简易类型系统

λ演算虽然是一门功能齐备的编程语言，但绝对称不上完善和易用。它的两大缺点已在上一章提及：

1° 没有区分不同性质的对象。比如，将数字0、真值False和函数λf.λg.g混为一谈。固然，只要程序员在正确的位置使用了正确的对象，程序仍可正确地运行；可是，只要~~稍~~稍有不慎，程序就将被引入不可控的境地。更糟糕的是，人们难以发现它的运行有误，或许一直被蒙在鼓里。

2° 没有静态检查机制。给定一个程序，哪怕它再简单也好，λ演算都没有方法检查它是否能「正常终止」，除非把程序运行一下试试。这不利于人们调试。

类型系统就是为解决这两个问题而生的。它扩充了纯而又纯的λ演算，使之除却「函数」以外还有诸如「真值」、「整数」、「列表」等一系列构造，既区分开了不同性质的常用对象，也在一定程度上帮人们检查出不正确的程序。就拿真值来举例：True和False至此不再通过函数来定义，而摇身变成与函数平起平坐的一等公民，不可再分。为了能够操作之，语言中将新增if-then-else语句，功用形同Jump。

引入类型系统大大增加了语法和语义的复杂性，这是为了换取开发效率所付出的代价。

本章介绍一个简易——甚至简陋——的类型系统，仅仅~~增~~增加了对真值的支持。后续章节将扩充之。

初步图景是这样的：把它扩充为 $\{\lambda, \cdot, (,), True, False, if, else, then\} \cup V$，把语法扩充为

$$
\begin{aligned}
t ::=\ & x \\
|\ & (\lambda x.t) \\
|\ & (t\ t) \\
|\ & True \\
|\ & False \\
|\ & (if\ t\ then\ t\ else\ t).
\end{aligned}
$$

再在语义上加入

$$\frac{t_1 \to t_1'}{if\ t_1\ then\ t_2\ else\ t_3 \to if\ t_1'\ then\ t_2\ else\ t_3}$$

$$\frac{}{if\ True\ then\ t_2\ else\ t_3 \to t_2}$$

$$\frac{}{if\ False\ then\ t_2\ else\ t_3 \to t_3}$$

这三条。看起来没什么困难的。

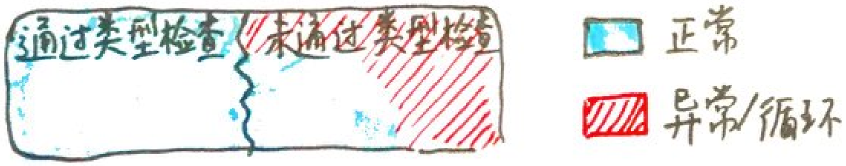
可是，这么做至多只是提升了True和False的地位，从符号上区分开了真值和函数；问题2°仍旧无法得到解决。比方说，程序员写了这么一段程序：

$\lambda b.$
$\quad if\ (if\ b\ then\ 0\ else\ True)$ 仍与以前一样用函数定义
$\quad then\ 36\ else\ 64$

（假定输入b是True或False），那么，在输入True时程序会卡住，而输入False时则不会。归根结底，「if b then 0 else True」一句的「返回值」类型不统一，导致了潜在的异常行为。

当程序非常庞大时，类似的错误很难排查。我们又绕回了问题2°：能否添加一套「类型约束」，使得运行以前就能检查出诸如类型不统一这样的错误？凡通过检查的程序，在类型上没有纰漏，则可担保其必可正常终止（而不会半途卡住或无尽循环）。没有通过检查的程序，则在类型上有纰漏，或许有可能出错。

remark. 因为λ演算是图灵完备的，所以不要指望有一套万能的类型系统能区分开正常终止的程序与异常/循环的程序。我们能做的，仅仅是下图所示之区分。



同时我们还希望，左半边不能太小，即：不能只有极少数的程序通得过检查。否则，这类型系统就缺乏指导意义了。

约定

自本章起，小写字母 t 默认[redacted]满足 $term(t)$；大写字母 T 默认指代类型（后面会定义）；x, y, z 默认指代 V 中的变元符号；v 默认[redacted]满足 $val(v)$（后面会定义）。这有助于我们省去眼花缭乱的前提。

def. 字符集：

$$\Sigma ::= \{\lambda, ., (,), True, False, if, then, else\} \cup \{:, Bool, \blacktriangleright\} \cup V$$

def 类型.

判断形式 $type(T)$

$$\frac{}{type(Bool)}$$

$$\frac{type(T_1) \quad type(T_2)}{type((T_1 \blacktriangleright T_2))}$$

BN 记号

$$T ::= Bool \mid T \blacktriangleright T$$

意即：基础类型只有Bool，由此衍生出许多函数类型，比如 $Bool \blacktriangleright (Bool \blacktriangleright Bool)$。

def 值.

判断形式 $val(v)$

$$\frac{}{val(\lambda x.t)} \qquad \frac{}{val(True)} \qquad \frac{}{val(False)}$$

BN 记号

$$v ::= \lambda x.t \mid True \mid False$$

「值」的含义即：唯有True, False和函数是被我们认可的、识别的。它们才具备实用价值。程序终态为值，方为「正常终止」。

def 语言 λ_B

判断形式 $term(t)$

$$\frac{}{term(x)} \quad \frac{}{term(True)} \quad \frac{}{term(False)}$$

$$\frac{term(t) \quad type(T)}{term(\lambda x{:}T.t)} \quad \frac{term(t_1) \quad term(t_2)}{term(t_1 t_2)}$$

$$\frac{term(t_1) \quad term(t_2) \quad term(t_3)}{term(if\ t_1\ then\ t_2\ else\ t_3)}$$

BN 记号

$$t ::= x \mid True \mid False \mid \lambda x{:}T.t \mid tt \mid if\ t\ then\ t\ else\ t$$

除了添加 True, False, if-then-else 以外，我们还给函数的参数打上了类型记号。$\lambda x{:}T.t$ 代表我们定义了函数 $x \mapsto t$ 且假定 x 的类型为 T。该记号将帮助我们进行类型分析。

def 单步执行关系.

判断形式 $t \to t'$. 规则如下：

① $$\frac{t_1 \to t_1'}{if\ t_1\ then\ t_2\ else\ t_3 \to if\ t_1'\ then\ t_2\ else\ t_3}$$

② $$\frac{}{if\ True\ then\ t_2\ else\ t_3 \to t_2}$$ ③ $$\frac{}{if\ False\ then\ t_2\ else\ t_3 \to t_3}$$

④ $$\frac{t_1 \to t_1'}{t_1 t_2 \to t_1' t_2}$$ ⑤ $$\frac{t_2 \to t_2'}{v t_2 \to v t_2'}$$ ⑥ $$\frac{}{(\lambda x{:}T.t)v \to t[x/v]}$$

大致说来，就是「先左后右，然后代入」。遇到 if-then-else 时，先评估条件，将其化为 True / False 以后再转移至 then / else。目前为止，一切都与原先相似。

下面讲解类型系统的精髓：类型检查。给定一个依前面定义书写的程序 t，怎么能检查出其类型正确性呢？先看两个例子。

e.g.1

$$\lambda a{:}Bool.\ \lambda b{:}Bool.\ if\ ((\lambda x{:}Bool.\ if\ x\ then\ False\ else\ True)\ a)\ then\ True\ else\ b$$

该程序接收两个值 a 与 b，并去计算 $\neg a \vee b$。类型检查器不理会，也无力理会程序的功能。它只想检查程序的类型是否合法。刚开始，它手中空空如也，只有一些

最基本的公理（如True : Bool）。它读入程序的第一行，便建立了假设{a:Bool, b:Bool}。在第二行的 if 内部，它进一步将假设扩充为{a:Bool, b:Bool, x:Bool}，然后分析出「if $\underset{\text{Bool}\checkmark}{x}$ then $\underset{\text{Bool}}{\text{False}}$ else $\underset{\text{Bool}}{\text{True}}$」的类型为Bool。接下去，又分析出 if $\underset{\text{Bool}\checkmark}{\ldots}$ then $\underset{\text{Bool}}{\text{True}}$ else $\underset{\text{Bool}}{b}$ 的类型为Bool。因此，整个程序的类型是

$\text{Bool} \blacktriangleright (\text{Bool} \blacktriangleright \text{Bool})$，没有纰漏。

e.g.2 $\lambda a:\text{Bool}\blacktriangleright\text{Bool}.\ \lambda b:\text{Bool}.$

$\text{if } b \text{ then } a\,b \text{ else } \lambda x:\text{Bool}.x$

步骤	假设	备注
(1)	$\emptyset$	
(2)	$\{a:\text{Bool}\blacktriangleright\text{Bool}\}$	
(3)	$\{a:\text{Bool}\blacktriangleright\text{Bool},\ b:\text{Bool}\}$	推得 $ab:\text{Bool}$.
(4)	$\{a:\text{Bool}\blacktriangleright\text{Bool},\ b:\text{Bool},\ x:\text{Bool}\}$	推得 $x:\text{Bool}$
(5)	$\{a:\text{Bool}\blacktriangleright\text{Bool},\ b:\text{Bool}\}$	进一步得 $(\lambda x:\text{Bool}.x):\text{Bool}\blacktriangleright\text{Bool}$
(6)		if-then-else中，then与else的类型不一致，故该句无类型。

这进一步造成整个程序无类型。检查器向用户发生类型警报。

从e.g.1,2看出，类型检查简单而言即从空假设 $\emptyset$ 入手，逐步深入分析组分的类型，并推出整个程序的类型。

以下定义严格规范了类型检查的实质。

def 类型推导.

判断形式 $\Gamma \vdash t:T$

（这定义了一个三元关系。含义为「由假设集 Γ 出发，可以推导出 t 在任何情形下均具有类型 T）

① $\dfrac{x:T \in \Gamma}{\Gamma \vdash x:T}$　② $\dfrac{}{\Gamma \vdash \text{True}:\text{Bool}}$　③ $\dfrac{}{\Gamma \vdash \text{False}:\text{Bool}}$

④ $\dfrac{\Gamma \vdash t_1:\text{Bool} \quad \Gamma \vdash t_2:T \quad \Gamma \vdash t_3:T}{\Gamma \vdash (\text{if } t_1 \ \cancel{\text{then}}\ t_2 \text{ else } t_3):T}$

⑤ $\dfrac{\Gamma \vdash t:T_1\blacktriangleright T_2 \quad \Gamma \vdash t':T_1}{\Gamma \vdash (t\,t'):T_2}$

因为可以换元，故这个条件没有多少约束。

⑥ $\dfrac{\Gamma \cup \{x:T_1\} \vdash t:T_2}{\Gamma \vdash (\lambda x:T_1.t):T_1\blacktriangleright T_2}\ (x \notin \text{dom}\,\Gamma)$

def 类型检查

给定程序t，寻找T满足 $\emptyset \vdash t:T$ 的过程，即类型检查。若T存在，则称程序t通过检查；否则，称t未通过检查。

我们逐条解释①—⑥。

① 说明，凡是Γ中已经包含的有关变元类型的假定，都能原样导出。

②③ 说明，在任何情况下 True和False都具有类型Bool.

④ 说明，如果Γ能推得t_1的类型为Bool，而t_2和t_3的类型一致为T，那么Γ亦能推得「if t_1 then t_2 else t_3」无论怎样都具有类型T。

⑤ 说明，在假设Γ下，把类型为$T_1 \blacktriangleright T_2$的函数$t$作用于类型为$T_1$的参数$t'$上，具有类型$T_2$。

⑥ 最有意思，不妨从下往上看。欲知Γ下函数 $\lambda x:T_1.t$ 的类型，先假设 $x:T_1$ 已知，看看此时 t 的类型是什么，然后再串起来构成整体类型。

下面是一些例子：

e.g.

$$\emptyset \vdash \mathit{True}:\mathrm{Bool}$$

$$\emptyset \vdash (\mathit{if}\ \mathit{True}\ \mathit{then}\ \mathit{False}\ \mathit{else}\ \mathit{True}):\mathrm{Bool}$$

$$\emptyset \vdash (\lambda a:\mathrm{Bool}\blacktriangleright\mathrm{Bool}.\ a\ \mathit{True}):(\mathrm{Bool}\blacktriangleright\mathrm{Bool})\blacktriangleright\mathrm{Bool}$$

$$\{x:\mathrm{Bool}\blacktriangleright\mathrm{Bool}\} \vdash (\lambda y:\mathrm{Bool}.\ xy):\mathrm{Bool}\blacktriangleright\mathrm{Bool}$$

请你给出它们的生成过程。

Lemma 1

若 $\Gamma \vdash t:T$，则其生成过程唯一。

proof. 显然. ■

Corollary 2 (Inversion)

(1) 若 $\Gamma \vdash x:T$ 则 $x:T \in \Gamma$

(2) 若 $\Gamma \vdash True:T$ 则 $T = Bool$ (False同理)

(3) 若 $\Gamma \vdash (\text{if } t_1 \text{ then } t_2 \text{ else } t_3)$，则

$\Gamma \vdash t_1 : Bool$ 且

$\exists T$, $\Gamma \vdash t_2 : T$ 及 $\Gamma \vdash t_3 : T$

(4) 若 $\Gamma \vdash (t\,t') : T_2$ 则 $\exists T_1$ 使

$\Gamma \vdash t : T_1 \to T_2$，且 $\Gamma \vdash t' : T_1$.

(5) 若 $\Gamma \vdash (\lambda x:T_1.t) : T_1 \to T_2$ $(x \notin dom\,\Gamma)$

则 $\Gamma \cup \{x:T_1\} \vdash t : T_2$

remark. 该推论为我们提供了类型检查算法。

Lemma 3 (Uniqueness)

任意取定Γ和t. 如果存在T使得$\Gamma \vdash t:T$ 那么T是唯一的。(要求Γ中不能有冲突)

换言之，在一致的假设集下，同一个程序至多只能有一种类型。

proof. 由归纳证明立即得到. ■

Lemma 4 (Weakening)

若 $\Gamma \vdash t:T$，且 $x \notin dom\,\Gamma$，则

$\Gamma \cup \{x:T'\} \vdash t:T$

换言之，原来就能推出的类型不会因假设的增加而无效。

proof. 由归纳证明立得. ■

Lemma 5 (Weak Substitution)

设Γ_0中不含y与z。若 $\Gamma_0 \cup \{y:T^*\} \vdash t:T'$

那么 $\Gamma_0 \cup \{z:T^*\} \vdash t[y/z] : T'$。换言之，

「同构」的假设集与程序在类型上无差别。

y和z的地位完全等同。

proof. 对$\vdash$的结构作归纳。

CASE ①. 我们有 $\Gamma_0 \cup \{y:T^*\} \vdash x:T$，希望证明 $\Gamma_0 \cup \{z:T^*\} \vdash x[y/z] : T$.

(a) 若 $x=y$，则 $\Gamma_0 \cup \{y:T^*\} \vdash x:T^*$，但据 Lemma 3，必然有 $T=T^*$. 因此，$\Gamma_0 \cup \{z:T^*\} \vdash z:T$ 即 $x[y/z]:T$

(b) 若 $x \neq y$，则由Corollary 2知 $x:T \in \Gamma_0$，因而

$\Gamma_0 \vdash x:T$，再由 Lemma 4 知 $\Gamma_0 \cup \{z:T^*\} \vdash x:T$

即 $x[y/z]:T$.

CASE②③ 显然

CASE④ $\Gamma_0 \cup \{y:T^*\} \vdash (\text{if } t_1 \text{ then } t_2 \text{ else } t_3):T$

$$\begin{cases} \Gamma_0 \cup \{y:T^*\} \vdash t_1 : Bool \\ \Gamma_0 \cup \{y:T^*\} \vdash t_2 : T \\ \Gamma_0 \cup \{y:T^*\} \vdash t_3 : T \end{cases}$$

$$\xrightarrow{I.H.} \begin{cases} \Gamma_0 \cup \{z:T^*\} \vdash t_1[y/z] : Bool \\ \Gamma_0 \cup \{z:T^*\} \vdash t_2[y/z] : T \\ \Gamma_0 \cup \{z:T^*\} \vdash t_3[y/z] : T \end{cases}$$

$$\xrightarrow{④} \Gamma_0 \cup \{z:T^*\} \vdash (\text{if } t_1 \text{ then } t_2 \text{ else } t_3)[y/z] : T$$

CASE⑤ 与④类似

CASE⑥ $\Gamma_0 \cup \{y:T^*\} \vdash (\lambda x:T_1.t): T_1 \to T_2$

$\Gamma_0 \cup \{y:T^*, x:T_1\} \vdash t:T_2$

即 $\Gamma_0 \cup \{x:T_1, y:T^*\} \vdash t:T_2$

$\xrightarrow{I.H.} \Gamma_0 \cup \{x:T_1, z:T^*\} \vdash t[y/z]:T_2$

$\xrightarrow{⑥} \Gamma_0 \cup \{z:T^*\} \vdash (\lambda x:T_1.t[y/z]):T_2$

$\xrightarrow{x \notin dom\, \Gamma_0 \cup \{y:T^*\}} \Gamma_0 \cup \{z:T^*\} \vdash (\lambda x:T_1.t)[y/z]:T_2$ ■

Lemma 6 (Strong Substitution)

设 $\Gamma_0 \cup \{x:T^*\} \vdash t:T$ 且

$\Gamma_0 \vdash v:T^*$

那么 $\Gamma_0 \vdash t[x/v]:T$.

意思是：如果用与 x 同类型的 v 取代之，t 的类型不变。

proof. 对 $\vdash$ 的结构作归纳，结合 Lemma 5 即得。留作习题。■

Theorem 7 (Preservation)

单步运行不改变类型。即：若 $t \to t'$ 且 $\Gamma \vdash t:T$，那么 $\Gamma \vdash t':T$。

proof. 对 $\to$ 的结构作归纳。

CASE① 已知 $t_1 \to t_1'$，$\Gamma \vdash (\text{if } t_1 \text{ then } t_2 \text{ else } t_3):T$

欲证 $\Gamma \vdash (\text{if } t_1' \text{ then } t_2 \text{ else } t_3):T$

这由 Corollary 2 和 I.H 直接推得。

CASE②③ 显然。

CASE ④⑤ 与①类似。

CASE ⑥

$$\Gamma \vdash (\lambda x:T^*.\, t)\, v : T_2$$

$$\xRightarrow[2]{\text{Corollary}} \exists T_1 \text{ 使 } \begin{cases} \Gamma \vdash (\lambda x:T^*.\, t) : T_1 \blacktriangleright T_2 \\ \Gamma \vdash v : T_1 \end{cases}$$

$$\xRightarrow[2]{\text{Corollary}} \begin{cases} \Gamma \cup \{x:T^*\} \vdash t : T_2 \\ \Gamma \vdash (\lambda x:T^*.\, t) : T^* \blacktriangleright T_2 \end{cases}$$

故由Lemma3知 $T_1 = T^*$。总结起来，我们有

$$\begin{cases} \Gamma \vdash v : T^* \\ \Gamma \cup \{x:T^*\} \vdash t : T_2 \end{cases}$$

由Lemma6便有 $\Gamma \vdash t[x/v] : T_2$ ■

经过艰苦跋涉，我们终于得到第一个重大定理。它表明：只要一个程序t通过了类型检查，那么任它怎么运行，它始终保有原始类型。这一性质称作「类型在运行中的不变性」。

与这一定理相搭配的，是下面一个定理。

Theorem 8 (Progress) 即 $\emptyset \vdash t:T$

若t通过了检查，则要么 val(t)，要么t能继续单步运行。

proof. 我们对 ⊢ 的结构作归纳

CASE ① 不存在.

CASE ②③ True和False本就是值.

CASE ④

$\emptyset \vdash t_1 : Bool$ 已知 有两种情况：

(a) val(t_1). 那么t_1要么是True/False，要么是(λ...)。可是后者不可能，因为Corollary 2 告诉我们(λ...)的类型必形如...▸...而不可能为Bool。这样一来，t_1只能为True/False，故 if t_1 then t_2 else t_3 可以单步运行。

(b) 否则，据归纳假设，t_1可单步运行，故 if t_1 then t_2 else t_3 可单步运行。

CASE⑤ 类似

CASE⑥ $(\lambda x{:}T.\ t)$ 本就是值

结合Theorem 7,8，便知：凡是通过检查的程序t，要么已经是值，要么能继续运行，且运行后仍能通过检查，如此等等。亦即：通过类型检查的程序，若能终止，总能够终止于某个值，而决不会半途卡住。而适当修改Theorem 8即可证明：凡通过检查的程序总能终止。因此，我们有以下结论：

Theorem 9

若t通过检查（即$\exists T$使$\emptyset \vdash t : T$），那么t总能终止于一个值，而且类型始终为T。

这时，类型系统的威力就显示出来了。如果程序员能设计出通过类型检查的程序，且不论功能是否正确，至少能确保无低级错误，并可保证终止。

Theorem 7,8 作为最为核心的定理，必须被任何实用的类型系统所满足。

remark. 直观上，「通过检查者必能终止」不难理解 —— 一个程序t能通过检查，必因其各部分具备有限长的类型 ···▶···▶·······。每在单步运行时「调用」一个λ项，▶就「消掉」一个，因此不可能存在无穷多的调用。

你可以试着证明一下这个命题：$\lambda x.xx$ 是无法通过类型检查的。由这个命题你就能理解为何死循环、甚至递归，都不可能具备类型。