『論理と形式化』資料 No.9
導出原理

亀山幸義 (kam[at]cs.tsukuba.ac.jp)

1 Prolog プログラムの実行＝ホーン節に対する自動証明

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 (selective linear derivation) 導出と呼ばれる。

演習問題 0

以下のプログラムで定義される述語 `foo` はどういう意味であるか (`foo(X,Y,Z)` が真となる X,Y,Z はどういう関係があるか) Prolog 処理系でいろいろな引数に対してテストした上で答えなさい。(テスト結果も提出すること)

```
geq(X,0).
geq(s(X),s(Y)) :- geq(X,Y).
lt(0,s(X)).
lt(s(X),s(Y)) :- lt(X,Y).
sub(X,0,X).
sub(s(X),s(Y),Z) :- sub(X,Y,Z).
foo(X,Y,s(Z)) :- geq(X,Y),sub(X,Y,W),foo(W,Y,Z).
foo(X,Y,0) :- lt(X,Y).
```

2 Prolog の実行と証明探索: 命題論理の範囲

ここでは、命題論理の範囲で考える。つまり、述語記号 $P, Q, ...$ は「アリティ 0 (引数が 0 個)」とする。0 引数の述語 $P()$ は括弧を省略して、P と書く。

2.1 例 1

Prolog プログラム:

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

ただし、SWI-Prolog の処理系にかけるときは、大文字の P,Q,R を小文字の p,q,r として入力すること。

上記のプログラムを論理式として表現:

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

H1,H2,H3 を使った (仮定した)P の証明:

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

これを Prolog でやってみよう。

ゴール `?- P.` の実行:

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。

質問: 証明との対応を考えなさい。

ゴール`?- P,Q.` の実行:

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。

質問: 証明との対応を考えなさい。

2.2 例 2

Prolog プログラム:

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

論理式として表現：

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

ゴール`?- Q.` **の実行:**

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。

対応する証明:

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

2.3　**例** 3

Prolog プログラム:

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

ゴール`?- P.` **の実行:**

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。

ホーン節のヘッド: ヘッドは、`:-` の前にある原子論理式のこと。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。

導出に使うホーン節の選択: ゴール中の `Q` を選択したとき (それについて実行を進めるとき)、ヘッドが `Q` であるホーン節の中から 1 つを選ぶ (ヘッドが `Q` でないホーン節を使って 1 ステップで `Q` を導くことはできない。)

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 (backtrack) 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 処理系にも、バックトラックの機構が組み込まれている。

演習問題 1

- 例 2 において、ゴール $G_0 = \{Q, S\}$ を実行した過程を、上記と同様に書きなさい。また、実行が成功するとき、対応する証明を書きなさい。
- 以下のプログラムに対して、ゴール $G_0 = \{P\}$ に対する実行の過程と、それが成功するときの証明を書きなさい。

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

3 Prolog の実行と証明探索: 一階述語論理の範囲

3.1 例 4

Prolog プログラム:

```
H1   add(0,Y,Y).
H2   add(s(X),Y,s(Z)) :- add(X,Y,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 $\mathrm{add}(s(X), Y, s(Z))$ と完全一致はしないが、代入 $[X := 0, Y := s(s(0)), W := s(Z)]$ によりマッチする。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) のヘッドと完全一致はしないが、代入 $[Y := s(s(0)), Z := s(s(0))]$ によりマッチする。そこで (H1) を使って、新しいゴールは $G_2 = \{\}$ となる。

4 ゴールが空集合になったので、成功して終わる。

5 途中で出てきた代入たちを合成することにより、$W = s(s(s(0)))$ という答えを得る。

この述語論理のケースが、命題論理のケースと異なるのは、「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」については後で説明する。

上記で、成功した実行に対応する証明は以下の通りである。

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

補足. ゴールに含まれる変数 W は存在記号で束縛されているので、本来なら、上記の証明は以下のように書くべきである。

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

3.2 例 5

例 4 と同じプログラムとする。ゴール `?- add(V,s(0),s(s(0))).` に対する実行:

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

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

2 $\mathrm{add}(V, s(0), s(s(0)))$ は、(H2) のヘッド $\mathrm{add}(s(X), Y, s(Z))$ と、代入 $[V := s(X), Y := s(0), Z := s(0)]$ によりマッチする。新しいゴールは $G_1 = \{\mathrm{add}(X, s(0), s(0))\}$ となる。

3 $\mathrm{add}(X, s(0), s(0))$ は、(H1) のヘッド $\mathrm{add}(0, Y, Y)$ と、代入 $[X := 0, Y := s(0)]$ によりマッチする。

4 ゴールが空集合になったので、成功して終わる。

5 途中で出てきた代入により、$V = s(0)$ という答えを得る。

演習問題 2

- 例 5 において、ゴール $G = \{add(s(s(V)), s(0), W)\}$ を実行した過程を、上記と同様に書きなさい。また、実行が成功するとき、対応する証明を書きなさい。
- ゴール $G' = \{add(s(s(V)), s(0), s(V))\}$ を実行した過程を、上記と同様に書きなさい。また、実行が成功するとき、対応する証明を書きなさい。

4 単一化 (unification)

Prolog プログラムの実行において、「ゴールとヘッドが代入によりマッチする」という判定が必要だった。これは、単一化 (unification) と呼ばれる手続きである。

単一化問題の定義: 2 つの項 $p(t_1, \cdots, t_n)$ と $p(s_1, \cdots, s_n)$ の単一化問題とは、以下のような代入 θ が存在するかどうかを判定する問題のこと。

$$(\theta(t_1) = \theta(s_1)) \ \wedge \ \cdots \ \wedge \ (\theta(t_n) = \theta(s_n))$$

ただし、$\theta(t)$ は、項 t に代入 θ を適用して得られた項のこと。

代入の例: 代入 $[X := 0, Y := s(Z)]$ を項 $add(X, s(Y), s(Z))$ に適用すると、$add(0, s(s(Z)), s(Z))$ を得る。

単一化の計算例:

項 1	項 2	項 1=項 2 の単一化	得られた代入
`add(0,X,X)`	`add(s(0),Y,Y)`	失敗	-
`add(0,X,X)`	`add(Y,0,s(0))`	失敗	-
`add(0,X,X)`	`add(Y,s(0),s(Z))`	成功	`[X:=s(0),Y:=0,Z:=0]`
`add(0,X,X)`	`add(Y,s(W),s(Z))`	成功	`[X:=s(W),Y:=0,Z:=W]`
`add(s(X),Y,s(Z))`	`add(Z,X,Y)`	失敗	-
`add(0,0,0)`	`mul(0,0,0)`	失敗	-

単一化に関する性質:

- 与えられた 2 つの項に対する単一化が成功するとき、最も一般的な代入 (most general unifier, mgu) が存在する。
- mgu は (実質的には) 一意である。
- 単一化問題を有限時間で解き、かつ、単一化が成功するときは mgu を計算するアルゴリズムが存在する。

単一化は、Prolog の実行だけでなく、様々な記号処理に出現する有用な手続きである。(例: 型推論アルゴリズム)

5 導出原理 (resolution principle)

Prolog の実行の 1 ステップの論理的意味は、$A_1 \wedge \cdots \wedge A_n \supset B$ と $A_1, \cdots, A_n$ から B を得るという「かつの導入」と「ならばの除去」の組合せであった。

実はこのように簡単なのは、ホーン節に限定したからであり、一般の節の場合に対応した 1 ステップ分の実行は、導出原理、あるいは、導出と呼ばれる。

5.1 導出原理 (命題論理版)

ここでは、簡単のため、命題論理版の導出原理を説明する。

節 $C_1 = (A_1 \vee \cdots \vee A_n)$ と節 $C_2 = (B_1 \vee \cdots \vee B_m)$ が与えられたとする。ここで、A_i と B_j は、命題変数、あるいは、命題変数に否定記号をつけたものとする。このとき、ある A_i が命題変数 P であり、また、ある B_j が $\neg P$ であったとすると、C_1 と C_2 から、導出により得られる節は、

$$C_3 = (A_1 \vee \cdots \vee A_{i-1} \vee A_{i+1} \vee A_n \vee B_1 \vee \cdots \vee B_{j-1} \vee B_{j+1} \vee B_m)$$

である。

要するに、C_1 から A_i(これは P である) を除去し、C_2 から B_j(これは $\neg P$ である) を除去し、それらを $\vee$ でつないだ節を作るという操作である。

例: 節 $P \vee Q \vee \neg R$ と節 $\neg S \vee \neg P \vee \neg T$ に導出原理を適用すると、節 $Q \vee \neg R \vee \neg S \vee \neg T$ が得られる。

Prolog 実行は、ホーン節に対する証明探索手続きであり、これは、導出原理の特殊な場合である。これを説明するためには、「反駁による証明 (証明した論理式であるゴールに否定をつけた論理式を仮定して、矛盾を

導くことにより、ゴールを証明する方法)」を理解する必要があるので、ここでは言及に留める。

演習問題 3

- 節 $C_1 = (P \vee \neg A \vee \neg B)$ と節 $C_2 = (\neg C \vee \neg P \vee \neg D)$ に対して導出原理を適用して得られる節 C_3 を求めなさい。
- $C_1 \wedge C_2 \supset C_3$ が、命題論理で証明可能なことを示しなさい。

6 Prolog の不完全なところ

実は、Prolog 処理系 (標準的な処理系) は、ホーン節に対する証明探索手続きとして、完全でない。

導出原理そのものは、節に対する証明探索手続きとして「完全」である (証明できる場合はいつでも、導出原理をうまい順番で適用することにより、証明できる)。しかし、標準的な Prolog 処理系では、(1) ゴールの中を左から順に選択、(2) プログラムを上から (左から) 順に選択するという特定の実行順序を取っている。このため、成功すべき場合でも、Prolog の実行では無限ループすることがある。

例:

```
H0   add(X,Y,Z) :- add(X,Y,Z).
H1   add(0,Y,Y).
H2   add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

H0 は、$\forall X\ \forall Y\ \forall Z\ (add(X,Y,Z) \supset add(X,Y,Z))$ という証明可能な式であるので、H1,H2 から証明できる論理式の集合と、H0,H1,H2 から証明できる論理式の集合は同じである。しかし、Prolog のプログラムとしては、H0 が先頭にあると `?- add(a,b,c).` の形のゴールの計算は常に H0 を選択してしまい、停止しない。

7 (参考) Answer Set Programminug (ASP)

Prolog の派生的な論理プログラミング言語の 1 つに解集合プログラミング (ASP) がある。これは、Prolog とまったく同様に、プログラムは、いくつかのホーン節からなるのであるが、大きな違いとして、変数の変域が有限集合ということがある。(前の例では、0, $s(0)$, $s(s(0))$, $\cdots$, という無限個の項があったが、ASP では、この s のような関数記号は許されない。)

この性質のため、ホーン節 (一階述語論理の一部の論理式) であるにもかかわらず、命題論理の論理式として表現しなおすことができるので、先週学習した SAT ソルバにかけることで、証明探索をすることができる。

例: $\forall X\ (add(0,X,X))$ というホーン節は、X が動く範囲が、$0,1,2$ だけであれば、$add(0,0,0) \wedge add(0,1,1) \wedge add(0,2,2)$ と同値である。

ASP は、論理式でプログラムを書くことができて、様々な問題の制約を自然に表現できるという Prolog の特徴を引き継ぎつつ、与えられたゴールを満たす解があるかどうかが、必ず有限時間で判定できるという Prolog にはない特徴があり、現在も研究されている。

8 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

宣言型プログラム vs 手続き型プログラム: 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。