# $\lambda$ calculus with primitives(原语/原生项/原始项)

| 类别 | 表面语法 | 抽象语法 | 注释 |
| --- | --- | --- | --- |
| $\text{Term } t ::=$ | $x$ | $\text{Var } x$ | |
| | $\backslash\ x\ .\ t$ | $\text{Lam } (x, t)$ | $\lambda$ calculus 部分 |
| | $t_1\ t_2$ | $\text{App } (t_1, t_2)$ | |
| | $n$ | $\text{Int } n$ | 原生(非负)整数 |
| | $b$ | $\text{Bool } b$ | 原生布尔值 |
| | $t_1 + t_2$ | $\text{Add } (t_1, t_2)$ | |
| | $t_1 - t_2$ | $\text{Sub } (t_1, t_2)$ | |
| | $t_1 < t_2$ | $\text{Lt } (t_1, t_2)$ | |
| | $t_1 = t_2$ | $\text{Eq } (t_1, t_2)$ | Int 的相等判断 |
| | $\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \text{ end}$ | $\text{If } (t_1, t_2, t_3)$ | if 表达式 |

表面语法的优先级与结合性类似于 OCaml 的, 使用 `()` 调整优先级.

# Simply Typed Lambda Calculus, STLC

`3 + true` 会带来错误, 语言不会做隐式转换(implicit conversion).

| 类别 | 表面语法 | 抽象语法 | 注释 |
| --- | --- | --- | --- |
| $\text{Type}\ \tau ::=$ | $\text{Int}$ | $\text{Int}$ | |
| | $\text{Bool}$ | $\text{Bool}$ | |
| | $\tau_1 \to \tau_2$ | $\text{Lam}\ (\tau_1, \tau_2)$ | 一般是 $\backslash\ x : \tau_1\ .\ t$ 的类型 |
| $------$ | $------$ | $------$ | $-----------$ |
| $\text{Term}\ t ::=$ | $x$ | $\text{Var}\ x$ | |
| | $\backslash\ x : \tau\ .\ t$ | $\text{Lam}\ (x, \tau, t)$ | |
| | $t_1\ t_2$ | $\text{App}\ (t_1, t_2)$ | |
| | $\dots$ | $\dots$ | 余下与上页相同 |

## `fix` 拓展

- Y 组合子 `\ f. (\x. f(x x))(\x. f(x x))` 在 STLC 中无法赋型:
  式中 `x` 的类型必须是某种 `A->B`, 因为 `x x`, 但这使得 `A = A->B`!
- 因此引入 `fix` 拓展用以定义递归函数, 以 `sum n = 0 + 1 + ... + n` 为例:

| 类别 | 表面语法 | 抽象语法 | 注释 |
|---|---|---|---|
| $\text{Term } t ::=$ | $\ldots$ | $\ldots$ | 与前文相同 |
| | $\text{fix } f : \tau = t$ | $\text{Fix}\,(f, \tau, t)$ | |

```
fix sum : Int -> Int = \ n : Int .
    if n = 0 then 0 else n + sum (n-1) end
```

- 对比: $\text{fix } f : \tau = t$ 和 $\lambda\, x : \tau .\, t$ 都引入了绑定变量, 一个是 `f`, 一个是 `x`.

## 递归: 延伸阅读

- Y 组合子在 unityped lambda calculus 里可赋型, 见 PFPL 第 20 章与第 21 章第 4 节.
- `fix` 拓展来自于 PFPL 第 19 章.
- 可以添加其他的原始项来处理递归, 例如 PFPL 第 9 章系统 T 中的原始递归函数.
- (余)归纳类型也是一种选择, 你已经在 OCaml 里定义过不少归纳类型, 例如:

  ```
  type term =
        | Var of string
        | Lam of string * term
        | App of term * term
  ```

  在一些语言里(例如 Coq ), (余)归纳类型用于保证程序可停机. 见 PFPL 第 15 章.