§1 函数

函数就是两个集合之间特定的对应“关系”，这个特定体现在一旦输入给定，输出就不能发生变化.

定义 1. (全函数) 如果从 A 到 B 的函数是全函数，记作 $A \xrightarrow{t} B$，就是满足

- $\forall a \in A.\forall b, b' \in B.(((a,b) \in f \wedge ((a,b') \in f \implies b = b'))$. (函数)
- $\forall a \in A.\exists b \in B \text{ s.t. } (a,b) \in f$. (全都有定义)

注. (1) 第一条是满足函数的性质; 第二条说明这个函数在定义域上的每个元素都有定义.

(2) 对于一般的函数，如果 f 在 a 处没有定义，可以写为 $f(a) = \perp$. 如果把 $\perp$ 当做一种特殊取值的话，那么可以扩展为全函数 $f : A \xrightarrow{t} (B + \{\perp\})$.

定义 2. (函数的相等) 对于两个函数 $f, g : A \to B$, $f = g \iff \forall a \in A.f(a) = g(a)$.

定义 3. (函数的复合) 如果 $f : A \to B, g : B \to C$，那么函数 $g \circ f = A \to C$，对应法则为 $(g \circ f)(a) = g(f(a)), \forall a \in A$.

定义 4. (有限函数和函数值更新) 如果 $f : A \to B$ 是偏函数，$a_1, \cdots, a_n \in A; b_1, b_2, \cdots, b_n \in B.f[a_1 \mapsto b_1, a_2 \mapsto b_2, \cdots, a_n \mapsto b_n]$ 表示新的映射 $g : A \to B$ 满足 $g(x) = \begin{cases} b_i & \text{对某些}i\text{, 如果}x = a_i \\ f(x) & \text{对所有的}i\text{, 如果}x \neq a_i \end{cases}$.

注. 这看上去就像把函数 f 的值更新了. 有时候也称为继承.

问题 1. 数学里面的函数和计算机科学的函数有什么不同?

解答. 实际上普通程序语言的函数 (如 C, C++) 并不是纯正意义上的数学函数. 比如它可能会调用全局变量导致即使有相同的输入参数，结果也可能发生改变.

当然，如果认为全局变量 (或者栈帧) 也在输入的一部分，那他们确实是一样的.

§2 编程语言中的类型

简单起见，可以认为**类型是取值的集合**. 这个取值的集合暗示了我们希望具有这个类型变量的值取得什么. 比如通常的可认为是 `int` 类型的取值范围是 $[-2147383648, 2147483647]$.

2.1. 乘积和函数类型.

定义 5. (函数类型) 如果 f 是一个从 A 到 B 的偏函数，我们说 $f : A \to B$，并且说 f 的类型是 $A \to B$.

注. (1) A, B 都是某些类型.

(2) 如果 A, B 都不是函数类型，那么 $A \to B$ 称为一阶函数; 反之称为高阶函数.

(3) 如果是全函数可以记其类型为 $A \xrightarrow{t} B$.
(4) 函数类型的结合律在右侧. 即 $A \to (B \to C)$ 可以缩写为 $A \to B \to C$.
(5) 函数应用的结合律是左结合的. 即 fab 实际上指的是 $(f(a))(b)$.

定义 6. (乘积类型) 如果 $A_1, A_2, A_3, \cdots, A_n$ 是类型, 那么 $A_1 \times A_2 \times \cdots \times A_n$ 是乘积类型.

例 1. 高阶函数.

(1) 定义 $add(n) = p, p(x) = x + n$. 其中 add 的类型为 $\mathbb{N} \xrightarrow{t} \mathbb{N} \xrightarrow{t} \mathbb{N}$. 例如 $add(1)(7) = p(7)_{n=1} = 7 + 1 = 8$.

(2) 定义 $twice(f)(x) = f(f(x))$, 其具有类型 $(\mathbb{N} \to \mathbb{N}) \xrightarrow{t} (\mathbb{N} \to \mathbb{N})$; 如果 $square(x) = x^2 (\mathbb{N} \to \mathbb{N})$ 作为参数, 就有 $twice(square) = j$, 满足 $j(x) = x^4$.

函数的部分带入 (Curry) 对于二元及以上的函数, 我们可以带入部分表达式来获得求值原函数的效果. 例如, 定义 $plus(x, y) = x + y$ 的类型是 $(\mathbb{N} \times \mathbb{N}) \to \mathbb{N}$; 定义 $add(n)$(上例) 的类型就是 $\mathbb{N} \xrightarrow{t} (\mathbb{N} \xrightarrow{t} \mathbb{N})$.

2.2. 类型推断.

定义 7. (类型) **类型是取值的集合**. 约定 "x 的类型是 t" 简单记作 $x : t$. 如果一个表达式是 well-typed, 当且仅当所有的参数和参与运算的算符的类型都是合适的. 如果一个表达式 e 是 well-typed, 并且最后的值的类型是 t, 记作 $e : t$.

如果 e 是表达式, 出现了变量 x. 只有在做出某些假定下, x 的类型才可以确定. 这样的假定被称为类型环境, 记作 $\tau = [x \mapsto t, \cdots]$, 表示将 x 映射到它的类型: $\tau x = t$.

对于类型的推理可以记作 $\tau \vdash e : t$ 表示在环境 τ 中, e 是 well-typed, 并且具有类型 t. 或者可以竖着写作 $\frac{\tau}{e : t}$.

例 2. 类型的推理规则. 像数理逻辑那样, 只要 $x : \mathbb{N}$, 就有 $x + x : \mathbb{N}$ 可以记作 $[x \mapsto \mathbb{N}] \vdash x + x : \mathbb{N}$.

通常情况下只用对某些部分更新, 因此可以记作 $\tau[x \mapsto t] \vdash x : t$.

例 3. 若干实例.

1. Bool 表达式类型 $\mathbb{B} = \{\texttt{true}, \texttt{false}\}$, 有

$$\frac{\tau \vdash e : \mathbb{B}}{\tau \vdash not\ e : \mathbb{B}}.$$

2. 考虑两个整数类型的加法:

$$\frac{\tau \vdash e_1 : \mathbb{Z} \qquad \tau \vdash e_2 : \mathbb{Z}}{\tau \vdash e_1 + e_2 : \mathbb{Z}}.$$

3. 考虑有序对.

$$\frac{\tau \vdash e_1 : t_1 \qquad \tau \vdash e_2 : t_2}{\tau \vdash (e_1, e_2) : t_1 \times t_2}.$$

4. 考虑高阶函数.

$$\frac{\tau \vdash f : t \to t' \qquad \tau \vdash e : t}{\tau \vdash f(e) : t'}.$$

例 4. 多步骤的类型推导:

1. $(m+n, m-n), m, n \in \mathbb{Z}$.

$$\frac{\dfrac{\tau \vdash m : \mathbb{Z} \qquad \tau \vdash n : \mathbb{Z}}{r \vdash m+n : \mathbb{Z}} \qquad \dfrac{\tau \vdash m : \mathbb{Z} \qquad \tau \vdash n : \mathbb{Z}}{r \vdash m-n : \mathbb{Z}}}{\tau : (m+n, m-n) : \mathbb{Z} \times \mathbb{Z}}.$$

2. $\tau = [f \mapsto \mathbb{N} \to \mathbb{N}, x \mapsto \mathbb{N}]$, 有

$$\frac{\tau \vdash f : \mathbb{N} \to \mathbb{N} \qquad \dfrac{\tau \vdash f : \mathbb{N} \to \mathbb{N} \qquad \tau \vdash x : \mathbb{N}}{\tau : f(x) : \mathbb{N}}}{\tau \vdash f(f(x)) : \mathbb{N}}.$$

2.3. 和类型. 为什么提出和类型. 回顾 C 语言中, 空指针 `NULL` 只不过是 `0` 的又一个定义. 概念上就可以使用类似于 `Inttag(0)` 表示这是一个整数 0; 而 `Pointertag(0)` 可以概念上认为这是空指针 `NULL`.

可以看做这样的一个结构体:

```
struct numeric {
  enum numeric_kind kind; // INT or FLOAT
  union {
    int i;
    float f;
  } data;
};
```

定义 8. (和类型) 如果 C_i 是标记 (构造子), t_i 是类型, 那么有和类型

$$C_1 t_{11} \times \cdots \times C_1 t_{1k_1} | \cdots | C_n t_{n1} \times \cdots \times C_n t_{nk_n}$$

. 其中 $n \geq 1, k_i \geq 0$, 且 C_i 要不同, 我们说属于这个的类型是由 $C_i(v_{i1}, \cdots, v_{ik_i})$ 构造而来的 $(v_{i1} : t_{i1}, \cdots, v_{ik_i} : t_{ik_i})$. 它表示下列的值的集合:

$$\bigcup_{i=1}^{n} \{C_i(v_1, \cdots, v_{k_i}) | v_j : t_{ij}, j = 1, 2, \cdots, k_i\}.$$

注. (1) 为什么 C_i 要不同? 这是因为如果相同了等于重复了, 为了方便起见就没有包含相同的.

2.4. 递归的类型.

定义 9. (递归类型) 可以递归地定义如下的类型 (数据类型) 作为和类型 $T = C_1 t_{11} \times \cdots \times t_{1k_1} | \cdots | C_n t_{n1} \times \cdots \times t_{nk_n}$. 其中 t_{ij} 可以包含 T 或者其他类型的名字. 所有的构造子 $C_1, C_2, \cdots, C_n$ 必须是不同的.

注. (1) 递归的数据类型和和类型很像, 有什么区别? 注意到和类型中所有的 t_i 不会包含他自己; 但是这里的 t_i 可以包含自己, 所以它产生的集合可能会任意大.

例 5. List 类型. 有限列表被定义为 $As = \ nils \mid cons\ A \times As$, 其中 $nils, cons$ 为构造子. 这表明: As 的元素

- 要么是 nil;
- 要么有 $cons(a, as), a : A, as : As$ 的形式.

有限长度的列表, 元素在 A 中, 通常称为 A^*. 如 $cons(7, cons(9, cons(13, nil)))$ 具有类型 $\mathbb{Z}^*$. 也可以写作 $[7, 9, 13]$ 或者 $7 :: 9 :: 13$.

List 类型的语义. 上述内容的集合描述实际上是 $As = \{nil\} \cup \{cons(a, as) | a \in A \wedge as \in As\}$. 这可以通过求闭包得到.

例 6. 语法解析树. 从编译原理课上知道语法解析树有如下的形式: $Nexp = \ Int\ \mathbb{N} |\ Add\ Nexp \times Nexp |\ Mul\ Nexp \times Nexp$.