Jacques Garrigue, 2023 年 5 月 24 日

単一化と自動化

1　単一化と自動化

```
Lemma test x : 1 + x = x + 1.
  Check [eta addnC].
     : ∀ x y : nat, x + y = y + x
  apply: addnC.
Abort.                                  (* 定理を登録せずに証明を終わらせる *)
```

ゴールと定理 addnC の字面が異なっているのに，ここでなぜ apply が使えるのか．
実は apply は複数のことをしている．

1. 定理の中の ∀ で量化されている変数を「単一化用の変数」に置き換える
2. 置き換えられた定理の結論をゴールと「単一化」する
3. 定理に前提があれば,「単一化」の結果を代入した前提を新しいゴールにする．

ここでいう単一化とは,（単一化用の）変数を含んだ項同士をその変数の値を定めることで同じものにする．例えば，1 + x = x + 1 と $?m$+$?n$ =$?n$+$?m$ を単一化するには,
$?m = 1$, $?n =$ x と定めれば良い．
Coq 本来の apply で変数が定まらないと，エラーになる．しかし，SSReflect の apply: や apply/ を使えば，変数が残せる．

```
Lemma test x y z : x + y + z = z + y + x.
  Check etrans.
     : ∀ (A : Type) (x y z : A), x = y -> y = z -> x = z
  apply etrans.
Error: Unable to find an instance for the variable y.
  apply: etrans.              (* y が結論に現れないので，apply: に変える *)
   x + y + z = ?Goal
  apply: addnC.
  apply: etrans.
  Check f_equal.
     : ∀ (A B : Type) (f : A -> B) (x y : A), x = y -> f x = f y
  apply: f_equal.                                  (*   x + y = ?Goal0 *)
  apply: addnC.
  apply: addnA.
Restart.                                                (* 証明を元に戻す *)
  rewrite addnC.                                (* rewrite も単一化を使う *)
  rewrite (addnC x).
  apply: addnA.
Abort.
```

一階の項に関して，単一化は最適な代入を見つけてくれるという結果が知られている．

定義 1 ある代入 σ_1 が代入 σ_2 **より一般的**であるとは，σ_{12} が存在し，$\sigma_2 = \sigma_{12} \circ \sigma_1$ であることをいう．

定義 2 単一化問題 $\{t_1 = t'_1, \ldots, t_n = t'_n\}$ に対して，$\sigma(t_i) = \sigma(t'_i)$ であるとき，σ はその単一化問題の**単一子**だという．

定理 1 任意の単一化問題に対して，解が存在するときには**最も一般的な単一子**を返し，存在しないときにはそれを報告するアルゴリズムが存在する．

具体的には，アルゴリズム $\mathcal{U}$ を以下の買い替え規則で定義できる．ここでは定数記号を 0 引数の関数記号と同一視する．

$$
\begin{array}{ll}
E \cup \{f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)\} \to E \cup \{t_1 = t'_1, \ldots, t_n = t'_n\} & \\
E \cup \{f(t_1, \ldots, t_n) = g(t'_1, \ldots, t'_m)\} \to \bot & f \neq g \\
E \cup \{x = t\} \to [t/x]E \cup \{x = t\} & x \in vars(E) \wedge (t = y \Rightarrow y \in vars(E)) \wedge x \notin vars(t) \\
E \cup \{x = x\} \to E & \\
E \cup \{x = t\} \to \bot & x \in vars(t)
\end{array}
$$

E が書き換えを繰り返し，書き換えられない E' になれば，その E' が $\{x_1 = t''_1, \ldots, x_m = t''_m\}$ という形であり，それを代入 $\sigma = [t''_1/x_1, \ldots, t''_m/x]$ と見なし，$\mathcal{U}(E) = \sigma$．このときに，任意の $t = t' \in E$ について，$\sigma(t) = \sigma(t')$，かつ E の単一子になる任意の σ' について，σ が σ' より一般的である．また，$E' = \bot$ のとき，E には解がない．

上記の $\mathcal{U}$ は一階の項のためのものであるが，Coq はそれより強い**高階単一化**[1]を行っている．しかし，この場合には最も一般的な解が存在するとは限らない．

```
Goal
  (∀ P : nat -> Prop, P 0 -> (∀n, P n -> P (S n)) -> ∀n, P n) ->
     ∀ n m, n + m = m + n.
  move=> H n m.                                  (* 全ての変数を仮定に *)
  apply: H.                                            (* n + m = 0 *)
Restart.
  move=> H n m.
  pattern n.                              (* pattern で正しい述語を構成する *)
  apply: H.                                          (* 0 + m = m + 0 *)
Restart.
  move=> H n.                               (* forall n を残すとうまくいく *)
  apply: H.                                          (* n + 0 = 0 + n *)
Abort.
```

単一化は様々な作戦で使われている．`apply` 以外に `elim`，`rewrite` や `set` が挙げられる．

[1] 1970 年代に高階単一化の可能性を証明した Gérard Huet は Coq の最初のプロジェクトリーダーだった