时序 Agent 程序设计语言 TAOPLID 的离线解释器*)

李 斌[1,2] 唐小燕[1] 吴梅丽[1] 张东摩[2] 吕 建[2]
(扬州大学计算机科学与工程系 扬州225009)[1]
(南京大学计算机软件新技术国家重点实验室 南京210093)[2]

摘 要 AOPLID是一种面向agent程序设计语言。本文旨在对AOPLID语言进行时序扩充,使之能表达并处理带时间参数的并发行动,基于离线方式下AOPLID程序的语义,用Prolog语言实现时序AOPLID语言(TAOPLID)的离线解释器。首先,我们对经典情境演算进行适当改造,使之能描述含时间变元的行动,因为持续行动一般可认为是具有瞬时开始行动和瞬时终止行动的过程,所以可以将一个持续动作分解为两个时间上互不相交的瞬时动作,再引入一个新的关系流 刻画这两个瞬时动作的执行情况,从而可在扩充后的情境演算中表达带时间参数的并发行动。其次,为使TAOPLID离线解释器方便处理以集合方式表示的TAOPLID程序,设计并实现了TAOPLID预处理器,它将TAOPLID程序的集合形式转换成Prolog子句形式,然后通过TAPOLID离线解释器对其解释生成一可执行的原子行动序列。

关键词 面向agent程序设计语言,心智状态,情境演算,解释器,并发行动

Off-line Interpreter of Temporal Agent Programming Language TAOPLID

LI Bin[1,2] TANG Xiao-Yan[1] WU Mei-Li[1] ZHANG Dong-Mo[2] LV Jian[2]
(Department of Computer Science and Engineering, Yangzhou University, Yangzhou225009)[1]
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing210093)[2]

Abstract AOPLID is a novel agent-oriented programming language. In this paper, TAOPLID, a temporal extension of AOPLID, is given so as to represent and process concurrent actions with the time arguments. Based on off-line AOPLID program semantics, off-line TAOPLID interpreter is implemented in Prolog. First, the classical situation calculus is extended to the temporal situation calculus so as to enable a treatment of actions with explicit time, therefore, concurrent processes with explicit time can be represented in the extended situation calculus. Next, off-line interpreter of TAOPLID is implemented in order to execute an TAOPLID program with the time arguments. Because TAOPLID program is represented with set manner, it is difficult to be interpreted by TAOPLID interpreter, so TAOPLID pre-processing program is designed and implemented. Pre-processing program can translate TAOPLID program into Prolog sub-clause which can be interpreted and a executable primitive action sequence can be achieved.

Keywords Agent-oriented programming, Mental states, Situation calculus, Interpreter, Concurrent action

1 引言

AOPLID[1,2]是一种意向驱动式面向agent程序设计语言。该语言以改造的情境演算[3,4]为理论框架,吸收了GOLOG[5]的合理成分,加入对信念、能力、意向、策略等agent心智成份的处理,使用信念修正原语处理通信交互以及事件响应等外因行动,采用离线执行、在线执行以及在线-离线相结合的运行方式,从而解决了GOLOG语言在应用于面向agent程序设计时不能有效地描述处理agent心智状态,无法处理外因行动等问题。AOPLID语言具有很强的模块化设计特征,易于编程、扩展和动态化。

本文旨在对AOPLID语言进行时序扩充,即在AOPLID语言中引入时间概念,使之能表达并处理带有时间参数的并发行动。首先,对经典情境演算进行适当改造,使之能描述含时间变元的行动,因为持续行动一般可认为是具有瞬时开始和瞬时终止的过程,所以可以将一个持续动作分解为两个时间上互不相交的瞬时动作,再引入一个新的关系流刻画这两个瞬时动作的执行情况,从而可在扩充后的情境演算中表达带时间参数的并发行动。在此基础上,通过对原有的离线方式程序语义[1]作适当修改,实现了离线方式时序AOPLID语言(Temporal AOPLID,简记为TAOPLID)的解释器。

本文第2节介绍AOPLID语言的基本概念;第3节给出了经改造的情境演算系统称为时序情境演算(Temporal Situation Calculus)的语言和公理系统;第4节实现了离线方式TAOPLID解释器;最后总结和介绍进一步要进行的工作。

2 AOPLID语言简介

为方便后文叙述,本节首先简单介绍一些与AOPLID语言相关的基本概念,详细说明参见文[1,2]。

2.1 基于情境演算的agent模型

情境演算是McCarthy与Hayes于1969年提出的用于刻画动态系统变化规律的多型二阶逻辑系统,其基本思想是,当一个动态系统中所包含的每个原子行动的执行条件及对各种流的影响情况均预先给定时,系统在任何情境下的状态也随之确定,并可以通过推理的方式获知系统在任何情境下的系统状态。有关情境演算的详细说明请参见文[3,4]。

*)本项研究得到国家重点基础研究发展规划973(No.2002CB312002)及江协省自然科学基金的赞助。李 斌 博士,副教授,研究方向是软件agent技术及应用、人工智能。

设 L_{sc}为一个情境演算语言，L 为其压缩[1]，则称六元组 $M=(D,B,E,C,I,S)$为基于语言 L_{sc}的一个 agent 模型，其中：

① $D=(A,R,F)$，A 表示 L_{sc}中所有原子行动符号，R 表示 L 中所有的关系流(情境变元被隐藏掉)，F 表示 L 中所有函数流(情境变元被隐藏掉)。

② B：信念集，其中信念指 agent 对自身状态及环境的认识。

③ E：效应集，用于反映流受行动的影响情况。

④ C：能力集，用于决定 agent 的行为。

⑤ I：意向集，用于反映 agent 为实现某个目标而采用的行动。

⑥ S：策略集，其中策略用于定义 agent 的复杂行动。

2.2 AOPLID 程序组成

每个 AOPLID 程序由六个部分组成，其形式如下：

〈Program〉=〈Declarations〉,〈Capabilities〉,〈Beliefs〉,〈Intentions〉,〈Strategies〉,〈Effects〉

其中 AOPLID 程序的六个部分与 agent 模型的六个成分完全对应，因此我们也将一个 AOPLID 程序记为六元组 $P=(D,C,E,B,I,S)$，根据 AOPLID 程序的语义，一个 AOPLID 的执行过程实质上就是一个 agent 模型转变为另一个 agent 模型的过程。

2.3 AOPLID 程序的执行方式

AOPLID 程序可以按三种不同的方式执行：离线执行、在线执行及在线-离线结合方式执行。在线方式执行 AOPLID 程序是指逐条执行程序中的每个可行的原子行动；离线方式执行 AOPLID 程序并不是逐条执行 AOPLID 程序中的每个原子行动，而是对每个给定的 AOPLID 程序 P，通过定理证明器证明下列推理关系$\models\exists\ p',s\ Trans(p,[],p',s)$，其中：*Trans* 谓词定义参见文[1]。如果获得一个构造性证明，则可得到 s 的一个基例化 $s=[a_1,a_2,\cdots,a_n]$。根据 AOPLID 程序的语义，s 是一个相对于 P 可执行的行动序列，这一行动序列被转交给外部执行器执行。在线-离线方式则是上列两种方式的结合。

3 时序情境演算

并发行动的表示与处理是行动理论中的一个关键问题，文[6]提出了一种利用现有情境演算理论表达并发行动的方法，即将持续行动看成由瞬时开始行动和瞬时终止行动组成的一段过程，这样持续行动就可由关系流、两个原子行动(瞬时开始行动和瞬时终止行动)三部分刻画。例如，若用 *walk*(x,y)表示某机器人从地点 x 处走到地点 y 处，则 *walk*(x,y)为一个持续动作，可以用两个原子行动 *startWalk*(x,y)及 *endWalk*(x,y)分别表示这一动作的开始与结束，而用关系流 *walking*(x,y,s)反映这一动作执行的状态。再如 *sing* 这一动作可分解为原子行动 *startSing*、*endSing* 及关系流 *singing* 三个成份。这样表达机器人边走边唱的并发过程可用类似于下列情境表达式表示：

do(*endWalk*(*A*,*B*),*do*(*endSing*,*do*(*startSing*, *do*(*startWalk*(*A*,*B*),s_0))))：表示行动 *sing* 包含在行动 *walk*(*A*,*B*)中；

do(*endWalk*(*A*,*B*),*do*(*endSing*,*do*(*startWalk*(*A*,*B*), *do*(*startSing*,s_0))))：表示行动 *sing* 与行动 *walk*(*A*,*B*)交叉。

上述方法能表达任意的交叉并发的行动，但问题是这种表达方法尽管能表达一行动发生在另一行动之前或之后，但不能刻画动作的持续时间。如上面的第一个表达式虽可表示行动 *sing* 在行动 *walk* 开始后发生，但不能描述在行动 *walk* 开始后多久发生。为此，我们在上列处理方法的基础上再引入时间变元，如用 *startWalk*(*A*,*B*,2) 表示在时刻2机器人开始从 *A* 走向 *B*，*endSing*(10) 表示在时刻10停止唱歌。则上例可进一步进精化为：*do*(*endWalk*(*A*,*B*,8),*do*(*endSing*(5),*do*(*startSing*(3),*do*(*startWalk*(*A*,*B*,2),s_0)))))：表示行动 *walk* 发生在时刻2到时刻8，行动 *sing* 发生在时刻3到时刻5；*do*(*endWalk*(*A*,*B*,8),*do*(*endSing*(6),*do*(*startWalk*(*A*,*B*,4),*do*(*startSing*(2),s_0)))))：表示行动 *sing* 发生在时刻2到时刻6，行动 *walk* 发生在时刻4到时刻8。

根据上述思想，我们对经典情境演算进行适当扩充与改造，经改造的情境演算系统称为时序情境演算。

3.1 时序情境演算的语言构成

时序情境演算语言 L_{TSC}与经典情境演算语言 L_{SC}基本相似，只在下列几方面有所不同：① 增加时间型 time；② 原子行动符号的型改为 $object^n\times time\rightarrow action$；③ 引入下列两个原始函数符号：

a. 时间函数 timer：action→time。用于表示行动发生的时刻，如 timer(startWalk(A,B,t))=t

b. 开始函数 start：situation→time。用于表示情境的开始时刻。

3.2 时序情境演算的公理系统

时序情境演算公理系统是在经典情境演算的公理系统的基础上引入一条新的公理，其公理系统为：

AX1 $\forall\ P(P(S_0)\wedge\forall_{a,s}P(s)\rightarrow P(\mathrm{do}(a,s)))\rightarrow\forall\ s\ P(s)$

AX2 $\forall\ a,s(\mathrm{do}(a,s)=\mathrm{do}(a',s')\leftrightarrow a=a'\wedge s=s')$

AX3 $\forall\ s(\neg s\sqsubset s_0)$

AX4 $\forall\ s,a,s'(s\sqsubset\mathrm{do}(a,s')\leftrightarrow s\sqsubseteq s')$

AX5 *start*(do(*a*,*s*))=*timer*(*a*)

其中公理 AX1～AX4为经典情境演算公理，公理(AX5)为新引入的公理，表示情境 s'若为在情境 s 下采取行动 a 后所产生的后继情境(即 $s'=\mathrm{do}(a,s)$)，则情境 s'的开始时刻等于行动 a 的发生时刻。

定义1 时序情境演算中情境的可达关系"$<$"：

$s<s'=_{df}s\sqsubset s'\wedge(\forall\ a,s^*)(s\sqsubset\mathrm{do}(a,s^*)\sqsubseteq(s'\rightarrow poss(a,s^*)\leqslant start(s^*)\leqslant timer(a))$

表示从情境 s 可以通过可行行动序列到达情境 s'，且可行行动序列的发生时刻非递减。特别地，$S_0<s$ 表示情境 s 可以在初始情境 S_0下通过一可行行动序列到达。

除了上述与领域无关的公理外，对一个应用领域而言，若试图用情境演算描述该应用领域的某个动态系统，还需提供如下与领域有关的公理：①行动的预决条件公理；②效应公理与框架公理；③初始情境公理；这些公理与经典情境演算相同。

4 时序 AOPLID 语言 TAOPLID

4.1 AOPLID 语言的时序功能扩充

通过上文可以知道，时序情境演算是对经典的情境演算进行扩充，引入了时间变元，增加了相应的情境公理，重新定义了情境的可达关系，这样就可在扩充后的时序情境演算中表达带时间参数的并发行动。而一个 AOPLID 程序的执行过程实质上就是由一个 agent 模型转变为另一个 agent 模型的

过程，在离线方式运行下，它是根据离线方式程序语义，通过离线 AOPLID 解释器解释执行的。因此为了在 AOPLID 语言中表示并处理带时间参数的并发行动，我们只要在 AOPLID 程序中给出有关行动的时间信息，并根据上述的时序情境演算实现离线方式下 AOPLID 解释器即可。因为 AOPLID 语言中行动用带参数的行动名表示，所以只要在行动参数表中增加时间参数即可表达带时间参数的行动，无须对 AOPLID 语言的语形作任何改变。

4.2 离线方式下 TAOPLID 解释器

文[1]给出了离线方式下 AOPLID 程序语义，我们只要对文[1]原子行动 α 的 *trans* 函数中加入判断条件 $start(s) \leqslant timer(\alpha)$ 即可，其余不变。如上文所述，以离线方式运行的 TAOPLID 程序解释器的核心部分为一个定理证明器，它通过证明生成一个行动序列，然后转交给外部执行器执行。与现有的其它程序设计语言相比，由于 Prolog 语言是一种逻辑型程序设计语言，因此用 Prolog 语言实现一个 TAOPLID 的解释器将更为方便。然而 TAOPLID 程序是一个六元组，而每个元组均是以集合形式表示的。因此，我们设计了一个 TAOPLID 程序预处理器，通过预处理器将 TAOPLID 程序中的每个集合转换为以 Prolog 语言编写的 TAOPLID 解释器能处理的子句形式，然后提交给 TAOPLID 解释器，生成一可执行行动序列。

4.2.1 离线执行方式下的 TAOPLID 预处理器　在 TAOPLID 解释器中，我们用谓词 *priact*($\alpha(x)$)表示 α 为原子行动；*capa*($\alpha(x)$,S)表示情境 S 下原子行动 α 可执行条件；*inte*(〈行动表达式〉,〈权值〉)表示〈行动表达式〉是该意向所打算采取的行动，〈权值〉作为 agent 采纳该意向的基准；*stra*(〈复杂行动名〉(〈变元〉*),〈行动表达式〉)表示〈行动表达式〉为此复杂行动的具体的实施方案。则 TAOPLID 预处理器对 TAOPLID 程序中各组成部分的处理方法为：

① 指称说明中关系流、普通谓词转换为同名的 Prolog 谓词；函数流、普通函数转换为同名的 Prolog 函词。

② 原子行动 $\alpha(x)$ 转换为相应的 Prolog 子句 *priact*($\alpha(x)$)。

③ 对每条能力规则 $c \in C$，若 c 形为(〈原子行动〉,〈条件〉)，则转换为相应的 Prolog 子句 capa(〈原子行动〉,S):-〈条件1〉。其中 S 为情境变元，〈条件1〉为〈条件〉恢复情境变元的结果。

④ 对每条效应规则 $e \in E$，若 e 形如(〈关系流名〉〈变元〉*,T,〈原子行动〉,F,〈条件〉)，则将其转换为相应的 Prolog 子句：〈关系流名〉(〈变元〉*,*trans*(A,S)):-关系流名(〈变元〉*,S),not A=〈原子行动〉,〈条件1〉。

若 e 形如(〈关系流名〉〈变元〉*,F,〈原子行动〉,T,〈条件〉)，则将其转换为相应的 Prolog 子句：〈关系流名〉(〈变元〉*,*trans*(A,S)):- A=〈原子行动〉,〈条件1〉;〈关系流名〉(〈变元〉*,S)。其中 S 为情境变元，〈条件1〉为〈条件〉恢复情境变元的结果。

函数流的转换规则类似。

如果某个流同时受若干原子行动影响，则需将子句右端析取即可。

⑤ 信念集的处理。在 TAOPLID 语言中，信念集采用全一阶语言描述，因此在推理复杂性方面有不可逾越障碍，在实现时必须在语形上加以适当限制。我们采用 Prolog 语言实现，因此限定 TAOPLID 程序的信念集以 Horn 子句的形式表示，在这一限定下，TAOPLID 程序中的每个信念将可以被直接转换为 Prolog 子句，其推理将采用 Prolog 语言所采用的失败即否定的方式进行。

⑥ 对每条意向规则 $i \in I$，若 i 形为(〈行动表达式〉,〈目标〉,〈动机〉,〈权值〉)，则转换为相应的 Prolog 子句：*inte*(〈行动表达式〉,〈权值〉):- *not*〈目标1〉,〈动机1〉。其中〈目标1〉,〈动机1〉是相应的〈目标〉,〈动机〉恢复情境变元的结果，每一意向子句按权值从大到小依次排列。

⑦ 对每条策略规则 $s \in S$，若 s 形如(〈复杂行动名〉(〈变元〉*),〈行动表达式〉)，则转换为相应的 Prolog 子句：*stra*(〈复杂行动名〉(〈变元〉*),〈行动表达式〉)。

⑧ 增加恢复情境变元子句：*reSitArg*(〈关系流名〉(〈变元〉*),S,〈关系流名〉(〈变元〉*,S)；*reSitArg*(〈函数流名〉(〈变元〉*),S,〈函数流名〉(〈变元〉*,S))。

⑨ 恢复 *start* 谓词的初始情境：*start*(〈*term*〉)修改为 *start*(s_0,〈*term*〉)。

⑩ 增加行动发生时间规则：

timer(〈*primitive_action_name*〉(〈*variable_type*〉*,T),T)

4.2.2 离线执行方式下的 TAOPLID 解释器　TAOPLID 解释器是根据文[1]给出的离线方式的 AOPLID 程序语义而设计的，用 Prolog 语言实现。其中，TAOPLID 行动表达式为以下形式表示：

① $d_1:d_2:\cdots:d_n$ 表示行动的串联；

② ?(p)测试条件表达式 p 是否为真；

③$\alpha \# \beta$ 表示非确定性地从 α 与 β 中选择其一执行；

④ if(p,α)表示如果条件 p 成立，则执行行动 α；

⑤ if(p,α,β)表示如果条件 p 成立，则执行行动 α，否则执行行动 β；

⑥ *star*(α) 表示非确定性地重复执行行动 α；

⑦ *while*(p,α)表示如果条件 p 成立，则循环执行行动 α；

⑧ *pi*(x,α)表示非确定性地选择 x 的一个值，用该值执行 α；

⑨ α 表示用户定义的原子行动或策略集中复杂行动。

TAOPLID 解释器的 Prolog 源代码如下：

```
/ * TAOPLID Intepreter * /
:-op(800,xfy,[&]). / * Conjunction * /
:-op(850,xfy,[V]). / * Disjunction * /
:-op(870,xfy,[=>]). / * Implication * /
:-op(880,xfy,[<=>]). / * Equivalence * /
:-op(950,xfy,[:]). / * Action Sequence * /
:-op(960,xfy,[#]). / * Nondeterministic action choice * /
trans(E1:E2,S,S1):-trans(E1,S,S2),trans(E2,S2,S1).
trans(?(P),S,S):- holds(P,S).
trans(E1#E2,S,S1):-trans(E1,S,S1);trans(E2,S,S1).
trans(if(P,E),S,S1):-trans(?(P):E,S,S1);trans(?(-P),S,S).
trans(if(P,E1,E2),S,S1):-trans(?(P):E1#?(-P):E2,S,S1).
trans(star(E),S,S1):-S1=S;trans(E:star(E),S,S1).
trans(while(P,E),S,S1):-trans(star(?(P):E):?(-P),S,S1).
trans(pi(X,E),S,S1):-sub(X,_,E,E1),trans(E1,S,S1).
trans(E,S,S1):- stra(E,E1),trans(E1,S,S1).
trans (E,S,trans(E,S)):-priact(E),capa(E,S),start(S,T1),timer
    (E,T2),T1<=T2.
transmain(S,S1):- inte(E,R),trans(E,S,S1).

/ * sub(Name,New,Term1,Term2):Term2 is Term1 with Name re-
placed by New * /
sub(X1,X2,T1,T2):- var(T1),T2=T1.
sub(X1,X2,T1,T2):- not var(T1),T1=X1,T2=X2.
sub (X1, X2, T1, T2 ) :- not T1=X1, T1=..[ F | L1 ], sub_list
    (X1,X2,L1,L2), T2=..[F|L2].
sub_list(X1,X2,[],[]).
sub_list(X1,X2,[T1|L1],[T2|L2]):- sub(X1,X2,T1,T2),sub_
  list(X1,X2,L1,L2).
```

(下转第154页)

具有接近的时间代价的情况下，取得了更好的聚类效果，因此，WRPC 算法在聚类效果和时间代价上取得了较好的折衷，实用性更好。

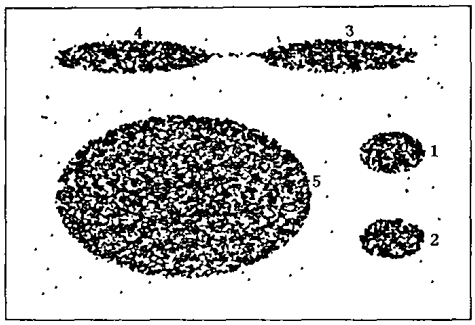

图6 实验2数据集

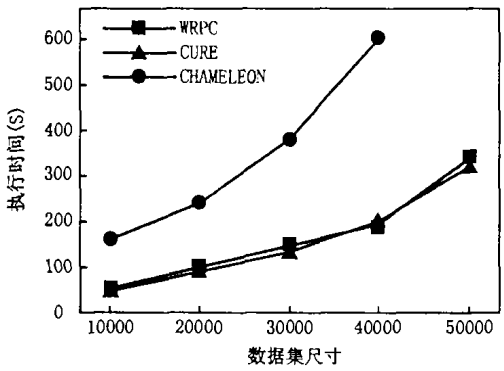

图7 实验2算法执行时间对比

结束语 本文对基于多代表点的聚类方法进行了研究，较为深入地讨论部分已有方法的优点和存在的不足，并在此基础上提出了 WRPC 算法，该算法采用了基于影响因子的代表点选取机制和基于 k-近邻方法的小簇合并机制。实验结果表明，WRPC 在时间代价与 CURE 相同的前提下，得到了接近于 CHAMELEON 的聚类效果，因此在算法的时间代价和聚类效果方面取得了更好的折衷。由于多代表点可以作为簇所反映的知识的浓缩，因此我们将对基于多代表点的可继承聚类算法做进一步的研究。

参 考 文 献

1 Kantardzic M. Data Mining concepts, models, method, and algorithms. 闪四清等译. 数据挖掘 概念、模型、方法和算法. 清华大学出版社，2003
2 Guha S, Rastogi R, Shim K. CURE: an efficient clustering algorithm for large databases. In: Proc. of 1998 ACM-SIGMOD Int. Conf. Management of Data, 1998
3 Karypis G, Han E-H, Kumar V. CHAMELEON: a hierarchical clustering algorithm using dynamic modeling. COMPUTER, 1999, 32: 68～75
4 Karypis G, Kumar V. Mulitilevel k-way hypergraph partitioning. In: Proc. of the Design and Automation Conf., 1999
5 Jarvis R A, Patrick E A. Clustering using a similarity measure based on shared nearest neighbors. IEEE Trans on Computers, 1973, C-22(11)
6 O'Callaghan L, Mishra N, Meyerson A, Guha S, Motwani R. Streaming-Data algorithms for high-quality clustering. In: 18th Intl. Conf. on Data Engineering, 2002
7 Ng R T, Han Jiawei. Efficient and effective clustering methods for spatial data mining. In: Proc. of the 20thVLDB Conf. Santiago, Chile, 1994
8 Li C R, Chen M S. A robust and efficient clustering algorithm based on cohesion self-merging. SIGKDD, Edmonton, Alberta, Canada, 2002

(上接第146页)

```
holds(P&Q,S):- holds(P,S),holds(Q,S).
holds(PVQ,S):- holds(P,S);holds(Q,S).
holds(P⇒Q,S):- holds(-PVQ,S).
holds(P⇔Q,S):- holds((P⇒Q)&(Q⇒P),S).
holds(-(-P),S):- holds(P,S).
holds(-(P&Q),S):- holds(-PV-Q,S).
holds(-(PVQ),S):- holds(-P&-Q,S).
holds(-(P⇒Q),S):- holds(-(-PVQ),S).
holds(-(P⇔Q),S):- holds(-((P⇒Q)&(Q⇒P)),S).
holds(-all(X,P),S):- holds(some(X,-P),S).
holds(-some(X,P),S):- not holds(some(X,P),S).
holds(-P,S):- isAtom(P),not holds(P,S).
holds(all(X,P),S):- holds(-some(X,-P),S).
holds(some(X,P),S):- sub(X,_,P,P1),holds(P1,S).

/* Restore Situation arguments to situation suppressed terms */
holds(A,S):-reSitArg(A,S,F),F;not reSitArg(A,S,F), isAtom(A).
isAtom(A):- not( A=-W;A=(W1&W2); A=(W1VW2); A=(W1=>W2); A=(W1<=>W2); A=some(X,W); A=all(X,W)).
reSitArg(capa(A),S,capa(A,S)).
reSitArg(start(T),S,start(S,T)).
reSitArg(now(T),S,now(S,T)).

/*"now" is a synonym for "start"*/
now(S,T):-start(S,T).
/*the start time of situation trans(A,B) is the time of A*/
start(trans(A,S),T):-timer(A,T).
```

总结 面向 agent 的理论与技术是当今人工智能与软件工程研究的重要内容，agent 理论、agent 结构、面向 agent 程序设计语言(AOPL)是 agent 理论与技术研究的核心内容[7]。

AOPLID 是一种建立在情境演算理论基础上的面向 agent 程序设计语言。与其它相关工作相比，该语言既有较完善的理论基础又有完整的程序设计语言的面向 agent 程序设计的解决方案[2]。本文主要对 AOPLID 进行时序扩充，使得 AOPLID 语言能够表示并处理带时间参数的并发行动。首先，在经典的情境演算中引入时间变元，将持续动作看成具有瞬时开始和瞬时终止的过程，从而在时序情境演算中表达带时间参数的并发行动；其次，对离线方式下 AOPLID 程序语义进行了适当改造，设计并实现了时序 AOPLID 离线解释器。

与此同时，在 agent 理论与 agent 结构研究方面，我们基于情境演算理论，提出了基于情境演算的智能体结构和刻画 agent 自主性的框架[8,9]；基于关系理论，提出了基于关系的两维意向结构[10]。

今后的工作主要有进一步扩充 TAOPLID 语言，研究如何在 TAOPLID 中表示其它 agent 的知识、不确定性的知识并进行相应推理；研究在 TAOPLID 中多 agent 规划的各种表示方法及其解释器的实现。

参 考 文 献

1 张东摩. 面向 Agent 的智能系统的理论研究：[博士后出站研究总结报告]. 南京大学研究生院，1998
2 郭磊，戈也挺，陈世福，张东摩. 一种意向驱动式面向 agent 程序设计语言. 软件学报，2003，14(3)：383～391
3 McCarthy J, Hayes P. Some Philosophical problems from the Standpoint of Artificial Intelligence. In: Meltzer B, Michie D. eds. Machine Intelligence 4, Edinburgh University Press, Edinburgh, Scotland, 1969. 463～502
4 Reiter R. Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems. Cambridge, MA: MIT Press, 2001. 35～105
5 Levesque HJ, Reiter R, Lesperance Y, et al. GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming, 1997, 31: 59～84
6 Reiter R. Sequential Temporal GOLOG. In: Proc. of KR'98, 1998. 547～556
7 刘大有，杨鲲，陈建中. Agent 研究现状与发展趋势. 软件学报，2000，11(3)：315～321
8 李斌，吕建，朱梧槚. 基于情境演算的智能体结构. 软件学报，2003，14(4)：733～742
9 李斌，陈韬略，吕建. 一个刻画 Agent 自主性的框架. 南京大学学报，2004，40(2)：137～145
10 李斌，朱朝晖，陈韬略，吕建，朱梧槚. 基于关系的两维意向结构. 软件学报，2004，15(4)：512～521