知識工学 (第 9 回)

二宮 崇 (ninomiya@cs.ehime-u.ac.jp)

一階述語論理 (9)

§9.5.4 融合法の完全性

融合法の完全性について説明をする。融合法の完全性を信じる人は読み飛ばしても良い。融合法は反駁完全(refutation completeness)である。すなわち、もし、一階述語論理の論理式が充足不能であるなら、融合法は必ず矛盾を導出することができる。これは背理法を想定しており、$KB \models Q$という伴意関係がもし成り立つならば、融合法は必ず証明に成功する、ということを意味している。

融合法は、$KB \models Q$が成り立つかどうかを示すために、$KB' = KB \wedge \neg Q$とし、KB'から矛盾(false)が導出されるかどうか調べる。KB'から矛盾が導出されれば背理法の仮定より$KB \models Q$が成り立つ。完全性を示すというここでの目標は、「KB'が充足不能であるならば、有限ステップで融合法は証明を終えることができる」ということを示すことである。

証明は次のステップにより示される。

1. エルブランの定理により充足不能な一階述語論理式には充足不能な命題論理式が存在することを示す。

2. 命題論理化された一階述語論理は、命題論理のための融合法により、有限時間でその判定を行うことができる。

3. 持ち上げ補題(lifting lemma)により、単一化を用いた一階述語論理のための融合法においても、命題論理のための融合法と等価な証明が存在することを示す。

○エルブランの定理

融合法の完全性を示す上で最も重要な箇所はエルブランの定理である。エルブランの定理を説明するためにはまずいくつかの概念を導入しないといけない。

エルブラン領域 (Herbrand universe): 融合法では最初に一階述語論理の論理式を CNF に変換し、節集合を生成する。節集合Sに対し、Sのエルブラン領域H_Sは以下から作られる全ての基礎項の集合である。

1. S中の関数記号。

2. S中の定数記号。もし定数がないのなら定数記号A。

例: $S = \{\neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B)\}$とする(節を一つだけ含む)。

$$H_S = \{A, B, F(A, A), F(A, B), F(B, A), F(B, B), F\big(A, F(A, A)\big), \cdots\}$$

つまり、節集合中で使われている全ての関数記号と定数記号を使って作ることのできる可能な項を全て列挙しているのがエルブラン領域である。

飽和(saturation): Sを節集合とし、Pを基礎項の集合とする。S中の変数に対し、矛盾が起きない可能な基礎項を代入して基礎節を得ることを飽和という。Sに対しPから選んで飽和させるとき、$P(S)$と書く。

エルブラン基底(Herbrand base): 節集合Sのエルブラン領域に対する飽和はSのエルブラン基底と呼ばれ、$H_S(S)$と書かれる。

例: $S = \{\neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B)\}$とする(上記の例と同じ)。

$$H_S(S) = \{\neg P\big(A, F(A, A)\big) \vee \neg Q(A, A) \vee R(A, B),$$

$$\neg P\big(B, F(B, A)\big) \vee \neg Q(B, A) \vee R(B, B),$$

$$\neg P\big(F(A, A), F(F(A, A), A)\big) \vee \neg Q(F(A, A), A) \vee R(F(A, A), B),$$

$$\neg P\big(F(A, B), F(F(A, B), A)\big) \vee \neg Q(F(A, B), A) \vee R(F(A, B), B), \cdots\}$$

エルブランの定理(Herbrand's theorem): 節集合Sが充足不能であるなら、充足不能な$H_S(S)$の有限部分集合が存在する。

節集合Sが充足不能なら、S中の変数にエルブラン領域の項を代入した基礎節(エルブラン基底)の中に充足不能な基礎節が存在する、ということである。つまり、Sが充足不能なら、うまくエルブラン領

域の項を代入していけば、有限時間で充足不能なSの基礎節が得られる、ということである。基礎節には変数が含まれないので、命題論理と等価な論理式となる。命題論理における融合法は完全であることがわかっているので、元々の節集合が充足不能であるなら、矛盾を必ず導出することができる。

〇持ち上げ補題

命題論理に対する融合法ではなく、一階述語論理の融合法に対しても完全性が成り立つことを示さないといけない。そのため持ち上げ補題と呼ばれる補題がある。

持ち上げ補題(lifting lemma): C_1とC_2を変数を共有しない二つの節とし、C_1'とC_2'をC_1とC_2の基底例とする。もし、C'がC_1'とC_2'の融合であるとすると、次のようなCが存在する。(1) CがC_1とC_2の融合であり、(2)C'がCの基底例である。

基底例	節
$\dfrac{C_1' \quad C_2'}{C'}$ (融合規則)	$\dfrac{C_1 \quad C_2}{C}$ (融合規則)

例:

$C_1 = \neg P(x, F(x, A)) \vee \neg Q(x, A) \vee R(x, B)$

$C_2 = \neg N(G(y), z) \vee P(H(y), z)$

$C = \neg N\big(G(y), F(H(y), A)\big) \vee \neg Q(H(y), A) \vee R(H(y), B)$

$C_1' = \neg P(H(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B)$

$C_2' = \neg N(G(B), F(H(B), A)) \vee P(H(B), F(H(B), A))$

$C' = \neg N(G(B), F(H(B), A)) \vee \neg Q(H(B), A) \vee R(H(B), B)$

従って、命題論理の融合法と一階述語論理の融合法には対応関係があり、命題論理の融合法において矛盾が導出されるなら、一階述語論理の融合法においても矛盾が導出される。従って、与えられた一階述語論理の論理式がもし充足不能であれば、一階述語論理の融合法において必ず矛盾が導出される。

§9.5.5 等号 (equality)

今まで等号に関する説明がなかったが、等号を扱うための 3 つのアプローチが存在する。1 つ目の方法は、等号無し一階述語論理に等号の公理を導入する方法である。

(同値関係の公理)

$$\forall x\ x = x$$

$$\forall x, y\ x = y\ \Rightarrow y = x$$

$$\forall x, y, z\ \ x = y \wedge y = z\ \Rightarrow x = z$$

(述語の同値関係)

$$\forall x, y\ \ x = y\ \Rightarrow \big(P_1(x) \Leftrightarrow\ P_1(y)\big)$$

$$\forall x, y\ \ x = y\ \Rightarrow \big(P_2(x) \Leftrightarrow\ P_2(y)\big)$$

$$\dots$$

(関数の同値関係)

$$\forall w, x, y, z\ \ w = y \wedge x = z\ \Rightarrow \big(F_1(w, x) = F_1(y, z)\big)$$

$$\forall w, x, y, z\ \ w = y \wedge x = z\ \Rightarrow \big(F_2(w, x) = F_2(y, z)\big)$$

$$\dots$$

このように公理を加えることで等号付き一階述語論理を実現することができる。また、上記の公理は等号の性質に関するわかりやすい説明ともなっている。

2 つ目の方法は、等号を扱う推論規則を追加する方法である。ここでは二つ紹介する。

デモジュレーション (demodulation): $x = y$という節があれば、そのほかの節αにおいて含まれるxを全てyに置き換える方法。例えば、

$$Father\big(Father(x)\big) = PaternalGrandfather(x)$$

$$Birthdate\big(Father\big(Fahter(Bella)\big), 1926\big)$$

であったとき、

$$Birthdate(PaternalGrandfather(Bella), 1926)$$

と推論する方法である。次のようにデモジュレーションは定義される。任意の項x, yに対し、

$$\frac{x = y \qquad m_1 \vee \cdots \vee m_n}{SUB(SUBST(\theta, x), SUBST(\theta, y), m_1 \vee \cdots \vee m_n)}$$

ただし、$UNIFY(x, z) = \theta$で、zはm_iに出現する任意の項である。$SUB(x, y, m)$は、m中に含まれるxをyに置き換える操作である。

パラモジュレーション (paramodulation): デモジュレーションは$x = y$という形の節にしか適用できないが、これを一般の節の形に拡張することができる。任意の項x, yに対し、

$$\frac{l_1 \vee \cdots \vee l_k \vee x = y \qquad m_1 \vee \cdots \vee m_n}{SUB(SUBST(\theta, x), SUBST(\theta, y), SUBST(\theta, l_1 \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_n)}$$

ただし、$UNIFY(x, z) = \theta$で、zはm_iに出現する任意の項である。パラモジュレーションは一階述語論理に対して完全な推論を与える。

3 つ目の方法は、単一化アルゴリズムを拡張して、等号推論を扱う方法である。例えば、$1 + 2 = 2 + 1$は一般に単一化できないが、$x + y = y + x$ということが成り立つことがわかっていれば単一化に成功する。

世の中に一階述語論理の定理証明器が多数存在し、優れたフリーソフトも多数存在している。これらを実際に用いれば、一階述語論理の推論の挙動を理解する大きな助けになるだろう。Otter, Prover9(Otter の後継), SPASS, E, Vampire, Waldmeister などがある。使いやすさの点から最初は Prover9 がおすすめである。

§9.4.2 論理プログラミング（logic programming）

一階述語論理をプログラミング言語として用いることができるように一階述語論理の表現に制限をかけ、推論をコントロールし、実行効率を非常に高くしたプログラミング言語を論理型プログラミング言語と呼ぶ。代表的な言語として Prolog が存在する。Prolog では次の形の節を持つ一階述語論理のみ扱う。

$$l_1 \wedge l_2 \wedge \cdots \wedge l_n \Rightarrow h$$

ただし、各l_iおよびhは正リテラルである。これらの節はホーン節と呼ばれる。$l_1 \wedge l_2 \wedge \cdots \wedge l_n$はボディ(body)と呼ばれ、$h$はヘッド(head)と呼ばれる。ボディがない節を定義することができ、ヘッドhだ

けの節を定義することもできる($true \Rightarrow h$という形の節と考えても良い)。ヘッドだけから成る節はファクト(fact)と呼ばれる。ホーン節に対する証明は簡単で、後ろ向き推論と呼ばれる推論方法で質問から順に証明をすることができる。Prolog では、証明に失敗したときは、証明を巻き戻して他の解を探しにいくことを行う。この巻き戻しのことをバックトラックという。

Prolog には次のような制限や機能がついている。

単一名仮説 (unique names assumption): 異なる定数は異なるオブジェクトを指す。つまり、$John = Daniel$というのは常に単一化に失敗する。また、関数も同様であり、項に対する等号は、$f(x_1, \cdots, x_n) = f(y_1, \cdots, y_n)$かつ$x_1 = y_1, \cdots, x_n = y_n$となるときにのみ成り立つ。つまり、Prolog における等号の処理は二つの項を単一化することに等しい。

失敗による否定(negation as failure)と閉世界仮説(closed-world assumption): 正リテラルのみしか扱えないのは不便である。そこで、Prolog には失敗による否定が導入されている。これはボディに負リテラル$\neg l$が含まれるとき、lの証明をまず行い、もし、lの証明ができなかったときは$\neg l$の証明ができたことにし、lの証明ができたときには$\neg l$を証明できなかったことにする方法である。一階述語論理の否定とは異なる否定であることに注意。また、この結果、知識ベースに定義されていないファクトは全て否定されたファクトとして定義されていることになる。これは閉世界仮説と呼ばれる。

カット(cut): 証明に失敗したとき、証明を巻き戻して(バックトラック)、他の解を探しにいくことになるが、そのバックトラックを局所的に強制的に止めることができる。この処理のことをカットという。カットを用いることで、プログラミング言語における if 文のような条件分岐処理を実現することができる。

Prolog ではホーン節$l_1 \wedge l_2 \wedge \cdots \wedge l_n \Rightarrow h$を次のように表す。

$$h \text{ :- } l_1, \ l_2, \ \cdots, \ l_n .$$

ファクトは次のように表す。

$$h .$$

変数の頭文字は大文字とし、述語、定数、関数の頭文字は小文字となるので注意。

例：

```
male(namihei).
male(katsuo).
male(tarao).
female(fune).
parent(namihei, katsuo).
parent(fune, katsuo).
father(X, Y) :- parent(X, Y), male(X).
```

実行例：

```
> ?- father(namihei, katsuo).
yes
> ?- father(namihei, tarao).
no
> ?- father(X, katsuo).
X: namihei
Enter ';' for more choices, otherwise press ENTER --> ;
no
> ?- parent(X, katsuo).
X: namihei
Enter ';' for more choices, otherwise press ENTER --> ;
X: fune
Enter ';' for more choices, otherwise press ENTER --> ;
no
```

例: リストの append

```
append([], Y, Y).
append([A|X], Y, [A|Z]) :- append(X, Y, Z).
```

実行例:

```
> ?- append([1,2,3], [4,5,6], X).
X: < 1, 2, 3, 4, 5, 6 >
Enter ';' for more choices, otherwise press ENTER --> ;
no
> ?- append(X, Y, [1,2,3,4]).
X: < >
Y: < 1, 2, 3, 4 >
```

```
Enter ';' for more choices, otherwise press ENTER --> ;
X: < 1 >
Y: < 2, 3, 4 >
Enter ';' for more choices, otherwise press ENTER --> ;
X: < 1, 2 >
Y: < 3, 4 >
Enter ';' for more choices, otherwise press ENTER --> ;
X: < 1, 2, 3 >
Y: < 4 >
Enter ';' for more choices, otherwise press ENTER --> ;
X: < 1, 2, 3, 4 >
Y: < >
Enter ';' for more choices, otherwise press ENTER --> ;
no
```