知識工学 (第 7 回)

二宮 崇 (ninomiya@cs.ehime-u.ac.jp)

一階述語論理 (9)

§9 一階述語論理による推論

§9.1 命題論理 対 一階述語論理

一階述語論理においてはモデルが無数に存在するためモデル検査による論理的同値関係の判定を行うことはできない。モデルを有限に限ったとしても、その解釈は爆発的数の解釈が存在することになる。

命題論理と一階述語論理における論理的同値関係と伴意関係をまとめると次のようになる。

	命題論理	一階述語論理
論理的同値関係$\alpha \equiv \beta$	真理値表が同一になる	
	$\alpha \Leftrightarrow \beta$がトートロジー	$\alpha \Leftrightarrow \beta$がトートロジー
	$\alpha \vDash \beta$かつ$\beta \vDash \alpha$	$\alpha \vDash \beta$かつ$\beta \vDash \alpha$
伴意関係$\alpha \vDash \beta$	$\alpha \equiv \alpha \wedge \beta$	$\alpha \equiv \alpha \wedge \beta$
	$\alpha \Rightarrow \beta$がトートロジー	$\alpha \Rightarrow \beta$がトートロジー

モデル検査を用いることができなくても、その伴意関係の公理を与えることは出来そうである。特に、αが真であるとき、βも常に真であるとき、伴意関係が成り立つといえることに注意しよう。命題論理において真理値表を用いずとも推論ができたように、一階述語論理においても論理的同値関係、伴意関係の公理を定義すればそれらを用いることにより推論が可能となる。

§9.1.1 限量子に対する推論規則

基礎項 (ground term): 変数を含まない項。

代入 $SUBST(\theta, \alpha)$: 文αに代入θを適用する。θは$\{v_1/g_1, \cdots, v_n/g_n\}$という形をしており、変数$v_i$に基礎項$g_i$を代入することを意味する。

全称具体化 (universal instantiation, UI): $\forall v\ \alpha$のvを任意の基礎項で置き換えた文αを推論する($\forall v\ \alpha$とvを任意の基礎項で置き換えた文αは伴意関係にある)。

$$\frac{\forall v\ \alpha}{SUBST(\{v/g\},\alpha)}$$

例:

$$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$

に対し、次の各式を伴意できる。(それぞれ、$\{x/John\}$、$\{x/Richard\}$、$\{x/Father(John)\}$による代入)

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$$

$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$$

$$King(Father(john)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$$

…

これは要するに、任意のvに対して文αが成り立つわけだから、vに任意の項を代入しても良い、というわけである。

存在具体化 (existential instantiation, EI): $\exists v\ \alpha$のvを新しい定数kで置き換えた文αを推論する($\exists v\ \alpha$とvを新しい定数kで置き換えた文αは伴意関係にある)。

$$\frac{\exists v\ \alpha}{SUBST(\{v/k\},\alpha)}$$

例:

$$\exists x\ Crown(x) \wedge OnHead(x, John)$$

に対し、次の文を伴意できる。

$$Crown(C) \wedge OnHead(C, John)$$

ただし、Cは知識ベースのどこにも現れていない新しい定数でなければならない。

これは要するに、あるvに対して、文αが成り立つわけだから、そのあるvに定数Cという名前をつけてやった、ということである。この新しい定数はスコーレム定数 (Skolem constant) と呼ばれる。

§9.1.2 命題論理への帰着

全称具体化と存在具体化を用いると、限量子のついた文から限量子のついていない文を推論することができる。伴意関係の性質から、推論によって得られた、限量子のついていない新しい文を知識ベースに加えることができる。限量子のついていない文だけを用いて推論することにより、一階述語論理を命題論理に帰着することができ、命題論理の推論問題として一階述語論理の問題を解くことができる。

限量子のつかない原子文は変数を持たない原子基礎文 (ground atomic sentence)となる。異なる原子基礎文 (ground atomic sentence) には異なる真偽値を自由に割り当てられる、と考えるならば、原子基礎文は命題論理と等価である。従って、一階述語論理の知識ベースに対し、全称具体化と存在具体化を適用することで、限量子のついていない知識ベースを推論することができ、限量子のついていない知識ベースに対し命題論理の推論を適用することで、一階述語論理の推論を行うことができる、ということである。

例: 次の知識ベースが成り立つとき、$Evil(John)$が成り立つかどうか？

R_1: $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

R_2: $King(John)$

R_3: $Greedy(John)$

R_4: $Brother(Richard, John)$

このとき、UI より、

R_5: $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$

が得られ、R_2とR_3とR_5に対し、モーダスポーネンスを適用し、

R_6: $Evil(John)$

が得られる。

この手法は命題論理化 (propositionalization) と呼ばれる。いかなる一階述語論理の知識ベースおよび質問は命題論理に置き換えることができるような気がするかもしれないが、実はそれは難しい。問題は、関数を含む項は無限に入れ子になった項(例えば、$Father(Father(\ldots\ (Father(John))\ \ldots))$

を作ることができてしまう、ということだ。つまり、UI によって、無限に新しい知識を生成することができてしまう。しかし、エルブランの定理という有名な定理のおかげで、伴意関係にある知識ベースと質問に対しては有限の大きさの命題論理に必ず置き換えることができることがわかっている。つまり、伴意関係にある知識ベースと質問に対しては、有限時間でその関係が成り立つかどうか判定できる、ということである。しかし、逆に言えば、伴意関係にあるかどうかを有限時間で判定するアルゴリズムは存在しない、ということでもある。これはチューリング機械の停止問題と同じであり、一階述語論理に対する伴意関係の問題は、半決定的 (semidecidable) である。すなわち、伴意関係にある文に対しては yes と答えるアルゴリズムが存在するが、伴意関係にない文に対して no と答えるアルゴリズムは存在しない。

§9.2 単一化と持ち上げ

§9.1.2 で示した例は答えまですぐにたどり着く推論を行ったが、例えば、次のような推論をしたとしたらどうだろう？

$$King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard)$$

この知識が何か害をなすわけではないが、無駄な推論といえよう。そこで、一階述語論理の形のままで推論する規則をいくつか用意することで無駄な推論をしなくてすむように工夫しよう。

§9.2.1 一階推論規則

一般化モーダスポーネンス: ある原子文p_i, p'_iとqに対して、全てのiに対して$SUBST(\theta, p'_i) = SUBST(\theta, p_i)$となる代入が存在するとき、次の推論が成り立つ。

$$\frac{p'_1, p'_2, \cdots, p'_n, (p_1 \land p_2 \land \cdots \land p_n \Rightarrow q)}{SUBST(\theta, q)}$$

例

R_1: $\forall x\ King(x) \land Greedy(x) \Rightarrow Evil(x)$

R_2: $King(John)$

R_3: $\forall y\ Greedy(y)$

この知識ベースに対し、

$$\frac{King(John), Greedy(y), King(x) \land Greedy(x) \Rightarrow Evil(x)}{SUBST(\{x/John, y/John\}, Evil(x)) = Evil(John)}$$

となり、$Evil(John)$を直接推論することができる。命題論理化に代わるこの推論規則はモーダスポーネンスの持ち上げ (lifting) 版と呼ばれる。

§9.2.2 単一化

持ち上げ推論規則を適用するには異なる論理表現を同一にする代入を見つけないといけない。この処理は単一化 (unification) と呼ばれる。単一化のアルゴリズムには様々な手法があるが、グラフ単一化の手法が一般的である。いずれにせよ、単一化のアルゴリズム$UNIFY$は二つの項p, qを受け取り$SUBST(\theta, p) = SUBST(\theta, q)$となる代入$\theta$を返す(または$SUBST(\theta, p)$を返す)。つまり、

$$UNIFY(p, q) = \theta \ where \ SUBST(\theta, p) = SUBST(\theta, q)$$

である。

例

R_1: $Knows(John, Jane)$

R_2: $\forall y \ Knows(y, Bill)$

R_3: $\forall y \ Knows(y, Mother(y))$

R_4: $\forall x \ Knows(x, Elizabeth)$

このとき、質問$\exists x \ Knows(John, x)$に対し、次の単一化が実行される。

$UNIFY\big(Knows(John, x), Knows(John, Jane)\big) = \{x/Jane\}$

$UNIFY\big(Knows(John, x), Knows(y, Bill)\big) = \{x/Bill, y/John\}$

$UNIFY\Big(Knows(John, x), Knows\big(y, Mother(y)\big)\Big) = \{y/John, x/Mother(John)\}$

$UNIFY\big(Knows(John, x), Knows(x, Elizabeth)\big) = fail$　(本当は fail しない)

最後の単一化が失敗しているのは変数xに$John$と$Elizabeth$の両方を代入しないといけないためである。これは、変数xの処理をきちんと行っていないためであって、本当に単一化に失敗するわけではない。元の$\forall x \ Knows(x, Elizabeth)$の式をみてみると、これは$\forall z \ Knows(z, Elizabeth)$としても等価であり、このように同じ変数名を使わないようにしないといけない。

$UNIFY\big(Knows(John, x), Knows(z, Elizabeth)\big) = \{x/Elizabeth, z/John\}$

このように変数名を書き換えておくことで、変数名の衝突問題を避けることができる。

しかし、まだもう一つ単一化に関して問題が残っている。例えば、

$$UNIFY\big(Knows(John, x), Knows(y, z)\big)$$

に対する答えはどうなるであろう？実に様々な可能な代入がありえるのだ。例えば、次の二つがあり得る。

$\{y/John, x/z\}$　(つまり、$Knows(John, x)$という代入結果になる)

$\{y/John, x/John, z/John\}$ (つまり、$Knows(John, John)$という代入結果になる)

この場合どちらの代入が好ましいのだろうか？前者のほうがより一般的でこの結果をまた別の推論に使える、という意味でも好ましい。後者には余計な推論(xとzを$John$としてしまうこと)が入っており、この代入が目的の推論に使えない場合はまったく役に立たない。そこで前者のようにより一般的な単一化結果を返す代入の方がより望ましいと考える。

最汎単一化子 (most general unifier, MGU): 項に対して代入を行うことで、項はより特殊な項となる。逆に特殊になる前の項は一般な項であったといえる。二つの項に対する可能な代入のうち最も一般な項を返す代入のことを最汎単一化子 (MGU) と呼ぶ。項の特殊-一般の関係は半順序関係となっており、単一化は上限(least upper bound)を求めていることになる。単一化において上限は唯一に定まる。

※上限は上界(upper bound)のうち最も下側の要素。情報数学 I の半順序関係を参照せよ。

参考: 単一化アルゴリズム

```
function UNIFY(x, y, θ):
    if θ = failure then return failure
    else if x = y then return θ
    else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
    else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
    else if COMPOUND?(x) and COMPOUND?(y) then
        return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
    else if LIST?(x) and LIST?(y) then
```

```
        return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, θ))
    else return failure

function UNIFY-VAR(var, x, θ):
    if {var/val}∈θ then return UNIFY(val, x, θ)
    else if {x/val}∈θ then return UNIFY(var, val, θ)
    else if OCCUR-CHECK?(var, x) then return failure
    else return add {var/x} to θ
```

※OCCUR-CHECK?(var, x)は変数 var が項 x の中に存在するかどうかチェックする述語である。これは単一化後、項の中にループ構造ができるどうかチェックしていることに相当する。ループができる場合は単一化に失敗する。多くの単一化アルゴリズムでは occur check は省略されている。