软件理论基础与实践

MORESTLC: More on the Simply Typed Lambda-Calculus

熊英飞
北京大学

扩展STLC

- STLC目前只有布尔类型和基本函数调用
- 加入更多语言成分
  - 自然数
  - Let
  - Pairs
  - Unit
  - Sums
  - Lists
  - 递归调用
  - Records

自然数

- 直接将Types部分定义的自然数语法、语义和类型规则加入即可

Let

let x = 1+5 in

let y = x + 1 in

y + 2

Let

Syntax:

```
t ::=                    Terms
    | ...                   (other terms same as before)
    | let x=t in t          let-binding
```

Reduction:

$$\frac{t_1 \rightarrow t_1'}{\texttt{let } x=t_1 \texttt{ in } t_2 \rightarrow \texttt{let } x=t_1' \texttt{ in } t_2} \quad \text{(ST\_Let1)}$$

$$\frac{}{\texttt{let } x=v_1 \texttt{ in } t_2 \rightarrow [x:=v_1]t_2} \quad \text{(ST\_LetValue)}$$

Typing:

$$\frac{\text{Gamma} \vdash t_1 \in T_1 \qquad x \mapsto T_1;\ \text{Gamma} \vdash t_2 \in T_2}{\text{Gamma} \vdash \texttt{let } x=t_1 \texttt{ in } t_2 \in T_2} \quad \text{(T\_Let)}$$

Unit

Syntax:

```
t ::=                   Terms
    | ...                   (other terms same as before)
    | unit                  unit

v ::=                   Values
    | ...
    | unit                  unit value

T ::=                   Types
    | ...
    | Unit                  unit type
```

Typing:

$$\frac{}{\text{Gamma} \vdash \text{unit} \in \text{Unit}} \quad \text{(T\_Unit)}$$

命令作为项

- 没有返回值的命令可以认为返回Unit
  - t1 := t2 : Unit
- 命令的序列可以看做函数调用的简写
  - t1;t2等价于(\x:Unit, t2) t1
- 之后在Reference章节会进一步学习

Pairs – 示例

```
\x : Nat*Nat,
   let sum = x.fst + x.snd in
   let diff = x.fst - x.snd in
   (sum,diff)
```

Pairs-语法

```
t ::=                     Terms
    | ...
    | (t, t)                  pair
    | t.fst                   first projection
    | t.snd                   second projection

v ::=                     Values
    | ...
    | (v, v)                  pair value

T ::=                     Types
    | ...
    | T * T                   product type
```

Pairs-语义

$$\frac{t_1 \rightarrow t_1'}{(t_1, t_2) \rightarrow (t_1', t_2)} \quad \text{(ST\_Pair1)}$$

$$\frac{t_2 \rightarrow t_2'}{(v_1, t_2) \rightarrow (v_1, t_2')} \quad \text{(ST\_Pair2)}$$

$$\frac{t_1 \rightarrow t_1'}{t_1.\mathrm{fst} \rightarrow t_1'.\mathrm{fst}} \quad \text{(ST\_Fst1)}$$

$$\frac{}{(v_1, v_2).\mathrm{fst} \rightarrow v_1} \quad \text{(ST\_FstPair)}$$

$$\frac{t_1 \rightarrow t_1'}{t_1.\mathrm{snd} \rightarrow t_1'.\mathrm{snd}} \quad \text{(ST\_Snd1)}$$

$$\frac{}{(v_1, v_2).\mathrm{snd} \rightarrow v_2} \quad \text{(ST\_SndPair)}$$

Pairs-类型

$$\frac{\text{Gamma} \vdash t_1 \in T_1 \qquad \text{Gamma} \vdash t_2 \in T_2}{\text{Gamma} \vdash (t_1, t_2) \in T_1 * T_2} \quad \text{(T\_Pair)}$$

$$\frac{\text{Gamma} \vdash t_0 \in T_1 * T_2}{\text{Gamma} \vdash t_0.\text{fst} \in T_1} \quad \text{(T\_Fst)}$$

$$\frac{\text{Gamma} \vdash t_0 \in T_1 * T_2}{\text{Gamma} \vdash t_0.\text{snd} \in T_2} \quad \text{(T\_Snd)}$$

Records – 示例

\x: {age:Nat, sex:Bool},

if x.age > 18 then tru else fls

Records-语法

```
t ::=                                  Terms
    | ...
    | {i1=t1, ..., in=tn}                 record
    | t.i                                 projection

v ::=                                  Values
    | ...
    | {i1=v1, ..., in=vn}                 record value

T ::=                                  Types
    | ...
    | {i1:T1, ..., in:Tn}                 record type
```

Records-语义

$$\frac{ti \longrightarrow ti'}{\{i_1=v_1,\ \ldots,\ im=vm,\ in=ti\ ,\ \ldots\} \longrightarrow \{i_1=v_1,\ \ldots,\ im=vm,\ in=ti',\ \ldots\}} \quad \text{(ST\_Rcd)}$$

$$\frac{t_0 \longrightarrow t_0'}{t_0.i \longrightarrow t_0'.i} \quad \text{(ST\_Proj1)}$$

$$\frac{}{\{\ldots,\ i=vi,\ \ldots\}.i \longrightarrow vi} \quad \text{(ST\_ProjRcd)}$$

Records

$$\frac{\text{Gamma} \vdash t_1 \in T_1 \qquad \ldots \qquad \text{Gamma} \vdash tn \in Tn}{\text{Gamma} \vdash \{i_1 = t_1, \ \ldots, \ in = tn\} \in \{i_1 : T_1, \ \ldots, \ in : Tn\}} \quad \text{(T\_Rcd)}$$

$$\frac{\text{Gamma} \vdash t_0 \in \{\ldots, \ i : Ti, \ \ldots\}}{\text{Gamma} \vdash t_0 . i \in Ti} \quad \text{(T\_Proj)}$$

Records可以表示为Pair和Unit

- {age=5, sex=tru}
  表示为
  (5, (tru, unit))

- 确实有编译器是这样实现Record的，不过更常见的是用偏移量

Sum-示例

div ∈ Nat -> Nat -> (Nat + Unit)

```
div =
 \x:Nat, \y:Nat,
  if iszero y then
   inr Nat unit
  else
   inl Unit (x / y)
```

Sum-语法

```
t ::=                   Terms
    | ...                   (other terms same as before)
    | inl T t               tagging (left)
    | inr T t               tagging (right)
    | case t of             case
        inl x => t
      | inr x => t

v ::=                   Values
    | ...
    | inl T v               tagged value (left)
    | inr T v               tagged value (right)

T ::=                   Types
    | ...
    | T + T                 sum type
```

Sum-语义

$$\frac{t_1 \to t_1'}{\texttt{inl}\ T_2\ t_1 \to \texttt{inl}\ T_2\ t_1'} \quad \text{(ST\_Inl)}$$

$$\frac{t_2 \to t_2'}{\texttt{inr}\ T_1\ t_2 \to \texttt{inr}\ T_1\ t_2'} \quad \text{(ST\_Inr)}$$

$$\frac{t_0 \to t_0'}{\begin{array}{c}\texttt{case}\ t_0\ \texttt{of inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \to \\ \texttt{case}\ t_0'\ \texttt{of inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2\end{array}} \quad \text{(ST\_Case)}$$

$$\frac{}{\begin{array}{c}\texttt{case}\ (\texttt{inl}\ T_2\ v_1)\ \texttt{of inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \\ \to [x_1 := v_1] t_1\end{array}} \quad \text{(ST\_CaseInl)}$$

$$\frac{}{\begin{array}{c}\texttt{case}\ (\texttt{inr}\ T_1\ v_2)\ \texttt{of inl}\ x_1 \Rightarrow t_1 \mid \texttt{inr}\ x_2 \Rightarrow t_2 \\ \to [x_2 := v_2] t_2\end{array}} \quad \text{(ST\_CaseInr)}$$

Sum-类型

$$\frac{\text{Gamma} \vdash t_1 \in T_1}{\text{Gamma} \vdash \text{inl } T_2 \ t_1 \in T_1 + T_2} \quad \text{(T\_Inl)}$$

$$\frac{\text{Gamma} \vdash t_2 \in T_2}{\text{Gamma} \vdash \text{inr } T_1 \ t_2 \in T_1 + T_2} \quad \text{(T\_Inr)}$$

$$\frac{\begin{array}{c}\text{Gamma} \vdash t_0 \in T_1+T_2 \\ x_1 \mapsto T_1;\ \text{Gamma} \vdash t_1 \in T_3 \\ x_2 \mapsto T_2;\ \text{Gamma} \vdash t_2 \in T_3\end{array}}{\text{Gamma} \vdash \text{case } t_0 \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \in T_3} \quad \text{(T\_Case)}$$

Variant

- 同Record类似，Sum也可以扩展为Variant
  - <some:Nat, none:unit>
- Variant和Record合用，可以起到Coq中Inductive定义的作用

```
Inductive rgb : Type :=
  | red
  | green
  | blue.
Inductive color : Type :=
  | black
  | white
  | primary (p : rgb).
```

```
<black:unit,
 white:unit,
 primary: {p:<red:unit,
              green:unit,
              blue:unit>}>
```

- 练习：用Variant定义natlist

List

- 在支持Universal Type和Recursive Type的编程语言中，List可以定义为用户定义类型
- 本课程不涉及以上两种类型，所以将List定义为语言扩展

```
Inductive list (X:Type) : Type :=
  | nil
  | cons (x : X) (l : list X).
```

List-语法

```
t ::=                     Terms
      | ...
      | nil T
      | cons t t
      | case t of nil    => t
                  | x::x => t

v ::=                     Values
      | ...
      | nil T                  nil value
      | cons v v               cons value

T ::=                     Types
      | ...
      | List T                 list of Ts
```

List-语义

$$\frac{t_1 \to t_1'}{\text{cons } t_1\ t_2 \to \text{cons } t_1'\ t_2} \quad \text{(ST\_Cons1)}$$

$$\frac{t_2 \to t_2'}{\text{cons } v_1\ t_2 \to \text{cons } v_1\ t_2'} \quad \text{(ST\_Cons2)}$$

$$\frac{t_1 \to t_1'}{\begin{array}{c}(\text{case } t_1 \text{ of nil} \Rightarrow t_2 \mid xh::xt \Rightarrow t_3) \to \\ (\text{case } t_1' \text{ of nil} \Rightarrow t_2 \mid xh::xt \Rightarrow t_3)\end{array}} \quad \text{(ST\_Lcase1)}$$

$$\frac{}{\begin{array}{c}(\text{case nil } T_1 \text{ of nil} \Rightarrow t_2 \mid xh::xt \Rightarrow t_3) \\ \to t_2\end{array}} \quad \text{(ST\_LcaseNil)}$$

$$\frac{}{\begin{array}{c}(\text{case (cons vh vt) of nil} \Rightarrow t_2 \mid xh::xt \Rightarrow t_3) \\ \to [xh:=vh, xt:=vt]t_3\end{array}} \quad \text{(ST\_LcaseCons)}$$

List-类型

$$\frac{}{\text{Gamma} \vdash \text{nil } T_1 \in \text{List } T_1} \quad \text{(T\_Nil)}$$

$$\frac{\text{Gamma} \vdash t_1 \in T_1 \qquad \text{Gamma} \vdash t_2 \in \text{List } T_1}{\text{Gamma} \vdash \text{cons } t_1\ t_2 \in \text{List } T_1} \quad \text{(T\_Cons)}$$

$$\frac{\begin{array}{c}\text{Gamma} \vdash t_1 \in \text{List } T_1 \\ \text{Gamma} \vdash t_2 \in T_2 \\ (h \mapsto T_1;\ t \mapsto \text{List } T_1;\ \text{Gamma}) \vdash t_3 \in T_2\end{array}}{\text{Gamma} \vdash (\text{case } t_1 \text{ of nil} \Rightarrow t_2 \mid h::t \Rightarrow t_3) \in T_2} \quad \text{(T\_Lcase)}$$

递归

- 递归的本质是要调用自己，但目前STLC中并没有“自己”这个概念
- 思路：把自己作为参数传入
  - ```
    fact = \self:Nat->Nat,
                  \x:Nat,
                    if x=0 then 1 else x * (self (pred x))
    ```
- 然后定义高阶函数负责传入“自己”
  - fix fact: Nat->Nat
- fix可以在lambda演算中定义，但其类型是递归类型，本课程不涉及
  - 作为语言成分定义
    ```

递归

Syntax:

```
t ::=                Terms
      | ...
      | fix t            fixed-point operator
```

Reduction:

$$\frac{t_1 \rightarrow t_1'}{\text{fix } t_1 \rightarrow \text{fix } t_1'} \quad \text{(ST\_Fix1)}$$

$$\frac{}{\text{fix } (\backslash xf{:}T_1.\, t1) \rightarrow [xf{:=}\text{fix } (\backslash xf{:}T_1.\, t1)]\ t_1} \quad \text{(ST\_FixAbs)}$$

Typing:

$$\frac{\text{Gamma} \vdash t_1 \in T_1\text{->}T_1}{\text{Gamma} \vdash \text{fix } t_1 \in T_1} \quad \text{(T\_Fix)}$$

Progess和Preservation

- 二者在扩展后的STLC上仍然成立

```
Theorem progress : forall t T,
     empty |- t \in T ->
     value t \/ exists t', t --> t'.
```

```
Theorem preservation : forall t t' T,
     empty |- t \in T  ->
     t --> t'  ->
     empty |- t' \in T.
```

- 具体证明留作作业

作业

- 完成MoreSTLC中standard非optional的6道习题