逻辑编程原理

通过一阶逻辑, 可定义一个子串(substr)的谓词, 该定义由以下的一组逻辑公式完成:

(1)$\forall x\, substr(x, x)$

(2)$\forall x \forall y\, suffix(x, y \cdot x)$

(3)$\forall x \forall y\, prefix(x, x \cdot y)$

(4)$\forall x \forall y \forall z\, (substr(x, y) \wedge suffix(y, z) \rightarrow substr(x, z))$

(5)$\forall x \forall y \forall z\, (substr(x, y) \wedge prefix(y, z) \rightarrow substr(x, z))$

将这组一阶逻辑公式转化为子句标准形:

(1)$substr(x,x)$

(2)$suffix(x,y\cdot x)$

(3)$prefix(x,x\cdot y)$

(4)$substr(x,y)\wedge suffix(y,z)\rightarrow substr(x,z)$

(5)$substr(x,y)\wedge prefix(y,z)\rightarrow substr(x,z)$

基于谓词substr的定义, 可以尝试求解这样的问题: 求满足$substr(X, a \cdot a \cdot a \cdot b \cdot c \cdot c)$的$x$.

这里称$substr(X, a \cdot a \cdot a \cdot b \cdot c \cdot c)$为目标(Goal).

求解该问题的思路是: 结合替换, 通过归结方法证否该目标的否命题。然后, 问题的解可通过相应的替换得到。

该目标的否命题是:

$$\neg(\exists X\, substr(w, a \cdot a \cdot a \cdot b \cdot c \cdot c))$$

即:

$$\forall X \neg substr(w, a \cdot a \cdot a \cdot b \cdot c \cdot c)$$

接下来, 以子句标准形为基础进行归结。

(6)$\neg substr(X, a \cdot a \cdot a \cdot b \cdot c \cdot c)$

(7)$\neg substr(X,\ y_1) \vee \neg suffix(y_1, a \cdot a \cdot b \cdot c \cdot c)$

$6,4,\{x \leftarrow X,\ y \leftarrow y_1,\ z \leftarrow a \cdot a \cdot b \cdot c \cdot c\}$

(8)$\neg substr(X,\ a \cdot b \cdot c \cdot c)$

$7,2,\{x \leftarrow a \cdot b \cdot c \cdot c,\ y \leftarrow a,\ y_1 \leftarrow a \cdot b \cdot c \cdot c\}$

(9)$\neg substr(X,\ y_2) \vee \neg suffix(y_2, a \cdot b \cdot c \cdot c)$

$8,5,\{x \leftarrow X,\ y \leftarrow y_2,\ z \leftarrow a \cdot b \cdot c \cdot c\}$

(10)$\neg substr(X,\ a \cdot b \cdot c)$

$$9,3,\{x \leftarrow a \cdot b \cdot c,\ y \leftarrow c,\ y_2 \leftarrow a \cdot b \cdot c\}$$

(11)□

$$10,1,\{x \leftarrow X,\ \boxed{X \leftarrow a \cdot b \cdot c}\}$$

于是, 得到一个替换$\{X \leftarrow a \cdot b \cdot c\}$, 正是这个替换使得目标的否不成立。也就是说替换$\{X \leftarrow a \cdot b \cdot c\}$是满足目标$substr(X, a \cdot a \cdot a \cdot b \cdot c \cdot c)$的一个解。

归结过程不是唯一的, 不同的归结过程可能带来不同的解。事实上, 目标$substr(X, a \cdot a \cdot a \cdot b \cdot c \cdot c)$也确实存在不同的解。

一个**霍恩子句**(Horn Clause)具有以下的标准形式:

$$A \leftarrow B_1, B_2, \cdots, B_n$$

霍恩子句等价于 $A \vee \neg B_1 \vee \neg B_2 \vee \neg \cdots \vee \neg B_n$.

左边的A是头部(Head), 右边$\neg B_i$称为身子(Body).

一个**事实**(Fact)是退化的霍恩子句: $A \leftarrow$.

一个**数据库**(Database)由若干个事实构成。

一个**目标子句**(Goal Clause)是另一种退化的霍恩子句:

$$\leftarrow B_1, B_2, \cdots, B_n.$$

在逻辑编程中, 一个**程序子句**(Program Clause)就是一个霍恩子句。

```
mother_child(juan, ting)./* m1 */

father_child(qiang, ting)./* f1 */
father_child(qiang, mei)./* f2 */
father_child(gang, qiang)./* f3 */

sibling(X, Y) :- parent_child(Z, X), parent_child(Z,
Y). /* s1 */

parent_child(X, Y) :- father_child(X, Y). /* p1 */
parent_child(X, Y) :- mother_child(X, Y). /* p2 */
```

运行:

```
?-sibling(ting, X).
```

其中,数据库包括事实m1和f1-f3; s1-s2和p1是程序子句。

目标等价于$\neg sibling(ting, X)$. 而子句s1等价于

$sibling(X, Y) \vee \neg parent\_child(Z, X) \vee \neg parent\_child(Z, Y)$.

所以目标将与s1进行合一操作, 得到替换:

$$\{X \leftarrow ting, Y \leftarrow X\}.$$

对所有子句进行替换后,继续进行下一步合一操作

……

等归结完成后,可得到替换$\{X \leftarrow mei\}$为所求的解。

不过,除了“mei”之外,也会得到平凡解“ting”.

```
?- sibling(ting, X).
X = ting ;
X = mei ;
X = ting.
```

可通过修改sibling谓词的定义完善此程序。

```
mother_child(juan, ting).
father_child(qiang, ting).
father_child(qiang, mei).
father_child(gang, qiang).
sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y), \+ X=Y.
parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

?- sibling(ting, X).
X = mei ;
false.
```

在增加“\+ X=Y”后，得到唯一解“mei”.