『論理と形式化』資料 No.2
導出原理

亀山幸義 (kam[at]cs.tsukuba.ac.jp)

1 Prologの実行

Prolog はホーン節の集合に対する自動証明手続きを、プログラムの処理と見なすことにより得られたプログラミング言語である。この自動証明手続きは、導出 (resolution) と呼ばれる。(Prolog の場合は，特に SLD 導出と呼ばれる。) 導出の簡単な場合について、この資料で説明する。

ゴールとは: 「ゴール」は「解くべき目標」を意味する単語である。Prolog においては、ゴールは、「原子論理式の有限集合 (空集合の場合もある)」として定式化し、これらの原子論理式たち全部を証明することが、処理の目標である。

Prolog の処理系は、与えられたゴールに対して、「導出」を繰返し適用し、以下の 3 つのいずれかとなる。

- ゴールが空集合になれば、実行は成功して終わる。(与えられたゴール中の全ての原子論理式が証明された。)
- 導出をどのように適用しても、ゴールが空集合にならなければ、実行は失敗して終わる。(与えられたゴールを証明することができない。)
- この過程が無限ループになることもあり、その場合は、与えられたゴールが証明可能かどうかがわからない。

Prolog のゴールは、

```
?- P1, P2, ..., Pn.
```

の形で与える。これは、原子論理式の集合 $\{P1, P2, \cdots, Pn\}$ がゴールであることを意味する。これらは、全部証明しないといけないので、論理的には、$P1 \wedge P2 \wedge \cdots \wedge Pn$ ということになる。

ゴールを、ここでは集合 G_i と書くことにする。

2 導出原理 (命題論理の範囲)

ホーン節が命題論理の範囲内であるとする。つまり、述語記号 $P, Q, ...$ がすべて「アリティ0 (引数が 0 個)」の場合である。0 引数の述語 $P()$ は括弧を省略して、P と書く。

2.1 例 1

以下の Prolog プログラムを考える。(各ホーン節には H1, H2, ... という名前をつけた。これは、説明のためのものであり、Prolog プログラムの一部ではない。)

```
H1   P :- Q.
H2   Q :- R.
H3   R.
```

これらを、論理式として表現すると、以下の通りである。

$$
\begin{array}{ll}
H1 & Q \supset P \\
H2 & R \supset Q \\
H3 & R
\end{array}
$$

この例では、項に対する変数 $x, y, \cdots$ が含まれないので、$\forall$ は現れない。

さて、目標はゴール $\{P\}$ を、H1, H2, H3 を使って (仮定して) 証明することであり、NK での証明 (の 1 つ) は以下のようになる。

$$\dfrac{\dfrac{(H1) \quad Q \supset P \qquad \dfrac{(H2) \quad R \supset Q \qquad (H3) \quad R}{Q} \supset\text{-E}}{P} \supset\text{-E}}{}$$

これを Prolog でやってみよう。

ゴール `?- P.` の実行 (導出):

0 最初のゴール: $G_0 = \{P\}$.

1 P に対して (H1) を使って: $G_1 = \{Q\}$.

2 Q に対して (H2) を使って: $G_2 = \{R\}$.

3 R に対して (H3) を使って: $G_3 = \{\}$. ゴールが空集合になったので、実行は成功。

証明図を、**下から**構成していることがわかる。

ゴール`?- P,Q.` の実行 (導出): これは、論理式で書けば、$P \wedge Q$ であるかどうかを導出するものである。

0 最初: $G_0 = \{P, Q\}$.

1 P に対して (H1) を使って: $G_1 = \{Q, Q\}$.

2 Q に対して (H2) を使って: $G_2 = \{R, Q\}$.

3 R に対して (H3) を使って: $G_3 = \{Q\}$.

4 Q に対して (H2) を使って: $G_4 = \{R\}$.

5 R に対して (H3) を使って: $G_5 = \{\}$. ゴールが空集合になったので、実行は成功。

2.2 例 2

以下の Prolog プログラムを考える。

```
H1   Q :- R, S.
H2   R :- S.
H3   S.
```

これらを、論理式として表現すると、以下の通りである。

$$\begin{array}{ll} H1 & (R \wedge S) \supset Q \\ H2 & S \supset R \\ H3 & S \end{array}$$

ゴール`?- Q.` の実行 (導出):

0 $G_0 = \{Q\}$.

1 Q に対して (H1) を使って: $G_1 = \{R, S\}$.

2 R に対して (H2) を使って: $G_2 = \{S, S\}$.

3 S に対して (H3) を使って: $G_3 = \{S\}$.

4 S に対して (H3) を使って: $G_4 = \{\}$. ゴールが空集合になったので、実行は成功。

この導出に対応する証明図は以下の通りである。

$$\dfrac{\dfrac{(H1)}{(R \wedge S) \supset Q} \qquad \dfrac{\dfrac{(H2)}{S \supset R} \quad \dfrac{(H3)}{S}\ \supset\text{-E}}{R} \quad \dfrac{(H3)}{S} \qquad \wedge\text{-I}}{Q}\ \supset\text{-E}$$

2.3 例 3

```
H1   P :- Q, R.
H2   P :- S.
H3   Q :- S.
H4   S.
```

ゴール?- P. の実行 (導出):

0 $G_0 = \{P\}$.
1 P に対して、2 通りの可能性があるので、まずは (H1) を使って: $G_1 = \{Q, R\}$.
2 Q に対して (H3) を使って: $G_2 = \{S, R\}$.
3 S に対して (H4) を使って: $G_3 = \{R\}$.
4 R に対して適用可能なルールがないので、失敗する。(R は、どのホーン節のヘッドとも一致しない。)
5 そこで、先ほど試さなかった選択肢まで戻る。$G_5 = G_0 = \{P\}$.
6 P に対して (H2) を使って: $G_6 = \{S\}$.
7 S に対して (H4) を使って: $G_7 = \{\}$. ゴールが空集合になったので、実行は成功。

この場合に特徴的なのは、「失敗したら戻る」という部分 (上記の 4-5) である。上記で、最初に H1 を使う場合を試し、それが失敗したあと、H2 を使う場合を試した。

なお、ホーン節の「ヘッド」とは、:- の前にある原子論理式のことである。たとえば、P :- Q,R,S. というホーン節のヘッドは P である。ただし、P. の形のホーン節のヘッドは、P とする。ゴールに Q が含まれていて、ヘッドが Q であるホーン節が存在しないなら、そのゴールは導出できない。

バックトラック: 「いくつかの選択肢があるとき、まず、1 つ試して、それが失敗したら、選択したポイントまで戻って次の選択肢を試す (それを、成功するまで、あるいは、選択肢がなくなるまで繰返す)」という仕組みを、バックトラック (backtracking) と呼び、証明探索や、解の探索などの探索アルゴリズムでは頻繁に現れる。Prolog も、バックトラックの機構が組み込まれていて、上記のように、「複数の選択肢を次々と試す」ということが、自動的におこなわれる。

3 演習問題

- 例 2 で、ゴール $G_0 = \{Q, S\}$ に対する導出を書きなさい。
- 以下のプログラムに対して、ゴール $G_0 = \{P\}$ に対する導出を書きなさい。

```
H1   P :- Q, R.
H2   P :- Q, T.
H3   Q :- S.
H4   Q.
H5   T.
```

4　述語論理へ

これまでの例は、命題論理の範囲だった。すなわち、論理式にあらわれる述語は「0 引数述語」のみであった。実際の Prolog は、引数を 1 個以上とる述語を使うことができる。すなわち、述語論理における原子論理式をつかって ホーン節を記述することができる。これは、 すでに add や times などの例で見た通りである。この場合でも、この資料で述べる導出原理は、適用できる (というより、この場合についての自動証明の手続きとして、導出原理が提案された)。述語論理のケースの導出言語についての一般論は、やや難しくなるので、ここれでは、例に基づいた説明を行なう。

4.1　例 4

以下の Prolog プログラムを考える。

```
H1   r(X,Y) :- a(X,Y).
H2   r(X,Z) :- a(X,Y), r(Y,Z).
H3   a(tokyo,ibaraki).
H4   a(ibaraki,fukushima).
```

これらを、論理式として表現すると、以下の通りである。(ここでは Prolog 風に、変数を大文字で、述語記号等を小文字で書いた。)

$$
\begin{array}{ll}
H1 & \forall X \forall Y\ (a(X,Y) \supset r(X,Y)) \\
H2 & \forall X \forall Y \forall Z\ ((a(X,Y) \wedge r(Y,Z)) \supset r(X,Z)) \\
H3 & a(\text{tokyo},\ \text{ibaraki}) \\
H4 & a(\text{ibaraki},\ \text{fukushima})
\end{array}
$$

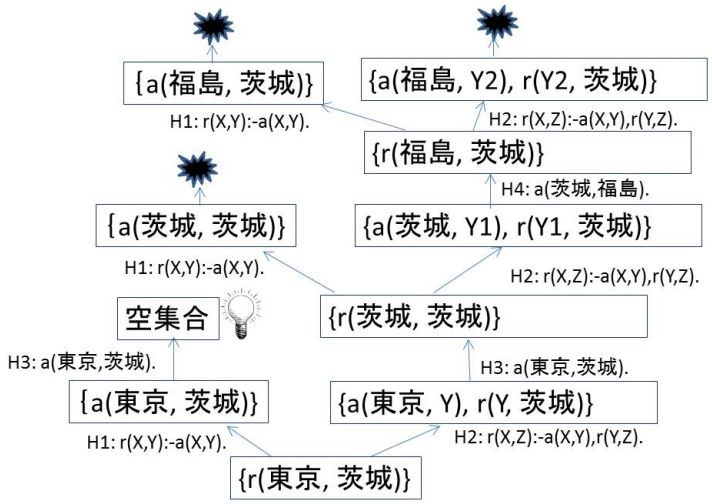
この例では、項に対する変数 x, y, z が含まれるので、$\forall$ が現れている。

r(tokyo, ibaraki) の導出:

H5, H6, H7, H8 から r(tokyo, ibaraki) を導く証明図は以下のようになる。

$$
\dfrac{\dfrac{\dfrac{(H1)}{\forall X \forall Y\ (a(X,Y) \supset r(X,Y))}}{a(\text{tokyo},\ \text{ibaraki}) \supset r(\text{tokyo},\ \text{ibaraki})}\ \forall\text{-E} \quad \dfrac{(H3)}{a(\text{tokyo},\ \text{ibaraki})}}{r(\text{tokyo},\ \text{ibaraki})}\ \supset\text{-E}
$$

Prolog での導出の過程は、以下のようになる。



導出の過程は、証明図を作るための試行錯誤 (バックトラック) の部分を含むため、証明図より若干大きくなる (また、証明図そのものではない。)

4.2 例 5

```
H1   add(0,Y,Y).
H2   add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

これらを、論理式として表現すると、以下の通りである。

$$H1 \quad \forall Y\ (\mathrm{add}(0, Y, Y))$$
$$H2 \quad \forall X \forall Y \forall Z\ (\mathrm{add}(X, Y, Z) \supset \mathrm{add}(s(X), Y, s(Z)))$$

ゴール `?- add(s(0),s(s(0)),W).` に対する導出:

0 $G_0 = \{\mathrm{add}(s(0), s(s(0)), W\}$.

1 $\mathrm{add}(s(0), s(s(0)), W)$ は (H1) のヘッドとはマッチしない。

2 $\mathrm{add}(s(0), s(s(0)), W)$ は (H2) のヘッド $\mathrm{add}(s(X), Y, s(Z))$ と一致はしないが、$X = 0, Y = s(s(0)), W = s(Z)$ **と置くとマッチする。**そこで (H2) を使って、新しいゴールは $G_1 = \{\mathrm{add}(0, s(s(0)), Z\}$ となる。

3 $\mathrm{add}(0, s(s(0)), Z$ は (H1) のヘッドと一致しないが、$Y = s(s(0)), Z = Y$ **と置くとマッチする。**そこで (H1) を使って、新しいゴールは $G_2 = \{\}$ となり、

4 ゴールが空集合になったので、実行は成功して終わる。

5 ここで、最初のゴールに含まれる変数 W は、途中の段階で $W = s(Z) = s(Y) = s(s(s(0)))$ となったので、$W = s(s(s(0)))$ という答えが得られる。

6 なお、上記 3 において (H2) を試すケースをまだやっていないが、これは**変数をどう置いてもマッチしない** ので、そちらは失敗する。(上記の答えを得たあと「n」とタイプしても、それ以外には答えは得られない。)

この述語論理のケースが、命題論理のときと異なるのは、「ゴールの中の原子論理式」と「ルールのヘッド」が完全に一致するかどうかどうかをチェックするのではなく、「変数に対する適当な代入のもとで一致する」かどうかを試している点である。

「適当な代入」を取るというのが不思議なところであるが、NK の証明図にもどしてみると、何をやっているかわかる。上記の導出 (成功したもの) に対応する証明図は以下の通りである。

$$\dfrac{\dfrac{(H2)}{add(0,s(s(0)),s(s(0))) \supset add(s(0),s(s(0)),s(s(s(0))))}\ \forall\text{-E} \quad \dfrac{(H1)}{add(0,s(s(0)),s(s(0)))}\ \forall\text{-E}}{\mathrm{add}(s(0),s(s(0)),s(s(s(0))))}\ \supset\text{-E}$$

「変数に対する適当な代入」というのは、証明図では、「$\forall$ の除去規則 ($\forall$-E 規則)」である。この場合、除去される変数 (もともと、$\forall$ で束縛されていた変数) には、どんな項を代入してもよいので、「適当な」代入となるのである。

補足. ここではきちんと述べなかったが、ゴール中に含まれる変数 W は、「すべての W」という意味ではなく、「ある W」という意味である。すなわち、上記の証明図は、より厳密にすると、以下のように書くべきである。

$$\dfrac{\dfrac{\dfrac{(H2)}{add(0,s(s(0)),s(s(0))) \supset add(s(0),s(s(0)),s(s(s(0))))}\ \forall\text{-E} \quad \dfrac{(H1)}{add(0,s(s(0)),s(s(0)))}\ \forall\text{-E}}{\mathrm{add}(s(0),s(s(0)),s(s(s(0))))}\ \supset\text{-E}}{\exists W.\ \mathrm{add}(s(0),s(s(0)),W)}\ \exists\text{-I}$$

5 Prologは一体何を計算しているか？

例 2 のプログラムの先頭に、`P :- P.` というホーン節を追加したとする。これは論理的には $P \supset P$ という (証明可能な) 論理式に対応するので、これを加えても、まったく問題なさそうである。しかし、これを追加すると、導出は停止しない。つまり、$G_0 = \{P\}$ というゴールに対して、Prolog の処理は停止しない。なぜか。

あるいは、add の定義を少し変更して、

```
H1   add(0,Y,Y).
H2'  add(X,Y,Z) :- add(s(X),Y,s(Z)).
```

としても正しいようにみえるが、このようにすると、add の計算はうまく行かない。

この現象は、導出原理により理解することができる。つまり、導出原理では、ゴールの中の原子論理式を、ルール H1,H2' により書き換えていくのだが、この際に「ルールのヘッドを、ボディに置きかえる」という方向に計算が進む。(ここで、ルールのボディとは、:- の右側の部分のことで、上記 H2' でいえば、add(s(X),Y,s(Z)) である。) よって、ルールのヘッドよりボディが、「より複雑」になっていれば、導出原理による計算は止まらない。一方、「より簡単」になっていれば、導出原理による計算は止まる (可能性がある)。

一般に Prolog のプログラムの計算は、「ヘッドをボディに書きかえる」という操作を繰返しておこなった極限として、定めることができる。

より詳細は、参考書等を参考にして自力で勉強されたい。

6 まとめ

- 論理プログラミング: 「推論」過程を「計算」と見なす。
- ホーン節: 一階述語論理の論理式のうち、ある特定の形のもの
- Prolog: ホーン節に対する効率良い推論手続き (導出原理) に基づくプログラミング言語

Prolog プログラムは、論理式としてのプログラムやゴールのみを記述し、そのゴールをどうやって推論するかを、プログラム中には一切書かない (処理系が勝手に探索してくれる) という意味で、「宣言型プログラミング」をおこなっている。一方で、C や Java などのプログラムは、「解を求めるための手順」を記述するので、「手続き型プログラミング」とよばれる。