软件理论基础与实践

TYPECHECKING: A Typechecker for STLC

熊英飞
北京大学

Type-Checking Extended STLC

- Extended STLC仍然具有类型唯一性并可以把has_type关系转换成函数

```
Theorem unique_types : forall Gamma e T T',
  Gamma |- e \in T ->
  Gamma |- e \in T' ->
  T = T'.
```

复习：判断类型等价

```
Fixpoint eqb_ty (T1 T2:ty) : bool :=
  match T1,T2 with
  | <{ Bool }> , <{ Bool }> =>
      true
  | <{ T11->T12 }>, <{ T21->T22 }> =>
      andb (eqb_ty T11 T21) (eqb_ty T12 T22)
  | _,_ =>
      false
  end.
```

复习：类型检查函数

```
Fixpoint type_check (Gamma : context) (t : tm) : option ty :=
  match t with
  | tm_var x =>
      Gamma x
  | <{\x:T2, t1}> =>
      match type_check (x |-> T2 ; Gamma) t1 with
      | Some T1 => Some <{T2->T1}>
      | _ => None
      end
  | <{t1 t2}> =>
      match type_check Gamma t1, type_check Gamma t2 with
      | Some <{T11->T12}>, Some T2 =>
          if eqb_ty T11 T2 then Some T12 else None
      | _,_ => None
      end
```

复习：类型检查函数

```
| <{true}> =>
    Some <{Bool}>
| <{false}> =>
    Some <{Bool}>
| <{if guard then t else f}> =>
    match type_check Gamma guard with
    | Some <{Bool}> =>
        match type_check Gamma t, type_check Gamma f with
        | Some T1, Some T2 =>
            if eqb_ty T1 T2 then Some T1 else None
        | _,_ => None
        end
    | _ => None
    end
end.
```

改进符号

大量嵌套的match不好书写，引入类似命令式语言的符号

```
Notation " x <- e1 ;; e2" := (match e1 with
                              | Some x => e2
                              | None => None
                              end)
         (right associativity, at level 60).

Notation " 'return' e "
  := (Some e) (at level 60).

Notation " 'fail' "
  := None.
```

即定义函数式语言中的Monad支撑机制在option上的的特化版本

改进符号

```
| <{\x:T2, t1}> =>
      match type_check (x |-> T2 ; Gamma) t1 with
      | Some T1 => Some <{T2->T1}>
      | _ => None
      end
```

```
| <{\x:T2, t1}> =>
    T1 <- type_check (x |-> T2 ; Gamma) t1 ;;
    return <{T2->T1}>
```

改进符号

```
| <{t1 t2}> =>
    match type_check Gamma t1, type_check Gamma t2 with
    | Some <{T11->T12}>, Some T2 =>
        if eqb_ty T11 T2 then Some T12 else None
    | _,_ => None
    end
```

```
| <{t1 t2}> =>
    T1 <- type_check Gamma t1 ;;
    T2 <- type_check Gamma t2 ;;
    match T1 with
    | <{T11->T12}> =>
        if eqb_ty T11 T2 then return T12 else fail
    | _ => fail
    end
```

扩展eqb_ty

```
Fixpoint eqb_ty (T1 T2 : ty) : bool :=
  match T1,T2 with
  | <{{Nat}}>, <{{Nat}}> =>
      true
  | <{{Unit}}>, <{{Unit}}> =>
      true
  | <{{T11 -> T12}}>, <{{T21 -> T22}}> =>
      andb (eqb_ty T11 T21) (eqb_ty T12 T22)
  | <{{T11 * T12}}>, <{{T21 * T22}}> =>
      andb (eqb_ty T11 T21) (eqb_ty T12 T22)
  | <{{T11 + T12}}>, <{{T21 + T22}}> =>
      andb (eqb_ty T11 T21) (eqb_ty T12 T22)
  | <{{List T11}}>, <{{List T21}}> =>
      eqb_ty T11 T21
  | _,_ =>
      false
  end.
```

eqb_ty的性质

```
Lemma eqb_ty_refl : forall T,
  eqb_ty T T = true.

Lemma eqb_ty__eq : forall T1 T2,
  eqb_ty T1 T2 = true -> T1 = T2.
```

问：那为什么还要定义eqb_ty？为什么不直接使用=？
答：在递归类型、精化类型等复杂类型系统中，语法上不等的类型也有可能等价

复习：在Coq中定义STLC

- <{...}>定义项
- <{{...}}>定义类型

练习：定义Sum的类型检查分支

```
Fixpoint type_check (Gamma : context) (t : tm) : option ty :=
  match t with
  (* Complete the following cases. *)
  (* sums *)
  | …  =>
    …
  | …
```

答案：定义Sum的类型检查分支

```
Fixpoint type_check (Gamma : context) (t : tm) : option ty :=
  match t with
  (* Complete the following cases. *)
  (* sums *)
  | <{inl Tr t}> =>
    Tl <- type_check Gamma t ;;
    return <{{Tl + Tr}}>
  | <{inr Tl t}> =>
    Tr <- type_check Gamma t ;;
    return <{{Tl + Tr}}>
  | <{case t0 of | inl x1 => t1 | inr x2 => t2}> =>
    T0 <- type_check Gamma t0 ;;
    match T0 with
    | <{{Tl + Tr}}> =>
        T1 <- type_check (x1 |-> Tl ; Gamma) t1 ;;
        T2 <- type_check (x2 |-> Tr ; Gamma) t2 ;;
        if eqb_ty T1 T2 then return T1 else fail
    | _ => fail
    end
```

证明Sum的类型检查正确性

```
Theorem type_checking_sound : forall Gamma t T,
  type_check Gamma t = Some T ->
  has_type Gamma t T.
```

复习： solve_by_invert

- 定义策略在任意命题上反复应用n次inversion

```
Ltac solve_by_inverts n :=
  match goal with | H : ?T |- _ =>
  match type of T with Prop =>
    solve [
      inversion H;
      match n with S (S (?n')) =>
        subst; solve_by_inverts (S n’)
      end ]
  end end.

Ltac solve_by_invert :=
  solve_by_inverts 1.
```

|-： 匹配目标（复习）
match type of X with
Type： 匹配类型
solve [策略]： 如果策略没有完成当前目标证明，就报错
（避免进入执行完策略但没有证明目标的情况）

复习：type_checking_sound中app的证明

在证明工具中演示

定义一系列的证明策略

```
Ltac invert_typecheck Gamma t T :=
  remember (type_check Gamma t) as TO;
  destruct TO as [T|];
  try solve_by_invert; try (inversion H0; eauto); try (subst; eauto).
```

用于消除T <- type_check Gamma t的运算

```
Ltac analyze T T1 T2 :=
  destruct T as [T1 T2| |T1 T2|T1| |T1 T2]; try solve_by_invert.
```

用于消除以下运算：

```
match T1 with
     | … => fail
     | <{ … }> => return …
     end
```

定义一系列的证明策略

```
Ltac case_equality S T :=
  destruct (eqb_ty S T) eqn: Heqb;
  inversion H0; apply eqb_ty__eq in Heqb; subst; subst; eauto.
```

用于消除如下运算

`if eqb_ty S T then return … else fail`

重写type_checking_sound中app的证明

在证明工具中演示

练习：定义Sum的类型检查证明分支

```
Theorem type_checking_sound : forall Gamma t T,
  type_check Gamma t = Some T ->
  has_type Gamma t T.
Proof with eauto.
  intros Gamma t. generalize dependent Gamma.
  induction t; intros Gamma T Htc; inversion Htc.
  …
(* sums *)
  - invert_typecheck Gamma t0 Tl.
  - invert_typecheck Gamma t0 Tr.
  - invert_typecheck Gamma t1 T0.
    analyze T0 Tl Tr.
    remember (s |-> Tl; Gamma) as Gamma1.
    invert_typecheck Gamma1 t2 T1.
    remember (s0 |-> Tr; Gamma) as Gamma2.
    invert_typecheck Gamma2 t3 T2.
    case_equality T1 T2.
```

作业

- 完成TypeChecking中
  - type_check_defn和ext_type_checking_sound的
    - Sums
    - Lists
    - Fix