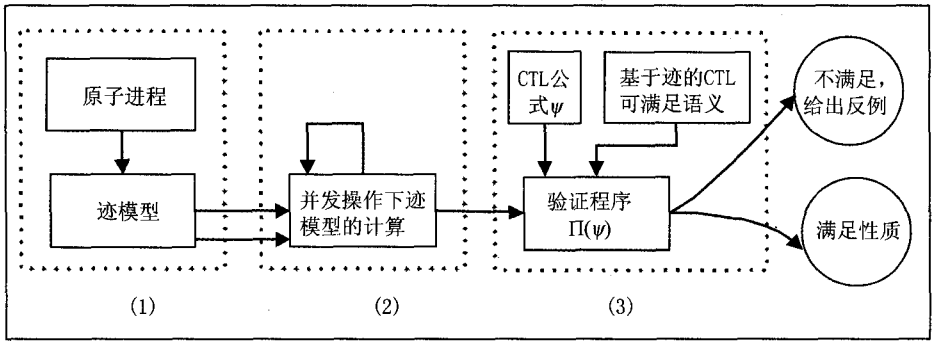

图2 基于CTL的CSP模型验证框架

2.2 迹模型下 CTL 的可满足语义

前期的研究中,提出了 LTL 公式的验证方法. LTL 对线性性质有较强的表示能力,但对于非线性结构则不能很好地表示,如 LTL 无法表示"某些路径上满足公式. φ". 为此,本文把 CTL 引入 CSP 模型验证框架,从而进一步加强该框架的性质刻画能力. CTL 公式从语法结构上可分为状态公式和路径公式,其状态公式如下:

$$\psi ::= p \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid E\varphi \mid A\varphi, \tag{1}$$

这里 φ 表示一个路径公式.

CTL 路径公式为:

$$\varphi ::= G\psi \mid F\psi \mid X\psi, \tag{2}$$

这里 ψ 表示状态公式.

为了方便说明，文中采用了 G，F 和 X3 个时态算子，以及 A 和 E 两个路径算子；其中，G 算子可用于刻画行为序列中的“未来的所有状态”，F 算子用于刻画行为序列中的“未来的某一状态”，X 算子则用于刻画行为须列中的“下一个状态”，A 算子表示“所有路径”，E 算子表示“存在某路径”. 用 traces(P)表示进程 P 的迹模型，π 来表示 traces(P)中的一条迹，则迹模型下 CTL 公式的可满足语义(traces(P)$\models S_{CTL}$)如下.

给定进程 P，ψ_1 和 ψ_2 是状态公式，φ 属于路径公式，$\pi_t \in$ trace(P)，traces(P)$\models S_{CTL}$ 可归结为：状态公式可满足关系和路径公式的可满足关系. 状态公式的可满足关系可定义如下：

traces(P)$\models a$　　iff　$\exists i.\ a=\pi_t(i)$　　traces(P)$\models \neg a$　　iff　$\pi_t \not\models a$

traces(P)$\models \psi_1 \wedge \psi_2$　iff　$\pi_t \models \psi_1$ 且 $\pi_t \models \psi_2$　　traces(P)$\models \psi_1 \vee \psi_2$ iff　$\pi_t \models \psi_1$ 或 $\pi_t \models \psi_2$

traces(P)$\models E\varphi$　　iff　$\exists \pi_t \in$ trace(P)且 $\pi_t \models \varphi$ traces(P)$\models A\varphi$　　iff　$\forall \pi_t \in$ trace(P)且 $\pi_t \models \varphi$

路径公式的可满足关系可定义为：

$\pi_t \models X\psi$　　iff　$\pi_t(2) \models \psi$　　$\pi_t \models F\psi$　　iff　$\exists i.\ \pi_t(i) \models \psi$

$\pi_t \models G\psi$　　iff　$\pi_t \models \neg F \neg \psi$

其中，$\pi_t \in$ trace(P)，$\pi_t(i)$ 刻画 π_t 的第 i 个事件.

在上述可满足语义的基础上，针对路径算子起始的 CTL 公式初步设计了验证方法，其具体过程是：首先，判定 CTL 公式的路径算子类型，“A”形式调用 ASP 谨慎推理模式，“E”形式调用 ASP 果敢推理模式；然后，由内向外分析 CTL 公式 ψ，并构建验证程序 $\Pi(\psi)$；最后，结合验证程序 $\Pi(\psi)$和迹模型进行性质验证.

为了实现验证的自动化，还需完成两项工作：1)利用 ASP 表示可满足语义对应的判定规则，如表 2 所示；2)利用 ASP 和判定规则，构建面向 CTL 公式的验证程序.

表 2　CTL 可满足语义的 ASP 规则库

公式 f	对应的 ASP 形式	辅助规则
p(p 为原子命题)	p_e : -$f(A,B,T)$. $s(A,B,T)$: -flow(A,B,T), $lp(A,T)$.	
$\neg p$	p_e : -$f(A,B,T)$. $s_1(A,B,T)$: -flow(A,B,T), $lp(A,T)$. $s(A,B,T)$: -not $s_1(A,B,T)$.	
$f_1 \wedge f_2$	f : -f_1, f_2.	e : -el(A,T).
$f_1 \vee f_2$	f : -f_1. f : -f_2.	elo(A,T) : -flow(B,A,T), flow(C,A,T).
$G f_1$	f : -e, not q. q : -$s_2(A,B,T)$. $s_2(A,B,T)$: -not $s_1(A,B,T)$, flow(A,B,T).	nl(A,B,T) : -flow(A,B,T), elo(B,T).
$F f1$	f : -e, q . q : -$f_1(A,B,T)$, be(A,B,T).	be(A,B,T) : -elo(A,T), flow(A,B,T).
$X f1$	f : -$f(C,A,T)$. $f(C,A,T)$: -$f1(A,B,T)$, flow(C,A,T), first(C,T).	be(B,C,T) : -flow(B,C,T), be(A,BT).
Ef	Brave reasoning: f is Bravely true, 即 f 在所有的回答集中都成立	
Af	Cautious reasoning: f is cautiously true, 即 f 在某些回答集上成立	

2.3　验证程序 $\Pi(\psi)$

通过表 2，可构建性质的验证程序 $\Pi_{ans}(\psi)$，进而把性质验证归结为回答集求解，具体求解方式与文文献[14]类似. $\Pi_{ans}(\psi)$的构建方法如下：首先对待验证的 CTL 公式 ψ 进行路径算子判定，若是 E 算子，则选择果敢推理模式，若为 A 算子选择谨慎推理模式，则选择对 ψ 取反为$\neg\psi$；继而由外向内解析 CTL 公式的基本结构，最后递归式地调用表 2 中相应规则，这样即可得到目标程序 $\Pi_{ans}(\neg\psi)$. 下面举例说明：

给定待验证进程 p，以及 CTL 公式 $F_1 = E\,F$ 1. up 和公式 $F_2 = A\,F$ (1. up$\wedge$2. up)，其中 1. up，2. up 是进程 p 的原子事件.

F_1 的验证程序可表示为：

f : -f_1. f_1 : -e, not q. q : -w_2(A,B,T). $w_2(A,B,T)$: -not $w_1(A,B,T)$, flow(A,B,T).

$w_1(A,B,T)$: -not $w_3(A,B,T)$. $w_3(A,B,T)$: -flow(A,B,T). lp (A,T). lp (1. up,p).

其中，F_1 的验证采用果敢推理方式.

F_2 的验证程序可表示为：

f : -f_1.　　f : -f_2.

f_1 : -e, not q.　　q : -$y_2(A,B,T)$. $y_2(A,B,T)$: -not $y_1(A,B,T)$, flow(A,B,T).

$y_1(A,B,T)$: -not $t_3(A,B,T)$. $y_3(A,B,T)$: -flow(A,B,T),lp (A,T). lp (1. up,p).

f_2 : -le,not p. p : -$z_2(A,B,T)$. $z_2(A,B,T)$: -not $z_1(A,B,T)$,flow(A,B,T).

$z_1(A,B,T)$: -not $s_3(A,B,T)$. $z_3(A,B,T)$: -flow(A,B,T),lp (A,T). lp (2. up,p).

其中,$F2$ 采用谨慎推理的方式.

3 实 验

本部分采用哲学家就餐模型来对 CTL 性质进行验证. 其中,哲学家就餐模型的 CSP 描述采用文献[13]的描述形式.

3.1 哲学家就餐模型

哲学家进程的 CSP 描述分为 3 种形式,分别为:递归、一般选择及其嵌套和非确定选择. 其中,每种形式的具体描述方式见表 3.

表 3 哲学家进餐问题的 CSP 模型

类型	具体描述形式
递归	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 注释:定义哲学家 Phi 满足以下行为标准:每个人初始处于坐下状态,然后拿左边叉子,继而拿右边叉子,然后进餐,接着放回左叉子,再放右叉子,最后离开座位.
一般选择及其嵌套	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$_c$)) 注释:哲学家进程初始时,对于叉子不做限制,当选定具体方式后,哲学家按照已选择方式进行. 其中,Phi$_c$ 表示从首事件沿所在路径循环进行的一进程.

3.2 性质验证

为了更好地阐明系统的验证效果,文中以哲学家就餐模型进行了相关的性质验证. 表 4 给出了模型 $M=ph1||ph2$ 对于上述性质 F_1 和 F_2 的验证结果. 为了比较不同类型性质(CTL 和 LTL)的验证效率,表 4 还给出了性质 $F_3=F_1$. up $\wedge$ F_2. up 的验证结果文中对不同 ASP 回答集求解器的验证效率进行了比较.

表 4 哲学家就餐模型 M 对 F_1 和 F_2 的验证结果

时态逻辑		DLV	SMODLES	CMODELS	验证结果
CTL	F_1	0.61	0.43	0.37	满足
	F_2	1.75	1.49	1.31	满足
LTL	F_3	1.47	1.29	1.09	满足

通过实验结果可知,3 种求解器都可对性质进行有效的性质验证. 其中,对于等效的性质($F_2\approx F_3$),系统对 LTL 的验证效率略高,这是由于 CTL 的验证转化较为复杂,故验证效率略低. 此外,该系统分别采用了 3 种求解器对性质进行了验证,结果显示 Cmodels 验证效率最高,Smodels 次之,DLV 最低.

4 总 结

本文提出了一种基于 CTL 的 CSP 模型检测方法,该方法对于分支时序性质有良好的表示能力,是对文献[9]中成果的有效补充. 此外,该验证体系与文献[9]具有完全的兼容性. 其不足之处是:进程的迹模型通常规模十分庞大. 这使得迹模型生成以及验证复杂度较高. 鉴于此,下一阶段的任务将集中在:1)进一步优化迹模型的生成机制,从而降低计算复杂度;2)对待验证性质进行预处理,缓解状态爆炸问题.

参 考 文 献

[1] Armando A, Arsac W, Avanesov T, et al. The AVANTSSAR platform for the automated validation of trust and security of service-ori-

ented architectures[J]. Lecture Notes in Computer Science,2012,7214(1):267-282

[2] Gligoric M, Majumdar R. Model checking database applications[C]//In Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-Verlag,2013:549-564.

[3] Wu Xi, Zhu Huibiao, Zhao Yongxin,et al. Modeling and verifying the Ariadne protocol using process algebra[J]. Comput Sci Inf Syst 2013,10(1): 393-421.

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

[5] Roscoe A W. Understanding Concurrent System[M]. Heidelberg: Springer-Verlag,2011.

[6] Armstrpng P, Goldsmith M H, Lowe G, et al. Recent developments in FDR[C]//International Conference on Computer Aided Verification. Berlin:Springer,2012:699-704.

[7] Parashkevov A N, Yantchev J. ARC-a tool for efficient refinement and equivalence checking for CSP[C]//IEEE Second International Conference on Algorithms & Architectures for Parallel Processing. Singapore:IEEE,1996:68-75.

[8] Jun Sun, Yang Liu, Jin Songdong,et al. Bounded model checking of compositional processes[C]. 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering, Los Angeles,2008.

[9] 赵岭忠，翟仲毅，钱俊彦. 基于进程迹的 CSP 模型验证框架[J]. 计算机科学，2013,40(11):181-186.

[10] Francesco R. Nicola LeoneDLV[EB/OL]. [2016-03-25]. http://www.dbai.tuwien.ac.at/proj/dlv.

[11] Patrik S. Smodels[EB/OL]. [2016-03-25]. http://www.tcs.hut.fi/Software/smodels/.

[12] Cryptonym. CModels[EB/OL]. [2016-03-25]. http://www.cs.utexas.edu/users/tag/cmodels.html.

[13] Hoare C A R. Communicating Sequential Processes [EB/OL]. [2016-03-25]. http://www.usingcsp.com/cspbooks.

[14] 赵岭忠，翟仲毅，钱俊彦，等. 基于关键迹和 ASP 的 CSP 模型检测[J]. 软件学报，2015,26(10):2521-2544.

Model Checking CSP Concurrent Systems Based on CTL

WANG Yali[1a,b], YANG Yujie[1a,b], ZHAO Lingzhong[2], ZHAIi Zhongyi[2]

(1. a. College of Computer and Information Engineering; b. Engineering Lab of Intelligence Business & Internet of Things, Henan Normal University, Xinxiang 453007,China;2. School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: CSP is one of the main methods of model checking. Denotational semantics and operational semantics are the two approaches adopted for model validation. Compared with operational semantics, denotational semantics computation model is intuitive, and easy to extend. This paper mainly completed the trace model's generation by denotational semantics and ASP mechanism, and developed a set of CSP model validation method based on CTL. Results showed that, this method has a good description ability for the properties of the branch type, and can ensure the accuracy of verification.

Keywords: model checking; trace model; computing tree logic; answer set programming