ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.12, December 2009, pp.3101−3116
doi: 10.3724/SP.J.1001.2009.03433

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于事件的Web服务组合方法*

李 鑫[1+], 程 渤[2], 杨国纬[1], 刘启和[1]

[1](电子科技大学 计算机科学与工程学院,四川 成都 610054)

[2](北京邮电大学 网络与交换技术国家重点实验室,北京 100876)

Method of Web Services Composition Based on Events

LI Xin[1+], CHENG Bo[2], YANG Guo-Wei[1], LIU Qi-He[1]

[1](School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China)

[2](State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

+ Corresponding author: E-mail: lixin.hblc@gmail.com

Li X, Cheng B, Yang GW, Liu QH. Method of Web services composition based on events. *Journal of Software*, 2009,20(12):3101−3116. http://www.jos.org.cn/1000-9825/3433.htm

Abstract: To obtain a service composition approach that satisfies the multiple requirements of users and can be easily realized, a novel method based on events is proposed in this paper. Firstly, a simple event language for services is introduced, which can be regarded as a language based on ECA (event-condition-action) rule. Secondly, a composite scheme that describes the composite services is constructed with the modularizing method based on the rule language. The scheme can not only solve the representation problem of services composition domain when adopting AI planning (artificial intelligent planning), but also the insufficiency problem in description capability when the technologies such as UML (unified modeling language) are adopted. Thirdly, to effectively present the composite scheme, the paper accomplishes the scheme's semantics definition and codes the scheme with answer set programs. Finally, the ASP (answer set programming) technology is utilized to present the composite trajectory.

Key words: simple event language for service; answer set programming; composite scheme; composite trajectory; preorder service set; mutually exclusive constraint

摘 要: 为获得一种既易于实现又能满足用户多样化需求的服务组合的有效途径,提出一种基于事件的服务组合方法.首先定义了一种基于 ECA(event-condition-action)规则的语言——简单服务事件语言.在这种语言基础上,通过模块化方法构造的用于描述组合服务的组合方案,不但解决了采用 AI 规划(artificial intelligent planning)时服务组合域表示困难的问题,而且解决了采用 UML(unified modeling language)等技术时描述能力不足的问题.随后,为有效地表示组合方案,完成了它的语义定义以及 answer set 程序编码工作.最后利用 answer set 编程(answer set

* Supported by the National Natural Science Foundation of China under Grant No.60702071 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2005AA114030 (国家高技术研究发展计划(863)); the Program for New Century Excellent Talents in University of China under Grant No.NCET-06-0811 (新世纪优秀人才支持计划); the China Postdoctoral Science Foundation under Grant No.20070410490 (中国博士后科学基金)

Received 2007-10-24; Revised 2008-05-19; Accepted 2008-08-07

programming)技术实现了对组合轨迹的表示.

关键词: 简单服务事件语言;answer set 编程;组合方案;组合轨迹;前序服务集;互斥约束
中图法分类号: TP311 文献标识码: A

Web服务的自动组合是一项具有挑战性的课题,其原因在于智能化目标、服务组合域表示和满足用户多样化需求之间的矛盾.目前主要有以下两类服务组合方法:(1) AI规划(artificial intelligent planning)或情景算子(situation calculus)[1,2];(2) UML(unified modeling language)[3]或模板[4]等.虽然前一类方法能够最大程度地实现智能化,但是很难有效地建立服务规划域,导致该类方法不易实现.例如,为表示拥有由a地到达b地的机票(机票预订服务的效果)与在b地(宾馆房间预订服务的前提)之间的逻辑关系,只能选择可能导致规划复杂化的间接因果链(indirect casual link)[5,6].更为重要的是,不完全信息往往会导致组合无法完成,即使可以借助上下文感知解决此问题[7],同时却带来了规划选择与上下文选择之间的矛盾.后一类方法因为不再需要严格的知识表示,而不存在以上问题,但是该类方法通常因描述能力不足而不能满足用户的多样化需求.另外,属于后一类的还有Petri网[8]和图搜索[9]方法,前者虽然描述能力较强,但并不适合描述复杂的组合情况.后者虽然提高了组合图搜索的效率,但同样存在描述能力不足的问题.

为了获得一种易于实现同时又能满足用户多样化需求的服务组合的有效途径,本文提出了一种基于事件的服务组合方法.首先需要定义一种基于ECA(event-condition-action)规则的语言——简单服务事件语言(simple event language for services,简称SELS).虽然文献[10]中也提出了能够用于服务组合的ECA语言EOVLP(evolving logic program),但是它过于复杂.本文根据用于描述实时策略的PDL(policy description language)[11,12]完成SELS的定义.由PDL构成的策略具有语法简单、语义明确和执行高效等优点,但是它并不适合于服务组合的描述.为此,SELS通过以下3个途径完成了对PDL的改造:(1) 扩充原有事件种类.不仅明确了内部事件,而且增加定义了服务事件;(2) 增加一种用于描述服务组合的规则;(3) 重新解释了复杂事件的发生条件.

在 SELS 规则基础上,利用新的构造方法建立用于描述组合服务的组合方案.一方面,这种组合方案不再需要建立类似 AI 规划中繁琐的知识表示,加之可通过外部事件感知外部信息,从而有效地解决了以上服务组合域表示困难带来的问题.而且一个服务可同时拥有多个前序服务集不仅满足了用户多样化的需求,同时还提高了组合方案的可靠性.另一方面,在组合方案中服务被区分为基本、复杂和组合 3 种不同的类型,其模块化的构造方法使得组合方案具有良好的结构性和可移植性.特别是利用过程构造子(procedural constructor)建立的复杂服务不但能够提高组合的效率,而且有利于实现那些描述困难但又需求频繁的服务的表示.最后,严格的形式语义定义也说明了组合方案具有良好的语义特性.

组合方案良好的语法和语义特性使得采用answer set编程(answer set programming,简称ASP)成为可能.ASP是根据一种重要的非单调逻辑编程(nonmonotonic logic programming,简称NLP)模型语义answer set[13,14]发展而来的NLP技术.存在以下 3 个理由促使本文需要实现组合方案的ASP表示:(1) ASP表示能力强大,使得组合方案具有丰富的描述能力;(2) 实现逻辑编程表示不仅是语义Web的要求[15],也为充分利用后者的本体库提供了可能[16];(3) 已开发出成熟的ASP推理系统,如Smodels[17]和DLV[18].在本文中,不仅完成了组合方案的answer set程序编码工作,而且形式地证明了组合方案与其answer set程序在语义上的等价性.

为了有效地表示组合轨迹,根据服务本身的特性,添加相应的互斥约束就显得十分必要.然而仅仅依靠实现组合方案的语言,如C++或Java,要完成对互斥约束的满足是不现实的.但是根据ASP技术,却可方便地实现对该约束的满足.所以,本文提出了用于转换组合方案 answer set 程序的 MECST(mutually exclusive constraints satisfaction transformation)算法.经过该算法处理后形成的 answer set 程序,由于其 answer set 与组合轨迹存在一一对应的关系,从而实现了对后者的有效表示.

1 SELS 语法

SELS 中包含基本事件、行动和常数 3 种相互互斥的符号集,同时,基本事件又被区分为相互互斥的外部事

件、内部事件和服务事件.外部事件、行动和常数是由系统决定的.其中,通过外部事件,组合方案感知到系统获得的外部信息,如服务搜索结果;行动既是组合方案对发生事件的反应,也是它影响外部世界的唯一手段.内部事件和服务事件都是由组合方案根据需要定义的.其中,内部事件主要用于对执行流程、状态等的描述;服务事件用于标识 Web 基本服务.所有事件和行动都可以拥有一个或者多个变元(argument),其可以是变量也可以是常数.另外,事件的变元也可看作相应事件的属性.

event causes *action* if *condition* (1)

event triggers *internal event* if *condition* (2)

event attaches *service event* if *condition* (3)

SELS 中存在形如式(1)~式(3)的 3 种规则.在解释这 3 种规则之前,首先完成以下 3 个定义:

定义 1. 事件(event)可以是一个基本事件,也可以是一个复杂事件.一个复杂事件被表示为 $e_1 \& ... \& e_n$,其中,$e_i\ (1 \le i \le n)$为事件文字,&为逻辑合取操作符.一个事件文字要么为基本事件 $e(t_1,...,t_n)$,称此为事件正文字,要么为 $!e(t_1,...,t_n)$,称此为事件负文字.正文字表示在某个时间点上事件 e 发生,而负文字则表示相反的意义.到最近时间为止,只有其中正文字代表的事件发生或已经发生,而负文字代表的事件并未发生的情况下,事件 *event* 才发生.事件实例(instance)是通过对变元完成填充(grounding)获得的.另外,本文中事件负文字只限于外部事件.

定义 2. 行动(action)表示为 $a(t_1,...,t_n)\ (n \ge 0)$,其中,$a$ 是行动名,$t_1,...,t_n$ 是它的 n 个变元.在式(1)中,变元 $t_1,...,t_n$ 必须出现在事件正文字中.行动实例定义与上同理.

定义 3. 条件(condition)可以看成是对以上 3 种规则的一种约束.它表示为$p_1,\ldots,p_n$,其中,$p_i\ (1 \le i \le n)$是表示比较关系的原子谓词,逗号表示逻辑合取.与行动一样,条件中的变元也必须出现在相应规则的事件正文字中.

在满足条件的情况下,事件的发生将分别导致规则(1)中的行动被执行、触发规则(2)中的内部事件(internal event)和添加规则(3)中的服务事件(service event)到服务事件序列中.最后需要说明的是,关于组合方案以及服务事件序列的严格形式定义将在下文完成.

2 组合方案

2.1 基本服务

在组合方案中,Web服务被区分为基本服务、复杂服务和组合服务 3 种类型.其中,基本服务就是只需要一次服务上下文交换就能完成的服务,如机票订购(flight booking,简称fb)或火车票订购(train booking,简称tb)服务等,它相当于OWL-S[19]中定义的原子服务;类似GOLOG[20]采用的方法,多个基本服务可通过过程构造子建立复杂服务,以提高组合的效率,如复杂服务if-!*fb*-then-*tb*就是根据构造子If-Then-Else建立的,它表示如果未能订购到机票就订购火车票;组合服务就是根据前两种服务的组合关系而形成的服务.

在组合方案中,首先,每个基本服务都存在一个唯一的服务事件,如可用服务事件*se_fb*和*se_tb*分别标识*bf*和*tb*.其次,无论是基本服务还是复杂服务,都存在一个被称为Proc的内部事件用以唯一标识它们的执行模块,称该模块为Proc模块,如可用*proc_fb*和*proc*_if-!*fb*-then-*tb*分别表示*fb*与if-!*fb*-then-*tb*的Proc模块.基本服务拥有能够唯一标识其服务实例的ID(identity),注意:服务实例不同于服务事件实例,只要ID值相同就为同一服务实例,而服务事件实例还受其他因素影响,如动态ID等.根据RDF[21],ID被定义为服务的URI(uniform resource identifier).另外,在本文中,服务事件和Proc事件分别用以‘*se*’和‘*proc*’为前缀的字符串表示.

假设所有变元值都是合法正确的,式(4)~式(12)完成了任意一个基本服务 *ws* 的 Proc 模块描述.这里只考虑式(4)到(10),剩下两式在本节介绍 While-do 构造子时说明.式(4)~式(10)按以下 4 步完成对 *ws* 执行流程的描述.

Step 1. 当 *proc_ws* 发生且其 *ID* 值为空时,规则(4)执行 *getNaDID_ws* 以获得 *ws* 的系统命名和动态 ID,同时规则(5)触发内部事件 *search_ws* 以使 *ws* 执行流程进入搜索状态.

Step 2. 在搜索状态中且能够顺利获得 *ws* 系统名称和动态 ID(由外部事件 *echo_GND_ws* 表示)的情况下,规则(6)执行 *ac_Search* 以要求系统为 *ws* 搜索合适的服务实例,同时,规则(7)触发内部事件 *conversation_ws* 以使

ws 执行流程进入会话状态.

Step 3. 在会话状态中且能顺利搜索到服务实例(由外部事件 *echo_Search* 表示)的情况下,通过规则(8)执行 *ac_Converse* 以完成会话协议所要求的任务(如,服务上下文交换),同时,还通过规则(9)使得流程进入应答状态(由内部事件 *answer_ws* 表示).

Step 4. 规则(10)表示在应答状态中且该服务实例同意此次服务交易(由外部事件 *echo_Conv* 以及相应条件表示)的情况下,将该服务事件实例添加到服务事件序列中.

$$proc\_ws(ID) \quad \text{causes} \quad getNaDID\_ws \quad \text{if} \quad empty(ID) \tag{4}$$

$$proc\_ws(ID) \quad \text{triggers} \quad search\_ws \quad \text{if} \quad empty(ID) \tag{5}$$

$$search\_ws \,\&\, echo\_GND\_ws(WSN, DID) \quad \text{causes} \quad ac\_Search(WSN, DID) \tag{6}$$

$$search\_ws \,\&\, echo\_GND\_ws(WSN, DID) \quad \text{triggers} \quad conversation\_ws(DID) \tag{7}$$

$$conversation\_ws(DID) \,\&\, echo\_Search(ID, DID) \quad \text{causes} \quad ac\_Converse(ID, DID) \tag{8}$$

$$conversation\_ws(DID) \,\&\, echo\_Search(ID, DID) \quad \text{triggers} \quad answer\_ws(ID, DID) \tag{9}$$

$$answer\_ws(ID, DID) \,\&\, echo\_Conv(ID, AR, DID) \quad \text{attaches} \quad se\_ws(ID, DID) \quad \text{if} \quad AR = approval \tag{10}$$

$$proc\_ws(ID, DID) \quad \text{causes} \quad ac\_Converse(ID, DID) \quad \text{if} \quad \neg empty(ID) \tag{11}$$

$$proc\_ws(ID, DID) \quad \text{triggers} \quad answer\_ws(ID, DID) \quad \text{if} \quad \neg empty(ID) \tag{12}$$

在式(6)和式(7)的 $echo\_GND\_ws(WSN, DID)$ 中,WSN 为服务名称变元,DID 为动态 ID(dynamic ID,简称 DID)变元.提供系统服务名称查询,可以有效克服不同命名带来的语义冲突.同时系统为基本服务 Proc 模块的一次执行提供唯一 DID 号,这样不但可方便系统识别服务实例的动态特征,而且对于组合中由同一服务实例不同执行产生的不同事件和行动,也可有效识别.式(4)~式(10)中,除 *search_ws* 和 *conversation_ws* 不拥有 ID 和 CT 变元外,内部事件和服务事件至少应该拥有表示 ID、DID、父 ID(father ID,简称 FID)、计数(counter,简称 CT)的变元,本文只在相关环境中列出必要变元.可以通过定义用于记录变元内容的内部事件以减少事件包含的变元数,这有利于信息的传递,使得对服务的描述灵活、方便.但为简洁起见,本文忽略该类事件.另外,变元缺省值(本文中为 –1)表示空.

2.2 复杂服务

提供过程构造子不仅可以提高服务组合的效率,更为重要的是,可以利用它表示组合关系描述困难但又需求频繁的服务.本文提供 3 种用于建立复杂服务的过程构造子,它们分别是线序(linear sequence)、If-Then-Else 和While-do.线序构造子又被区分为两种子类型,其中一种为一般线序构造子,用符号‘;’表示.而另一种则为特殊线序构造子,用符号‘;*’表示.这两个构造子的区别在于前者表示的服务组合顺序与服务执行顺序是一致的,而在后者中这两种顺序并不一致.比如,复杂服务*fb*;*txb*表示航班到达后还需要乘出租车到达目的地,其中出租车预定服务由txb(taxi booking)表示.而复杂服务*txb*;**fb*表示需要先乘坐出租车到达机场后再搭乘航班.在*txb*;**fb*中,按照组合顺序,显然*txb*应该在*fb*之前,但是由于机票资源的稀缺性,从而由它形成的上下文约束更为重要,所以应该先执行*fb*然后才执行*txb*,以使后者满足前者形成的服务上下文,而不是相反.设存在基本服务s_1和s_2,式(13)和式(14)完成$s_1;s_2$的Proc模块描述.

$$proc\_s_1;s_2 \quad \text{triggers} \quad proc\_s_1 \tag{13}$$

$$se\_s_1(ID\_s_1) \quad \text{triggers} \quad proc\_s_2(ID\_s_1) \tag{14}$$

在式(14)中,$proc_s_2$通过PSS_ID变元记录s_1的ID值,关于PSS_ID变元将在第 2.3 节介绍.式(15)~式(17)完成$s_1;*s_2$的Proc模块描述.

$$proc\_s_1;*s_2 \quad \text{triggers} \quad proc\_s_2 \tag{15}$$

$$se\_s_2(ID\_s_2) \quad \text{triggers} \quad proc\_s_1(ID\_s_2) \tag{16}$$

$$se\_s_1 \,\&\, se\_s_2(ID\_s_2) \quad \text{attaches} \quad se\_vs(ID\_s_2) \tag{17}$$

式(17)中se_vs为虚拟服务vs的服务事件,vs只拥有服务事件.注意,不同于$proc_s_1$通过PSS_ID变元记录s_2的

ID值,*se_vs*是通过ID变元.

根据构成条件判断的内容,If-Then-Else构造子也被区分为两种子类型.if-!s_1-then-s_2表示s_1和s_2通过第 1 类If-Then-Else构造子建立的复杂服务,而if-*C*-then-s_1-else-s_2则是它们通过第 2 类If-Then-Else构造子建立的复杂服务.它们的区别在于前者中If条件判断的内容为一个基本服务的执行状况,如上文中的if-!*bf*-then-*tb*.而在后者中的相应内容为一个感知(sensing)服务提供的关键信息,关于基本服务被区分为感知服务与改变世界状态(world-altering)服务的详细介绍见文献[1].如复杂服务 if-distance>500-then-*fb*-else-*tb* 表示任意两地相距如果超过 500km,就订购机票,否则订购火车票.其中 distance 值由交通信息服务 ti (transportation information)提供.

式(18)和式(19)完成if-*C*-then-s_1-else-s_2的Proc模块描述.

$$proc\_\text{if-}C\text{-then-}s_1\text{-else-}s_2(C,IM) \quad \text{triggers} \quad proc\_s_1 \quad \text{if} \quad C(IM) \tag{18}$$

$$proc\_\text{if-}C\text{-then-}s_1\text{-else-}s_2(C,IM) \quad \text{triggers} \quad proc\_s_2 \quad \text{if} \neg C(IM) \tag{19}$$

在以上两式中,*IM*是由在s_1和s_2之前执行的一个感知服务获得的关键信息,*C*(*IM*)表示在 *IM* 上的条件判断.当*C*(*IM*)或¬*C*(*IM*)为复杂逻辑式(即由包含逻辑否定符¬或逻辑析取符∨的多个谓词构成)时,首先形成它的否定范式,然后将该范式转化为析取范式,最后通过为析取范式中的每个合取子范式建立类似式(18)或式(19)的规则即可完成等价逻辑表示.

式(20)~式(23)完成if-!s_1-then-s_2的Proc模块描述.

$$proc\_\text{if-!}s_1\text{-}then\text{-}s_2 \quad \text{triggers} \quad proc\_s_1 \tag{20}$$

$$search\_s_1\,\&!echo\_GND\_s_1(WSN,DID) \quad \text{triggers} \quad proc\_s_2 \tag{21}$$

$$conversation\_s_1(DID)\,\&!echo\_Search(ID,DID) \quad \text{triggers} \quad proc\_s_2 \tag{22}$$

$$answer\_s_1(ID,DID)\,\&!echo\_Conv(ID,approval,DID) \quad \text{triggers} \quad proc\_s_2 \tag{23}$$

在以上 4 式中,一方面, $proc\_\text{if-!}s_1\text{-then-}s_2$ 通过规则(20)触发$proc_s_1$;另一方面,规则(21)~规则(23)确保了只有当s_1运行失败时才执行s_2.如果用规则 $proc\_\text{if-!}s_1\text{-then-}s_2\,\&!se\_s_1$ triggers $proc_s_2$取代以上 3 个规则,将会导致s_1和s_2并发的错误.造成这种错误的原因是,组合方案采用了在知识工程中广泛接受的闭世界假设(closed world assumption,简称CWA).CWA意味着对于任何事件或行动,组合方案都假设其没有发生,只有当系统告知一个外部事件发生,或按照其规则推理出一个内部(或服务)事件发生,或执行了一个行动时,组合方案才确定其发生.

s_1通过While-do构造子建立的复杂服务可以描述为while-*M*-*N*-do-s_1,它表示s_1的同一实例需要被循环执行 $N-M+1$ 次.式(24)~式(26)完成了while-*M*-*N*-do-s_1的Proc模块描述.

$$proc\_\text{while-do-}s_1(M,N) \quad \text{triggers} \quad proc\_s_1(-1,-1,M) \tag{24}$$

$$proc\_\text{while-do-}s_1(M,N)\,\&\,se\_s_1(ID,DID,CT) \quad \text{triggers} \quad proc\_s_1(ID,DID,CT+1) \quad \text{if} \quad CT \le N \tag{25}$$

$$proc\_\text{while-do-}s_1(M,N)\,\&\,se\_s_1(ID,DID,CT) \quad \text{attaches} \quad se\_vs(ID,DID,CT-1) \quad \text{if} \quad CT = N+1 \tag{26}$$

由于$proc_s_1$的ID值为空,所以式(24)导致s_1的Proc模块中的式(4)~式(10)被执行.由于$proc_s_1$的ID值不再为空,所以式(25)只可能导致循环执行式(11)、式(12)、式(9)、式(10)*N*–*M*次,从而保证了 while-*M*-*N*-do-s_1 始终只执行s_1的一个实例.与s_1;*s_2一样,while-*M*-*N*-do-s_1利用虚拟服务结束其Proc模块流程表示.

称用于构造复杂服务的基本或虚拟服务为前者的子服务,相应地称前者为后者的父服务.复杂服务也拥有定义为 URI 的 ID 值,子服务的事件始终是通过 FID 变元记录该值.当然,还可以根据以上构造子归纳定义更为复杂的递归服务,如复杂服务 *ti*;if-*distance*>500-then-*fb*-else-*tb* .但由于这些并不是本文讨论的重点,所以不考虑该定义.

2.3 组合服务

在完成对基本和复杂服务的描述后,就可以利用SELS规则完成组合服务的表示.组合方案始终是由一个外部事件触发启动的,如在由a地到b地的旅游服务中,就是由外部事件*travel*(*a*,*b*)触发启动的.称这种事件为初始外部事件*ixe*,同时称由*ixe*直接触发的服务为初始服务.当*n*(*n*≥1)个基本(或复杂)服务$s_1,\ldots,s_n$为初始服务时,可用形如式(27)的*n*个规则描述:

ixe triggers $proc\_s_i$ $1 \le i \le n$ (27)

在上述旅游服务中,既可选择搭乘飞机也可选择乘坐火车到达 b 地,则用两个规则 $travel(a,b)$ triggers $proc\_fb(a,b)$ 和 $travel(a,b)$ triggers $proc\_tb(a,b)$ 完成该表示.

定义 4. 在服务组合中,一个基本(或复杂)服务 s 的前序服务集(preorder service set,简称 PSS)是由能够直接导致它发生的一个或者多个基本(或复杂)服务构成的最小集合,同时称 s 为其 PSS 的后序服务.

当s的一个PSS包含多个服务时,表示只有在这些服务都发生的情况下,才会导致s的发生.而集合的最小性意味着在PSS中去掉任意一个服务,将不会导致s的发生.注意:在PSS中添加别的服务也可能导致s的不发生.除初始服务不存在PSS以外,组合方案中的其他任何服务都必须拥有一个或多个PSS(多PSS情况).多PSS情况为组合提供了多种实现方式,从而满足了用户多样化的需求.另外,由于多个PSS被看作是复杂服务的,同时其Proc模块内部并不产生多PSS情况,所以不需要考虑其子服务的多PSS情况.首先不考虑复杂服务,设$s_1,\ldots,s_{k+1}$ ($k \ge 1$)为$k+1$个基本服务,根据PSS的基值,分以下两种情况讨论.

如果 $PSS(s_2)=\{s_1\}$ ($PSS(s_2)$ 表示 s_2 的一个 PSS),即它们按线序 $sequence(\{s_1\},s_2)$ 关系组合.用式(28)描述.

$se\_s_1(ID\_s_1)$ triggers $proc\_s_2(ID\_s_1)$ (28)

如果 $PSS(s_{k+1})=\{s_1,\ldots,s_k\}$,即它们按聚合 $join(\{s_1,\ldots,s_k\},s_{k+1})$ 关系组合.用式(29)来描述:

$se\_s_1(ID\_s_1)\,\&\ldots\&\,se\_s_k(ID\_s_k)$ triggers $proc\_s_{k+1}(ID\_s_1,\ldots,ID\_s_k)$ (29)

在以上两规则中, $proc\_s_2$ 和 $proc\_s_{k+1}$ 都拥有用于记录 s_2 和 s_{k+1} 的 PSS 元素 ID 值的变元,称这种变元为 PSS_ID 变元.在式(4)~式(10)中,内部事件(除 $answer\_ws$ 以外)以及行动 $ac\_Converse$ 拥有与 $proc\_ws$ 相同数目的 PSS_ID 变元.这种变元不仅能够满足会话协议的要求也能够有效区分不同 PSS 导致的不同组合.如第 2.1 节所述,可以为每个服务的 Proc 模块定义用于记录 PSS_ID 变元内容的内部事件.与之同理,本文忽略该类事件而要求 Proc 事件包含 PSS_ID 变元.

当$PSS(s_1)=\{s_{k+1}\},\ldots,PSS(s_k)=\{s_{k+1}\}$时,即它们按分支 $split(\{s_{k+1}\},(s_1,\ldots,s_k))$ 关系组合,则可用形如式(30)的 k 个规则描述.如果 $split$ 中PSS基大于 1,则可结合式(29)和式(30)描述.

$se\_s_{k+1}(ID\_s_{k+1})$ triggers $proc\_s_i(ID\_s_{k+1})$ $1 \le i \le k$ (30)

如果是在一个PSS拥有的 k ($2 \le k$)个后序服务$s_1,\ldots,s_k$中任选一个组合,即它们是按选择choice(PSS,$(s_1,\ldots,s_k)$)关系组合的.本文称该关系为OR组合关系,这种关系也满足了用户多样化的需求.它的SELS规则描述与分支情况类似,本文第 4 节将最终完成它逻辑意义的表示.

复杂服务的 Proc 事件中也拥有 PSS_ID 变元,并且这些 PSS_ID 值会被传递给其 Proc 模块中最先被执行的子服务.对于复杂服务与其 PSS 的表示,与以上情况类似.但当它们成为另一个服务的 PSS 元素时,就需要作特别处理.对于服务 s,如果 $PSS(s)=\{\text{if-!}s_1\text{-then-}s_2\}$ 或 $PSS(s)=\{\text{if-}C\text{-then-}s_1\text{-else-}s_2\}$,则都需要两个规则 $se\_s_1(ID\_s_1)$ triggers $proc\_s(ID\_s_1)$ 和 $se\_s_2(ID\_s_2)$ triggers $proc\_s(ID\_s_2)$ 描述;如果 $PSS(s)=\{s_1;s_2\}$,需要规则 $se\_s_2(ID\_s_2)$ triggers $proc\_s(ID\_s_2)$;如果 $PSS(s)=\{s_1;{*}s_2\}$ 或 $PSS(s)=\{\text{while-}M\text{-}N\text{-do-}s_2\}$,都需要规则 $se\_vs(ID\_s_2)$ triggers $proc\_s(ID\_s_2)$.另外,以上 PSS_ID 变元的传递原则同样适合复杂服务 Proc 模块中的子服务,如式(14)和式(16).

类似 AI 规划,最后需要一个目标服务事件 $goal$,用以表示一次组合的成功完成.$goal$ 不拥有任何变元,也可将 $goal$ 看作一个特殊的虚拟服务事件.当 $PSS(goal)=\{s_1,\ldots,s_k\}$ ($1 \le k$)时,用式(31)来描述:

$se\_s_1\,\&\ldots\&\,se\_s_k$ attaches $goal$ (31)

本节中的所有 SELS 规则构成了组合模块,它与基本和复杂服务的 Proc 模块一起构成的 SELS 规则有限集合被称为组合方案.一个组合问题被定义为一个三元组 $\langle CS,ixe,goal\rangle$,其中,$CS$ 表示组合方案,ixe 和 $goal$ 分别表示初始外部事件和目标服务事件.最后,关于 DID 作如下说明:在 CS 的一次执行中,基本服务的 Proc 模块一次执行获得的 DID 值是唯一的,所以除 while-M-N-do-s 外,在 CS 中多次被执行的基本服务的相同实例获得的 DID 值互不相同.

根据 PSS 还可用有向图 $G=\langle V,U\rangle$表示 CS 的组合模块,其中 V 的任意节点 s 表示 CS 中的 ixe,或 $goal$,或一个

服务,U中的一条有向边$\langle v,s\rangle$表示v是s的一个PSS元素,如果$v=iex$,则s为一个初始服务.在G中ixe的入度和$goal$的出度都为0,要保证CS正常运行,G中至少不能存在圈.定义以ixe为起点到达s的最大路长为s的最晚加入步(latest joined step,简称LJS).所有初始服务的LJS都为1,该概念将在第4节定理2的证明中被用到.

3 语义及ASP表示

3.1 组合方案的语义

在组合方案CS执行过程中,由事件发生形成的所有事件实例必然在时间上对应于一个事件段(event interval)序列$EI_0,...,EI_n,...$在该序列中,存在两种不同性质的事件段,一种发生了外部事件,其长度(即事件段的值)由系统决定;而另一种并未发生外部事件,其长度为执行相应规则的时间.因此,CS的执行形成了一个事件序列$E_0,...,E_n,...$,其中E_i($0\le i$)是事件实例集,且对应的事件段是EI_i.显然$E_0=\{ixe\}$,严格的表示应为$E_0=\{ground(ixe)\}$,其中$ground$为填充函数,但方便起见,本文忽略该函数.由相同事件触发的多个不同事件由于对应于同一个事件段,所以必然在同一个事件实例集中,如上文旅游服务中,都由$travel(a,b)$触发的$proc_fb(a,b)$和$proc_tb(a,b)$实例.另外,行动实例应在与引起它被执行的事件实例所在的事件段中.

根据事件序列$E_0,...,E_n,...$,可得到只包含服务事件实例的序列$SE_1,...,SE_m,...$,称该序列为服务事件序列.其中,SE_i($1\le i$)为服务事件实例集.显然服务事件序列是事件序列的一子序列,因而存在一个一一影射$\pi(k):SE_k\subseteq E_{\pi(k)}$,其中,$0\le k$,$\pi(0)=0$,$SE_0=\varnothing$.所以存在一个服务段(service interval)序列$SI_1,...,SI_m,...$,其中,SI_i($1\le i$)唯一对应于事件段序列中的一子序列为$EI_{\pi(i-1)},...,EI_{\pi(i)-1}$.用$XE_{i,j}(CS)$表示属于服务段$SI_i$且发生在事件段$EI_j$中的外部事件.设$CS$产生的服务事件序列长度为$m$,则$XE(CS)=\bigcup_{i=1}^{m}\bigcup_{j=\pi(i-1)}^{\pi(i)-1}XE_{i,j}(CS)$表示$CS$执行中发生的所有外部事件.这里需要说明的是,错误的CS描述可导致无限长度的事件序列,所以需要合法性检测以防止这种异常现象的发生.但限于篇幅,关于CS的合法性检测算法将另文讨论.

如采用文献[11,12]中省略事件段序列而直接定义事件序列的话,就不能明确反映出事件段与事件实例集之间的内在关系,也未能区分以上两种性质不同的事件段.更为重要的是,这样会破坏CS语义的严谨性.

定义 5. 如果一个服务事件序列$S_1,...,S_n,S_{n+1},...$满足以下两个条件,(1) 除S_1中的元素(即初始服务事件实例)外的任意一个服务事件实例在该序列中都只存在一个PSS实例;(2) 在由同一个OR组合关系引起的后序服务事件实例中,最多只存在一个实例在该序列中,则称$S_1,...,S_n,S_{n+1},...$为组合序列.如果$goal\in S_{n+1}$,称序列$S_1,...,S_n,goal$为CS的一个组合轨迹(composite trajectory).其中,S_n在$goal$的一个PSS实例中.

在组合序列$S_1,...,S_n,...$中,S_i($2\le i$)中的任意一个元素,其PSS实例中至少存在一个元素在S_{i-1}中.由于CS中存在多PSS情况或OR组合关系,所以一个服务事件序列可包含多个组合序列.

定义 6. 组合方案CS的语义为转换函数$\pi_{CS}:CS\times XE(CS)\to CTset(CS)$.其中,$CTset(CS)$表示$CS$形成的所有组合轨迹构成的集合.

从定义6也可看到组合问题$\langle CS,ixe,goal\rangle$的解就是$CS$语义中的值域$CTset(CS)$.显然,组合服务的一个实例就是$CTset(CS)$中的一个组合轨迹.

3.2 Answer set程序编码

设GCS为组合方案CS一次执行后形成的填充实例,为完成其ASP表示,需要引入以下4个原子谓词:

(1) $occ(xe,EIN,SIN)$.表示外部事件实例$xe\in E_{EIN}$且$\pi(SIN-1)\le EIN\le\pi(SIN)-1$;

(2) $exec(a,EIN,SIN)$.表示行动a在事件段E_{EIN}中被执行且$\pi(SIN-1)\le EIN\le\pi(SIN)-1$;

(3) $def(ie,EIN,SIN)$.表示内部事件实例$ie\in E_{EIN}$且$\pi(SIN-1)\le EIN\le\pi(SIN)-1$;

(4) $join(se,EIN,SIN)$.表示服务事件实例$se\in SE_{SIN}$且$\pi(SIN)=EIN$.

在以上4个谓词中,变量EIN和SIN分别记录事件段和服务段在各自序列中的序号.下面称在answer set程序中的规则为NLP规则.设事件实例$event=e_1\,\&\,...\,\&\,e_i\,\&\,!e_k\,...\,\&\,!e_n$,条件$condition=p_1,...,p_m$,其中,$1\le i\le k\le n$,$0\le m$.简洁起见,这里假设$event$中只包含外部事件实例.可按式(32)~式(34)描述的方法将GCS中

的 SELS 规则实例转换为相应的 NLP 规则.

event causes *a* if *condition*:

$$exec(a,EIN,SIN)\leftarrow occ(e_1,EIN_1,SIN_1),...,occ(e_i,EIN_i,SIN_i),p_1,...,p_m,\\ \text{not } occ(e_k,EIN_k,SIN_k),...,\text{not } occ(e_n,EIN_n,SIN_n),\\ ein(EIN_1),...,ein(EI_i),sin(SI_1),...,sin(SIN_i),\max\_ein(EIN),\max\_sin(SIN) \tag{32}$$

event triggers *ie* if *condition*:

$$def(ie,EIN+1,SIN)\leftarrow occ(e_1,EIN_1,SIN_1),...,occ(e_i,EIN_i,SIN_i),p_1,...,p_m,\\ \text{not } occ(e_k,EIN_k,SIN_k),...,\text{not } occ(e_n,EIN_n,SIN_n),\\ ein(EIN_1),...,ein(EIN_i),sin(SIN_1),...,sin(SIN_i),\max\_ein(EIN),\max\_sin(SIN) \tag{33}$$

event attaches *se* if *condition*:

$$join(se,EIN+1,SIN+1)\leftarrow occ(e_1,EIN_1,SIN_1),...,occ(e_i,EIN_i,SIN_i),p_1,...,p_m,\\ \text{not } occ(e_k,EIN_k,SIN_k),...,\text{not } occ(e_n,EIN_n,SIN_n),\\ ein(EIN_1),...,ein(EIN_i),sin(SIN_1),...,sin(SIN_i),\max\_ein(EIN),\max\_sin(SIN) \tag{34}$$

在本文中,用not操作符表示事件负文字!*e*.当然也可采用逻辑否定符号¬表示,但是需要增加表示CWA的NLP规则 $\neg b\leftarrow \text{not } b$ (b 为任意谓词).在CWA前提下,以上两种表示逻辑意义等价,且前者可减少用于描述answer set程序域的规则数量.如果 *event* 中的任意 e_l($1\le l\le i$)为内部(服务)事件实例,则只需要用 $def(e_l,EIN_l,SIN_l)$ ($join(e_l,EIN_l,SIN_l)$)取代以上 3 个规则体(body)[14,18]中的 $occ(e_l,EIN_l,SIN_l)$ 即可.

在以上 3 个NLP规则中,域谓词*ein*和*sin*分别表示事件段和服务段的序号,谓词 $\max\_ein(EIN)$ 和 $\max\_sin(SIN)$ 表示 *EIN* 和 *SIN* 分别为相应规则正体(positive body)[14,18]中事件段和服务段序号的最大值.除需要对s_1;*s_2的Proc模块产生的服务段序列作一次置换处理以外,*GCS*中的规则均按以上方法表示.以下两个NLP规则表示式(15)和式(16),从中可以看到在服务事件序列上将 se_s_1 和 se_s_2 作了一次互换处理,以反映它们的组合顺序.

$$def(proc\_s_2,EIN+1,SIN+1)\leftarrow def(proc\_s_1;*s_2,EIN,SIN),ein(EIN),sin(SIN),\max\_ein(EIN),\max\_sin(SIN)$$

$$def(proc\_s_1(id\_s_2),EIN+1,SIN-2)\\ \leftarrow join(se\_s_2(id\_s_2),EIN,SIN),ein(EIN),sin(SIN),\max\_ein(EIN),\max\_sin(SIN)$$

最后,为完成推理,还需要一个关于 *goal* 的约束(constraint)[14] 或完整约束(integrity constraint)[18],该约束由式(35)来描述:

$$\leftarrow \text{not } join(goal,EIN,SIN),ein(EIN),sin(SIN) \tag{35}$$

将由以上方法得到的answer set程序记为 $\prod(GCS)$.由定义 3、*CS*中SELS规则、NLP规则以及约束的建立情况可知,在 $\prod(GCS)$ 的任意规则中,条件、头(head)[14,18]和负体(negative body)[14,18]中的变量始终都出现在正体中,所以该程序满足安全性要求[18].将完成 *occ* 表示的*XE*(*CS*)记为 $OCC(XE(CS))$,它构成 $\prod(GCS)$ 的论据(fact)库.另外,将 $\prod(GCS)$ 中表示基本(复杂)服务Proc模块的NLP规则集合记为 $\prod(Proc)$.

事实上,以上方法也适合*CS*的转换以形成 $\prod(CS)$,但这会破坏ASP的一阶性要求[18]或者导致未被填充函数的存在,不过将 $\prod(GCS)$ 看成是由*CS*填充 $\prod(CS)$ 中事件和行动形成的更为合适.

引理 1. $\prod(GCS)$ 满足局部 stratification.

证明:表示式(21)~式(23)的 NLP 规则显然不会导致 $\prod(GCS)$ 依赖图(dependency graph)中出现负圈的情况,所以 $\prod(GCS)$ 满足 stratification.再由文献[22]中引理 6.12 可得 $\prod(GCS)$ 满足局部 stratification.关于逻辑程序依赖图的概念见文献[22]. □

定理 1. 设存在一个组合问题 $\langle CS,ixe,goal\rangle$,*CS* 的语义为 $\pi_{CS}: CS\times XE(CS)\to CTset(CS)$,则 $CTset(CS)\neq\varnothing$ 当且仅当 $\prod(GCS)\cup OCC(XE(CS))$ 拥有唯一的 answer set *AS*.

详细证明过程见附录.

在定理 1 中,设 $ct: S_1,...,S_n,goal$ 为$CTset(CS)$中的任意一个组合轨迹,如果对于该轨迹中的任意服务事件实例集 $S_i=\{se_i^1,...,se_i^{n_i}\}$ 且 $join(se_i^j,ei_i^j,i)\in AS$,其中,$1\le i\le n,1\le j\le n_i$,则称*AS*能够表示*ct*.

引理 2. 设存在一个组合问题 $\langle CS, ixe, goal\rangle$,CS 的语义为 $\pi_{CS}: CS \times XE(CS) \to CTset(CS)$. 如果 $\prod(GCS) \cup OCC(XE(CS))$ 拥有 answer set AS,则 $CTset(CS)$中的任意组合轨迹都能由 AS 表示.

证明方法与对定理 1 的证明类似.此处略. □

定理 1 和引理 2 形式地说明了 CS 与 $\prod(GCS) \cup OCC(XE(CS))$ 在语义上的等价性.

4 互斥约束

上文只是实现了对组合轨迹集 $CTset(CS)$的表示,并没有完成对组合轨迹的表示.存在以下两个原因造成了这种缺陷,其一为未能有效地区分多 PSS 情况,其二为未能实现对 OR 组合关系的表示.为此,首先将能够达到相同效果或者由于时间(或空间)的重叠而不能组合在一起的服务定义为客观互斥服务,称此种关系为客观互斥关系.比如,在以上旅游服务中,$fb(a,b)$,$tb(a,b)$和 if-!fb-then-$tb(a,b)$ 相互之间就是客观互斥的,因为它们都完成从 a 地到 b 地的相同功能.又如 b 地的两个时间相互重叠的旅游项目.相互客观互斥的服务要么 PSS 相同要么都为初始服务,不然会导致组合的混乱.除初始服务外,相互客观互斥的服务虽然是通过 OR 组合关系描述的,但是由于后者还需要反映用户主观随意的需求,所以其中的后序服务并不一定满足以上客观互斥服务定义要求,定义这种为满足用户主观随意需求而不能组合在一起的服务为主观互斥服务,称此种关系为主观互斥关系,统称以上两种关系为互斥关系.互斥关系显然满足传递性和对称性,同一个 PSS 中也不可能包含相互互斥的服务.

命题 1. 互斥关系构成了多 PSS 情况和 OR 组合关系存在的必要条件.

对于主观互斥关系而言,命题 1 显然成立.在排除主观互斥关系的情况下,对于客观互斥关系,可以作如下简单的反证推理说明:如果不存在客观互斥服务,必定不存在 OR 组合关系.更为重要的是,由于不存在服务组合实现方式的多样性,从而多 PSS 情况就不可能发生.另外,经验也可以帮助说明以上命题的合理性.例如,在以上旅游服务中,预定 b 地宾馆房间服务 $hb(b)$ 的 PSS 显然与以上多个客观互斥服务存在函数关系,其中,宾馆房间预定服务由 hb (hotel booking)表示.由以上分析可得 PSS 的如下性质:

命题 2. 如果 PSS_1 和 PSS_2 是服务 s 的任意两个 PSS,则在 PSS_1/PSS_2 中至少存在一个服务与 PSS_2/PSS_1 中的一个服务互斥,且在 PSS_1 和 PSS_2 中不存在其他服务与它们互斥(唯一性).唯一性是由互斥关系的传递性、对称性和同一个 PSS 中不可能包含相互互斥的服务共同决定的.

为表示服务互斥关系,对组合方案 CS 需要添加形如式(36)的互斥约束:

$$\text{choice} \quad proc\_s_1, proc\_s_2 \quad \text{if} \quad condition \tag{36}$$

$$\text{choice} \quad proc\_fb(OP_1, DP_1), proc\_tb(OP_2, DP_2) \quad \text{if} \quad OP_1 = OP_2, DP_1 = DP_2 \tag{37}$$

式(36)表示在满足条件$condition$的情况下,由Proc事件表示的服务s_1和s_2是相互互斥的,所以它们或它们子服务的服务事件实例不能同时存在于一个的组合轨迹中.一条互斥约束只表示两个服务之间的互斥关系,可根据互斥关系的传递性和对称性推导出多个服务之间的互斥关系.式(37)表示了fb和tb之间的互斥关系,该式中变元OP_i和DP_i($1 \le i \le 2$)分别表示出发地点和到达地点,所以在以上旅游服务中,表示由b地返回的$fb(b,a)$和$tb(b,a)$也是相互互斥的.

设 MECS(mutually exclusive constraints set)为 CS 的所有互斥约束构成的集合,则组合问题 $\langle CS, ixe, goal\rangle$ 扩展定义为 $\langle CS, ixe, goal, MECS\rangle$.

定义 7. 如果 MECS 能够识别 CS 中的所有互斥关系,则称 MECS 相对 CS 是完备的.同时,如果 CS 形成的任何组合服务实例都满足 MECS 的要求,则称 CS 满足 MECS.

Table 1 MECST algorithm

表 1 MECST 算法

1.	function MECST ($\prod(GCS)$,*MECS*, $DG(\Pi(GCS)\setminus\prod(Proc))$): answer set program $\coprod(GCS)$
2.	var $\coprod(GCS)$: null; *R*: NLP rule set; *A*: predicate set; *a*: predicate;
	begin
3.	while $DG(\prod(GCS)\setminus\prod(Proc))\neq\varnothing$ do
4.	if $r\in\prod(GCS)\setminus\prod(Proc)$ and the indegree of any predicate in is zero then note: $B(r)$is the body of *r*
5.	for each predicate $a\in B(r)$ do to delete these nodes representing *a* in $DG(\prod(GCS)\setminus\prod(Proc))$;
6.	*R*=ruleWithSameBody($\prod(GCS)$,$B(r)$);
7.	*A*=ruleHeadSet(*R*);
8.	transformation($B(r)$,*A*,*MECS*, $\coprod(GCS)$);
9.	end_if
10.	end_while
11.	copy $\prod(GCS)$ and formula(35) to $\coprod(GCS)$;
12.	return $\coprod(GCS)$;
	end_function
13.	procedure transformation ($B(r)$,*A*,*MECS*, $\coprod(GCS)$)
14.	var *C*: predicate set; *E*: null predicate set;
	begin
15.	for *C* = max*MECSet*(*A*,MECS) and $C\neq\varnothing$ do
16.	$1\le\{C\}\le 1\leftarrow B(r)$ is inserted into $\coprod(GCS)$;
17.	$E=E\cup C$;
18.	end_for
19.	for any $a\in A-E$ do $a\leftarrow B(r)$ is inserted into $\coprod(GCS)$;
	end_procedure

Note:
1. Function *ruleWithSameBody*($\prod(GCS)$,$B(r)$): the rule set in which the body of any rule in $\prod(GCS)$ is $B(r)$;
2. Function *ruleHeadSet*(*R*): the heads of all rules in rule set *R*;
3. Function *maxMECSet*(*A*,*MECS*): a mutually exclusive constraints set in *A* according to MECS;
4. Procedure transformation ($B(r)$,*A*,*MECS*, $\coprod(GCS)$) accomplishes to satisfy the exclusive constraint and insert these NLP rules into $\coprod(GCS)$.

显然,仅依靠实现 *CS* 的软件语言,如C++或者Java,要满足 *MECS* 是不现实的,但是利用ASP技术却可方便实现对它的满足.为此,提出了由表 1 描述的MECST(mutually exclusive constraints satisfaction transformation)算法.该算法利用 $\prod(GCS)\setminus\prod(Proc)$ 的依赖图 $DG(\prod(GCS)\setminus\prod(Proc))$ 和基数约束规则(cardinality constraint rule)[17,23],将 $\prod(GCS)$ 转化为能够满足 *MECS* 的 $\coprod(GCS)$.由于基本和复杂服务的Proc模块内始终满足 *MECS*,所以只需要对 $\prod(GCS)$ 中表示组合模块的部分进行转换.需要说明的是:在MECST算法中形成的基数约束规则(即第 16 步中的 $1\le\{C\}\le 1\leftarrow B(r)$)等价于析取规则,采用前者的原因是为了方便以后的扩展工作.从表 1 可知,MECST算法始终是从以 $occ(ixe,0,0)$ 为体的NLP规则开始执行的,不断地在 $\coprod(GCS)$ 中构造满足MECS的基数约束规则直至 $DG(\prod(GCS)\setminus\prod(Proc))$ 为空.

定理 2. 存在一个组合问题 $\langle CS,ixe,goal,MECS\rangle$, *CS* 的语义为 $\pi_{CS}: CS\times XE(CS)\to CTset(CS)$.如果 MECS 相对 *CS* 是完备的,则对于 $CTset(CS)$ 中的任意一个组合轨迹, $\coprod(GCS)\cup OCC(XE(CS))$ 都拥有唯一的一个 answer set 表示该轨迹.

证明:令 $\Omega=(\coprod(GCS)\setminus F(35))\cup OCC(XE(CS))$,其中 $F(35)$ 是式(35).设 *CS* 中存在 k ($2\le k$)个相互互斥的服务 $s_1,...,s_k$,由上文可知,它们的 PSS 相同,所以它们的 LJS 相同.利用归纳法证明:(1) $s_1,...,s_k$ 为初始服务(即它们的 LJS 为 1)时,由 MECS 的完备性,MECST 算法以及文献[14]中命题 2.1 对析取规则语义性质的定义,可得表示它们的 Proc 事件实例的 *def* 谓词分别在 Ω 的 k 个不同 answer sets 中;(2) 假设当 $s_1,...,s_k$ 的 LJS$\le n$ ($2\le n$)时,表示它们 Proc 事件实例的谓词分别在 Ω 的互不相同的 answer set 中;(3) 当 $s_1,...,s_k$ 的 LJS 为 $n+1$,如果 $s_1,...,s_k$ 只有一个 PSS,与(1)同理分析,可得与其相同的结论.如果它们的 PSS 数大于 1,由命题 2,它们的任意两个 PSS 中至少存在两个相互互斥的不同服务,而这两个互斥服务的 LJS 一定小于 $n+1$,所以由(2)得 $s_1,...,s_k$ 不同的 PSS 实例应由 Ω 互不相同的 answer set 表示,与(1)同理分析可得,表示 $s_1,...,s_k$ 的 Proc 事件实例的谓词分别在 Ω 的互不

相同的 answer set 中.由命题 1 可知,结合服务的 Proc 事件和服务事件之间的关系,再根据定义 5、定理 1 和引理 2,最后得证. □

由定理 2 可知, $\coprod(GCS)\cup OCC(XE(CS))$ 实现了对组合轨迹的有效表示.最后需要说明的是,如果在所有相互互斥的服务上用户能够形成严格全序偏好,则可利用权约束规则(weight constraint rule)[17,23]实现组合轨迹优化,从而获得满足用户最优偏好的唯一组合轨迹.

5 示范举例

为解决文中旅游服务组合问题 $\langle CS\_Travel, travel, goal, MECS\rangle$,在 Linux(Red Hat Linux 3.2.2-5)上,利用 gcc3.2.2 建立了 *CS_Travel* 运行的虚拟环境.存在以下 3 类可被用于 *CS_Travel* 中的基本服务,一是属于交通的服务 *fb*(机票订购)、*tb*(火车票订购)和 *txb*(出租车预定);二是属于住宿的服务 *hb*(预定宾馆房间);最后是 *b* 地两个相互互斥的旅游项目服务 *tim*1 和 *tim*2 ,它们之间的互斥关系可能是由时间重叠导致的,也可能是由为满足用户主观随意需求导致的.另外,还存在通过构造子建立的以下 3 个复杂服务:表示如果未能预订到机票就预订火车票的 if-!*fb*-then-*tb* 、表示先预订机票再预定出租车的 *fb*;*txb* 和表示先预定出租车再预订机票的 *txb*;**fb*.*Proc* 事件及服务事件分别具有以下形式: $proc(OP, DP, PSS\_ID_1, PSS\_ID_2)$, $se(OP, DP, ID)$.其中变元 *OP* 和 *DP* 分别为出发地和到达地, PSS_ID_1 和 PSS_ID_2 为两个 PSS_ID 变元,*ID* 为 ID 变元.为简洁起见,忽略那些只在 Proc 模块中传递值的变元,以及 *fb*;*txb* 和 *txb*;**fb* 中分别表示出租车起始和终止地点的变元.分以下 5 步完成 *CS_Travel* 组合模块的描述:

Step 1: 4 个表示由 *a* 到 *b* 的服务 $fb(a,b)$, $tb(a,b)$, if-!*fb*-then-$tb(a,b)$ 和 *txb*;* $fb(a,b)$ 为 *CS_Travel* 的初始服务,所以可用如下形式的 4 个 SELS 规则表示.注意,表示其他 3 个服务时,用相应服务的 *Proc* 事件取代 $proc\_fb(a,b,-1,-1)$ 即可.

$travel(a,b)$ triggers $proc\_fb(a,b,-1,-1)$

Step 2: 每个旅游项目服务 $tim1(-1,b)$ 和 $tim2(-1,b)$ 都拥有 4 个同样的 PSS, PSS 元素分别为以上 4 个初始服务.由于 if-!*fb*-then-$tb(a,b)$ 和 *txb*;* $fb(a,b)$ 为复杂服务,根据前文,所以对于 $tim1(-1,b)$ 与它的 PSS 可用如下形式的 3 个规则表示,对 $tim2(-1,b)$ 与此类似.其中,*vs* 为 *txb*;* $fb(a,b)$ 的虚拟服务.

$se\_fb(a,b,ID\_fb)$ triggers $proc\_tim1(a,b,ID\_fb,-1)$

$se\_tb(a,b,ID\_tb)$ triggers $proc\_tim1(a,b,ID\_tb,-1)$

$se\_vs(a,b,ID\_fb)$ triggers $proc\_tim1(a,b,ID\_fb,-1)$

Step 3: 宾馆房间预订服务 $hb(-1,b)$ 拥有的 8 个 PSS,每个 PSS 拥有两个元素,一个为以上任意初始服务,另一个为任意旅游项目服务.与步骤 2 同理,当 $tim1(-1,b)$ 为 PSS 元素时,可用如下形式的 3 个 SELS 规则表示,对 $tim2(-1,b)$ 与此类似.

$se\_fb(a,b,ID\_fb)\,\&\,se\_tim1(a,b,ID\_tim1)$ triggers $proc\_hb(a,b,ID\_fb,ID\_tim1)$

$se\_tb(a,b,ID\_tb)\,\&\,se\_tim1(a,b,ID\_tim1)$ triggers $proc\_hb(a,b,ID\_fb,ID\_tim1)$

$se\_vs(a,b,ID\_fb)\,\&\,se\_tim1(a,b,ID\_tim1)$ triggers $proc\_hb(a,b,ID\_fb,ID\_tim1)$

Step 4: 表示回程的 4 个服务 $fb(b,a)$, $tb(b,a)$, if-!*fb*-then-$tb(b,a)$ 和 $fb;txb(b,a)$ 的 PSS 元素都为 $hb(-1,b)$,用如下形式的 4 个规则表示.注:表示其他 3 个服务时,用相应服务的 Proc 事件取代 $proc\_fb(b,a,ID\_hb,-1)$ 即可.

$se\_hb(a,b,ID\_hb)$ triggers $proc\_fb(b,a,ID\_hb,-1)$

Step 5: 目标服务 *goal* 存在 4 个 PSS,它们的 PSS 元素分别为步骤 4 中 4 个表示回程的服务,即它们为终止服务.与步骤 2 同理,可用以下 3 个规则表示:

$se\_fb(b,a,ID\_fb)$ attaches *goal*

$se\_tb(b,a,ID\_tb)$ attaches *goal*

$se\_txb(-1,a,ID\_txb)$ attaches *goal*

只需 23 个 SELS 规则就完成了 *CS_Travel* 中组合模块的描述.限于篇幅,在此忽略 Proc 模块的描述.在 MECS 中,除存在如式(37)表示交通服务之间的 4 个互斥约束以外,还存在约束 choice $proc\_tim1(X,b)$, $proc\_tim2(Y,b)$ 以表示 *tim*1 和 *tim*2 之间的互斥关系.

根据 *CS_Travel* 运行产生的实例,利用Lparse 1.0.17[24]和Smodels 2.32[24]实现了 *CS_Travel* 的answer set程序编码工作,并根据MECST算法,形成了满足MECS的answer set程序.该程序最多可拥有 32 个answer sets,这与 *CS_Travel* 语义是相符合的.例如,在这个answer set程序中表示以上步骤 1 中SELS规则且满足MECS的NLP规则为

$$1\{def(proc\_fb(a,b,-1,-1),EIN+1,SIN),def(proc\_tb(a,b,-1,-1),EIN+1,SIN),$$
$$def(proc\_IfNotFbThenTb(a,b,-1,-1),EIN+1,SIN),def(proc\_TxbElsFb(a,b,-1,-1),EIN+1,SIN)\}1$$
$$:-\ occ(travel(a,b),EIN,SIN),ein(EIN),sin(SIN),\max\_ein(EIN),\max\_sin(SIN).$$

虽然组合方案*CS*的运行、*CS*的answer set编码和MECST算法都是多项式时间复杂度,但是根据文献[14]中的表 2 以及文献[17]中的定理 3.3,可知为一个析取逻辑程序计算answer set是NP完备的,因而本文模型属于NP完备问题.然而,一是由于理论的计算复杂性始终考虑的是最坏情况;二是由于*CS*的answer set程序的规模不会太大,所以answer set的计算困难并不妨碍本文的应用价值.成熟的answer set推理系统和已被广为接受的析取Datalog [25]都能够说明以上观点的合理性.如果去掉多PSS情况和OR组合关系,根据文献[14]中的表 2,由于此时*CS*的answer set程序不含析取规则且满足stratification,所以计算answer set是多项式时间复杂度,因而模型也是多项式时间复杂度,但这是以降低服务组合能力为代价的.

6 结束语

本文依次完成了以下 4 个方面的工作:(1) SELS 的定义;(2) 组合方案 *CS* 的构造;(3) *CS* 的语义定义及其 ASP 表示;(4) 组合轨迹的有效表示.从上文也可得到,采用 SELS 构造的 *CS* 具有如下 5 个优点:(1) 语法简单;(2) 语义明确;(3) 良好的感知能力.通过定义行动和外部事件,解决了外部信息表示的困难;(4) 表示的丰富性.不仅具有描述多种组合关系的能力,而且还完成了组合实现方式多样性的表示;(5) 良好的结构性.无论是用基本服务表示复杂服务,还是用它们表示组合服务,它们的 Proc 模块都具有良好的封装性.而且,基本或复杂服务的 Proc 模块均可被应用到不同的组合方案中.

对于服务组合的完成,*CS*与其ASP表示是缺一不可.两者之间的关系可以理解为,通过为后者获得海布兰空间(Herbrand universe)[26],前者帮助后者形成了它的海布兰库(Herbrand base)[26],而后者为前者提供了更为丰富的知识表示与推理能力.如果没有ASP表示,*CS*至少会丧失满足用户多样化需求的能力(即不再允许多PSS情况和OR组合关系).如果去掉*CS*而直接利用ASP表示组合服务,比如,可采用answer set规划[6]完成对服务组合的描述,但是会因为无法通过流(fluent)命题感知到组合过程中必要的外部信息,而无法成功.

在实际应用中,一方面,*CS*的成功还有赖于可靠的系统支持,如服务上下文系统[27];另一方面,还需要可靠的工业协议支持.

References:

[1] Mcilraith S, Son TC, Zeng HL. Semantic Web services. IEEE Intelligent Systems, Special Issue on the Semantic Web, 2001, 46−53.

[2] Sirin E, Parsia B, Wu D, Hendler J, Nau D. HTN planning for Web service composition using SHOP2. Web Semantic: Science, Services and Agents on the World Wide Web, 2004,1(4):377−396.

[3] Maamar Z, Benatallah B, Mansoor W. Service chart diagrams-description & application. In: Proc. of the 12th Int'l World Wide Web Conf. (WWW 2003). Budapest: ACM, 2003.

[4] Hu HT, Li G, Han YB. An approach to business-user-oriented larger-granularity service composition. Chinese Journal of Computers, 2005,28(4):694−703 (in Chinese with English abstract).

[5] Gelfond M, Lifschitz V. Action languages. Electronic Trans. on Artificial Intelligence, 1998,2(3-4):193−210.

[6] Son TC, Baral C, Tran N, Mcilraith S. Domain-Dependent knowledge in answer set planning. ACM Trans. on Computational Logic, 2006,7(4):613−657.

[7] Qiu L, Shi Z, Lin F. Context optimization of AI planning for services composition. In: Proc. of the IEEE Int'l Conf. on e-Business Engineering (ICEBE 2006). IEEE, 2006. 610−617.

[8] Guo YB, Du YY, Xi JQ. A CP-net model and operation properties for Web service composition. Chinese Journal of Computers, 2006,29(7):1067−1075 (in Chinese with English abstract).

[9] Deng SG, Wu J, Li Y, Wu ZH. Automatic Web service composition based on backward tree. Journal of Software, 2007,18(8):1896−1910 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1896.htm

[10] Antoniou G, Baldoni M, Baroglio C, Patti V, Baumgartner R, Eiter T, Herzog M, Schindlauer R, Tompits H, Bry F, Schaffert S, Henze N, May W. Reasoning methods for personalization on the semantic Web. Annals of Mathematics, Computing & Teleinformatics, 2004,1(2):1−24.

[11] Lobo L, Bhatia R, Naqvi S. A policy description language. In: Proc. of the 16th National Conf. on Artificial Intelligence (AAAI-99). AAAI Press, 1999. 291−298.

[12] Chomicki J, Lobo J, Naqvi S. Conflict resolution using logic programming. IEEE Trans. on Knowledge and Data Engineering, 2003,15(1):244−249.

[13] Gelfond M, Lifschitz V. The stable model semantics for logic programming. In: Kowalski R, Bowen K, eds. Proc. of the 5th Int'l Conf. and Symp. on Logic Programming. MIT Press, 1988. 1070−1080.

[14] Gelfond M, Leone N. Logic programming and knowledge representation——The A-Prolog perspective. Artificial Intelligence, 2002,138:3−38.

[15] Grosof B, Horrocks I, Volz R, Decker S. Description logic programs: Combining logic programs with description logic. In: Proc. of the 12th Int'l World Wide Web Conf. ACM, 2003. 48−57.

[16] Eiter T, Lukasiewicz T, Schindlauer R, Tompits H. Combining answer set programming with description logics for the semantic Web. INFSYS RR-1843-03-13: Vienna/Austria, Institut für Informationssysteme, Technische Universität Wien, 2003. http://www.kr.tuwien.ac.at/research/reports/index.html

[17] Simons P, Niemelä I, Soininet T. Extending and implementing the stable model semantics. Artificial Intelligence, 2002,138: 181−234.

[18] Leone N, Pfeifer G, Faber W, Eiter T, Gottlob G, Perri S, Scarcello F. The DLV system for knowledge representation and reasoning. ACM Trans. on Computational Logic, 2006,7(3):499−562.

[19] Matin D, Burstein M, Hobbs J, Lassila O, McDermott D, McIlraith S, Narayanan S, Paolucci M, Parsia B, Payne T, Sirin E, Srinivasan N, Sycara K. OWL-S: Semantic markup for Web services. W3C Member Submission, 2004. http://www.w3.org/Submission/OWL-S/

[20] Burgard W, Cremers A, Fox D, Hahnel D, Lakemeyer G, Schulz D, Steiner W, Thrun S. The interactive museum tour-quide robot. In: Proc. of the 15th National Conf. on Artificial Intelligence (AAAI-98). Philadelphia: AAAI Press, 1998. 11−18.

[21] Manola F, Miller E. RDF Primer. W3C recommendation, 2004. http://www.w3.org/TR/rdf-primer/

[22] Apt K, Bol R. Logic programming and negation: A survey. CS-R9402: Netherlands, CWI (Centrum Wiskunde & Informatica), 1994. http://db.cwi.nl/rapporten/index.php?jaar=1994&dept=11

[23] Niemelä I, Simons P, Soininet T. Stable model semantics of weight constraint rules. In: Proc. of the 5th Int'l Conf. on Logic Programming and Nonmonotonic Reasoning. Springer-Verlag, 1999. 317−331.

[24] http://www.tcs.hut.fi/Software/smodels/

[25] Eiter T, Gottlob G, Mannila H. Disjunctive datalog. ACM Trans. on Database Systems, 1997,22(3):364−418.

[26] Nilsson U, Maluszynski J. Logic, Programming and Prolog. 2th ed., John Wiley & Sons Ltd., 1995. 24−25. http://www.ida.liu.se/~ulfni/lpp

[27] Maamar Z, Benslimane D, Thiran P, Ghedira C, Dustdar S, Sattanathan S. Towards a context-based multi-type policy approach for Web services composition. Data & Knowledge Engineering, 2007,62:327−351.

附中文参考文献:

[4] 胡海涛,李刚,韩燕波.一种面向业务用户的大粒度服务组合方法.计算机学报,2005,28(4):694−703.

[8] 郭玉彬,杜玉越,奚建清.Web 服务组合的有色网模型及运算性质.计算机学报,2006,29(7):1067−1075.

[9] 邓水光,吴健,李莹,吴朝晖.基于回溯树的Web服务自动组合.软件学报,2007,18(8):1896−1910.http://www.jos.org.cn/1000-9825/18/1896.htm

附录

定理 1 证明:设 $\Omega=\prod(GCS)\cup OCC(XE(CS))$,由引理 1 得 $\prod(GCS)$ 满足局部 stratification,根据文献[14]中的定理 2.1,所以 $\Omega\setminus F(35)$ 拥有唯一的 answer set.其中, $F(35)$ 是式(35).由于易于完成变量 *EIN* 和 *SIN* 的填充,以下证明中认为 $\prod(GCS)$ 已完成填充.

充分性($\Rightarrow$). 利用反证法证明.假设 $CTset(CS)\neq\varnothing$ 且 Ω不拥有answer set.由于 $CTset(CS)\neq\varnothing$,则在 $CTset(CS)$ 中至少存在一个组合轨迹 $ct:S_1,...,S_n,goal$.由此 *goal* 存在一个实例为 $\{se_1,...,se_m\}$ 的 PSS_1,其中 $se_i\in S_w$, $1\le i\le m$, $1\le w\le n$,且 $S_n\subseteq\{se_1,...,se_m\}$.根据式(31),式(A1)为 $\prod(GCS)$ 中表示 *goal* 和 PSS_1 实例关系的NLP规则.由于ID,DID,FID,CT变元值可以有效地表示服务Proc模块中的事件和行动实例(比如,两个相同服务实例都等待在同一个事件段中的外部事件 $echo\_Conv(ID,approval,DID)$,虽然它们的ID值相同,但是DID值却不同),同时为简洁起见,所以除复杂服务 if-*C*-then-s_1-else-s_2 和 while-*M*-*N*-do-s_1 的Proc事件中分别表示条件判断和循环步长的变元外,不考虑*CS*中的其他变元.另外,也不考虑SELS规则中的条件以及 $\prod(GCS)$ 中的无关谓词.

$$join(goal,ein+1,n+1)\leftarrow join(se_1,ein_1,sin_1),...,join(se_m,ein_m,sin_m),\max\_ein(ein),\max\_sin(n) \tag{A1}$$

在规则(A1)体中至少存在一个 *join* 谓词不在 $\Omega\setminus F(35)$ 的 answer set AS_1 中,否则 $join(goal,ein+1,n+1)\in AS_1$,$\Omega$就拥有answer set AS_1,与前提矛盾.假设 $join(se_k,ein_k,sin_k)\notin AS_1$ ($1\le k\le m$),如果 se_k 不为虚拟服务事件实例,se_k为基本服务 s_k 的服务事件实例,设任意两个基本服务 s_x 和 s_z.根据 se_k 被添加到 *ct* 的方式,分以下 6 种情况讨论.

情况 1. 如果 $s_k\in PSS_1$.根据式(4)~式(10),式(A2)~式(A8)为 $\prod(GCS)$ 中表示 s_k 的 Proc 模块的 NLP 规则.

$$exec(getNaDID\_s_k,ein_k-4,sin_k-1)\leftarrow def(proc\_s_k,ein_k-4,sin_k-1) \tag{A2}$$

$$def(search\_s_k,ein_k-3,sin_k-1)\leftarrow def(proc\_s_k,ein_k-4,sin_k-1) \tag{A3}$$

$$exec(ac\_Search,ein_k-3,sin_k-1)\leftarrow def(search\_s_k,ein_k-3,sin_k-1),\ occ(echo\_GND\_s_k,ein_k-3,sin_k-1) \tag{A4}$$

$$def(conversation\_s_k,ein_k-2,sin_k-1)\leftarrow def(search\_s_k,ein_k-3,sin_k-1),\ occ(echo\_GND\_s_k,ein_k-3,sin_k-1) \tag{A5}$$

$$exec(ac\_Converse,ein_k-2,sin_k-1)\leftarrow def(conversation\_s_k,ein_k-2,sin_k-1),\ occ(echo\_Search,ein_k-2,sin_k-1) \tag{A6}$$

$$def(anwser\_s_k,ein_k-1,sin_k-1)\leftarrow def(conversation\_s_k,ein_k-2,sin_k-1),\ occ(echo\_Search,ein_k-2,sin_k-1) \tag{A7}$$

$$join(se_k,ein_k,sin_k)\leftarrow def(anwser\_s_k,ein_k-1,sin_k-1),occ(echo\_Conv,ein_k-1,sin_k-1) \tag{A8}$$

假设 $def(proc\_s_k,ein_k-4,sin_k-1)\in AS_1$.由以上 7 个 NLP 规则的逻辑关系,得 $join(se_k,ein_k,sin_k)\notin AS_1$ 的原因只可能是 $occ(echo\_Conv,ein_k-1,sin_k-1)\notin OCC(XE(CS))$,或 $occ(echo\_Search,ein_k-2,sin_k-1)\notin OCC(XE(CS))$,或 $occ(echo\_GND\_s_k,ein_k-3,sin_k-1)\notin OCC(XE(CS))$.从而 $echo\_Conv\notin XE_{sin_k-1,ein_k-1}(CS)$,或 $echo\_Search\notin XE_{sin_k-1,ein_k-2}(CS)$,或 $echo\_GND\_s_k\notin XE_{sin_k-1,ein_k-3}(CS)$,根据 *CS* 的语义定义, $ct\notin CTset(CS)$,与前提矛盾,所以 $def(proc\_s_k,ein_k-4,sin_k-1)\notin AS_1$.

情况 2. 如果 if-!s_x-then-$s_k\in PSS_1$ 或 if-!s_k-then-$s_x\in PSS_1$.

情况 2.1. 当 if-!s_x-then-$s_k \in PSS_1$ 时,根据式(20)~式(23),式(A9)~式(A12)为 $\prod(GCS)$ 中表示 if-!s_x-then-s_k 的 Proc 模块的 NLP 规则.

$$def(proc\_s_x, EIN+1, sin_k-1) \leftarrow def(proc\_\text{if-}!s_x\text{-then-}s_k, EIN, sin_k-1) \tag{A9}$$

$$def(proc\_s_k, EIN+3, sin_k-1) \leftarrow def(search\_s_x, EIN+2, sin_k-1), not\ occ(echo\_GND\_s_x, EIN+2, sin_k-1) \tag{A10}$$

$$def(proc\_s_k, EIN+4, sin_k-1) \leftarrow def(conversation\_s_x, EIN+3, sin_k-1), not\ occ(echo\_Search, EIN+3, sin_k-1) \tag{A11}$$

$$def(proc\_s_k, EIN+5, sin_k-1) \leftarrow def(answer\_s_x, EIN+4, sin_k-1), not\ occ(echo\_Conv, EIN+4, sin_k-1) \tag{A12}$$

为匹配 ein_k 值,当 (A10) 使得 $def(proc\_s_k, ein_k-4, sin_k-1) \in AS_1$ 时, $EIN = ein_k - 7$,当 (A11) 使得 $def(proc\_s_k, ein_k-4, sin_k-1) \in AS_1$ 时, $EI=ein_k-8$,当(A12)使得 $def(proc\_s_k, ein_k-4, sin_k-1) \in AS_1$ 时,$EI=ein_k-9$. 情况 1 说明导致 $join(se_k, ein_k, sin_k) \notin AS_1$ 的情况只能是 $def(proc\_s_k, ein_k-4, sin_k-1) \notin AS_1$. 假设 $def(proc\_\text{if-}!s_x\text{-then-}s_k, EIN, sin_k-1) \in AS_1$,根据以上 4 个NLP规则的逻辑关系,在以上 3 种情况中,能够导致 $def(proc\_s_k, ein_k-4, sin_k-1) \notin AS_1$ 的情况只可能是

$$occ(echo\_Conv, EIN+4, sin_k-1) \in OCC(XE(CS)),$$

$$occ(echo\_Search, EIN+3, sin_k-1) \in OCC(XE(CS)),$$

$$occ(echo\_GND\_s_x, EIN+2, sin_k-1) \in OCC(XE(CS)).$$

根据规则(A2)~(A8),以上 3 种结果都得到 $join(se_x, EIN+5, sin_k) \in AS_1$,由于 if-!$s_x$-then-$s_k \in PSS_1$,则 $join(goal, ein+1, n+1) \in AS_1$,与前提矛盾,所以 $def(proc\_\text{if-}!s_x\text{-then-}s_k, EIN, sin_k-1) \notin AS_1$.

情况 2.2. 当 if-!s_k-then-$s_x \in PSS_1$ 时,将式(A9)到式(A12)中字符串后缀 s_k 和 s_x 互换,直接可得,导致 $def(proc\_s_k, ein_k-4, sin_k-1) \notin AS_1$ 的原因只能是 $def(proc\_\text{if-}!s_k\text{-then-}s_x, ein_k-5, sin_k-1) \notin AS_1$.

情况 3. 如果 if-C-then-s_x-else-$s_k \in PSS_1$ 或者 if-C-then-s_k-else-$s_x \in PSS_1$.

情况 3.1. 当 if-C-then-s_x-else-$s_k \in PSS_1$ 时,根据式(18)和式(19),式(A13)和式(A14)为表示 $\prod(GCS)$ 中 if-C-then-s_x-else-s_k 的 Proc 模块的 NLP 规则.

$$def(proc\_s_x, ein_k-4, sin_k-1) \leftarrow def(proc\_\text{if-}C\text{-then-}s_x\text{-else-}s_k(c, im), ein_k-5, sin_k-1), c(im) \tag{A13}$$

$$def(proc\_s_k, ein_k-4, sin_k-1) \leftarrow def(proc\_\text{if-}C\text{-then-}s_x\text{-else-}s_k(c, im), ein_k-5, sin_k-1), \neg c(im) \tag{A14}$$

假设 $def(proc\_\text{if-}C\text{-then-}s_x\text{-else-}s_k(im), ein_k-5, sin_k-1) \in AS_1$,则导致 $def(proc\_s_k, ein_k-4, sin_k-1) \notin AS_1$ 的原因是 $AS_1 \nvdash c(im)$,所以 $def(proc\_s_x, ein_k-4, sin_k-1) \in AS_1$,它可能导致 $join(se\_s_x, ein_k, sin_k)$ 在或不在 AS_1 中的两种情况. 由情况 1 和情况 2.1 的分析可得这都会导致矛盾. 所以只存在 $def(proc\_\text{if-}C\text{-then-}s_x\text{-else-}s_k(c, im), ein_k-5, sin_k-1) \notin AS_1$ 的可能.

情况 3.2. 当 if-C-then-s_k-else-$s_x \in PSS_1$ 时,与情况 3.1 同理可得,如果 $def(proc\_s_k, ein_k-4, sin_k-1) \notin AS_1$,则 $def(proc\_\text{if-}C\text{-then-}s_k\text{-else-}s_x(im), ein_k-5, sin_k-1) \notin AS_1$.

情况 4. 如果 $s_x;s_k \in PSS_1$.根据式(13)和式(14),式(A15)和式(A16)为 $\prod(GCS)$ 中表示 $s_x;s_k$ 的 Proc 模块的 NLP 规则.

$$def(proc\_s_x, ein_k-9, sin_k-2) \leftarrow def(proc\_s_x;s_k, ein_k-10, sin_k-2) \tag{A15}$$

$$def(proc\_s_k, ein_k-4, sin_k-1) \leftarrow join(se\_s_x, ein_k-5, sin_k-1) \tag{A16}$$

设 $def(proc\_s_x;s_k, ein_k-10, sin_k-2) \in AS_1$. 导致 $def(proc\_s_k, ein_k-4, sin_k-1) \notin AS_1$ 的原因为 $join(se\_s_x, ein_k-5, sin_k-1) \notin AS_1$,与情况 1 同理分析可得, $def(proc\_s_x, ein_k-9, sin_k-2) \notin AS_1$.所以得到 $def(proc\_s_x;s_k, ein_k-10, sin_k-2) \notin AS_1$.

情况 5. 如果 $s_x;{*}s_z \in PSS_1$, se_k 是虚拟服务事件.与情况 4 同理分析可得,如果 $def(proc\_s_k, ein_k-4, sin_k-1) \notin AS_1$,

则 $def(proc\_s_x;*s_z,ein_k-11,sin_k-3)\notin AS_1$.

情况 6. 如果 while-M-N-do-$s_x\in PSS_1$, se_k 为虚拟服务服务事件.只需对 s_x 的 Proc 模块,进行 $n-m+1$ 次类似情况 1 和情况 4 中的分析可得如果 $def(proc\_s_k,ein_k-4,sin_k-1)\notin AS_1$,则

$$def(proc\_\text{while-do-}s_x(m,n),ein_k-(n-m)\times 3-6,sin_k-(n-m+2))\notin AS_1.$$

通过以上情况的分析说明,导致 $join(se_k,ein_k,sin_k)\notin AS_1$ 的原因只可能是表示 s_k 或其父服务 Proc 事件实例的def谓词不在AS_1中.由CS组合模块的构造原则,对于情况 1,可得 $join(se_v,\ ein_k-5,sin_k-1)\notin AS_1$, $se_v\in S_{sin_k-1}$ 且服务s_v或其父服务是s_k的一个PSS元素;对于情况 2(或 3)~情况 6,也可分别得

$$join(se_v,ein_k-5,sin_k-1)\notin AS_1,$$
$$join(se_v,ein_k-11,sin_k-2)\notin AS_1,$$
$$join(se_v,ein_k-12,sin_k-2)\notin AS_1,$$
$$join(se_v,ein_k-(n-m)\times 3-7,sin_k-(n-m+3))\notin AS_1,$$

且 se_v 分别在ct中的第 sin_k-1, sin_k-2, sin_k-3, $sin_k-(n-m+3)$ 服务事件集中,同时服务s_v或其父服务为以上复杂服务的一个PSS元素.所以可根据以上方法不断回朔直到以 $occ(ixe,0,0)$ 为体的NLP规则,最终得到 $occ(ixe,0,0)\notin AS_1$,从而 $ixe\notin XE_{0,0}(CS)$,这显然与事实矛盾.所以 $CTset(CS)\neq\varnothing$,则Ω存在answer set AS,且 $AS=AS_1$.

必要性($\Leftarrow$). 利用反证法证明.设Ω存在answer set AS且 $CTset(CS)=\varnothing$,所以 $join(goal,ein,sin)\in AS$.则在 $\prod(GCS)$ 中至少存在一个的NLP规则r,该规则 $H(r)=\{join(goal,ein,sin)\}$ 同时 $B(r)=\{join(se_1,ein_1,sin_1),...,join(se_m,ein_m,sin_m)\}$ ($ein=\max\_ein(ein_1,...,ein_m)+1$, $sin=\max\_sin(sin_1,...,sin_m)+1$), 且 $join(se_1,ein_1,sin_1)\in AS\wedge...\wedge join(se_m,ein_m,sin_m)\in AS$,由此可知,$goal$存在一个$PSS_1$且其实例为 $\{se_1,...,se_m\}$.在CS中从任意 $se_k\in\{se_1,...,se_m\}$ 出发,采用与以上证明类似的方法,不断回溯到包含ixe的SELS规则,最后可得到 $occ(ixe,0,0)\notin AS$ 的矛盾. □

李鑫(1973－),男,湖北利川人,博士生,工程师,主要研究领域为知识工程,逻辑编程,Web 服务组合.

杨国纬(1938－),男,教授,博士生导师,主要研究领域为自然语言处理,计算机网络.

程渤(1975－),男,博士,讲师,主要研究领域为网络智能服务,分布式计算,工作流技术.

刘启和(1973－),男,博士,副教授,CCF 会员,主要研究领域为数据挖掘,图像处理.