『論理と形式化』 補足資料 (Prolog 演習)

この資料では、Prolog のサンプル・プログラム (basic.pl) について論理的な観点から解説する。

1 事実を単純に羅列する場合: human

問題設定:

- Alice と Bob と Chris は人間である。

ここで必要な言語

- 定数: alice, bob, chris (定数の名前は、小文字で始めることにする)
- 関数記号: (なし)
- 述語記号: human(2 引数)

論理式 (Horn 節) による形式化

$$\texttt{human(alice)} \wedge \texttt{human(bob)} \wedge \texttt{human(chris)}$$

Prolog プログラム

```
human(alice).
human(bob).
human(chris).
```

3 つの節は「かつ (∧)」でつながっていると考えることに注意せよ。

Query の例

```
?- human(X).
X = alice  next
...
```

上記の Query の論理的な意味

$$\exists X.\ \texttt{human}(X)$$

これに対する X= alice という解は以下のように導ける (bob, chris も同様)。

$$\frac{\texttt{human(alice)}}{\exists X.\ \texttt{human}(X)}$$

2 隣接関係: adj と reach

問題設定:

- 東京の (ある方向の) 隣は、茨城および埼玉である。
- 埼玉の (ある方向の) 隣は、栃木である。

- 茨城の (ある方向の) 隣は、埼玉、栃木、福島である。
- 栃木の (ある方向の) 隣は、福島である。
- X の隣に Y があれば、X から Y に到達可能である。
- X の隣に Y があり、Y から Z に到達可能であれば、X から Z に到達可能である。

ここで必要な言語

- 定数: `tokyo`, `saitama`, `ibaraki`, `tochigi`, `fukushima`
- 関数記号: (なし)
- 述語記号: `adj`(2 引数、隣接をあらわす), `reach`(2 引数, 到達可能性をあらわす)

論理式 (Horn 節) による形式化

Horn 節 H1, H2, ..., H9 を以下のように置くと、上記の問題設定は、全体として、

$$H1 \wedge H2 \wedge \cdots \wedge H9$$

として形式化できる。

$$H1 : \texttt{adj}(\texttt{tokyo}, \texttt{ibaraki})$$
$$H2 : \texttt{adj}(\texttt{tokyo}, \texttt{saitama})$$
$$H3 : \texttt{adj}(\texttt{saitama}, \texttt{tochigi})$$
$$H4 : \texttt{adj}(\texttt{ibaraki}, \texttt{saitama})$$
$$H5 : \texttt{adj}(\texttt{ibaraki}, \texttt{tochigi})$$
$$H6 : \texttt{adj}(\texttt{ibaraki}, \texttt{fukushima})$$
$$H7 : \texttt{adj}(\texttt{tochigi}, \texttt{fukushima})$$
$$H8 : \forall X.\forall Y.(\texttt{adj}(X, Y) \supset \texttt{reach}(X, Y)))$$
$$H9 : \forall X.\forall Y.\forall Z.((\texttt{adj}(X, Y) \wedge \texttt{reach}(Y, Z)) \supset \texttt{reach}(X, Z)))$$

Prolog プログラム

```
adj(tokyo,ibaraki).
adj(tokyo,saitama).
adj(saitama,tochigi).
adj(ibaraki,saitama).
adj(ibaraki,tochigi).
adj(ibaraki,fukushima).
adj(tochigi,fukushima).
reach(X,Y) :- adj(X,Y).
reach(X,Z) :- adj(X,Y), reach(Y,Z).
```

Query の例

```
?- reach(tokyo,Y).
...
Y = fukushima next
...
```

上記の Query の論理的な意味

$$\exists Y.\ \texttt{reach}(\texttt{tokyo}, Y)$$

これに対する Y= fukushima という解は以下のように導ける。

$$\dfrac{\dfrac{H9}{(\texttt{adj}(t,i) \wedge \texttt{reach}(i,f)) \supset \texttt{reach}(t,f)} \quad \dfrac{\dfrac{H1}{\texttt{adj}(\texttt{tokyo},\texttt{ibaraki})} \quad \dfrac{H6}{\texttt{reach}(\texttt{ibaraki},\texttt{fukushima})}}{\texttt{adj}(\texttt{tokyo},\texttt{ibaraki}) \wedge \texttt{reach}(\texttt{ibaraki},\texttt{fukushima})}}{\dfrac{\texttt{reach}(\texttt{tokyo},\texttt{fukushima})}{\exists X.\ \texttt{reach}(\texttt{tokyo}, Y)}}$$

途中であまりにも長くなるので、適宜省略した。

3 自然数の足し算 add

問題設定:

- 自然数 X の「次の自然数」を s(X) とあらわす。
- どんな自然数 Y に対しても、$0 + Y = Y$
- どんな自然数 X,Y,Z に対しても、$X + Y = Z$ ならば $s(X) + Y = s(Z)$

形式化:

- Prolog では、関数記号は、普通の意味での「関数」ではない。つまり、a+b を計算して c にしてくれるような「関数の計算」を定義することができない。(実際の Prolog 処理系は、少し拡張されていて、ある程度そういう「関数の計算」をやってくれるのだが、ここでは、Prolog のコア部分だけの話をする。)
- a+b の計算が表現できないので、そのかわりに、「a+b=c を意味する 1 つの述語」として、add を導入する。
- つまり、add(a,b,c) は a+b=c となるとき真で、そうでないとき偽になるように形式化する。
- また、add の定義において、「第一引数 a がどんどん減少する」ように定義するとうまくいく。(定義を展開したとき、いずれ、停止するので。)

この最後のポイントがやや難しい。
たとえば、以下のものは、良い定義であるが、

```
add(0,Y,Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

これを、以下のように定義してしまうと (論理的には「正しい」内容ではあるのだが)、Prolog のプログラムとしてはうまく動かない。

```
add(0,Y,Y).
add(X,S(Y),s(Z)) :- add(X,Y,Z).
```

問題: なぜか考えてみよう？

もちろん、「第 2 引数がどんどん減少する」という定義でもよいので、以下の定義でもよい。

```
add(X,0,X).
add(X,S(Y),s(Z)) :- add(X,Y,Z).
```

以下のようなものは、全然だめである。

```
add(X,0,X).
add(X,Y,Z) :- add(X,s(Y),s(Z)).
```

たしかに 「X+s(Y)=s(Z) ならば X+Y=Z」は成立するのではあるが、これでは、定義をどんどん展開していっても停止しないので、うまくいかない。

参考: ここで書いたことは、実は論理プログラミングだけでなく関数プログラミングでも同様であり、「再帰的定義がいつ停止するか、再帰的定義は何を定義しているか」という問題に関連する。これについて詳しく知りたい人は、「最小不動点」というキーワードで文献を検索することをお勧めする。