

第 6 章 类型安全

绝大部分编程语言是安全的（类型安全，或强类型的）。非正式地说，这意味着执行过程中不会出现某些不匹配。举例说明：规定 E 类型安全，绝对不会出现数字加上字符串，或连接两个数字的情况。这两个操作都不是有意义的。

类型安全一般表述静态和动态的一致性。静态可以看成预计一个表达式的值具有某个形式，以使动态下的这个表达式是良定义的。因此，分析不能“卡在”在一个无可能转换的状态下，在实现上对应执行时“非法指令”错误的缺失。通过显示每一步保留可类型化的转换，或显示可类型化的规定是良定义的，安全性可以得到证明。因此，分析绝不会“消失”，也绝不会遇到非法指令。

对于语言 E 的类型安全规定如下：

定理 6.1 (类型安全)

1. 如果 $e : \tau$ 且 $e \rightarrow e'$, 那么 $e' : \tau$
2. 如果 $e : t$, 那么 e 是值, 或存在 e' 满足 $e \rightarrow e'$

第一条称为 *保留性*，指分析的每一步能保持类型的归类；第二条称作 *进展性*，保证表达式要么已经是值，要么可以进一步分析。安全性是以上两者的结合。

我们认为表达式 e 是 *中止的*，如果 e 不是值且没有 e' 使 $e \rightarrow e'$ 成立。根据安全性定理，中止的状态一定是弱类型的。换种说法，良类型的状态不会中止。

6.1 保留性

关于在第 4 章和第 5 章中定义的 E 的保留性理论已通过在转换系统上使用归纳法（规则 5.4）得到证明。

定理 6.2 (保留性) 如果 $e : \tau$ 且 $e \rightarrow e'$, 那么 $e' : \tau$

证明：给出两种情况的证明，其余留给读者。考虑规则 5.4b

$$\frac{e1 \rightarrow e1'}{\text{plus}(e1; e2) \rightarrow \text{plus}(e1'; e2)}$$

假设 $\text{plus}(e1; e2) : \tau$. 通过类型转化，我们有 $\tau = \text{num}$, $e1 : \text{num}$, 和 $e2 : \text{num}$ 。

归纳可得 $e1' : \text{num}$, 且由此有 $\text{plus}(e1'; e2) : \text{num}$. 连接操作的处理与此相似。

现在考虑规则 5.4h

$$\overline{\text{let } (e1; x.e2) \rightarrow [e1/x] e2}$$

假设 $\text{let } (e1; x.e2) : \tau 2$. 由引理 4.2, 设 $e1 : \tau 1$, 故有 $x : \tau 1 \vdash e2 : \tau 2$.
通过替换引理 4.4 中的 $[e1/x] e2 : \tau 2$, 可以得证。

很容易得到原语操作全是类型保留的。比如, $a : \text{nat}$ 且 $b : \text{nat}$, $a+b=c$, 那么 $c : \text{nat}$

保留性的证明由转化形式判断上的归纳构成, 因为它的论证在于对一个表达式所有可能转化形式的检测。有时候我们会尝试通过对 e 的结构归纳, 或类型归纳来得到证明, 但经验告诉我们, 这往往会导致得不到想要的结论, 或者根本不能付诸操作。

6.2 进展性

进展性理论持有的观点是良类型程序不会“中止”。

证明的关键是以下引理, 它们描述了每种类型的值。

引理 6.3 (规范形式) 如果 e 是值且 $e : \tau$, 那么有

1. 如果 $\tau = \text{num}$, 那么对某个数字 n , $e = \text{num}[n]$.
2. 如果 $\tau = \text{str}$, 那么对某个字符串 s , $e = \text{str}[s]$.

证明: 在规则 (4.1) 和 (5.3) 上归纳。

进展性通过归纳规则 (4.1) 定义语言的静态得证。

定理 6.4 (进展性) 如果 $e : \tau$, 那么 e 是值, 或存在 e' 满足 $e \rightarrow e'$.

证明: 证明通过类型推导归纳进行, 只考虑一种情况, 对于规则 (4.1d),

$$\frac{e1 : n \text{ um} \quad e2 : n \text{ um}}{\text{plus}(e1; e2) : n \text{ um}}$$

上下文为空, 因为我们不考虑闭项。

归纳可得 $e1$ 是值, 或者存在 $e1'$ 满足 $e1 \rightarrow e1'$. 在后一种情况下, 根据要求, 它遵循 $\text{plus}(e1; e2) \rightarrow \text{plus}(e1'; e2)$. 在前一种情况下, 同样归纳得到 $e2$ 是值, 或者存在 $e2'$ 满足 $e2 \rightarrow e2'$. 后一情况, 有 $\text{plus}(e1; e2) \rightarrow \text{plus}(e1; e2')$; 对于前一情况, 通过引理 6.3, 有 $e1 = \text{num}[n1]$ 和 $e2 = \text{num}[n2]$, 因此有:

$$\text{plus}(\text{num}[n1]; \text{num}[n2]) \rightarrow \text{num}[n1 + n2].$$

因为表达式的类型规则是语法制导的, 所以进展性定理可以同样由对 e 的结构归纳来证明, 每一步都要使用反演定理。但这种方法在非语法制导的类型规则下不适用, 也就是说, 一个给定的表达式形式有多个规则。这样的规则没有任何困难, 只要证明是在类型规则上归纳, 而不是在表达式的结构上。

总之, 保留性与进展性的结合构成安全的证明。进展定理确保了良类型的表达式不会在弱定义的状态下“卡住”, 而保留定理则确保了步骤执行后, 结果仍是良类型的(类型保持相同)。因此, 这两个部分共同工作, 以确保静态和动态是一致的, 并且在评估良类型的表达

式时不会遇到弱定义的状态。

6.3 运行时错误

假设我们想用一个除 0 无意义的商运算扩展 \mathcal{E} ，商的自然类型规则由以下规则给出：

$$\frac{e1 : n \text{ um } e2 : n \text{ um}}{\text{div}(e1; e2) : n \text{ um}}$$

但表达式 $\text{div}(\text{num}[3]; \text{num}[0])$ 是良类型的，却中止了！有两个选项去纠正这个错误

1. 增强类型系统，使任何良类型程序都不能除以 0。
2. 添加动态检查，将以零为除数的除法作为评估结果的一个错误信号。

两个选择都是原则上可行的，但最常见的方法是第二种。第一条要求类型检查器证明表达式为非零，然后才允许在一个商的分母中使用它。要做到这点必须将很多程序划为弱定义的。我们不能静态地预测表达式在计算时是否为非零，因此第二种方法在实际中最常用。

总体思路是区分检测错误与非检测错误。一个非检测错误是被类型系统排除的错误。不采取运行时检测以确保不会发生这种错误，因为类型系统排除了发生这种错误的可能性。例如，动态不需要检查，当执行加法时，它的两个参数实际上是数字，而不是字符串，因为类型系统的保证。另一方面，商的动态必须检查零除数，因为类型系统没有排除这种可能性。

对检测错误进行建模的一种方法是给出判断 $e \text{ err}$ 的归纳定义，指出表达式 e 可能检测出的运行时错误，如除以 0。以下是一些具有代表性的规则，将在此判断的完全归纳定义中出现：

$$\frac{e1 \text{ val}}{\text{div}(e1; n \text{ um}[0]) \text{ err}} \quad (6.1a)$$

$$\frac{e1 \text{ err}}{\text{div}(e1; e2) \text{ err}} \quad (6.1b)$$

$$\frac{e1 \text{ val } e2 \text{ err}}{\text{div}(e1; e2) \text{ err}} \quad (6.1c)$$

规则(6.1a)表示除零错误。其他规则向上传播这个错误：如果一个子表达式分析是检测错误，那么整个表达式同样。

一旦有了错误判断，我们还可以考虑一个表达式错误。它强制引入一个错误，使用以下静态和动态语义：

$$\frac{}{\Gamma \not\vdash \text{error} : \tau} \quad (6.2a)$$

$$\frac{}{\text{error} \text{ err}} \quad (6.2b)$$

保留定理不受检测错误的影响。但进展性的声明与证明需要修改。

定理 6.5 (考虑错误的进展性): 如果 $e : \tau$ ，那么 $e \text{ err}$ ，或者 e 是值，或者存在 e' 满足

$e \rightarrow e'$

证明：通过类型归纳证明，与前面给出的证明类似，只是在每一点的证明上需要考虑三个情况。

6.4 小结

类型安全的概念最初是由 [Milner\(1978\)](#) 提出的，他提出了“良定义的程序不会出错”。Milner 依靠一个明确的“出错”的概念来表达类型错误的概念，而 [Wright 和 Felleisen\(1994\)](#) 观察到我们可以相反地表明，在一个良类型程序中不可能弱定义的状态，这就产生了“良类型化程序不会被卡住”的说法。然而，他们的提论依赖于一项表明没有一个中止状态是良类型的分析。进展理论，依赖于 [Martin-Löf\(1980\)](#) 风格规范形式的描述，消除了这一分析。

习题：

- 6.1. 完成定理 6.2 的详细证明
- 6.2 完成定理 6.4 的详细证明
- 6.3. 给出定理 6.5 的几个例子，来说明在类型安全证明中是如何处理检测错误的。