Jacques Garrigue, 2014 年 7 月 7 日

# 単一化とその応用

## 1 単一化

単一化は二つの項を同じものにする代入を求めるアルゴリズムである．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 $\sigma_1$ が代入 $\sigma_2$ **より一般的**であるとは，$\sigma_{12}$ 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，$\sigma$ 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 $\mathcal{U}$ を以下の書き換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．

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

$(E, id)$ から始まり，書き換えを繰り返すことで，$(\emptyset, \sigma)$ になった場合，$\mathcal{U}(E) = \sigma$ は $E$ の最も一般的な単一子になる．$\bot$ になった場合，単一子が存在しない．

## 2 実装

今日の実装と前回の答えは `http://www.math.nagoya-u.ac.jp/~garrigue/lecture/2014_SS/` においてある．

```
(* 単一化 *)
Require Import Arith List ListSet.

Definition var := nat.
Notation "x == y" := (eq_nat_dec x y) (at level 70).

(* 定数・関数記号 *)
Inductive symbol : Set := Symbol : nat -> symbol.
Definition symbol_dec : forall a b : symbol, {a = b} + {a <> b}.
  pose eq_nat_dec.
  decide equality.
Defined.

(* 項は木構造 *)
Inductive tree : Set :=
  | Var : var -> tree
  | Sym : symbol -> tree
  | Fork : tree -> tree -> tree.
```

```
(* t' [x := t] *)
Fixpoint subs (x : var) (t t' : tree) : tree :=
  match t' with
  | Var v => if v == x then t else t'
  | Sym b => Sym b
  | Fork t1 t2 => Fork (subs x t t1) (subs x t t2)
  end.

(* 代入は変数代入の繰り返し *)
Fixpoint subs_list (s : list (var * tree)) t : tree :=
  match s with
  | nil => t
  | (x, t') :: s' => subs_list s' (subs x t' t)
  end.

Lemma subs_list_app : forall s1 s2 t,
  subs_list (s1 ++ s2) t = subs_list s2 (subs_list s1 t).
Proof.
  induction s1; simpl; auto.
  destruct a. congruence.
Qed.

(* 単一子の定義 *)
Definition unifies s (t1 t2 : tree) := subs_list s t1 = subs_list s t2.

(* 例 : (a, (x, y)) [y := z] [x := (y, b)] = (a, ((y,b), z)) *)
Definition atree := Fork (Sym (Symbol 0)) (Fork (Var 0) (Var 1)).
Definition asubs := (0, Fork (Var 1) (Sym (Symbol 1))) :: (1, Var 2) :: nil.
Eval compute in subs_list asubs atree.

(* 和集合 *)
Definition union := set_union eq_nat_dec.

(* 出現変数 *)
Fixpoint vars (t : tree) : list var := nil. (* 定義せよ *)

(* 出現しない変数は代入されない *)
Parameter subs_same : forall v t' t,
  ~In v (vars t) -> subs v t' t = t.

(* 出現判定 *)
Definition occurs (x : var) (t : tree) : {In x (vars t)}+{~In x (vars t)}.
  case_eq (set_mem eq_nat_dec x (vars t)); intros.
    left; apply (set_mem_correct1 _ _ _ H).
  right; apply (set_mem_complete1 _ _ _ H).
Defined.
Extraction occurs.

(* 等式の大きさの計算 *)
Fixpoint size_tree (t : tree) : nat :=
  match t with
  | Fork t1 t2 => 1 + size_tree t1 + size_tree t2
  | _ => 1
  end.

Fixpoint size_pairs (l : list (tree * tree)) :=
  match l with
  | nil => 0
  | (t1, t2) :: r => size_tree t1 + size_pairs r
  end.

Definition subs_pair x t (p : tree * tree) :=
  let (t1, t2) := p in (subs x t t1, subs x t t2).

Require Import Recdef.

(* 正直なやり方 *)
Function unify s l {measure size_pairs l} : option (list (var * tree)) :=
```

```
  match l with
  | nil => Some s
  | (Var x, Var x' as t) :: r =>
    if x == x' then unify s r else unify ((x,t)::s) (map (subs_pair x t) r)
  | (Var x, t) :: r =>
    if occurs x t then None else unify ((x,t)::s) (map (subs_pair x t) r)
  | (t, Var x) :: r =>
    if occurs x t then None else unify ((x,t)::s) (map (subs_pair x t) r)
  | (Sym b, Sym b') :: r =>
    if symbol_dec b b' then unify s r else None
  | (Fork t1 t2, Fork t1' t2') :: r =>
    unify s ((t1, t1') :: (t2, t2') :: r)
  | l => None
  end.
intros. simpl. auto with arith.
admit. admit.
Abort.                                                    (* 減らない場合がある *)
Reset unify_F.

Section Unify1.

(* 代入を行ったときの再帰呼び出し *)
Variable unify2 : list (tree * tree) -> option (list (var * tree)).

(* 代入して再帰呼び出し. x は t に現れてはいけない *)
Definition may_cons (x : var) (t : tree) r :=
  match r with
  | None => None
  | Some s => Some ((x,t) :: s)
  end.
Definition unify_subs x t r :=
  if occurs x t then None else may_cons x t (unify2 (map (subs_pair x t) r)).

(* 代入をせずに分解 *)
Fixpoint unify1 (h : nat) (l : list (tree * tree))
  : option (list (var * tree)) :=
  match h with
  | 0 => None
  | S h' =>
    match l with
    | nil => Some nil
    | (Var x, Var x') :: r =>
      if x == x' then unify1 h' r else unify_subs x (Var x') r
    | (Var x, t) :: r =>
      unify_subs x t r
    | (t, Var x) :: r =>
      unify_subs x t r
    | (Sym b, Sym b') :: r =>
      if symbol_dec b b' then unify1 h' r else None
    | (Fork t1 t2, Fork t1' t2') :: r =>
      unify1 h' ((t1, t1') :: (t2, t2') :: r)
    | l => None
    end
  end.
End Unify1.

(* 代入したときだけ再帰 *)
Fixpoint unify2 (h : nat) l :=
  match h with
  | 0    => None
  | S h' => unify1 (unify2 h') (size_pairs l + 1) l
  end.

Fixpoint vars_pairs (l : list (tree * tree)) : list var :=
  match l with
  | nil => nil
  | (t1, t2) :: r => union (union (vars t1) (vars t2)) (vars_pairs r)
  end.
```

```
(* 変数の数だけ unify2 を繰り返す *)
Definition unify t1 t2 :=
  let l := (t1,t2)::nil in unify2 (length (vars_pairs l) + 5) l.

(* 例 *)
Eval compute in unify (Sym (Symbol 0)) (Var 1).

Eval compute in
  unify (Fork (Sym (Symbol 0)) (Var 0))
        (Fork (Var 1) (Fork (Var 1) (Var 2))).

(* 全ての等式の単一子 *)
Definition unifies_pairs s l :=
  forall t1 t2, In (t1,t2) l -> unifies s t1 t2.

(* subs_list と Fork が可換 *)
Parameter subs_list_Fork : forall s t1 t2,
  subs_list s (Fork t1 t2) = Fork (subs_list s t1) (subs_list s t2).

(* unifies_pairs の性質 *)
Parameter unifies_pairs_same : forall s t l,
  unifies_pairs s l -> unifies_pairs s ((t,t) :: l).
Parameter unifies_pairs_swap : forall s t1 t2 l,
  unifies_pairs s ((t1, t2) :: l) -> unifies_pairs s ((t2, t1) :: l).

(* unify2 の健全性 *)
Theorem unify2_sound : forall h s l,
  unify2 h l = Some s -> unifies_pairs s l.
Proof.
  induction h; simpl; intros.
    discriminate.
  remember (size_pairs l + 1) as h'.
  clear Heqh'.
  revert l H.
  induction h'; simpl; intros.
    discriminate.
  destruct l.
    intros t1 t2 Hin. elim Hin.
  destruct p as [t1 t2].
  destruct t1; destruct t2; try discriminate.
  (* VarVar *)
  destruct (v == v0).
    subst v0.
    auto using unifies_pairs_same.
Lemma unify_subs_sound : forall h v t l s,
  (forall l s, unify2 h l = Some s -> unifies_pairs s l) ->
  unify_subs (unify2 h) v t l = Some s ->
  unifies_pairs s ((Var v, t) :: l).
Admitted.
  apply (unify_subs_sound h); auto.
  (* VarSym *)
  apply (unify_subs_sound h); auto.
  (* VarFork *)
  apply (unify_subs_sound h); auto.
  (* SymVar *)
  apply unifies_pairs_swap.
  apply (unify_subs_sound h); auto.
Admitted.

(* 単一化の健全性 *)
Corollary soundness : forall t1 t2 s, unify t1 t2 = Some s -> unifies s t1 t2.
Admitted.
```

**練習問題 2.1**　*1.* vars を定義せよ.

*2.* 証明の中の Parameter を Theorem に変え，Admitted を Qed に変えよ.
unify_subs_sound が最も重要な補題である.