باسم تعالی

طراحی مفسر Let و صحت‌سنجی آن به واسطه Coq

محمد ایزدی ۹۹۱۰۱۲۶۸

دنیا روشن ضمیر ۹۹۱۷۰۴۶۷

فاز اول:

با دو تابع کمکی تابع evaluate را می‌نویسیم

۱- Evaluate_with_env: تابعی بازگشتی که یک عبارت و یک Environment ورودی می‌گیرد. اگر عبرات ورودی از نوع عدد بود، مقدار آن را بر می‌گرداند. اگر متغیر بود مقدار آن در env فعلی را بر می‌گرداند و در صورت وجود نداشتن ارور بر می‌گرداند. اگر Plus بود مقدار دو عبارت را حساب کرده، اگر یکی ارور بود ارور بر گردانده و در غیر این صورت مجموع حاصل دو عبارت را بر می‌گرداند. در نهایت اگر Let بود مقدار عبارت initializer را حساب کرده، در متغیری با نام داده شده به env اضافه می‌کند و مقدار عبارت جدید را بر می‌گرداند و اگر جایی به ارور خورد ارور بر می‌گرداند.

```
Fixpoint evaluate_with_env (expr: Expression) (env: list (string * nat)) : EvalResult :=
  match expr with
  | Num n => Ok n
  | Var v => match lookup_variable v env with
             | Some n => Ok n
             | _ => Error
             end
  | Plus l r => match (evaluate_with_env l env, evaluate_with_env r env) with
                | (Ok n1, Ok n2) => Ok (n1 + n2)
                | _ => Error
                end
  | Let var initializer result =>
    match evaluate_with_env initializer env with
    | Ok n => evaluate_with_env result ((var, n) :: env)
    | _ => Error
    end
  end.
```

۲- lookup_variable: یک env و نام یک متغیر ورودی می‌گیرد و به صورت بازگشتی در env به دنبال مقدار متغیری با نام داده شده می‌گردد و اگر به انتهای آن رسید بدون آن که متغیر را پیدا کند None بر می‌گرداند.

```
Fixpoint lookup_variable (var: string) (env: list (string * nat)) : option nat :=
  match env with
  | nil => None
  | (v, n) :: rest => if string_dec var v then Some n else lookup_variable var rest
  end.
```

برای تابع evaluate تنها کاری که می‌کنیم این است که تابع evaluate_with_env را روی عبارت مربوطه با env خالی صدا می‌کنیم.

```
Definition evaluate (expr: Expression) : EvalResult :=
  evaluate_with_env expr nil.
```

فار دوم:

۱- evaluate_plus: برای این اثبات کافیست تعریف evaluate را در فرض و حکم باز کنیم، حکم را ساده‌سازی کنیم (که شاخه مربوط به evaluate عبارت از نوع plus اجرا شود) و مقادیر فرض را جایگذاری کنیم تا حکم بدیهی شود.

```
Lemma evaluate_plus: forall n1 n2 e1 e2, evaluate e1 = Ok n1 ->
evaluate e2 = Ok n2
-> evaluate (Plus e1 e2) = Ok (n1 + n2).
Proof.
intros n1 n2 e1 e2.
intros H1 H2.
unfold evaluate in *.
simpl.
rewrite H1.
rewrite H2.
reflexivity.
Qed.
```

۲- evaluate_plus_comm: با باز کردن evaluate و ساده کردن عبارت حاصل و حالت‌بندی روی حاصل evaluate_with_env برای عبارت‌هایی که داریم، در صورتی که مقدار حداقل یکی از عبارت‌ها ارور باشد حکم بدیهی است و در صورتی که مقدار هر دو عبارت عدد باشد با استفاده از خاصیت جابه‌جایی جمع که coq به صورت پیش فرض دارد می‌توان حکم را ثابت کرد.

```
Lemma evaluate_plus_comm: forall e1 e2, evaluate (Plus e1 e2) =
evaluate (Plus e2 e1).
Proof.
intros e1 e2.
unfold evaluate in *.
simpl.
destruct evaluate_with_env.
destruct evaluate_with_env.
-rewrite Nat.add_comm. reflexivity.
-reflexivity.
-destruct evaluate_with_env.
  +reflexivity.
  +reflexivity.
Qed.
```

۳- evaluate_error: دو طرف را جداگانه ثابت می‌کنیم.

```
Lemma evaluate_error: forall e, evaluate e = Error <-> exists v,
IsFree v e.
Proof.
intros e. split.
- apply error_implies_free.
- apply Free_implies_error.
Qed.
```

برای اثبات free بدون در صورت وجود ارور داریم:

```
Lemma error_implies_free: forall e,
 evaluate e = Error -> exists v : string, IsFree v e.
intro e. intro H. induction e.
  + inversion H.
  + exists v. apply IsFreeVar.
  + destruct (evaluate e1) eqn:E1.
    -- destruct (evaluate e2) eqn:E2.
      ++ pose proof (evaluate_plus n n0 e1 e2).
         rewrite E1 in H0. rewrite E2 in H0.
       rewrite H in H0. discriminate H0. reflexivity. reflexivity.
      ++ assert (exists v : string, IsFree v e2). apply IHe2. reflexivity.
          destruct H0 as [v1 F]. exists v1. apply IsFreePlusRight. exact F.
    -- assert (exists v : string, IsFree v e1). apply IHe1. reflexivity.
          destruct H0 as [v1 F]. exists v1. apply IsFreePlusLeft. exact F.
  + destruct (evaluate e1) eqn:E1.
    -- clear IHe1. unfold evaluate in H. simpl in H. unfold evaluate in E1. rewrite E1 in H.
      clear E1. destruct (evaluate e2) eqn:Eq2.
      ++ clear IHe2. pose proof (extended_env_error_implies_env_error e2 var n).
          assert (evaluate e2 = Error). rewrite H0. reflexivity. exact H.
           rewrite H1 in Eq2. discriminate.
      ++ assert (exists v : string, IsFree v e2). apply IHe2. reflexivity. clear IHe2.
         destruct H0 as[v1 F]. exists v1. apply IsFreeLetBody. exact F.
          unfold not. intros HF. rewrite HF in F. clear HF v1.
admit.
    -- assert (exists v : string, IsFree v e1). apply IHe1. reflexivity.
        ++ destruct H0 as [v1 H0']. exists v1. apply IsFreeLetInit. exact H0'.
Admitted.
```

```
Lemma extended_env_error_implies_env_error: forall e2 var n,
evaluate_with_env e2 ((var, n) :: nil) = Error -> evaluate e2 = Error.
Proof.
Admitted.
```

روی عبارتی که ارزیابی می‌شود استقرا می‌زنیم.

در صورت عدد یا متغیر بودن آن حکم بدیهی است.

اگر عبارت جمع بود، روی مقدار هر یک از دو عبارت داخل آن حالت‌بندی می‌کنیم. اگر مقدار هر دو OK بود، در فرض‌ها به تناقض می‌رسیم و اگر مقدار یکی ارور بود می‌توان از فرض استقرا فهمید یک متغیر آزاد در آن داریم که در عبارت اصلی هم آزاد خواهد بود.

برای حالت let روی حاصل عبارت اول حالت‌بندی می‌کنیم. اگر ارور بود می‌توان از فرض استقرا استفاده کرد و حکم را نتیجه گرفت. اگر مقدارش OK بود روی مقدار عبارت دوم حالت‌بندی می‌کنیم (در حالی که از فرض استقرا می‌دانیم مقدار عبارت دوم در

env گسترش داده شده ارور دارد) . اگر ارور بود در شرایط تناقض داریم (موفق به اثبات نشدیم) و اگر OK بود از این که خطا داشتن عبارت ارزیابی شده در env گسترش داده شده به معنای خطا در env معمولی است (موفق به اثبات نشدیم) به تناقض می‌رسیم.

برای اثبات ارور داشتن در صورت آزاد بودن:

```
Lemma Free_implies_error : forall e,
(exists v : string, IsFree v e) -> evaluate e = Error.
Proof.
intros e.
intro H.  induction e.
- destruct H as [v F]. inversion F.
- destruct H as [v0 F]. inversion F.
destruct (lookup_variable v nil) as [n |] eqn:L. discriminate.
unfold evaluate. unfold evaluate_with_env. rewrite L. reflexivity.
- destruct H as [v F]. unfold evaluate. simpl. destruct (evaluate e1) eqn:E1.
  + unfold evaluate in E1. rewrite E1. assert (evaluate e2 = Error).
    -- rewrite IHe2. reflexivity. clear IHe1. admit.
    -- unfold evaluate in H. rewrite H. reflexivity.
  + unfold evaluate in E1. rewrite E1. reflexivity.
- destruct H as [v F]. unfold evaluate. simpl. destruct (evaluate e1) eqn:E1.
  + unfold evaluate in E1. rewrite E1. admit.
  + unfold evaluate in E1. rewrite E1. reflexivity.
Admitted.
```

با استقرا روی عبارت.

اگر عدد باشد تناقض در فرض داریم.

اگر متغیر باشد، در خود آزاد است و حکم بدیهی است.

اگر جمع باشد، روی دو عبارت حالت‌بندی می‌کنیم. اگر در یکی متغیر آزاد داشته باشیم مشکلی در اثبات نداریم و در صورتی که هیچ عبارتی شامل متغیر آزاد نباشد تناقض داریم (موفق به اثبات نشدیم).

اگر let باشد، با آزاد بودن متغیر در عبارت اول اثبات بدیهی است. و در صورت آزاد نبودن و نداشتن ارور موفق به اثبات نشدیم.