一种意向驱动式面向 agent 程序设计语言*

郭 磊[1], 戈也挺[1], 陈世福[1+], 张东摩[2]

[1](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093)

[2](南京航空航天大学 计算机科学与工程系,江苏 南京 210016)

An Agent-Oriented Programming Language with Intention Driver

GUO Lei[1], GE Ye-Ting[1], CHEN Shi-Fu[1+], ZHANG Dong-Mo[2]

[1](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

[2](Computer Science and Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

+ Corresponding author: E-mail: chensf@nju.edu.cn

http://www.nju.edu.cn

Received 2002-03-19; Accepted 2002-05-24

Guo L, Ge YT, Chen SF, Zhang DM. An agent-oriented programming language with intention driver. *Journal of Software*, 2003,14(3):383~391.

Abstract: An agent-oriented programming language with intention driver is proposed, which is called AOPLID. Based on open situation calculus, AOPLID can be regarded as an improvement of GOLOG that is based on situation calculus. AOPLID can formalize some elements of the agent's mental state, namely belief, intention, capability and strategy. A belief revision operator is introduced in AOPLID to deal with the communication and exogenous events. AOPLID solves the problems GOLOG faces, such as inconvenience of describing the agent's mental state, lack of communication. The syntax of the AOPLID and its semantic under the OSC are presented. An example program of AOPLID that describes the coffee machine is given too.

Key words: agent-oriented programming; exogenous action; situation calculus; belief revision

摘 要: 提出了一种意向驱动式面向 agent 程序设计语言——AOPLID(agent-oriented programming language with intention driver).该语言基于开放式情景演算 OSC(open situation calculus),吸收了 GOLOG 的合理成分,加入对信念、意向、能力、策略等 agent 心智成分的处理,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,并采用了一种新颖的离线规划和在线执行相结合的运行方式,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.给出了 AOPLID 语言语法结构,基于 OSC 的 AOPLID 程序语义以及 AOPLID 程序实例.

关键词: 面向 agent 程序设计;外因行动;情景演算;信念修正

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60003010 (国家自然科学基金); the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No.97028428 (国家教育部博士点基金)

第一作者简介: 郭磊(1974－),男,江苏启东人,博士,主要研究领域为分布式人工智能,agent 逻辑基础,行动推理.

面向 agent 的理论与技术研究是人工智能、软件工程、并行与分布计算、Internet 技术、机器人技术等众多研究领域的重要研究内容.面向 agent 的程序设计(agent oriented programming,简称 AOP)是 agent 理论与技术研究的核心内容之一.

首先提出面向 agent 程序设计概念的是 Shoham,他提出了一种“面向 agent 的程序设计语言(AOP)”[1],将意向、承诺、能力等概念引入到 Common List 中,使其能够处理 agent 的某些心智属性.此后基于不同的理论,应用背景提出了各种面向 agent 的程序设计语言,目前大致可以分为如下几类:

(1) List like.典型代表为 AGENT-0/1[1]及在此基础上发展起来的其他语言系统,如 PLACAS[2].优点在于语形、语义理论完善,缺点是实现困难.

(2) Prolog like.典型代表是 GOLOG[3],优点在于便于表示 agent 的行动,缺点是缺乏有效处理 agent 的心智状态的功能.

(3) C++ like.典型代表为 April[4]与 DAISY[5],其优点在于实现方便,缺点在于过程性成分太强,表示与处理 agent 的心智状态的能力较差.

(4) Rule based.典型代表是 MyWorld[6],其优点在于知识表示形式简单,推理复杂性较小,缺点是理论与实现之间的差距较大.

(5) Speech act based.典型代表是 KQML[7],其优点在于处理 agent 之间的通信与交互的能力较强,但未提供表示与处理 agent 心智属性的功能.

其中,加拿大多伦多大学认知机器人研究小组提出的 GOLOG 语言[3]受到广泛的关注.该语言融合了过程性程序设计语言与逻辑程序设计语言,以情景演算为理论基础,将过程性控制语句解释为情景演算中的复杂行动,利用情景演算的推理机制将其分解为一组可执行的原子行动序列,然后通过外部执行器逐条执行该行动序列中的各原子行动.GOLOG 语言的这种离线执行方式可用于规划生成器.GOLOG 程序可以先试探性地生成一个行动规划,检测目标是否达到,如果未达到再选择其他行动序列,如此循环,直至生成一个可实现目标的行动序列.此外,应用 GOLOG 语言编程,用户只需给出所有原子行动的预决条件及执行结果(对应于情景演算的预决条件公理和后继状态/效应公理),而不必关心每个原子行动具体的实现方式及运行过程.

虽然 GOLOG 语言在某种程度上兼备了过程性语言的可控制性和逻辑程序设计语言的高抽象性的优点,但要应用于面向 agent 程序设计仍存在一些不可忽视的局限性,主要表现在以下几个方面:

(1) 不便描述信念、意向等 agent 的心智成分[8].

(2) 无法感知外界变化、响应事件.事件的发生不可预料,对 agent 心智状态的影响也无法预知,因此无法给出其预决条件和结果,也就无法用 GOLOG 语言编程.

(3) 缺乏通信机制.由于接受到的信息的内容是不可预知的,因此 GOLOG 无法对此进行编程.

GOLOG 语言的上述局限性很大程度上与其理论基础情景演算有关,经典的情景演算缺乏对 agent 通信交互以及事件响应等行为的刻画能力.为了刻画 agent 的这些基本行为,我们在对经典情景演算系统作必要的修正与扩充[9]的基础上引入广义信念修正理论[10,11],提出了一种开放式情景演算系统 OSC(open situation calculus)[12].本文以 OSC 为理论基础,吸收 GOLOG 语言的合理成分,提出一种意向驱动式的面向 agent 程序设计语言 AOPLID(agent oriented programming language with intention driver).

1 OSC 系统简介

1.1 情 景

与经典情景演算不同,OSC 中的情景是 agent 在某一时刻的心智状态.OSC 系统中情景分为两种:一种是 agent 离线规划时所“设想”的可能情景,另一种是 agent 在线执行的过程中实际改变心智状态后的实情景.

当 agent 在情景 s 下执行了一个原子行动以后,s 的后继情景便由离线规划时的可能情景转化为实情景;当交互通信或者事件发生时,其结果将直接影响到 agent 的心智状态,因而交互通信或者事件处理完成后的相应状态是一个实情景,这个实情景在 agent 的规划过程中是没有“设想”到的,不存在相应的可能情景.在出现了没有

“设想”到的情景的时候,agent 上次规划时所“设想”的尚未达到的各种可能情景将成为“不可能达到的可能情景”.行动队列中剩余的行动也将不再可行,这时 agent 必须根据当前的情景再次进行离线规划以生成新的行动序列,并“设想”出一连串新的可能情景.

1.2 外因行动

对事件响应的前提及其结果都是不可预期的,通信交互等行动的结果也是不可预期的,基于其结果的不可预期性这一共同特点,我们把通信交互行动以及事件响应处理等统称为外因行动.外因行动只在 agent 在线执行的过程中发生,直接对 agent 的心智状态产生影响,所以外因行动无法进行离线规划,也无法用一般的封闭式行动逻辑理论进行刻画.信念修正是一种可以很好地刻画 agent 在接受新信息时信念集改变的开放式理论,我们在 OSC 系统中引入信念修正理论刻画外因行动.

1.3 情景流result_of(a,s)和bel_rev(ψ,s)

值域为情景集的流称为情景流,流 result_of(a,s)表示在情景 s 下执行行动 a 后的后继情景.它的值可以是实情景也可以是可能情景.流 bel_rev(ψ,s)表示在情景 s 下用外因行动的结果ψ对 agent 的信念集进行修正后的后继情景.

1.4 公理系统

OSC 系统保留了经典情景演算系统的所有公理,同时,根据广义信念修正理论引入用于刻画外因行动及情景流 bel_rev(ψ,s)的 8 条公理.广义信念修正理论保证 agent 的信念集在发生外因行动后仍能保持其协调性.限于篇幅,本文不详细讨论 OSC 系统公理.由于信念修正操作的非单调性,原规划中行动的先决条件将不一定满足,因此必须再次进行规划,以产生新的行动序列.

2 AOPLID 语言的构成

2.1 基本构成

AOPLID 程序由指称集、能力集、效应集、信念集、意向集和策略集 6 个部分组成,其形式规范如下:

⟨*program*⟩ ::=⟨*declarations*⟩,⟨*capabilities*⟩,⟨*effects*⟩,⟨*beliefs*⟩,⟨*intentions*⟩,⟨*strategies*⟩

⟨*declarations*⟩ ::=**DECLARATIONS**

primitive action [⟨*primitive-action-name*⟩(⟨*variable-type*⟩*)]*

relational fluent [⟨*relational-fluent-name*⟩(⟨*variable-type*⟩*)]*|

[⟨*relational-fluent-name*⟩(⟨*variable*⟩*)=⟨*formula*⟩]*

functional fluent [⟨*function-type*⟩⟨*functional-fluent-name*⟩(⟨*variable-type*⟩*)]*|

[⟨*functional-fluent-name*⟩(⟨*variable*⟩*)=⟨*term*⟩]*

predicate [⟨*predicate-name*⟩(⟨*variable-type*⟩*)]*|

[⟨*predicate-name*⟩(⟨*variable*⟩*)=⟨*formula*⟩]*

function [⟨*function-type*⟩⟨*function-name*⟩(⟨*variable-type*⟩*)]*|

[⟨*function-name*⟩(⟨*variable*⟩*)=⟨*pure-term*⟩]*

⟨*capabilities*⟩ ::=**CAPABILITIES**

(⟨*primitive-action*⟩,⟨*sentence*⟩)*

⟨*effects*⟩ ::=**EFFECTS**

(⟨*effect-rule*⟩)*

⟨*beliefs*⟩ ::=**BELIEFS**

(⟨*sentence*⟩)*

⟨*intentions*⟩ ::=**INTENTIONS**

(⟨*closed-action-expression*⟩,⟨*sentence*⟩,⟨*sentence*⟩,⟨*rating*⟩)*

⟨*strategies*⟩ ::=**STRATEGIES**

(⟨*complex-action-name*⟩(⟨*variable*⟩*),⟨*action-expression*⟩)*

⟨*effect-rule*⟩ ::=(⟨*relational-fluent-name*⟩(⟨*variable*⟩*),T,⟨*primitive-action*⟩,F,⟨*formula*⟩)|

(⟨*relational-fluent-name*⟩(⟨*variable*⟩*),F,⟨*primitive-action*⟩,T,⟨*formula*⟩)|

(⟨*functional-fluent-name*⟩(⟨*variable*⟩*),⟨*variable*⟩,⟨*primitive-action*⟩,⟨*variable*⟩,⟨*formula*⟩)

⟨*action-expression*⟩ ::=⟨*atomic-action*⟩|
⟨*action-expression*⟩;⟨*action-expression*⟩|
⟨*action-expression*⟩|⟨*action-expression*⟩|
⟨*condition*⟩?|
while ⟨*condition*⟩ do ⟨*action-expression*⟩ endwhile|
if ⟨*condition*⟩ then ⟨*action-expression*⟩ else ⟨*action-expression*⟩ endif|
pi ⟨*variable*⟩⟨*action-expression*⟩|
star ⟨*action-expression*⟩

⟨*rating*⟩ ::=rational number

⟨*sentence*⟩ ::=first order formula without free variables, in which all predicates (functions) are declared in the declarations as the predicates or relational fluents (functions or functions fluents)

⟨*formula*⟩ ::=first order formula (maybe with free variables), in which all predicates (functions) are declared in the declarations as the predicates or relational fluents (functions or functional fluents)

⟨*atomic-action*⟩ ::=⟨*primitive-action*⟩|⟨*complex-action*⟩

⟨*primitive-action*⟩ ::=⟨*primitive-action-name*⟩(⟨*term*⟩*)

⟨*primitive-action-name*⟩::=string declared in the declarations as a primitive action

⟨*complex-action*⟩ ::=⟨*complex-action-name*⟩(⟨*term*⟩*)

⟨*complex-action-name*⟩::=string declared in the strategies

⟨*closed-action-expression*⟩::=action-expression without free variable

⟨*term*⟩ ::=⟨*variable*⟩|⟨*function-name*⟩(⟨*term*⟩*)

⟨*pure-term*⟩ ::=term without the occurrence of functional fluents

⟨*functions-name*⟩ ::=⟨*functional-fluent-name*⟩|⟨*function-name*⟩

⟨*functional-fluent-name*⟩::=string declared in the declarations

⟨*function-name*⟩ ::=string declared in the declarations

⟨*variable-type*⟩ ::=int|real|bool|sentence|agent

⟨*function-type*⟩ ::=int|real|bool|sentence|agent

2.2 各语言成分的含义

上述形式规范中 AOPLID 程序(⟨*program*⟩)中各组成部分的含义如下:

指称集(⟨*declarations*⟩).说明在本程序中所出现的所有原子行动(primitive actions)、关系流(rational fluents)、函数流(functional fluents)及普通函数(function)与普通谓词(predicate).程序的其余部分中出现的这些对象,除保留行动、保留谓词或函数外,均必须在指称集中加以说明.

能力集(⟨*capabilities*⟩).刻画原子行动执行的前提条件,对应于 OSC 系统中的预决条件公理.如果某个原子行动的能力集中为空,则认为该原子行动可无条件地执行.

效应集(⟨*effects*⟩).刻画流受行动影响的情况,对应于 OSC 系统中的效应公理.只有关系流和函数流才有相应的效应规则.所有效应规则(⟨*effect-rule*⟩)中条件的判断均以行动执行前的信念状态为标准.

信念集(⟨*beliefs*⟩).agent 的信念集合,程序运行前的信念集是 agent 的初始信念,对应于 OSC 系统中的初始情景公理.agent 运行的过程中其信念集可以被改变.

意向集(⟨*intentions*⟩).agent 的意向集合,对应于 OSC 系统中的意向公理.在意向项中,第 1 项是不含自由变元的行动表达式,为达到该意向的目标应执行的行动;第 2 项表示该意向的目标是否满足,其值为真表示已满足,为假表示未满足,若第 2 项为 false,则表示目标永远不能满足;第 3 项表示该意向的动机,其值为真表示实现该意向的动机永不消失,反之则表示该意向满足一次即可;第 4 项为意向的初始优先级,在程序运行中意向的优先级可以被改变.

策略集(⟨*strategies*⟩).一个命名的复杂行动称为策略,类似于过程性程序设计语言中的过程或函数.目前,我们规定了一个复杂行动名(⟨*complex-action-name*⟩)在策略集中只能有惟一的解释项,在以后的扩展中可以考虑一个复杂行动有多个对应解释项,调用时按优先级或设定条件来进行选择.

2.3 信念修正原语与外因行动的处理

外因行动的最基本特征是直接改变 agent 的心智状态.当 agent 收到一组新信息时,会试图将这组信息融入自己的信念集中,这个过程可以由信念修正理论进行刻画.AOPLID 语言提供了信念修正原语,以便对外因行动

引起的信念修正操作进行编程处理,其语形为

BELREV(⟨*sentence*⟩*)

表示用语句集⟨*sentence*⟩*修正 agent 当前信念集,信念修正原语对应于 OSC 系统中的情景流 bel_rev(ψ,*s*).

定义一个外因行动首先应在指称集中说明一个与执行单元相对应的原子行动,然后将外因行动作为一个复杂行动在策略集中加以解释,外因行动的描述与一般复杂行动的描述的区别在于可以使用信念修正原语.

在 AOPLID 中预定义了简单异步通信行动 SEND,READ,开发者可以定义其他复杂外因行动.SEND 和 READ 对应的原子行动是 sendinf,readinf,关系流 MOREINF 表示接收消息队列中是否还有未处理的消息.谓词 TRUST 和 OBL 用于表示 agent 之间的信任和义务关系,这些关系可以编程确定,具体定义如下:

```
DECLARATIONS
    primitive action
    sendinf (agent sentence)   //向一个 agent 发送一条带编号的消息
    readinf (agent sentence)   //从接收消息队列中读一条消息
    relational fluent
       MOREINF
       predicate
            TRUST (agent)   OBL (agent)
    …
BELIEFS
    (TRUST (agt1))   (TRUST (agt2))   (OBL (agt3))
    …
STRATEGIES
    (SEND (agent agt sentence inf) , sendinf (agt inf))
    (READ, while MOREINF do
            readinf (agt inf); if TRUST(agt)∨OBL(agt) then BELREV(inf) endif endwhile)
```

3 AOPLID 程序的运行状态与例程

3.1 离线规划状态和在线执行状态

为了处理通信交互以及事件响应等外因行动,AOPLID 语言采用了一种新颖的在、离线结合的运行方式.一个 AOPLID 程序有两种运行状态:离线(off-line)规划状态和在线(on-line)执行状态.

在离线规划状态下,执行单元不工作,原子行动不被真正执行.程序根据当前的状态在意向的驱动下生成一组可执行的原子行动序列.AOPLID 程序在离线规划状态下是可回溯的.

在在线执行状态下,执行单元将实际执行规划生成的行动序列中的原子行动,并修改程序的信念集等成分.程序是不可回溯的.此时,程序可以处理通信交互及事件响应等直接影响 agent 心智状态的外因行动.离线规划状态和在线执行状态的关系如图 1 所示.

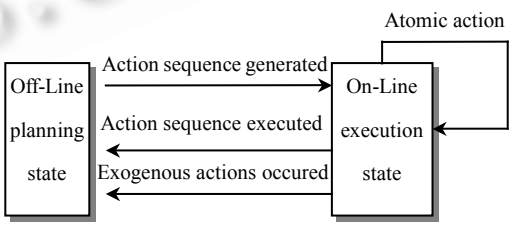

Fig.1 The states of an AOPLID program

图 1 AOPLID 程序运行状态

3.2 咖啡机器人控制程序

咖啡机器人的例子被许多文章引用,这里我们给出 AOPLID 语言编写的咖啡机器人控制程序.

```
DECLARATIONS
    primitive action
        pickupCoffee            //端起一杯咖啡
        giveCoffee(string)      //给某人一杯咖啡
```

```
        goto(string)              //走到某个位置
    relational fluent
        has_coffee(string)        //某人已经有了咖啡了
        holding_coffee            //机器人正端着一杯咖啡
    functional fluent
        string robot_location     //机器人所处的位置
    predicate
        wants_coffee(string)      //某人想要一杯咖啡
        clerk(string)             //某人是本部门的员工
        no_coffee                 //咖啡机器里没有咖啡了
    function
        string office(string)     //某人所在的办公室
CAPABILITIES
    (pickupCoffee,¬holding_coffee∧robot_location=CoffeeMachine)
    (giveCoffee(person),holding_coffee∧robot_location=office(person))
    (goto(location),robotlacation≠location)
EFFECTS
    (has_coffe(person),F,giveCoffee(person),T,true)
    (holding_coffee,F,pickupCoffee,T,true)
    (holding_coffee,T,giveCoffee(person),F,true)
    (robot_location,location1,goto(location2),location2,true)
BELIEFS
    (robot_location=CoffeeMachine)(¬holding_coffee)(¬no_coffee)
    (clerk(Mary))(clerk(David))(clerk(Bob))
    (TRUST(Mary))(TRUST(David))
INTENTIONS
    (deliverCoffee,∃p(clerk(p)∧wants_coffee(p)→has_coffee(p)),no_coffee,1)
STRATEGIES
    (deliverCoffee,if ∃p(clerk(p)∧wants_coffee(p)) then (pi p)((clerk(p)∧wants_coffee(p))?;delicerOneCoffee(p)) else
                READ endif; deliverCoffee)
    (deliverOneCoffee(person), if robot_location≠CoffeeMachine then goto(CoffeeMachine) endif;
                pickupCoffee;goto(office(person));giveCoffee(person)))
```

在上述程序的能力集中,pickupCoffee 动作执行的条件是机器人当前未持有咖啡,并且机器人正处在咖啡机器(CoffeeMachine)处;giveCoffee(*person*)动作的执行条件为机器人持有一杯咖啡,并且在 *person* 的办公室内;goto(*location*)动作的执行条件是机器人当前的位置不在 *location* 处.

效应集中包含 4 条效应规则,分别表示 giveCoffee(*person*)可使关系流 has_coffee(*person*)由假变为真;pickupCoffee 可使关系流 holding_coffee 由假变真,而 giveCoffee(*person*)使之由真变假;goto(*location*)可使函数流 robot_location 的值变为 *location*.

策略集中包含了两个复杂行动的解释,deliverOneCoffee(*person*)是机器人装咖啡、移动到目的地然后把咖啡给 *person* 等一串原子行动所组成的复杂行动;deliverCoffee 检查是否有人需要咖啡,如果有,则送去一杯咖啡,否则调用预定义外因行动 READ 处理接收到的信息,这时可能会有新的“要咖啡”请求(一个或多个)加入到信念集中来,从而实现了对请求的动态响应.

意向集中包含了一条意向,即 deliverCoffee,其目标是使每位需要咖啡的职员都有一杯咖啡,意向的保留动机是咖啡机里仍有咖啡.

AOPLID 程序与 GOLOG 程序的最大区别在于处理外因行动的能力.GOLOG 程序控制的咖啡机器人无法处理外因行动,不能进行通信,所以它只能满足在初始状态中有“要咖啡”请求的职员的需求,如果在程序开始运行后再有职员发出“要咖啡”请求的话只能修改其程序并重新启动执行.而采用 AOPLID 控制程序的咖啡机器人就能够完成当发生“要咖啡”的请求时,咖啡机器人即送上一杯咖啡.显然,具备处理通信能力的机器人更“聪明”.

4 AOPLID 程序的语义

为行文方便,我们约定几个表示方法:一个 AOPLID 程序记为 $P=(D,C,E,B,I,S)$,$P.D$,$P.C$,…,$P.S$ 分别表示程序 P 的指称集、能力集…策略集.若 X 为 C,E,B,I,S 之一,则用 $P|^{X}_{X'}$ 表示在 P 中用 X' 替换 X 后得到的程序.

4.1 离线规划状态下的AOPLID程序的语义

离线规划状态下的 AOPLID 程序与 GOLOG 程序类似,情景的转换是可能情景之间的转换,两者的区别主要在于 GOLOG 程序是由确定的目标驱动的,而 AOLID 程序是在意向的驱动下进行规划的,其意向在程序运行的过程中可以动态变化.首先给出 AOPLID 程序 P 在情景 s 下转换到行动δ被执行后的后继情景 s'下的程序 P' $Trans(\delta,P,s,P',s')$的定义:

定义 1. 递归定义五元谓词 $Trans(\delta,P,s,P',s')$如下:

(1) 若δ为原子行动,$\delta=a$,则

$$Trans(\delta,P,s,P',s') \stackrel{\text{def}}{=} Poss(a,s)\wedge P'=P|^{B}_{BEL(s')}\wedge s'=\text{result\_of}(a,s)$$

其中 $Poss(a,s)=\exists c(c\in C\wedge c.[1]=a\wedge(\wedge P.B\rightarrow c.[2]))$,$\wedge P.B$ 表示 $P.B$ 中所有公式的合取;$c.[1]$表示能力项 c 的第 1 个参数.

(2) 若δ为测试行动,$\delta=\varphi?$,则

$$Trans(\delta,P,s,P',s') \stackrel{\text{def}}{=} (\wedge P.B\rightarrow\varphi)\wedge P'=P\wedge s'=s;$$

(3) 若δ为串联行动,$\delta=\delta_1;\delta_2$,则

$$Trans(\delta,P,s,P',s') \stackrel{\text{def}}{=} \exists P'',s''(Trans(\delta_1,P,s,P'',s'')\wedge Trans(\delta_2,P'',s'',P',s'));$$

(4) 若δ为并联行动,$\delta=\delta_1|\delta_2$,则

$$Trans(\delta,P,s,P',s') \stackrel{\text{def}}{=} (Trans(\delta_1,P,s,P'',s'')\vee Trans(\delta_2,P'',s'',P',s'));$$

(5) 若δ为非确定性参数选择,$\delta=\text{pi}\ \vec{x}\ \delta_1$,则

$$Trans(\delta,P,s,P',s') \stackrel{\text{def}}{=} \exists\vec{x}(Trans(\delta_1,P,s,P',s'));$$

(6) 若δ为非确定性迭代,$\delta=\text{star}\ \delta_1$,则

$$Trans(\delta,P,s,P',s') \stackrel{\text{def}}{=} \forall R(\forall P_1,s_1(R(P,s,P_1,s_1)\wedge Trans(\delta_1,P,s,P_1,s_1))\wedge$$
$$\forall P_2,s_2,P_3,s_3(R(P,s,P_2,s_2)\wedge Trans(\delta_1,P_2,s_2P_3,s_3)\rightarrow R(P,s,P_3,s_3))\rightarrow R(P,s,P',s'));$$

(7) 若δ为复杂行动,$\delta=\alpha(\vec{x})$,则

$$Trans(\delta,P,s,P',s') \stackrel{\text{def}}{=} \exists st(st\in P.S\wedge st.[1]=\alpha(\vec{x})\wedge Trans(st.[2](\vec{x}),P,s,P',s')).$$

if-then 和 while-do 可以通过下列定义解释为特殊的复杂行动:

if φ then δ_1 else δ_2 endif $\stackrel{\text{def}}{=}$ $(\varphi?;\delta_1)|(\neg\varphi?;\delta_1)$

while φ do δ endwhile $\stackrel{\text{def}}{=}$ $\text{star}(\varphi?;\delta)|\neg\varphi?$

下面给出 AOPLID 程序 P 在情景 s 下由意向 i 驱动转换到后继情景 s'下的程序 P' $trans(i,P,s,P',s')$ 的定义.

定义 2. $trans(i,P,s,P',s') \stackrel{\text{def}}{=} \exists\delta(\delta=i.[1]\wedge Trans(\delta,P,s,P',s')\wedge(\wedge P'.B\rightarrow i.[2])\wedge(\wedge P.B\rightarrow\neg i.[3]))$.

一个意向 i 是可行的是指在 OSC 系统中可以证明下列推理关系成立:

$$|=\exists P',s'(trans(i,P,s,P',s')).$$

AOPLID 程序的意向集中的意向通常不是惟一的,因此在离线规划时必须选择一个权值最大的可行意向.下面给出 AOPLID 程序 P 在情景 s 下可进行离线规划 Planable(P,s)的定义.

定义 3. Planable(P,s) $\stackrel{\text{def}}{=}$ $\exists P',s',i(i\in P.I\wedge trans(i,P,s,P',s')\wedge\neg i'(i'\in P.I\wedge i'.[4]>i.[4]))$.

一个 AOPLID 程序在情景 s 下是否可以进行离线规划即指是否可以在 OSC 系统中证明推理关系|= Planable(P,s)成立,也即证明|= $\exists P',s',i(i\in P.I\wedge trans(i,P,s,P',s')\wedge\neg i'(i'\in P.I\wedge i'.[4]>i.[4]))$.

若可以在 OSC 系统中用二阶逻辑的定理证明器得到一个构造性证明,则存在量词 s'的基例:

$$s'=\text{result\_of}(\delta_n,\ldots,\text{result\_of}(\delta_2,\text{result\_of}(\delta_1,s))\ldots)=s[\delta_1,\delta_2,\ldots,\delta_n]$$

中的行动序列$\delta_1,\delta_2,\ldots,\delta_n$即是本次规划所生成的规划行动序列.

4.2 在线运行状态下的AOPLID程序的语义

AOPLID 程序在在线运行状态下,原子行动的执行结果将改变其信念集等成分,此时情景的转换是实情景之间的转换.一个 AOPLID 程序 P 在 s 情景下执行信念修正原语 BELREV(ψ)的语义为:情景转换为实情景 s'=bel_rev(ψ,s),同时程序转换为 $P'=P|^{B}_{BEL(s')}$.AOPLID 程序在线运行状态下的语义如下:

```
AOPLID(P)
  S:=∅;
  while (PI≠∅∧Plannable(P,s)) do
    s1:=planning(P,s)
      while (s≠s1) do
        执行 s1 中行动序列的下一行动δi,
        根据δi 修改 P
        if (δi 是外因行动)
          s:=bel_rev(ψ,s)
        else
          s:=result_of(δi,s);s1:=s
        endif
      endwhile
  endwhile

planning(P,s)
      证明|=Planable(P,s)
  return s′,s′为证明过程中所得到的存在量词基例
```

上述语义的基本思想是,对任一给定的程序,首先试图以离线方式让定理证明器生成一个可执行的方案,主程序根据证明器提交的行动序列依次执行该序列中的每个原子行动,如果发生了外因行动,则说明后续的行动序列不可靠,因此由定理证明器在新的环境下重新生成新的行动队列.

5 相关工作比较与小结

一般认为[13],agent 应该具有信念、意向、目标等心智成分以及交互通信能力、感知能力和对事件的响应处理能力.而 GOLOG 语言在对 agent 的上述性质与能力的描述与处理上存在缺陷.不少研究者对此作了深入的研究,如:Levesque,Shapiro[14,15]探讨了如何在情景演算中加入信念、知识、目标等心智成分,为 GOLOG 语言处理心智成分提供理论基础;Lakemeyer,Levesque[16]就在 GOLOG 语言中引入感知行动提出了解决方案;Shapiro[17]讨论了 GOLOG 语言的通信问题.但在上述解决方案中,没有一种既具有完善的理论基础,又提供了便于进行面向 agent 程序设计的程序语言.

本文和文献[12]在经典情景演算的基础上引入了广义信念修正理论,在 GOLOG 基础上,加入信念、意向、能力、策略等心智成分,引入信念修正原语以处理交互通信和事件的响应处理等外因行动,从而克服了 GOLOG 语言不便描述处理 agent 的心智成分,无法刻画 agent 的交互通信能力和对事件的响应等问题,给出了一个既有较完善的理论基础又有较为完整的程序设计语言的面向 agent 程序设计的解决方案.

我们已经通过 JAVA 编程实现了 AOPLID.限于篇幅,本文未对与之相关的 agent 模型、agent 结构以及 AOPLID 语言的实现技术与运行环境等问题多作探讨,我们将在后续文章中对此进行进一步的讨论.

References:

[1] Shoham, Y. Agent-Oriented programming. Artificial Intelligence, 1993,60(1):51~92.

[2] Thomas R. The PLACA agent programming language. In: Wooldridge M, Jenings N, eds. Intelligent Agents. LNAI 890, Springer-Verlag, 1994. 355~370.

[3] Levesque H, Reiter R, Lespérance Y, Lin F, Scherl R. GOLOG: a logic programming language for dynamic domains. Journal of Logic Programming, 1997,31(1):59~84.

[4] McCabe F, Clark K. April-Agent based systems. In: Wooldridge M, Jenings N, eds. Intelligent Agents. LNAI 890, Springer-Verlag, 1994. 293~317.

[5] Poggi A. DAISY: an object-oriented system for distributed artificial intelligence. In: Wooldridge M, Jenings N, eds. Intelligent Agents. LNAI 890, Springer-Verlag, 1994. 341~354.

[6] Wooldridge M. MYWORLD: the logic of an agent-oriented DAI testbed. In: Wooldridge M, Jenings N, eds. Intelligent Agents. LNAI 890, Springer-Verlag, 1994. 160~178.

[7] Labrou Y, Finin T. A proposal for a new KQML specification. Technique Report, TRCS-97-03, University of Maryland Baltimaore County, 1997.

[8] Hu SL, Shi, CY. Agent-BDI logic. Journal of Software, 2000,11(10):1353~1360 (in Chinese with English Abstract).

[9] Guo L, Zhang DM, Li B. A feasible solution to the frame problem. Pattern Recognition and Artificial Intelligence, 2000,13(2): 121~127 (in Chinese with English Abstract).

[10] Zhang DM. A general framework for belief revision. In: Bai S, ed. Proceedings of the 4th International Conference for Young Computer Scientists. Beijing: Peking University Press, 1995. 574~581.

[11] Zhang DM. Belief revision by sets of sentences. Journal of Computer Science and Technology, 1996,11(2):1~19.

[12] Guo L. Studies on the situation calculus and belief revision [Ph.D. Thesis]. Department of Computer Science and Technology, Nanjing: Nanjing University, 2002 (in Chinese with English Abstract).

[13] Liu, DY, Yang, K, Chen, JZ. Agents: present status and trends. Journal of Software, 2000,11(3):315~321 (in Chinese with English Abstract).

[14] Levesque HJ. What is planning in the presence of sensing? In: Weld D, Clancey B, eds. Proceedings of the 13th National Conference on Artificial Intelligence (AAAI'96). 1996. 1139~1146.

[15] Shapiro S, Lespérance Y, Levesque HJ. Goals and rational action in the situation calculus-a preliminary report. In: Working Notes of the AAAI Fall Symposium on Rational Agency: Concepts, Theories, Models, and Applications. Cambridge, MA: AAAI Press, 1995.

[16] Lakemeyer G, Levesque HJ. AOL: a logic of acting, sensing, knowing, and only knowing. In: Cohn AG, Schubert LK, Shapiro SC, eds. Principles of Knowledge Representation and Reasoning: Proceedings of the 6th International Conference (KR'98). 1998. 316~329.

[17] Shapiro S, Lespérance Y, Levesque HJ. Specifying communicative multi-agent systems with ConGolog. In: Traum D, ed. Working Notes of the AAAI Fall 1997 Symposium on Communicative Action in Humans and Machines. Cambridge, MA: AAAI Press, 1997. 75~82.

附中文参考文献:

[8] 胡山立,石纯一.Agent BDI 逻辑.软件学报,2000,11(10):1353~1360.

[9] 郭磊,张东摩,李斌.框架问题的一种可行解.模式识别与人工智能,2000,13(2):121~127.

[12] 郭磊.情景演算与信念修正的研究及其应用[博士学位论文].南京:南京大学计算机科学与技术系,2002.

[13] 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.