Egison を用いた自動定理証明

05-181023 服部桃子

May 28, 2019

Contents

Contents

特長

- 「パターンマッチ指向」言語
- Non-free なデータ型に対してパターンマッチできる
- 非線形パターンマッチができる

Non-free なデータ型

標準形がないデータ型のこと

- set, multiset は Non-free
  - multiset$\{1,2,3\}$ は $\{2,1,3\}$ などとも書ける
- list は cons に関しては free
  - `[1; 2] = 1 :: 2 :: []`　(※ OCaml)
- list は append に関しては Non-free
  - `[1; 2] = [] @ [1; 2] = [1] @ [2] = [1; 2] @ []`　(※ OCaml)

例

```
(match-all {1 2 3} (list integer) [<cons $x $xs> [x xs]])
;=>{[1 {2 3}]}

(match-all {1 2} (list integer) [<join $xs $ys> [xs ys]])
;=>{[{} {1 2}] [{1} {2}] [{1 2} {}]}

(match-all {1 2 3} (multiset integer) [<cons $x $xs> [x xs]])
;=>{[1 {2 3}] [2 {1 3}] [3 {1 2}]}

(match-all {1 2} (multiset integer) [<join $xs $ys> [xs ys]])
;=>{[{} {1 2}] [{1} {2}] [{2} {1}] [{1 2} {}]}

(match-all {1 2 3} (set integer) [<cons $x $xs> [x xs]])
;=>{[1 {1 2 3}] [2 {1 2 3}] [3 {1 2 3}]}

(match-all {1 2} (set integer) [<join $xs $ys> [xs ys]])
;=>{[{} {1 2}] [{1} {1 2}] [{2} {1 2}] [{1 2} {1 2}]}
```

※`{...}`は collection、`[...]` は tuple

非線形パターンマッチ

- パターン中に同じ変数が複数回出現することを許す
- 例

```
(define $twin-primes
  (match-all primes (list integer)
    [<join _ <cons $p <cons ,(+ p 2) _>>> [p (+ p 2)]]))
```

matcher の定義

```
(define $unordered-pair
  (lambda [$m] ; 引数も matcher
    (matcher
      {[<pair $ $> [m m] ; マッチの形式と matcher
         {[<P $x $y> {[x y] [y x]}]}] ; next-targets を返す
       [$ [something] ; 分解せずに返す場合
         {[$tgt {tgt}]}]})))

(match-all <P 1 2> (unordered-pair integer) [<pair $x $y> [x y]])
;=> {[1 2] [2 1]}
(match-all <P 2 1> (unordered-pair integer) [$x x])
;=> {<Pair 2 1>}
```

Contents

定義: Resolvent

- リテラル ... 命題変数 P またはその否定 $\neg P$
- 節 (clause) ... リテラルの有限集合
- リテラル L の complement L^* を以下で定義

$$L^* :\equiv \begin{cases} \neg P & (L \equiv P) \\ P & (L \equiv \neg P) \end{cases}$$

- 節 c, c_1, c_2 に対し以下を満たす L が存在する時、c を c_1, c_2 の **resolvent** という
    - $L \in c_1$ かつ $L^* \in c_2$
    - $c = (c_1 \setminus \{L\}) \cup (c_2 \setminus \{L^*\})$

例

$$\frac{\{P, \neg Q, \neg R\} \qquad \{\neg P, \neg Q, R, S\}}{\{\neg Q, \neg R, R, S\}} \qquad \frac{\{P, \neg Q, \neg R\} \qquad \{\neg P, \neg Q, R, S\}}{\{P, \neg P, \neg Q, S\}}$$

$S = \{c_1, \ldots, c_n\}$ を節の有限集合としたとき、

$$c(S) := \left(\bigvee c_1\right) \wedge \cdots \wedge \left(\bigvee c_n\right)$$

健全性・完全性

$c(S) \Rightarrow$ が LK で導出可能 $\Leftrightarrow$ S の resolution tree で根が $\emptyset$ のものが存在

命題論理との違い

- リテラルは命題変数ではなく原子式 $P(t_1, \ldots, t_n)$ になる
- resolvent の定義: 節 c, c_1, c_2 に対し以下を満たす 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、$c = (c_1\theta \setminus \{L_1\theta\}) \cup (c_2\theta \setminus \{L_2^*\theta\})$

例

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

Contents

term の定義

$$t ::= x \in \mathbf{Var} \mid f(t_1, \ldots, t_n)$$

```
(define $term
  (matcher
    {[<var $> integer ; variable の場合 (variable は integer で定義)
      {[<Var $i> {i}]
       [_ {}]}]
     [<compound $ $> [string (list term)] ; 関数に適用されている場合
      {[<Compound $s $l> {[s l]}]
       [_ {}]}]
     [<unify ,$t $> something ; resolution の時に使う
      {[$s (unify t s)]}]
     [$ something
      {[$tgt {tgt}]}]}))
```

3. Egison での実装

```
(define $subst
  (lambda [$u $t] ; t に代入 u を適用
    (match t term
      {[<var $n>
         (match u (multiset [integer term])
           {[<cons [,n $t] _> t] ; あてはまる規則があれば代入
            [_ <Var n>]})]       ; なければそのまま
       [<compound $f $xs> ; t が関数適用された項の場合、関数の全引数に代入
        <Compound f (map (lambda $x (subst u x)) xs)>]})))
```

```
(define $x 0)
(define $y 1)
(define $z 2)
; P(g(x), y)
(define $t1 (app "P" (app "g" (var x)) (var y)))

; [x/y][z/x] P(g(x), y) = P(g(z), x)
(subst {[y <Var x>] [x <Var z>]} t1)
;=> <Compound "P" {<Compound "g" {<Var 2>}> <Var 0>}>
```

unify

```
; 失敗なら{}、成功なら{単一化子}を返す (optional 型の気持ち)
(define $unify
  (match-lambda (unordered-pair term)
    {[<pair <var $x> <var ,x>>
      {{}}] ; 成功 (空の単一化子を返す)
     [<pair <var $x> (& $t !(occur ,x))> ; 自由出現しない場合
      {{[x t]}}] ; [t/x] を返す
     [<pair <compound $f <nil>> <compound ,f <nil>>>
      {{}}] ; 成功
     [<pair <compound $f <cons $x $xs>> <compound ,f <cons $y $ys>>>
      (match (unify x y) (list something)
       {[,{} {}] ; unify がすでに失敗している
        [<cons $u1 _> ; unifiable な場合、単一化子を singleton から取り出す
         (match (unify (subst u1 <Compound f xs>) (subst u1 <Compound f ys>))
                (list something)
          {[,{} {}] ; 失敗
           [<cons $u2 _> {{@u1 @u2}}]})]})] ; {@u1 @u2}は u1 ++ u2
     [_ {}]})) ; 失敗
```

3. Egison での実装

resolution

```
(define $resolution ; resolvent を求める
  (match-all-lambda (multiset (multiset literal))
     ; リテラル t
    {[<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 $σ> $s)> $ys>
         ; 残りの節は無視
         _>>
      {@(subst-clause σ xs) @(subst-clause σ ys)}]}))
```

solve

```
(define $solve
  (match-lambda (multiset (multiset literal))
    {[<cons <nil> _> ; 空節が導出できた
      <REFUTE>]
     [$p
      (let {[$q (resolution (rename-problem ∅ p))]}
        (if (include?/m (multiset literal) p q) ; q ⊆? p
          <SATURATED> ; 新しい節がない→ resolve できる節がなかった
          (solve {@p @q})))]}))  ; resolution 前の節も残しておく
```

rename-problem

```
> (show-clauses p1)
"{{~P(g(x), y), P(x, f(x, y))}
 {~P(c, y)}
 {P(g(g(c)), c)}}"
> (show-clauses (rename-problem 0 p1))
"{{~P(g(x), y), P(x, f(x, y))}
 {~P(c, z)}
 {P(g(g(c)), c)}}"
```

- 次のような問題を考える
  - Kate is a girl.
  - Joe's friends are all tall.
  - Harry loves any girl that is tall.
  - Kate is a friend of Joe.
  - Does Harry love any of Joe's friends? If he does, who does he love?
- 命題にして連言標準形にすると

$$\{G(k)\}\,\{\neg F(j,x), T(x)\}\,\{\neg G(y), \neg T(y), L(h,y)\}\,\{F(j,k)\}\,\{\neg L(h,z), \neg F(j,z)\}$$

- Egison で書くと

```
(define $p7
  {{<Lit 1 (app "G" (app "k"))>}
   {<Lit -1 (app "F" (app "j") (var x))> <Lit 1 (app "T" (var x))>}
   {<Lit -1 (app "G" (var y))> <Lit -1 (app "T" (var y))>
      <Lit 1 (app "L" (app "h") (var y))>}
   {<Lit 1 (app "F" (app "j") (app "k"))>}
   {<Lit -1 (app "L" (app "h") (var z))>
      <Lit -1 (app "F" (app "j") (var z))>}})
```

- 実行してみる

```
> (solve p7)
...
{{G(k)}
 {~F(j, x), T(x)}
 {~G(y), ~T(y), L(h, y)}
 {F(j, k)}
 {~L(h, z), ~F(j, z)}
 {~T(k), L(h, k)}
 {~F(j, z), ~G(z), L(h, z)}
 {T(k)}
 {~F(j, k), L(h, k)}
 {~G(5), ~T(5), ~F(j, 5)}
 {~L(h, k)}
 {~T(k), ~F(j, k)}
 {~G(k), L(h, k)}
 {~F(j, 10), ~G(10), ~F(j, 10)}
 {~F(j, 5), ~G(5), ~F(j, 5)}
 {L(h, k)}
 {~G(k), ~T(k)}
 {~F(j, k), ~F(j, k)}
 {~T(k)}
 {~F(j, k), ~G(k)}
 {~F(j, k)}
 {~G(k)}
 {}}
<REFUTE>
```

- 3 回目の `solve` で空節が導かれた
- 節がとても多くなる = 探索空間が広くなりすぎる

3. Egison での実装 / 例 2

- 群の公理から導けるはずの命題を証明してみる
- 群の公理:

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$
$$x \cdot x^{-1} = e$$
$$x \cdot e = x$$

- 等式についての公理を追加:

$$x = x$$
$$x = y \lor x \neq y$$
$$x = y \land y = z \to x = z$$
$$x = y \land z = w \to x \cdot z = y \cdot w$$
$$x = y \to x^{-1} = y^{-1}$$

- $e \cdot x = x, \quad x^{-1} \cdot x = x$ を証明してみたい

3. Egison での実装 / 例 2

- これらを Egison で表現すると以下

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

(define $p4
  {@axioms-of-groups
   @axioms-of-equality
   ; (negation of) the goal: x^{-1} · x = e
   {<Lit -1 <Compound "="
     {<Compound "p" {<Compound "i" {<Compound "e2" {}>}>
                    <Compound "e2" {}>}>
      <Compound "e" {}>}>>}
```

- 探索空間が広過ぎて答えが導けない
- これを等式論理の性質などを用いて解けるようにするのが来週以降の目標