

文章编号:1001-9081(2010)11-2932-05

从经典逻辑知识构建 ASP 知识库的新方法

赵岭忠,王雪松,钱俊彦,蔡国永

(桂林电子科技大学 计算机科学与工程学院, 广西 桂林 541004)

(zhaolingzhong163@163.com)

摘要:回答集程序设计(ASP)是一种主流的非单调知识表示工具。为了能够在利用 ASP 求解问题过程中使用现有的以经典逻辑表示的知识,给出了一种把以谓词逻辑公式表示的约束型知识和定义型知识转化为 ASP 程序或知识库的新方法,并以实例说明了其有效性。该方法满足转化后 ASP 程序的回答集与原公式集的模型具有一一对应关系。在实际应用中,该方法提供了一项从现存的以谓词逻辑为表示语言的知识库,构建以 ASP 为知识表示语言的非单调知识库的技术。

关键词:谓词逻辑;谓词公式;回答集程序设计;ASP 知识库

中图分类号: TP18; TP311.13 **文献标志码:** A

New method for building ASP knowledge base from knowledge in classical logic

ZHAO Ling-zhong, WANG Xue-song, QIAN Jun-yan, CAI Guo-yong

(School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin Guangxi 541004, China)

Abstract: Answer Set Programming (ASP) is now a mainstream tool for the representation of non-monotonic knowledge. In order to make use of the existing knowledge in classic logic in the process of using ASP for problem solving, a method was proposed for translating knowledge in classic logic formulas to an ASP program or ASP knowledge base so that the models of the formulas and the answer sets of the ASP program were in one-to-one correspondence. Some examples were presented to illustrate the effectiveness of the method. Two classes of knowledge were distinguished in this paper, i. e. constraint knowledge that requires a formula to be satisfactory and definition knowledge that defines a predicate. In practice, the method provides a way of building non-monotonic ASP knowledge bases from the existing knowledge bases that use predicate logic as representation language.

Key words: predicate logic; predicate formula; Answer Set Programming (ASP); ASP knowledge base

0 引言

国际学术界对非单调知识库的研究方兴未艾,当前关注的重点主要集中在非单调知识库的管理、维护、验证等方面^[1-2]。由于传统的单调知识库中存在丰富的、以谓词逻辑公式表示的知识资源,所以在使用非单调知识库的过程中,用户通常希望拥有访问该单调知识的能力。把以谓词逻辑公式表示的知识转化为等价的、以非单调语言表示的非单调知识是一种可行的解决方法。回答集程序设计(Answer Set Programming, ASP)是采用回答集语义的逻辑程序设计技术^[3-4],能够利用默认否定机制方便地表示非单调知识。目前,ASP已经成为一种主流的知识表示和推理工具^[5-6]。因此,研究如何在一定的语义约束下把以谓词逻辑表示的知识转化为对应的 ASP 程序,成为基于现有知识,构建以 ASP 为知识表示语言的非单调知识库技术的关键。

本文区分两种类型的知识:约束型知识和定义型知识。在待求解问题的规格中,约束型知识规定一个问题解需要满足的条件;而定义型知识则规定一个概念是如何精确定义的。约束型知识可以是任意逻辑公式。比如,约束型知识 $(p \wedge q)$ 要求在一个问题的解中命题 p 和 q 同时成立。如果使用谓词表示

某个概念,则关于该概念的定义型知识则给出了该谓词的一个定义。通常,定义型知识具有以下形式:

$$F(X_1, \dots, X_n) \equiv \alpha$$

其中: F 是一个 n 元谓词, α 是一个具有自由变量 X_1, \dots, X_n 的逻辑公式。上式称为原子公式 $F(X_1, \dots, X_n)$ 的定义。使用以上两种类型的知识可以获得层次分明的问题规格。比如, N -皇后问题可以由一条约束型知识 α 和一条定义型知识 β 来描述,其中:

$$\alpha: \forall X. \exists Y. \text{hasqueen}(X, Y)$$

要求每一行必须有一个皇后;

$$\beta: \text{hasqueen}(X, Y) \equiv$$

$$\begin{aligned} & \forall Z. W. (((X = Z) \wedge (Y \neq W) \rightarrow \\ & \neg \text{hasqueen}(Z, W)) \wedge ((X \neq Z) \wedge (Y = W) \rightarrow \\ & \neg \text{hasqueen}(Z, W)) \wedge (\text{abEqu}(X, Y, Z, W) \rightarrow \\ & \neg \text{hasqueen}(Z, W))) \end{aligned}$$

则要求没有任何两个皇后在同一行、同一列,或者在同一对角线上。其中 $\text{abEqu}(X, Y, Z, W)$ 表示坐标点 (X, Y) 和 (Z, W) 在同一对角线上。

具体而言,本文研究的核心问题可描述如下:已知约束型知识和定义型知识的集合 S ,如何求得一个对应的 ASP 程序

收稿日期:2010-05-10;修回日期:2010-07-20。 基金项目:国家自然科学基金资助项目(60803033;60903079)。

作者简介:赵岭忠(1977-),男,河南社旗人,副教授,博士,主要研究方向:知识表示、知识推理;王雪松(1981-),女,江苏沛县人,讲师,硕士,主要研究方向:知识表示、符号计算;钱俊彦(1973-),男,浙江嵊州人,教授,博士,主要研究方向:软件验证、模型检验;蔡国永(1971-),男,广西平乐人,教授,博士,主要研究方向:软件工程、自治计算。

P 使得知识集 S 的模型和 P 的回答集具有一一对应关系。Niemi 和 You 等人^[7-8]研究了以上问题的一个子问题,即把命题逻辑子句集转化为逻辑程序,并使得子句集的模型与逻辑程序的回答集具有一一对应关系,其中命题子句相当于本文的约束型知识。Pettorossi 和 Proietti^[9]给出了一种把谓词公式转化为采用完美模型语义的逻辑程序的算法。Lin 和 Zhao^[10]研究了以上问题的逆问题,即在保持语义的情况下把一个 ASP 程序转化为一个命题逻辑子句集,并基于该研究成果开发了基于 SAT 的回答集求解器 ASSAT。

显然,为了把以逻辑公式表示的约束型知识和定义型知识转化为对应的 ASP 程序,需要把命题逻辑子句到 ASP 规则的转化技术扩展应用于谓词逻辑公式到 ASP 规则的转化。为此,首先给出了一种语法树制导的逻辑公式翻译技术;然后在此基础上给出约束型知识和定义型知识的翻译;此后证明了按照本文知识转化技术所生成的 ASP 程序语义与原有知识的模型之间具有一一对应关系。最后,以 N -皇后问题为例说明了本文技术的应用。

1 基础知识

本文考虑不含函词的谓词逻辑公式。若 a 是原子,则 a 称为正文字, $\neg a$ 称为负文字。称 a 和 $\neg a$ 为互补的文字。如果一个文字集合 S 中不包含互补的文字,则称 S 是一致的。若 A 和 B 均为集合,则 $A \setminus B$ 定义为: $A \setminus B = \{a \mid a \in A, a \notin B\}$ 。

谓词逻辑的词汇包含以下互不相交的集合:

- 1) 变量集;
- 2) 谓词符号集,其中 $n \geq 0$ 元谓词 F 表示为 $F(X_1, \dots, X_n)$,其中 X_1, \dots, X_n 是变量集中的元素;
- 3) 个体符号集,该集合是有限集。

0 元谓词又称为命题。本文考虑的逻辑符号包括:存在量词 \exists , 逻辑连接词 \neg 、 \wedge 、 \vee 。在此基础上,逻辑公式可定义如下:

- 1) 若 F 是一个 n 元谓词符号, t_1, \dots, t_n 为变量或个体符号,则 $F(t_1, \dots, t_n)$ 是一个原子公式;
- 2) 若 α 和 β 是两个逻辑公式,则 $\neg \alpha$ 、 $(\alpha \wedge \beta)$ 和 $(\alpha \vee \beta)$ 也是逻辑公式;
- 3) 若 α 是公式,则 $(\exists X. \alpha)$ 是逻辑公式。

为了便于讨论,以上公式均不含全称量词 \forall ,以及联系词蕴涵 \rightarrow 和等价 \leftrightarrow 。形如 $(\forall X. \alpha)$ 的公式被视为 $\neg(\exists X. \neg \alpha)$ 的简写;蕴涵式 $(\alpha_1 \rightarrow \alpha_2)$ 被视为 $(\neg \alpha_1 \vee \alpha_2)$ 的简写;等价公式 $(\alpha_1 \leftrightarrow \alpha_2)$ 被视为 $(\alpha_1 \rightarrow \alpha_2) \wedge (\alpha_2 \rightarrow \alpha_1)$ 的简写。

变量 X 在公式 α 中的某次出现是约束出现,如果它出现在形如 $\exists X. \beta$ 的 α 的子公式中;否则称 X 的本次出现为自由的。公式集 S 的 Herbrand 基 U_S 定义为由 S 中出现的谓词符号所构成的所有不包含变量的原子的集合。 U_S 的任意子集均称为 Herbrand 解释。

已知 Herbrand 解释 I , 逻辑公式在 I 中的真值定义如下:

- 1) 如果原子公式 a 不含变量且 $a \in I$,则 a 在 I 中为真;
- 2) 如果负文字 $\neg a$ 不含变量且 $a \notin I$,则 $\neg a$ 在 I 中为真;
- 3) 如果公式 $(\alpha \wedge \beta)$ 不含变量且 α 和 β 在 I 中均为真,则 $(\alpha \wedge \beta)$ 为真;
- 4) 如果公式 $(\alpha \vee \beta)$ 不含变量且 α 或 β 在 I 中为真,则 $(\alpha \vee \beta)$ 为真;
- 5) 如果存在 α 的一个不含变量的实例在 I 中为真,则公

式 $(\exists x. \alpha)$ 在 I 中为真。

I 被称为 S 的 Herbrand 模型,或简称模型,如果对任意公式 $f \in S$, f 在 I 中为真。对 Herbrand 基 U_S 中的每一个原子 $p(l_1, \dots, l_k)$,关联一个新的原子 $p'(l_1, \dots, l_k)$,表示 $p(l_1, \dots, l_k)$ 的否定。下文中,这些新原子的集合表示为 U_S' ,即:

$$U_S' = \{p'(l_1, \dots, l_k) \mid p(l_1, \dots, l_k) \in U_S\}$$

下面给出 U_S 和 U_S' 中原子的默认定义 $DDef(S)$ 。

定义 1 默认定义。令 S 是一个公式, $DDef(S)$ 定义如下:

$$DDef(S) = \{p(l_1, \dots, l_k) :- \text{not } p'(l_1, \dots, l_k). \mid p(l_1, \dots, l_k) \in U_S\} \cup \{p'(l_1, \dots, l_k) :- \text{not } p(l_1, \dots, l_k). \mid p'(l_1, \dots, l_k) \in U_S'\}$$

例如:

$$DDef(\{a \vee \neg b \vee c\}) = \{a :- \text{not } a'. \mid a' :- \text{not } a. \mid b :- \text{not } b'. \mid b' :- \text{not } b. \mid c :- \text{not } c'. \mid c' :- \text{not } c. \}$$

可见, $DDef(\{\alpha\})$ 可视为由 α 中出现的谓词所构成的所有不含变量的原子的默认定义。该定义仅规定,对于任意原子 $p(l_1, \dots, l_k)$, $p(l_1, \dots, l_k)$ 和 $p'(l_1, \dots, l_k)$ 之中有且只有一个在回答集(定义见下文)中为真。在实际问题求解过程中,当这些原子存在更具体的定义时, $DDef()$ 可以省略。

回答集程序设计是采用回答集语义的逻辑程序设计技术。本文假定读者熟悉逻辑程序设计的基本知识。在逻辑程序的回答集语义中,包含变量的规则被视为该规则所有不含变量实例的集合的简写。一个正规逻辑程序(Normal Logic Program, NLP) P 是一组规则的集合,其中每一条规则具有如下形式:

$$L_0 :- L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n$$

其中: $n \leq m \leq 0$, L_i 是正文字, $\text{not } L_i$ 被称为 L_i 的默认否定。对于任意规则 r , $\text{head}(r) = L_0$ 是规则 r 的头部, $\text{body}(r) = \{L_1, \dots, L_m, \text{not } L_{m+1}, \dots, \text{not } L_n\}$ 是 r 的规则体,同时定义 $\text{pos}(r) = \{L_1, \dots, L_m\}$, $\text{neg}(r) = \{L_{m+1}, \dots, L_n\}$, $\text{lit}(r) = \text{pos}(r) \cup \text{neg}(r) \cup \{\text{head}(r)\}$ 。特别地,若 $\text{body}(r) = \emptyset$,则称 r 为事实;若 r 没有头部,则称 r 为约束。

下面首先给出不包含默认否定的 NLP 的回答集的定义。若 π 是一个不包含 not 的 NLP, LIT 是 π 的 Herbrand 基 U_π 。 π 的回答集是满足下面条件的 LIT 的任意最小子集 S : 对每一个规则 $r \in \pi$, 如果 $\text{pos}(r) \subseteq S$, 则 $\text{head}(r) \in S$ 。由定义可知,不包含 not 的 NLP 的回答集,如果存在则必然是唯一的。

对以上定义进行拓展可得到包含 not 的 NLP 的回答集的定义。若 π 是一个包含 not 的 NLP, LIT 是 π 的 Herbrand 基 U_π 。对任意 $S \subseteq LIT$, π^S 定义如下:

$$\pi^S = \{\text{head}(r) :- \text{pos}(r) \mid r \in \pi \wedge \text{neg}(r) \cap S = \emptyset\}$$

显然, π^S 不包含 not, 假设其回答集为 ANS , 如果 $S = ANS$, 则 S 是 π 的一个回答集。

已知程序 P 约束集 C , $P \cup C$ 的回答集是 P 的回答集中满足 C 中约束的集合。

例 下面的 ASP 程序 P :

$$p :- \text{not } q. \quad q :- \text{not } p.$$

有 $\{p\}$ 和 $\{q\}$ 两个回答集。如果向 P 中增加约束“ $:-p.$ ”, 则 $\{p\}$ 就不再是 $P \cup \{:-p.\}$ 的一个回答集。

2 逻辑公式的转化

下面讨论把逻辑公式转化为 ASP 规则的方法。首先,每一

个逻辑公式将表示为一棵语法树;然后将基于树结构产生与该逻辑公式对应的 ASP 规则。

一棵语法树是满足下述条件的树结构:

1) 每一个内部节点都有一个与之关联的逻辑符号做标记,且如果标记为“ \neg ”或“ $\exists X$ ”,其孩子节点唯一;

2) 每一个叶子节点都有一个与之关联的命题符号或者原子命题做标记。

下文中,节点 N 的标记记做 $label(N)$ 。语法树的每一个节点 N 均对应一个逻辑公式 $f(N)$,其定义如下:

1) 如果 N 是叶子节点, $f(N) = label(N)$; 否则:

2) 如果 N 标记为“ \wedge ”且其孩子节点为 N_1, \dots, N_m , 则 $f(N) = f(N_1) \wedge \dots \wedge f(N_m)$;

3) 如果 N 标记为“ \vee ”且其孩子节点为 N_1, \dots, N_m , 则 $f(N) = f(N_1) \vee \dots \vee f(N_m)$;

4) 如果 N 标记为“ \neg ”且其孩子节点为 N' , 则 $f(N) = \neg(f(N'))$;

5) 如果 N 标记为“ $\exists X$ ”且其孩子节点为 N' , 则 $f(N) = \exists(X. f(N'))$ 。

可见,每一个逻辑公式都有一个与之相对于的语法树;公式 α 的语法树的每一个节点均对应于 α 的一个子公式,且与某个节点对应的逻辑公式是该节点的父节点对应逻辑公式的直接子公式(immediate sub-formula)。

对于含有全称量词和蕴涵词的公式 β , 如:

$\beta: \exists X. (animal(X) \wedge \forall Y. (grass(Y) \rightarrow like(Y, X)))$

可首先把公式 β 转化为:

$\exists X. (animal(X) \wedge \neg (\exists Y. \neg (grass(Y) \rightarrow like(Y, X)))) = \exists X. (animal(X) \wedge \neg (\exists Y. (grass(Y) \wedge \neg like(Y, X))))$

以上公式对应的语法树如图 1 所示。

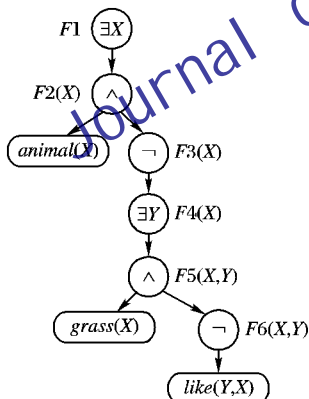


图1 逻辑公式 β 的语法树

本文不讨论构造逻辑公式对应语法树的细节,而是假定:存在一个以逻辑公式 α 为输入,并返回与之对应的语法树的算法 $Parse(\alpha)$ 。下文中,树 t 的根表示为 $root(t)$ 。为了方便,为语法树的每一个节点 N 关联一个新的原子公式 $AtomicF(N)$,表示与 N 对应的逻辑公式 $f(N)$ 。如果 N 是叶子节点, $AtomicF(N) = label(N)$ 。显然,与节点 N 相关联的原子公式形如 $F(X_1, \dots, X_n)$,其中 $\{X_1, \dots, X_n\}$ 是 $f(N)$ 中的自由变量。在图 1 中,与内节点相关联的原子公式被标记在节点的外部。例如,与表示子公式“ $\exists Y. (grass(Y) \wedge \neg like(Y, X))$ ”的节点相关联的原子公式为 $F4(X)$,因为该公式中 X 是唯一的自由变量。

显然,任意逻辑公式集 S 均可转化为一个语法树的集合

T ,且满足与 T 中语法树内部节点相关联的原子公式均互不相同。

基于上面的讨论可给出基于逻辑公式的语法树把该公式转化为 ASP 规则的算法。令 $N = root(Parse(\alpha))$,则下面的算法 $Tr(N)$ 将产生与逻辑公式 α 对应的 ASP 程序。

```

Tr(N)
{ global S =  $\emptyset$ ;
  If label(N) = ' $\neg$ '
  { S := S  $\cup$  { AtomicF(N) :- not AtomicF(N'). | N' is the only
    child node of N };
    return(S); }
  Else if label(N) = ' $\wedge$ '
  { S := S  $\cup$  { AtomicF(N) :- AtomicF(N1), ...,
    AtomicF(Nk). }  $\cup$  Tr(N1)  $\cup$  ...  $\cup$  Tr(Nk). | N1, ..., Nk
    are the child nodes of N };
    return(S); }
  Else if label(N) = ' $\vee$ '
  { S := S  $\cup$  { AtomicF(N) :- AtomicF(N1), ..., AtomicF(Nk). :-
    AtomicF(Nk). }  $\cup$  Tr(N1)  $\cup$  ...  $\cup$  Tr(Nk). | N1, ..., Nk
    are the child nodes of N };
    return(S); }
  Else if label(N) = ' $\exists X.$ '
  { S := S  $\cup$  { AtomicF(N) :- AtomicF(N'). }  $\cup$  Tr(N'). | N' is
    the only child node of N };
    return(S); }
  Else if N is a leaf node
  return( $\emptyset$ );
}

```

例 从图 1 所示的语法树所得 ASP 程序为:

```

Tr(root(Parse( $\beta$ ))) = {
  F1 :- F2(X).
  F2(X) :- animal(X), F3(X).
  F3(X) :- not F4(X).
  F4(X) :- F5(X, Y).
  F5(X, Y) :- grass(X), F6(X, Y).
  F6(X, Y) :- not like(Y, X). }

```

下面讨论如何把知识转化为对应的 ASP 规则。

定义 2 与约束型知识对应的 ASP 规则。已知约束型知识 α ,其对应的 ASP 规则集 $ASP_{con}(\alpha)$ 定义如下:

$ASP_{con}(\alpha) = \{ :- not AtomicF(root(Parse(\alpha))). \} \cup Tr(root(Parse(\alpha)))$

例

$ASP_{con}((p \wedge q) \vee r) = \{ :- not F1.$
 $F1 :- F2.$
 $F1 :- r.$
 $F2 :- p, q.$
 $p :- not p'. p' :- p.$
 $q :- not q'. q' :- q.$
 $r :- not r'.$
 $r' :- r. \}$

例 考虑公式

$\beta: \forall X. (human(X) \rightarrow \exists Y. (man(Y) \wedge is_father_of(Y, X)))$

则: $ASP_{con}(\beta) = \{ :- F1. \} \cup Tr(root(Parse(\beta)))$

下面的定理表明:对任意约束型知识 α ,当且仅当 α 有 Herbrand 模型 $ASP_{con}(\alpha) \cup DDef(\{\alpha\})$ 有回答集。

定理 1 若 α 是一条约束型知识, $DDef = DDef(\alpha)$, U 是 $\{\alpha\}$ 的 Herbrand 基, 则有:

1) 如果 S 是 $ASP_{con}(\alpha) \cup DDef$ 的一个回答集, 则 $(S \cap U)$ 是 α 的一个模型;

2) 如果 M 是 α 的一个模型, 则唯一存在一个 $ASP_{con}(\alpha) \cup DDef$ 的回答集 S 满足 $M \subseteq S$ 。

证明 利用结构归纳法对 α 的结构进行归纳。证明细节略。

定义 3 与定义型知识对应的 ASP 规则。已知定义型知识: $F(X_1, \dots, X_n) \equiv \alpha$, 其对应的 ASP 规则集 $ASP_{def}(F(X_1, \dots, X_n) \equiv \alpha)$ 定义如下:

$$ASP_{def}(F(X_1, \dots, X_n) \equiv \alpha) = \\ Tr(root(Parse(\alpha) \uparrow F(X_1, \dots, X_n)))$$

其中: $t \uparrow (F(X_1, \dots, X_n))$ 是把与树 t 的根节点 $root(t)$ 相关联的原子公式 $AtomicF(root(t))$ 替换为 $F(X_1, \dots, X_n)$ 之后得到的语法树。

下面的定理表明以上给出的对定义型知识的转化是正确的。

定理 2 若 $D: F(X_1, \dots, X_n) = \alpha$ 是一个谓词定义, $DDef = DDef(\{\alpha\})$, U 是 $\{\alpha\}$ 的 Herbrand 基, α' 是利用个体符号 a_1, \dots, a_n 分别替代公式 α 中的自由变量 X_1, \dots, X_n 得到的公式 α 的实例, S 是 $ASP_{def}(D) \cup DDef(\{\alpha\})$ 的一个回答集, 则有: 当且仅当 $(S \cap U)$ 是 α' 的模型, $F(a_1, \dots, a_n) \in S$ 。

证明 略。

下文算法 GenASP(S) 把由约束型知识与定义型知识构成的知识集合转化为 ASP 程序。

```
GenASP(S)
{ global P = {};
  For each expression exp of S, do {
    If exp is a constraint, P := P  $\cup$  ASPcon(exp);
    If exp is a definition, P := P  $\cup$  ASPdef(exp);
  }
  return(P);
}
```

利用定理 1 和定理 2, 可证明算法 GenASP(S) 的正确性。

定理 3 令 S 是一个约束型知识和定义型知识的集合, 则有: 当且仅当 GenASP(S) 有一个回答集, S 有一个模型。

3 实例

例 N -皇后问题。可以使用两个公式描述该问题:

1) $\alpha: \forall X. \exists Y. hasqueen(X, Y)$;

2) $\beta: hasqueen(X, Y) \equiv \forall Z. W. (((X = Z) \wedge (Y \neq W) \rightarrow \neg hasqueen(Z, W)) \wedge ((X \neq Z) \wedge (Y = W) \rightarrow \neg hasqueen(Z, W)) \wedge (abEqu(X, Y, Z, W) \rightarrow \neg hasqueen(Z, W)))$

其中 $abEqu(X, Y, Z, W)$ 定义为 $|X - Z| = |Y - W|$ 。

对公式 α :

$$\forall X. \exists Y. hasqueen(X, Y) \equiv \\ \neg (\exists X. \neg (\exists Y. hasqueen(X, Y)))$$

令: $N_1 = root(Parse(\alpha))$

则有:

$$Tr(N_1) = \{r11: \neg not\ r12. \ r12: \neg r13(X). \\ r13(X): \neg not\ r14(X). \\ r14(X): \neg hasqueen(X, Y). \}$$

由定义 2:

$$ASP_{con}(\alpha) = \{:- not\ AtomicF(root(Parse(\alpha))). \} \cup \\ Tr(root(Parse(\alpha))) = \\ \{:- not\ AtomicF(N_1). \} \cup Tr(N_1)$$

对公式 β , 首先删除其中的全称量词和蕴涵词:

$$hasqueen(X, Y) \equiv \neg (\exists Z. (\exists W. ((X = Z \wedge Y \neq \\ W \wedge hasqueen(Z, W)) \vee (X \neq Z \wedge Y = \\ W \wedge hasqueen(Z, W)) \vee (X \neq Z \wedge Y \neq \\ W \wedge abEqu(X, Y, Z, W) \wedge hasqueen(Z, W))))))$$

根据定义 3, $hasqueen(X, Y)$ 的定义可转化为下述程序:

```
{ hasqueen(X, Y): - not r21(X, Y).
  r21(X, Y): - r22(X, Y, Z).
  r22(X, Y, Z): - r23(X, Y, Z, W).
  r23(X, Y, Z, W): - X = Z, Y \neq W, hasqueen(Z, W).
  r23(X, Y, Z, W): - X \neq Z, Y = W, hasqueen(Z, W).
  r23(X, Y, Z, W): - X \neq Z, Y \neq W, abEqu(X, Y, Z, W),
  hasqueen(Z, W). }
```

如果使用下面实现谓词 $abEqu(X, Y, Z, W)$ 的 ASP 规则:

```
abEqu(X, Y, Z, W): - M = X + W, N = Y + Z,
  X >= Z, Y >= W, M = N.
abEqu(X, Y, Z, W): - M = X - Y, N = Z + W,
  X >= Z, Y <= W, M = N.
abEqu(X, Y, Z, W): - M = Z + W, N = Y + X,
  X <= Z, Y >= W, M = N.
abEqu(X, Y, Z, W): - M = Z + Y, N = W + X,
  X <= Z, Y <= W, M = N.
```

就可以得到下述求解 N -皇后问题的 ASP 程序:

```
% the rules obtained from the first formula.
:- not r11.
r11: - not r12.
r12: - r13(X).
r13(X): - not r14(X), range(X).
r14(X): - hasqueen(X, Y).
% the rules obtained from the second formula.
hasqueen(X, Y): - not r21(X, Y), range(X), range(Y).
r21(X, Y): - r22(X, Y, Z).
r22(X, Y, Z): - r23(X, Y, Z, W).
r23(X, Y, Z, W): - X = Z, Y \neq W, hasqueen(Z, W), range(X),
range(Y), range(Z), range(W).
r23(X, Y, Z, W): - X \neq Z, Y = W, hasqueen(Z, W), range(X),
range(Y), range(Z), range(W).
r23(X, Y, Z, W): - X \neq Z, Y \neq W, abEqu(X, Y, Z, W), hasqueen
(Z, W).
% implementation of abEqu(X, Y, Z, W).
abEqu(X, Y, Z, W): - M = X + W, N = Y + Z, X >= Z, Y >= W, M = N.
abEqu(X, Y, Z, W): - M = X - Y, N = Z + W, X >= Z, Y <= W, M = N.
abEqu(X, Y, Z, W): - M = Z + W, N = Y + X, X <= Z, Y >= W, M = N.
abEqu(X, Y, Z, W): - M = Z + Y, N = W + X, X <= Z, Y <= W, M = N.
range(1..8).
```

在上面的程序中, 对每一个形如“ $F1(X_1, \dots, X_n): \neg not\ F2(X_1, \dots, X_n) \dots$ ”的规则, 有一个规定变量 $X_i (i \in \{1, \dots, n\})$ 取值范围的谓词 $range()$ 。对 m -皇后问题, X_i 的取值范围为 $\{1, \dots, m\}$ 。把规定范围的谓词 $range()$ 插入对应规则体中的另一个目的是为了保持规则的安全性。按照 dlw 语言对安全性的要求, 每一个出现在规则头部的变量至少在同一规则

的规则体的正文中出现一次,且该正文不是比较谓词。因此,形如“ $F1(X_1, \dots, X_n) :- \text{not } F2(X_1, \dots, X_n).$ ”的规则在 dlw 语言中是不安全的。

在 dlw^[11]上运行该程序,可得两个4-皇后问题的解:

```
{hasqueen(3,1), hasqueen(1,2), hasqueen(4,3), hasqueen(2,4)}
{hasqueen(2,1), hasqueen(4,2), hasqueen(1,3), hasqueen(3,4)}
```

利用下面的命令:

```
C:\>dlw mingw -silent -N=16 -pfilter=hasqueen nqueen.txt
```

运行该程序,dlw 返回 92 个 8-皇后问题的解,其中前 8 个解如下所示:

```
{hasqueen(3,1), hasqueen(7,2), hasqueen(2,3), hasqueen(8,4),
hasqueen(5,5), hasqueen(1,6), hasqueen(4,7), hasqueen(6,8)}
{hasqueen(3,1), hasqueen(6,2), hasqueen(2,3), hasqueen(7,4),
hasqueen(5,5), hasqueen(1,6), hasqueen(8,7), hasqueen(4,8)}
{hasqueen(6,1), hasqueen(3,2), hasqueen(1,3), hasqueen(7,4),
hasqueen(5,5), hasqueen(8,6), hasqueen(2,7), hasqueen(4,8)}
{hasqueen(3,1), hasqueen(6,2), hasqueen(8,3), hasqueen(1,4),
hasqueen(5,5), hasqueen(7,6), hasqueen(2,7), hasqueen(4,8)}
{hasqueen(7,1), hasqueen(3,2), hasqueen(8,3), hasqueen(2,4),
hasqueen(5,5), hasqueen(1,6), hasqueen(6,7), hasqueen(4,8)}
{hasqueen(6,1), hasqueen(3,2), hasqueen(1,3), hasqueen(8,4),
hasqueen(5,5), hasqueen(2,6), hasqueen(4,7), hasqueen(7,8)}
{hasqueen(4,1), hasqueen(7,2), hasqueen(1,3), hasqueen(8,4),
hasqueen(5,5), hasqueen(2,6), hasqueen(6,7), hasqueen(3,8)}
{hasqueen(6,1), hasqueen(4,2), hasqueen(2,3), hasqueen(8,4),
hasqueen(5,5), hasqueen(7,6), hasqueen(1,7), hasqueen(3,8)}
```

在上面的命令行中,“-N=max”规定了算术运算中遇到最大合法整数值,dlw 只能正确识别在该范围内的整数。对 N-皇后问题,max 应为 2N。

4 结语

本文给出了一种把以谓词逻辑公式表示的约束型知识和定义型知识转化为 ASP 程序的方法。该方法允许首先利用约束型知识和定义型知识对待求解问题进行规格,然后利用现有的回答集求解器来求得问题的部分或者全部解。虽然本文采用的回答集求解器是 dlw,但只需对相关算法进行简单的修改便可生成能够在其他回答集求解器^[12-13]上运行的逻辑程序。在构造以 ASP 为知识表示语言的非单调知识库的过程中,本文方法可作为一个把现有以谓词逻辑公式表示的知识自动转化为 ASP 规则和约束的工具。未来,将把本文方法应用于大规模工业应用领域,如机械装配序列规划问题的求解^[14],以测试本文方法的时空效率和其他性能;并利用该方法建立用于求解机械装配序列规划问题的专用 ASP 知识库^[15]。

参考文献:

- [1] ACOSTA GUADARRAMA J C. Maintaining knowledge bases at the object level [C]// MICAI 2007: the 2007 6th Mexican International Conference on Artificial Intelligence, Special Session. Washington, DC: IEEE Computer Society, 2008: 3-13.
- [2] SCHAUB T. Model-based knowledge representation and reasoning via answer set programming [M]. Berlin, Heidelberg: Springer-Verlag, 2008: 1-2.
- [3] GELFOND M, LIFSCHITZ V. Classic negation in logic programs and disjunctive databases [J]. Next Generation Computing, 1991, 9(3/4): 365-385.
- [4] GELFOND M, LIFSCHITZ V. The stable model semantics for logic programming [C]// Proceedings of the ICLP'88. Cambridge, MA: The MIT Press, 1988: 1070-1080.
- [5] BARAL C, GELFOND M. Logic programming and knowledge representation [J]. Journal of Logic Programming, 1994, 19: 73-148.
- [6] LIFSCHITZ V. Answer set programming and plan generation [J]. Artificial Intelligence, 2002, 138(1/2): 39-54.
- [7] NIEMELÄ I. Logic programs with stable model semantics as a constraint programming paradigm [J]. Annals of Mathematics and Artificial Intelligence, 1999, 25(3/4): 241-273.
- [8] YOU J, CARTWRIGHT R, LI M. Iterative belief revision in extended logic programs [J]. Theoretical Computer Science, 1996, 170(1/2): 383-406.
- [9] PETRODSSI A, PROietti M. Perfect model checking via unfold/fold transformations [C]// Proceedings of the First International Conference on Computational Logic. Berlin: Springer-Verlag, 2000: 613-628.
- [10] LIN F, ZHAO Y. ASSAT: Computing answer sets of a logic program by SAT solvers [J]. Artificial Intelligence, 2004, 157(1/2): 115-137.
- [11] LEONE N, PFEIFER G, FABER W, et al. The DLV system for knowledge representation and reasoning [J]. ACM Transactions on Computational Logic, 2006, 7(3): 499-562.
- [12] LIN Z, ZHANG Y, HERNANDEZ H. Fast SAT-based answer set solver [C]// Proceedings of AAAI '06. Cambridge, MA: The MIT Press, 2006: 92-97.
- [13] GIUNCHIGLIA E, LIERLER Y, MARATEA M. Cmodels - 2: SAT-based answer set programming [EB/OL]. [2009-12-12]. <http://www.star.dist.unige.it/~enrico/ftp/04aaai.pdf>.
- [14] GOTTIPOLU R B, GHOSH K. A simplified and efficient representation for evaluation and selection of assembly sequences [J]. Computer in Industry, 2003, 50(3): 251-264.
- [15] ZHAO L Z, QIAN J Y, CHANG L, et al. Using ASP for knowledge management with user authorization [J]. Data and Knowledge Engineering, 2010, 69(8): 737-762.

(上接第2923页)

- [7] DELLNITZ M, HOBMAN A. A subdivision algorithm for the computation of unstable manifolds and global attractors [J]. Numerische Mathematik, 1997, 75: 293-317.
- [8] KRAUSKOPF B, OSINGA H M, DOEDEL E J, et al. A survey of methods for computing (un) stable manifolds of vector fields [J]. International Journal of Bifurcation and Chaos, 2005, 15(3): 763-791.
- [9] KRAUSKOPF B, OSINGA H M. Growing unstable manifolds of planar maps [EB/OL]. [2009-12-12]. <http://www.ima.umn.edu/preprints/OCT97/1517.pdf>.
- [10] KRAUSKOPF B, OSINGA H M. Computing geodesic level sets on

- global (un) stable manifolds of vector fields [J]. Journal on Applied Dynamical Systems, 2003, 2(4): 546-569.
- [11] HOBSON D. An efficient method for computing invariant manifolds of planar maps [J]. Journal of Computational Physics, 1993, 104(1): 14-22.
- [12] KRAUSKOPF B, OSINGA H M. Growing 1D and quasi 2D unstable manifolds of maps [J]. Journal of Computational Physics, 1998, 146(1): 406-419.
- [13] GUCKENHEIMER J, WOLFOLK P. Dynamical systems: Some computational problems [M]// Bifurcations and Periodic Orbits of Vector Fields. Dordrecht, Netherlands: Kluwer Academic Publishers, 1993: 241-277.