Contents

Chapter 1

Library UnificationExercises.Unification

```
Require Import List.
Require Import Basics.
Require Import Logic.
Require Import Arith.EqNat.

Import ListNotations.

Definition var := nat.

Inductive ter : Type :=
  | V : var → ter
  | T : ter → ter → ter.

Definition eqn := prod ter ter.

Implicit Types x y z : var.
Implicit Types s t u v : ter.
Implicit Type e : eqn.
Implicit Types A B C : list eqn.
Implicit Types sigma tau : ter → ter.
Implicit Types m n k : nat.

Definition subst sigma : Prop :=
  ∀ s t, sigma (T s t) = T (sigma s) (sigma t).

Definition unif sigma A : Prop :=
  subst sigma ∧ ∀ s t, In (s,t) A → sigma s = sigma t.

Definition unifiable A : Prop :=
  ∃ sigma, unif sigma A.

Definition principal_unifier sigma A : Prop :=
  unif sigma A ∧ ∀ tau, unif tau A → ∀ s, tau (sigma s) = tau s.
```

```
Lemma subst_term_var_agreement :
  ∀ sigma tau, (subst sigma) → (subst tau) →
    (∀ x, sigma (V x) = tau (V x)) →
        ∀ s, (sigma s) = (tau s).
Proof.
  intros sigma tau sub1 sub2 var_agree s. induction s.
    - apply var_agree.
    - unfold subst in sub1. unfold subst in sub2. rewrite sub1. rewrite sub2. rewrite
IHs1. rewrite IHs2. reflexivity.
Qed.

Lemma principle_unif_idempotent :
  ∀ sigma A, principal_unifier sigma A → (∀ t, (sigma (sigma t)) = (sigma t)).
Proof.
  intros. unfold principal_unifier in H. destruct H. apply H0. apply H.
Qed.

Lemma unif_fact_a :
  ∀ A t s sigma, unif sigma ((s, t) :: A) ↔ (sigma s) = (sigma t) ∧ unif sigma A.
Proof.
  intros. split.
    - intros. firstorder.
    - intros. firstorder. inversion H2. symmetry in H4. symmetry in H5. rewrite
H4. rewrite H5. apply H.
Qed.

Lemma unif_fact_b :
  ∀ A B sigma, unif sigma (A ++ B) ↔ (unif sigma A) ∧ (unif sigma B).
Proof.
  intros. split.
    - intros. split.
      + induction B.
        × rewrite app_nil_r in H. apply H.
        × apply IHB. unfold unif in H. destruct H. unfold unif. split. apply H.
intros. apply H0. apply in_app_or in H1. apply in_app_iff with (l':= a :: B). destruct
H1.
          { left. apply H1. }
          { right. apply in_cons. apply H1. }
      + induction A.
        × simpl in H. apply H.
        × apply IHA. unfold unif in H. destruct H. unfold unif. split. apply H.
intros. apply H0. apply in_app_or in H1. apply in_app_iff with (l := a :: A). destruct
H1.
          { left. apply in_cons. apply H1. }
          { right. apply H1. }
```

```
    - intros. unfold unif in *. destruct H. destruct H. destruct H0. split. apply
H. intros. apply in_app_or in H3. destruct H3.
      + apply H1. apply H3.
      + apply H2. apply H3.
Qed.

Lemma sublist_non_unifiable :
  ∀ A B, incl A B → (unifiable B) → (unifiable A).
Proof.
  intros. unfold unifiable in *. firstorder.
Qed.


Fixpoint v_term t :=
match t with
  | (V x) ⇒ [x]
  | (T s t) ⇒ (v_term s) ++ (v_term t)
end.

Fixpoint v_list A :=
match A with
  | nil ⇒ nil
  | st :: A' ⇒ (v_term (fst st)) ++ (v_term (snd st)) ++ (v_list A')
end.

Fixpoint domain A :=
match A with
  | nil ⇒ nil
  | (V x, _) :: A' ⇒ x :: (domain A')
  | _ :: A' ⇒ nil
end.

Definition disjoint {X} (A B : list X) : Prop :=
  ¬ (∃ x:X, In x A ∧ In x B).

Inductive solved : list eqn → Prop :=
  | solved_nil : solved nil
  | solved_cons : ∀ x s A, ~(In x (v_term s)) → ~(In x (domain A)) → (disjoint (v_term
s) (domain A)) → (solved A) → (solved ((V x, s) :: A)).

Fixpoint var_term_replace s x t :=
match s with
  | (V y) ⇒
    if (beq_nat x y)
      then t
    else (V y)
  | (T u v) ⇒ (T (var_term_replace u x t) (var_term_replace v x t))
end.
```

```
Fixpoint var_list_replace A x t :=
match A with
  | nil ⇒ nil
  | uv :: A' ⇒ ((var_term_replace (fst uv) x t), (var_term_replace (snd uv) x t)) ::
(var_list_replace A' x t)
end.

Fixpoint phi A s :=
match A with
  | nil ⇒ s
  | (V x, t) :: A' ⇒ var_term_replace (phi A' s) x t
  | (u, v) :: A' ⇒ s
end.

Definition bad_equation e : Prop :=
  ∃ x s, (e = (V x, s)) ∧ ((V x) ≠ s) ∧ (In x (v_term s)).

Lemma solved_principle_unifier :
  ∀ A, (solved A) → (principal_unifier (phi A) A).
Proof.
  intros.
Admitted.

Fact var_term_no_replacement :
  ∀ x s t,
    ¬ (In x (v_term s)) → (var_term_replace s x t) = s.
Proof.
  intros. unfold not in H. induction s.
    - simpl. simpl in H. firstorder. destruct (beq_nat x v) eqn:H0.
      + apply beq_nat_true in H0. exfalso. apply H. symmetry in H0. apply H0.
      + reflexivity.
    - simpl in *. apply f_equal2.
      + firstorder.
      + firstorder.
Qed.

Fact var_list_no_replacement :
  ∀ x A t,
    ¬ (In x (v_list A)) → (var_list_replace A x t) = A.
Proof.
  intros. unfold not in H. induction A.
    - simpl. reflexivity.
    - simpl. apply f_equal2.
      + destruct a. simpl. apply f_equal2.
        × simpl in H. apply var_term_no_replacement. intros H0. apply H. apply
in_or_app. left. apply H0.
```

```
      × simpl in H. apply var_term_no_replacement. intros H0. apply H. apply
in_or_app. right. apply in_or_app. left. apply H0.
    + apply IHA. firstorder. apply H. apply in_or_app. right. apply in_or_app.
right. apply H0.
Qed.

Fact term_list_domain_agreement :
  ∀ x A t,
    ¬ (In x (domain A)) → (domain (var_list_replace A x t)) = (domain A).
Proof.
  intros. unfold not in H. induction A.
    - simpl. reflexivity.
    - destruct a. destruct t0.
      + destruct (beq_nat x v) eqn:H0.
        × exfalso. apply H. simpl. left. apply beq_nat_true in H0. symmetry in H0.
apply H0.
        × simpl. rewrite H0. apply f_equal2.
          { reflexivity. }
          { apply IHA. intros. apply H. simpl. right. apply H1. }
      + firstorder.
Qed.

Fact subst_replacement :
  ∀ sigma s x t,
    (subst sigma) → (sigma (V x)) = (sigma t) → (sigma (var_term_replace s x t)) =
(sigma s).
Proof.
  intros. induction s.
  - destruct (beq_nat x v) eqn:H1.
    + apply beq_nat_true in H1. symmetry in H1. rewrite H1. rewrite H0. simpl.
destruct beq_nat.
      × reflexivity.
      × apply H0.
    + simpl. rewrite H1. reflexivity.
  - simpl. unfold subst in *. rewrite H. rewrite IHs1. rewrite IHs2. firstorder.
Qed.

Fact lambda_subst :
  ∀ x t, (subst (fun s ⇒ var_term_replace s x t)).
Proof.
  intros. unfold subst. intros. reflexivity.
Qed.

Fact phi_A_subst :
  ∀ A, (subst (phi A)).
```

```
Proof.
  intros. unfold subst. intros. induction A.
  - reflexivity.
  - destruct a. destruct t0.
    + simpl. rewrite IHA. reflexivity.
    + reflexivity.
Qed.

Fact phi_domain_vars_disjoint :
  ∀ A s, (disjoint (domain A) (v_term s)) → (phi A s) = s.
Proof.
  intros. unfold disjoint in H. unfold not in H. induction A.
  - simpl. reflexivity.
  - destruct a. destruct t.
    + simpl. rewrite IHA.
      × apply var_term_no_replacement. unfold not. intros. apply H. ∃ v. split.
        { left. reflexivity. }
        { apply H0. }
      × intros [x]. apply H. ∃ x. split.
        { right. apply H0. }
        { apply H0. }
    + simpl. reflexivity.
Qed.

Fact solved_A_phi_A_unifier :
  ∀ A, (solved A) → (unif (phi A) A).
Proof.
  intros. split.
  - apply phi_A_subst.
  - intros. destruct A.
    + simpl. inversion H0.
    + destruct e. destruct H0.
      × inversion H0.
Admitted.

Fact sigma_A_unifier :
  ∀ sigma A, (unif sigma A) → (∀ s, (sigma (phi A s)) = (sigma s)).
Proof.
  intros.
Admitted.

Fact solved_A_phi_A_principal_unifier :
  ∀ A, (solved A) → (principal_unifier (phi A) A).
Proof.
  intros.
```

```
Admitted.

Fixpoint size s : nat :=
match s with
  | (V _) ⇒ 1
  | (T s u) ⇒ (size s) + (size u)
end.

Lemma sigma_x_vs_s_size :
  ∀ x s sigma, (In x (v_term s)) → (subst sigma) → ((size (sigma (V x))) ≤ (size (sigma
s))).
Proof.
  intros. destruct s.
    - firstorder. symmetry in H. rewrite H. reflexivity.
    - unfold size.
Admitted.

Lemma no_bad_equations_unifiable :
  ∀ e, (bad_equation e) → ~(unifiable [e]).
Proof.
  intros. unfold bad_equation in H. unfold unifiable. unfold not. firstorder. apply
H2.
Admitted.

Fact domain_A_sublist_A :
  ∀ A, (incl (domain A) (v_list A)).
Proof.
  intros. unfold incl. intros. destruct A.
    - simpl. contradiction.
    - destruct e. simpl.
Admitted.

Fact appending_variable_lists :
  ∀ A B, (v_list (A ++ B)) = (v_list A) ++ (v_list B).
Proof.
  intros. induction A, B.
    - simpl. reflexivity.
    - simpl. reflexivity.
    - simpl. rewrite app_nil_r. rewrite app_nil_r. reflexivity.
    - simpl. rewrite ← app_assoc. rewrite ← app_assoc. rewrite IHA. simpl. reflexivity.
Qed.

Fact variable_subsets :
  ∀ s t A, (In (s,t) A) → ((incl (v_term s) (v_list A)) ∧ (incl (v_term t) (v_list A))).
Proof.
  intros. unfold incl in *. split.
  - intros. induction A.
```

```
    + simpl. contradiction.
    + firstorder. destruct H. apply in_or_app. left.
Admitted.

Fact sublist_implies_variable_sublist :
  ∀ A B, (incl A B) → (incl (v_list A) (v_list B)).
Proof.
  intros. unfold incl in *. intros. induction A, B.
  - apply H0.
  - simpl. firstorder.
  - simpl in *. apply IHA.
    + intros.
Admitted.

Definition gen x : ter := (V x).

Lemma non_unifiable_gen_different :
  ∀ m n, m ≠ n → ¬ (unifiable [(gen m, gen n)]).
Proof.
  intros. unfold not. unfold unifiable. intros.
Admitted.

Lemma disjoint_solved_lists :
  ∀ A B, (disjoint (v_list A) (domain B)) → (solved A) → (solved B) → (solved (A ++
B)).
Proof.
  intros.
Admitted.
```