paramodulation と superposition

服部桃子

June 11, 2019

Contents

前回までのあらすじ

- 第 1 回: 導出原理と Egison での実装
- 第 2 回: 項書き換え系の話
  - 停止性、合流性
  - Lexicographic Path Order
  - Knuth-Bendix 完備化

前回までのあらすじ / 導出原理

定義: resolvent

- resolvent の定義: 節 c, c_1, c_2 に対し以下を満たすリテラル L_1, L_2 が存在するとき、c を c_1, c_2 の resolvent という
  - $L_1 \in c_1$ かつ $L_2^* \in c_2$
  - L_1, L_2 の最汎単一化子 θ が存在し、$c = (c_1\theta \setminus \{L_1\theta\}) \cup (c_2\theta \setminus \{L_2^*\theta\})$

$$\frac{\{P(a,b)\} \qquad \{\neg P(a,x), Q(x)\}}{\{Q(b)\}} \ \theta = [b/x]$$

```
(define $resolution ; resolvent を求める
  (match-all-lambda (multiset (multiset literal))
    {[<cons <cons <lit ,1 $t> $xs>
       ; t との最汎単一化子が σ であるようなリテラル ¬s
       <cons <cons <lit ,-1 (& <unify ,t $sigma> $s)> $ys>
         _>>
      {@(subst-clause sigma xs) @(subst-clause sigma ys)}]}))
```

前回までのあらすじ / 群の性質を示す

- 群の公理から群の性質を示す
  - 等式は単なる 2 項述語なので 等式の公理 も必要

```
(define $p3
  {@axioms-of-groups
   @axioms-of-equality
   @congruence-groups
   ; (negation of) the goal: e * x = x
   {<Lit -1 <Compound "=" <Compound "*" {e e1}> e1>>}})

> (solve p3) ; => 現実的な時間では終わらない
```

目標

Egison で実装した導出原理のプログラムを改良し、探索空間を減らす

- paramodulation
- superposition

paramodulation

$$\frac{C \vee s \doteq t \qquad D \vee P[s']}{\sigma(C \vee D \vee P[t])}$$

- 導出原理で等式を直接扱う手法
- 上のような規則を追加
  - $s \doteq t$ は「$s = t$ または $t = s$」
  - σ は s と s' の最汎単一化子 ($\sigma = \mathrm{mgu}(s, s')$)
- 等式の公理は reflexivity(反射性) 以外は不要になる
  - functional reflexivity も必要とされる
    - $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$

定理: Refutational completeness

S が normal model[a] を持たないならば、S に reflexivity と functional reflexivity を加えたものから paramodulation と導出原理によって空節を導くことができる

[a] "="が等号として解釈されるモデル

証明 (概略):

普通の導出原理の完全性を利用して、「導出原理 + 等式の公理で導出可能」ならば「導出原理 + paramodulation + reflexivity で導出可能」を示す

paramodulation / completeness

- 例えば導出原理+(1-ary)functional congruence を用いている場合...

$$\frac{\neg(x = x') \lor f(\ldots, x, \ldots) = f(\ldots, x', \ldots) \qquad C \lor s = t}{C \lor f(\ldots, s, \ldots) = f(\ldots, t, \ldots)}$$

paramodulation + functional reflexivity で次のように示せる

$$\frac{f(\ldots, x, \ldots) = f(\ldots, [x], \ldots) \lor C \qquad s = t}{f(\ldots, s, \ldots) = f(\ldots, t, \ldots) \lor C} \ \sigma = [s/x]$$

- 他、(1-ary) predicate congruence を用いている場合や、対称性・推移性を用いている場合も同様の議論

prop

```
; predicate とは別に equation を用意する
(define $prop
  (matcher
    {[<pred $ $> [string (list term)]
       {[<Pred $p $a> {[p a]}]
        [_ {}]}]
     [ <equation $ $>  [term term]
       ; unordered-pair のようにマッチ (l ≐ r を表現)
      {[<Equation $l $r> { [l r] [r l] }]
       [_ {}]}]
     [<unify ,$t $> something
      {[$s (punify t s)]}]
     [<subterm $ $> [term something]
      {[$s (psubterm s)]}]
     [$ something
       {[$tgt {tgt}]}]}))
```

subterm

```
(define $prop
  (matcher
    {(...)
     ; subterm とその「環境」(term -> prop の関数) のペアにマッチ
     [<subterm $ $> [term something]
      [$s (psubterm s)]]
     [$ something
       {[$tgt {tgt}]}]}))
```

```
;; term -> {term, term -> term}
(define $subterm
  (match-all-lambda term
    {[<compound $f <join $xs <cons <subterm $x $env> $xs'>>>
       (let {[$env'
               (lambda [$x] <Compound f (append xs (cons x xs'))>)]}
         [x (compose env' env)])]
     [$t [t id]]}))
```

resolution

```
(define $resolution
  (match-all-lambda (multiset (multiset literal))
     ; paramodulation
    {[<cons <cons <lit ,1 <equation $s $t>> $xs>
       <cons <cons <lit $sign
            ; env[s'], s'は変数でない, sigma = mgu(s, s')
            <subterm (& !<var _> <unify ,s $sigma> $s') $env>> $ys>
         _>>
      (csubst sigma {@xs @ys <Lit sign (env t)>})
     ]
     ; 普通の resolution
     [<cons <cons <lit ,1 $t> $xs>
       <cons <cons <lit ,-1 (& <unify ,t $sigma> $s)> $ys>
         _>>
      (csubst sigma {@xs @ys})
     ]}))
```

(余談) 導出原理の最適化