FLUX 中负析取约束的研究与实现 *

刘一松[1], 朱会娟[1], 朱 芒[2], 徐艳群[3]
(1. 江苏大学 计算机科学与通信工程学院, 江苏 镇江 212013; 2. 镇江市排水管理处, 江苏 镇江 212000; 3. 南阳理工学院 计算机科学与技术系, 河南 南阳 473004)

摘 要: FLUX 是基于流演算的逻辑程序语言,实现 agents 在不完全状态下对其动作和感知信息进行逻辑推理。FLUX 利用不同的约束来编码不完全状态,但现有的约束并不能覆盖所有流演算状态公式,这势必影响 FLUX 的应用范围。针对以上问题,在 FLUX 中引入负析取约束,利用约束处理规则集(CHRs)加以实现,并基于流演算基础语义分析了负析取约束的正确性,从而提高了 FLUX 对不完全状态的表达能力。

关键词: 行动推理; 流演算; FLUX; 约束求解
中图分类号: TP301.2 **文献标志码**: A **文章编号**: 1001-3695(2010)08-2980-04
doi:10.3969/j.issn.1001-3695.2010.08.044

Research and implementation on negative disjunction constraints in FLUX

LIU Yi-song[1], ZHU Hui-juan[1], ZHU Mang[2], XU Yan-qun[3]
(1. *School of Computer Science & Communication Engineering, Jiangsu University, Zhenjiang Jiangsu* 212013, *China*; 2. *Management Dept. of Drainage, Zhenjiang Jiangsu* 212000, *China*; 3. *Dept. of Computer Science & Technology, Nanyang Technical Institute, Nanyang Henan* 473004, *China*)

Abstract: FLUX is a constraint logic programming language based on fluent calculus, using which agents can reason logically from their actions and sensor information in incomplete states. The incomplete state is encoded by the constraints in FLUX. However, the existing constrains in FLUX are not complete to cover all cluent calculus state formulas, which to some extent restricts the range of applications of FLUX. This paper addressed this problem by adding negative disjunction constraints into FLUX, implemented it by constraint handling rules (CHRs), and proved the correctness of negative disjunction constraints using the semantics of the cluent calculus, which enhanced the ability of FLUX to express incomplete states.

Key words: reasoning about action; fluent calculus; FLUX; constraint solving

0 引言

在人工智能领域中,赋予智能 agents 在不完全状态环境下对行动和感知信息的逻辑推理能力是当前研究的热点问题之一。基于情景演算的逻辑程序语言 Golog 采用回归(regression)推理机制,判断某个属性 ϕ 是否成立需要回归到初始情景且整个动作历史序列需要参与计算。因此,回归推理机制并不适合 agents 在大规模状态空间和动作空间中执行。基于流演算的逻辑程序语言 FLUX 采用前推(progression)推理机制,在当前情景下就可直接判断某个属性是否成立,其推理效率优于 Golog[1~3]。

FLUX 通过约束对不完全状态加以描述,目前,FLUX 中的约束有:否定约束(not_holds),描述不完全状态中不成立的流;正析取约束(or_holds),描述不完全状态中成立流的析取式;全称约束(all_holds),描述不完全状态中多个成立流的合取式;全称否定约束(all_not_holds),描述不完全状态中多个不成立流的合取式;条件约束(if_then_holds 和 if_then_or_holds),描述不完全状态中在一定条件下成立的流的析取式(可能为单个流)[4~6]。

FLUX 中现有约束的不完整致使一些重要问题无法解决,这必然会大大限制 FLUX 的适用范围。例如,基于流演算理论,某个动作或感知信息会引入析取状态公式,包括 $\vee_{i=1}^{k}\text{holds}(F_i,Z)$(正析取状态公式),FLUX 中 or_holds 约束予以实现;又包括 $\vee_{i=1}^{k}\neg\ \text{holds}(F_i,Z)$(负析取状态公式),但 FLUX 现有约束无法解决此类问题,流演算的创立者 Thielscher 将实现负析取状态公式列为需要进一步研究的重要问题之一[4~6]。针对以上问题,本文对流演算理论和 FLUX 深入研究后,在 FLUX 现有约束基础上引入负析取约束(or_not_holds),解决了流演算中负析取状态公式问题,并基于流演算基础语义证明负析取约束的正确性。

1 流演算及 FLUX

1.1 流演算

流演算(fluent calculus)[5,7] 可以看做是为了解决框架问题(行动推理所伴随的三大问题之一)而对情景演算的一种改进。流演算引入状态概念,利用状态更新公理,既解决了表示框架问题又解决了推理框架问题。

流演算包含四种保留型,即流型(fluent)、状态型(state)、

收稿日期: 2010-01-07; **修回日期**: 2010-02-04 **基金项目**: 江苏省社会发展计划资助项目(BS2001046);江苏省高校自然科学研究计划资助项目(03kjd520075)

作者简介:刘一松(1966-),男,湖南长沙人,副教授,博士,主要研究方向为人工智能、智能虚拟主体(liuyisong@ujs.edu.cn);朱会娟(1984-),女,河南洛阳人,硕士研究生,主要研究方向为人工智能、流演算;朱芒(1975-),男,工程师,主要研究方向为智能控制.

动作型(action)和情景型(situation),其中流型是状态型的子型。

流演算中状态是由成立的流构成,用常量∅和特殊二元函数∘:state × state→state 表示。其中,"∘"的含义为将两个状态合并为一个状态,不含任何流的状态称为空状态(∅),仅含一个流的状态称为单一状态。流演算中引入宏 holds(f, Z)表示流 f 在状态 Z 下成立,宏 holds 用等词关系表示成 holds(f, Z) = $_{\text{def}}$ ($\exists Z'$)$Z = f \circ Z'$,即如果状态 Z 能分解成两个状态,且其中一个是含流 f 的单一状态,那么在 Z 下 f 成立;反之,如果不能以这种方法分解,那么在 Z 下 f 不成立。

1.2　FLUX

FLUX[6] 由于不完全状态的约束表示形式和前推推理机制,致使其计算复杂度随状态空间和动作空间的增大仅呈线性增大。因此 FLUX 非常适合用于大规模、动态环境下的规划问题求解[1]。

FLUX 中的不完全状态可表示为列表$[F_1,\cdots,F_n|Z]$,即除已知成立流 $F_1,\cdots,F_n$ 外,其他成立和不成立的流(变量 Z)不为主体(或机器人)所认知。通过对列表或列表中尾部变量 Z 加以约束来描述不完全状态,如表 1 所示。

表 1　FLUX 中现有的约束和语义

约束	语义
not_holds(F,Z)	¬ holds(F,Z)
all_holds(F,C,Z)	$\forall(\bar{x})(c(\bar{x})\supset \text{holds}(F,Z))$,$\bar{x}$ 是流 F 中的变量
all_not_holds(F,C,Z)	$\forall(\bar{x})(c(\bar{x})\supset \neg\ \text{holds}(F,Z))$,$\bar{x}$ 是流 F 中的变量
or_holds($[F_1,\cdots,F_k]$,Z)	$\vee_{i=1}^{k}\text{holds}(F_i,Z)$
if_then_holds(F,G,Z)	holds(F,Z)⊃holds(G,Z)
if_then_or_holds(F,$[G_1,\cdots,G_k]$,Z)	holds(F,Z)⊃$\vee_{i=1}^{k}$holds(G_i,Z)

表 1 中 Z 表示不完全状态,$F,G,G_1,\cdots,G_k,F_1,\cdots,F_k$ 等都是流,流的参数可以包含变量也可以不包含变量,$c(\bar{x})$ 为算术表达式。

1.3　CHRs

FLUX 中约束通过 CHRs 予以实现。CHRs[8] 是基于一种称为变迁的过程语义根据用户申明的规则修改约束库来进行约束处理,CHRs 的通用规则如下列式子:

$$H_1,\cdots,H_m \Leftrightarrow G_1,\cdots,G_k|B_1,\cdots B_n \tag{1}$$

$$H_1 \backslash H_2 \Leftrightarrow G|B \tag{2}$$

$$H_1,\cdots,H_m \Rightarrow G_1,\cdots,G_k|B_1,\cdots,B_n \tag{3}$$

式(1)(2)为简化规则,式(3)为衍生规则。$H_1,\cdots,H_m$ 称为 CHRs 的头(head)部分,是一组约束;$G_1,\cdots,G_k$ 为守卫(guard)部分,是一组 Prolog 语句;$B_1,\cdots,B_n$ 为主体(body)部分,是一组约束或 Prolog 语句。式(1)(3)的说明性解释分别由式(4)(5)给出:

$$(\forall \vec{x})(G_1\wedge\cdots\wedge G_k\supset[H_1\wedge\cdots\wedge H_m\equiv (\exists \vec{y})(B_1\wedge\cdots\wedge B_n)]) \tag{4}$$

$$(\forall \vec{x})(G_1\wedge\cdots\wedge G_k\rightarrow[H_1\wedge\cdots\wedge H_m\supset (\exists \vec{y})(B_1\wedge\cdots\wedge B_n)]) \tag{5}$$

其中:$\vec{x}$是在头部分和守卫部分中出现的变量,$\vec{y}$是在主体部分中出现的变量,且与 $\vec{x}$不同名。直观地说,如果头部分可以与约束库中元素匹配,经推导守卫部分为真,则式(1)是用主体部分代替约束库中头部分,式(3)是主体部分被加入约束库中,式(2)是 $H_1,H_2\Leftrightarrow G|H_1,B$ 的缩写形式。

2　负析取约束

2.1　引入负析取约束的原因

FLUX 现有的约束并不能充分表达不完全状态,例如在清扫机器人例子中[6],其场景为一个包括办公室和走廊的正方形办公楼层,如图 1(a)所示。办公楼层可以划分成 5 × 5 个方格,每个办公室占一个方格,如方格(1,1)。走廊由若干方格组成,如方格(1,2)。有些办公室被占领如(3,1),处在被占领的办公室上下左右格子可感知到灯光,如图 1(b)所示。假设初始状态办公室(1,3),(2,3),(5,5)中肯定有某些房间没被占领,但不知道具体是哪些,即

¬ occupied(1,3) ∨ ¬ occupied(2,3) ∨ ¬ occupied(5,5)

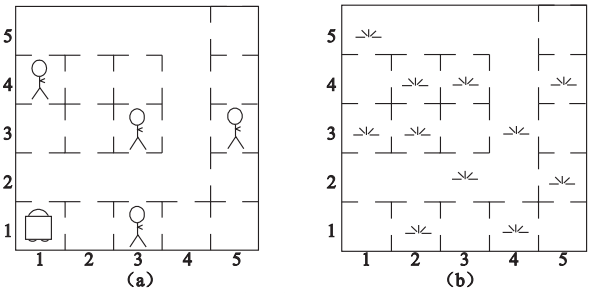

图1　办公楼层布局图

FLUX 现有约束无法处理这类问题,这势必影响 FLUX 的应用范围和其表达问题的能力,因此本文中引入负析取约束,其表示形式和语义如表 2 所示。

表 2　负析取约束和语义

约束	语义
or_not_holds($[F_1,\cdots,F_k]$,Z)	$\vee_{i=1}^{k}$¬ holds(F_i,Z)

2.2　负析取约束实现

在负析取约束实现过程中,需要在析取列表中构建一个新元素[neq(x,y),not_holds(F,Z)],即 or_not_holds(V,Z)中的负析取列表 V 含有流和[neq(x, y),not_holds(F,Z)]两种类型的元素。

负析取约束 or_not_holds($[\delta_1,\delta_2,\cdots,\delta_n]$,$Z$)等价于

$$\vee_{i=1}^{n}\begin{cases}\text{not\_hold}(f,Z)\text{如果 }\delta_i\text{ 是一个具体的流}\\ x\neq y\wedge\text{not\_holds}(f,Z)\text{如果 }\delta_i\text{ 是[neq}(x,y),\text{not\_holds}(f,Z)]\end{cases}$$

当 or_not_holds(V,Z)中负析取列表 V 中的流包含变量参数时,须采用 CHRs 衍生规则得到[neq(x, y),not_holds(F,Z)]形式。例如,or_not_holds([$F(a)$,$F(x)$],[$F[y]|Z$]),通过 CHRs 衍生规则可以重写为

$$\text{or\_not\_holds}\left(\begin{bmatrix}[\text{neq}([x],[y]),\text{not holds}(F(x),Z)],\\ [\text{neq}([a],[y]),\text{not\_holds}(F(a),Z)]\end{bmatrix},z\right)$$

根据流演算基础公理和惟一名公理$\sum_{\text{state}}\cup\text{UNA}[F]$[5]可得出:

not_holds($F(a)$,$F(y)\circ Z$) ∨ not_holds($F(x)$,$F(y)\circ Z$) ≡

($a\neq y\wedge$ not_holds($F(a)$,Z)) ∨ ($x\neq y\wedge$ not_holds($F(x)$,Z))

它与下面的推导是一样的:

not_holds($F(a)$,$F_1(x_1)\circ\cdots\circ F_n(x_n)\circ Z$) ∨

not_holds($F(b)$,$F_1(x_1)\circ\cdots\circ F_n(x_n)\circ Z$)

根据与状态的第一个元素 $F_1(x_1)$ 进行比较,这个式子可

以等价为下面的四个约束：

$a \neq x_1 \vee b \neq x_1$

$a \neq x_1 \vee \text{not\_holds}(F(b), F_2(x_2) \circ\cdots\circ F_n(x_n))$

$b \neq x_1 \vee \text{not\_holds}(F(a), F_2(x_2) \circ\cdots\circ F_n(x_n))$

$\text{not\_holds}(F(a), F_2(x_2) \circ\cdots\circ F_n(x_n)) \vee$

$\text{not\_holds}(F(b), F_2(x_2) \circ\cdots\circ F_n(x_n))$

根据以上的推理可以得到负析取约束的实现代码如下：

```
or_not_holds([F],Z)⇔\+length(F,2)|not_holds(F,Z)                    %1
or_not_holds(V,Z)⇔\+(member(F,V),\+length(F,2))|
                    or_and_not_eq(V,D),call(D)                      %2
or_not_holds([F],[])⇔\+length(F,2)|true                             %3
or_not_holds(V,[])⇔member(F,V,W),\+length(F,2)|
                    or_not_holds(W,[])                              %4
or_not_holds(V,Z)⇔member(F,V),length(F,2),\+(or_and_
                    not_eq[F],D),\+call(D)|true                     %5
or_not_holds(V,Z)⇔member(F,V,W),length(F,2)\+
                    (or_and_not_eq(F,D),call(D))|
                    or_not_holds(W,Z)                               %6
or_not_holds(V,[F|Z])⇔or_not_holds(V,[],[F|Z])                      %7
or_not_holds([G|V],W,[F|Z])⇔
                    G==F -> or_not_holds(V,W,[F|Z]);
                    G\=F ->or_not_holds(V,[G|W],[F|Z]);
                    G=..[_|ArgX],F=..[_|ArgY],
                    or_not_holds(V,[[neq(ArgX,ArgY),
                    not_holds(G,Z)]|W],[F|Z])                       %8
or_not_holds([],W,[_|Z])⇔or_not_holds(W,Z)                          %9
not_holds(F,Z)\or_not_holds(V,Z)⇔member(G,V,W),
                    F==G|true                                       %10
all_holds(F,C,Z)\or_not_holds(V,Z)⇔
                    copy_fluent(F,C,F1,C1),member(G,V),
                    member(G,V,W),F1=G,\+call(#\+C1)|
                    or_not_holds(W,Z)                               %11
all_not_holds(F,C,Z)\or_not_holds(V,Z)⇔
                    copy_fluent(F,C,F1,C1),
                    member(G,V),
                    F1=G,\+call(#\+C1)|true                         %12
or_and_not_eq([],0#\=0)
or_and_not_eq([E|Eq],D1#\/D2):-
                    (length(E,2)->(member(A,E),
                    member(A,E,B),member(C,B),call(A),
                    call(C)->D1=(0#=0);D1=(0#\=0))),
                    or_and_not_eq(Eq,D2)
```

2.3 负析取约束代码的正确性分析

以上代码不仅实现了负析取约束，也实现了与 FLUX 中现有约束的统一。下面对负析取约束代码的正确性进行分析。

%1 处理负析取列表 V 中只有一个流形式元素，并通过%5加以验证。%2 处理列表 V 中只有[neq(x, y), not_holds(F,Z)]形式元素，其中的 neq(x, y)和 not_holds(F,Z)是合取关系，V 中相互元素之间是或的关系，需要调用辅助谓词 or_and_not_eq(V,D)，通过%8 加以验证。%3 处理空的状态下 V 中只有一个流形式元素，此约束等价为真。%4 处理空状态下 V 中有流形式元素，根据流演算空状态公理这是无意义的，因此从列表 V 中移除流形式元素。%3 和%4 可由式(6)(流演算空状态公理)证明：

$$\text{not\_holds}(F, \varnothing) \equiv \text{T} \tag{6}$$

%5 和%6 处理 V 中含有[neq(x, y), not_holds(F,Z)]形式元素，其正确性可由式(7)(8)证明：

$$(x \neq y \wedge \text{not\_holds}(F,Z)) \supset \{[(x \neq y \wedge \text{not\_holds}(F,Z)) \vee \xi] \equiv T\} \tag{7}$$

$$(x \neq y \vee \neg\ \text{not\_holds}(F,Z)) \supset \{[(x \neq y \wedge \text{not\_holds}(F,Z)) \vee \xi] \equiv \xi\} \tag{8}$$

%7 ~%9 总体来说是将 or_not_holds([F_1,…,F_n],[G_1,…,G_k|Z_1])($n \geqslant 0, k \geqslant 0$, Z_1 为变量，F_1,…,F_n 和 G_1,…,G_k 为流)的形式转换为

$$\text{or\_not\_holds}([R_1,\cdots,R_n], Z_1)$$

其中，R_i($0 \leqslant i \leqslant n, n \in N, n \geqslant 0$) 具有下列形式之一：

a)流 F_i，流形式，如果 F_i($1 \leqslant i \leqslant n$)与所有 G_j($1 \leqslant j \leqslant k$)都不相同，即 $F_i = \backslash = G_j$(“=\=”为 Prolog 谓词)；

b)元素[neq(x, y), not_holds(F,Z)]，列表行式，如果 F_i 与 G_j 可以合一，即 $F_i = G_j$，此处“=”为 Prolog 合一谓词，neq(x, y)指$x \neq y$, x为流 F_i 中的变量，y为流 G_j 中的变量，此处 not_holds(F,Z)只是一个列表元素不具有约束功能。

例如，约束 or_not_holds([$F(a)$,$F(x)$],[$F(y)$|Z])(a 为常量，x、y 为变量)形式转换步骤如下：

$$\text{or\_not\_holds}([F(a),F(x)],[F(y)|Z]) \xrightarrow{\%7}$$

$$\text{or\_not\_holds}([F(a),F(x)],[],[F(y)|Z|]) \xrightarrow{\%8}$$

$$\text{or\_not\_holds}[F(x)],[[\text{neq}([a],[y]),\ \text{not\_holds}(F(a),Z)]],[F(y)|Z] \xrightarrow{\%8}$$

$$\text{or\_not\_holds}([],[\text{neq}([x],[y]),\text{not\_holds}(F(x),z)],\ [\text{neq}([a],[y]),\text{not\_holds}(F(a),z)]],[F(y)|z]) \xrightarrow{\%9}$$

$$\text{or\_not\_holds}([[\text{neq}([x],[y]),\text{not\_holds}(F(x),Z)],\ [\text{neq}([a],[y]),\text{not\_holds}(F(a),Z)]],Z)$$

假设 F 是流演算中的流集合，设 or_not_holds(δ_1,[f|Z])为 ξ_1，or_not_holds(δ_1,Z)为 ξ_2，根据流演算基本状态公理 $\sum_{\text{state}}$ 和惟一名公理 UNA[F]可得出下列式子：

$$\xi_1 \equiv [\xi_1 \vee \text{or\_not\_holds}([],Z)] \tag{9}$$

$$[\text{not\_holds}(f, f \circ Z) \vee \xi_1] \vee \xi_2 = \xi_1 \vee \xi_2 \tag{10}$$

$$f_1 \neq f \supset \{[\text{not\_holds}(f_1, f \circ Z) \vee \xi_2 \equiv \xi_1 \vee [\text{not\_holds}(f_1, Z) \vee \xi 2] \tag{11}$$

$$[\text{not\_holds}(F(x), F(y) \circ Z) \vee \xi_1] \vee \xi_2 \equiv \xi_1 \vee [(x \neq y \wedge \text{not\_holds}(F(x), Z)) \vee \xi_2]; \tag{12}$$

$$[\text{or\_not\_holds}([],[f|Z]) \vee \xi_2] \equiv \xi_2 \tag{13}$$

%7 ~%9 采用 CHRs 中的衍生规则，%7 可由式(9)证明，%8 可由式(10) ~ (12)证明，%9 可由式(13)证明。%10 当负析取列表 V 中的某个流在约束库中已经申明不成立时，约束 or_not_holds(V,Z)等价为 true，可由式(14)证明：

$$\text{not\_holds}(f,Z) \supset [(\text{not\_holds}(f,Z) \vee \xi) \equiv \text{T}] \tag{14}$$

%12 是将%10 扩展到包含变量的一般性情况下，借助 Prolog 合一谓词匹配包含变量的流，其正确性可由式(15)来证明：

$$[\forall(x)(c(x) \supset \text{not\_holds}(f,Z))] \supset [(\text{not\_holds}(g,Z) \vee \xi) \equiv \text{T}] \quad ((\exists\theta) f\theta = g) \tag{15}$$

%11 当负析取列表 V 中某个流可以与约束库中已经申明成立的流合一时，约束 or_not_holds(V,Z)转换为 or_not_holds(W,Z)，W 是从列表 V 中除去成立流后所得的列表，即约束 all_holds与 not_holds 不可能同时成立，正确性可由式(16)证明：

$$\{[\forall(x)(c(x) \supset \text{holds}(f,Z))] \wedge \text{not\_holds}(g,Z)\} \equiv \text{F} \quad ((\exists\theta) f\theta = g) \tag{16}$$

3 应用实例

对机器人清扫例子加以改进，假设初始状态机器人在办公室(1,1)，在门 d1112(连接办公室(1,1)和走廊(1,2))处，现

要到达办公室(3,3)(如图 1 中,但被占领办公室有所变动),其中可达房间必须是无人占领的,即不可妨碍办公人员办公。机器人感知到办公室(1,3)、(3,3)中可能未被占领,但不知道具体是哪个。机器人执行程序如下:

初始状态简化描述为

```
init(Z0):-
Z0 =[in(1,1),at(d1112)|Z],
     not_holds(occupied(1,2),Z),
     not_holds(occupied(2,2),Z),
     not_holds(occupied(3,2),Z),
     not_holds(occupied(1,1),Z),
     or_not_holds([occupied(1,3),occupied(3,3)],Z),
     all_not_holds(occupied(_,0),0#=0,Z)
     ……
```

前提条件公理为

```
poss(enter(X1,Y1),[Light]),Z):-
     knows_val([D],at(D),Z),
     knows_val([X,Y],in(X,Y),Z),
     connect(X,Y,X1,Y1,D),
     knows_not(occupied(X1,Y1),Z)
poss(go(D),Z):-
     knows_val([X,Y],in(X,Y),Z),
     (oritate(X,Y,X1,Y1,D,Z),
     knows_not(at(D),Z);
     connect(X,Y,X1,Y1,D),
     knows_not(occupied(X1,Y1),Z),
     knows_not(at(D),Z))
```

状态更新公理为

```
state_update(Z1, go(D),Z2, []):-
     update(Z1, [at(D)], [at(_)], Z2)
state_update(Z1,enter(X,Y,[Light]),Z2):-
     update(Z1,[in(X,Y)],[in(_,_)],Z2),
     light(X,Y,Light,Z2).
```

辅助谓词:

```
c(1,1,1,2,d1112)
……
connect(X,Y,X1,Y1,D):-
     c(X,Y,X1,Y1,D);c(X1,Y1,X,Y,D)
oritate(X,Y,X1,Y1,D,Z):-
     knows_val([X_g,Y_g],goalroom(X_g,Y_g),Z),
     R1 is X_g -X, R2 is Y_g -Y,
     (R1 #>0 -> D1 is 1;
     R1 #<0 -> D1 is -1;
     R1 #=0 -> D1 is 0),
     (R2 #>0 -> D2 is 1;
     R2 #<0 -> D2 is -1;
     R2 #=0 -> D2 is 0),
     ((D1 =0;D2 =0),direct_x =D1,direct_y =D2;
     direct_x =D1,direct_y =0;direct_x =0,direct_y =D2),
     X1 is direct_x +X, Y1 is direct_y +Y,
     connect(X,Y,X1,Y1,D),
     knows_not(occupied(X1,Y1),Z).
light(X, Y, percept,Z):-
     X_east #= X+1, X_west #= X-1,
     Y_north #= Y+1, Y_south #= Y-1,
     (percept = false,
     not_holds(occupied(X_east,Y),Z),
     not_holds(occupied(X,Y_north),Z),
     not_holds(occupied(X_west,Y),Z),
     not_holds(occupied(X,Y_south), Z);
     percept = true,
     or_holds([occupied(X_east,Y),
     occupied(X,Y_north),
     occupied(X_west,Y),
     occupied(X,Y_south)],Z))
```

执行目标语句 main(3,3),即目标办公室为(3,3),执行完毕可得动作序列:enter(1,2)、go(c1222)、enter(2,2)、go(c2232)、enter(3,2)、go(d3233)、enter(3,3),以及下列约束:

```
not_holds(occupied(3,3),_)
not_holds(occupied(1,1),_)
not_holds(occupied(1,2),_)
…
```

在初始状态下执行 enter 动作到达走廊(1,2),感知到灯光,赋予 Light 真值,并在约束库中增加约束:or_holds([occupied(1,3), occupied(2,2), occupied(1,1),occupied(-1,0)],*Z*),约束 not_holds(occupied(1,1),*Z*)、not_holds(occupied(2,2),*Z*) 和 all_not_holds(_,0)已经存在于约束库中,这四个约束语句相结合推得 holds(occupied(1,3),*Z*),即把流 occupied(1,3)加入当前状态 *Z*(此时状态 Z 可以分解为流 occupied(1,3)和变量 *Z*1),并取消约束库的 or_holds 约束。约束库中负析取约束 or_not_holds([occupied(1,3),occupied(3,3)], occupied(1,3)∘*Z*1)经处理后将被约束 not_holds(occupied(3,3),*Z*1)替换,即办公室(3,3)不被占领,则机器人可以顺利到达办公室(3,3)。

4　结束语

不完全状态如何表达是关于动作和感知信息逻辑推理过程中要解决的关键问题之一,它直接影响 FLUX 的整体结构及其描述和处理问题的能力。本文引入负析取约束,通过 CHRs 加以实现,并基于流演算基础语义对负析取约束的正确性进行了分析,从而增强了 FLUX 对不完全状态的表达能力,提高了 FLUX 处理问题的能力。

参考文献:

[1] THIELSCHER M. Reasoning about actions with CHRs and finite domain constraints[C]//Proc of the 18th International Conference on Logic Programming. Berlin: Springer,2002: 70-84.

[2] THIELSCHER M. Logic-based agents and the frame problem: a case for progression[C]//HENDRICKS V F. First-Order Logic Revisited: Proc of the Conference "75 Years of First-Order Logic". 2004:323-336.

[3] SCHIFFEL S, THIELSCHER M. Interpreting golog programs in FLUX[C]//Proc of the 7th International Symposium on Logical Formalizations of Commonsense Reasoning. 2005.

[4] THIELSCHER M T. Handling implication and universal quantification constraints in FLUX[C]//BEEK P van. Proc of the 11th International Conference on Principles and Practice of Constraint Programming. 2005: 667-681.

[5] THIELSCHER M. Reasoning robots: the art and science of programming robotic agents[M]. [S. l.]: Kluwer Academic Publishers, 2005.

[6] THIELSCHER M. FLUX: a logic programming method for reasoning agents[J]. Theory and Practice of Logic Programming, 2005,5(4-5):533-565.

[7] THIELSCHER M. From situation calculus to fluent calculus: state update axioms as a solution to the inferential frame problem[J]. Artificial Intelligence, 1999, 111(1-2): 277-299.

[8] FRÜUHWIRTH T. Theory and practice of constraint handling rules [J]. Logic Programming, 1998, 37(1-3): 95-138.