控制流

常见的程序语言中除了最基本的分支、循环结构，往往还包含 break 和 continue 这样的语句。这些语句会对既定的执行顺序造成比较大的影响。

本文探讨如何将这种比较高级的控制语句加入到我们此前构建的几种程序语义中。

指称语义（进阶）

为了表现出一段程序指令终止方式的不同，将程序指称定义为 (st_1, e, st_2) 的三元组。其中

$$e ::= \mathrm{Normal} | \mathrm{Break} | \mathrm{Continue}$$

分别表示正常终止、因 break 终止和因 continue 终止三种终止方式。

对于原子语句，终止方式都是正常终止。对于 break 和 continue，前后程序状态不变，但是终止方式分别为因 break 终止和因 continue 终止。

对于分支语句，仍使用测试关系和关系的并，只是需要补充说明条件判定是正常终止。

对于串行执行 $c_1; c_2$，有两种可能：一是 c_1 正常终止，然后 c_2 以某种方式终止；二是 c_1 非正常终止。

对于循环体，由于引入了 break 和 continue，因此需要着重考虑。

（FILL IN HERE）

为了表示因 break 终止和因 continue 中止只在循环体内部发生，循环体的终止应当表示为正常终止。

霍尔逻辑（进阶）

考虑引入了 break 和 continue 之后怎么设计推理规则。

一种方法是：扩充霍尔三元组，扩充成为具有以下形式

$$\{P\}\, c\, \{Q_n\}\, \{Q_b\}\, \{Q_c\}$$

这表示如果 c 从某个符合断言 P 的状态开始执行，且

- c 正常终止时满足后条件 Q_n。
- c 因 break 终止时满足后条件 Q_b。
- c 因 continue 终止时满足后条件 Q_c。

对应的，需要扩充已有的推理规则。

条件修改

方便起见，省略了 $\forall$ 引入的变量。

$$\frac{P \vdash P', Q'_n \vdash Q_n, Q'_b \vdash Q_b, Q'_c \vdash Q_c, \{P'\}\, c\, \{Q'_n\}\, \{Q'_b\}\, \{Q'_c\}}{\{P\}\, c\, \{Q_n\}\, \{Q_b\}\, \{Q_c\}}$$

串行、分支

两部分的 Q_n, Q_b, Q_c 和后条件的都相同。

原子语句

由于不存在 break 和 continue 的可能，因此让对应后条件最强，即取 $Q_b = Q_c = \bot$。

这种定义的方式和当初处理死循环时比较像。正如我们不希望出现死循环一样，这种定义方式反映了之前指称语义时就有的想法：不允许整个程序非正常终止。

break 和 continue

这两个和空语句类似，但是终止方式不同。因而定义如下。

$$\forall P, \frac{}{\{P\}\,\mathrm{break}\,\{\bot\}\,\{P\}\,\{\bot\}}$$

$$\forall P, \frac{}{\{P\}\,\mathrm{continue}\,\{\bot\}\,\{\bot\}\,\{P\}}$$

循环

循环的推理规则是改动最大的（或者说和别的相比差别最大的）。

$$\forall b, P, I, c, \frac{\{I \wedge b\}\,c\,\{I\}\,\{P\}\,\{I\}}{\{I\}\,\mathrm{While}\ b\ \mathrm{Do}\ c\ \mathrm{EndWhile}\,\{P \vee (I \wedge \neg b)\}\,\{\bot\}\,\{\bot\}}$$

其中 Q_n 和 Q_c 相同，都是循环不变量 I，因为 continue 了之后循环还应执行。

> VST 在处理循环的时候会（FILL IN HERE）

小步语义（进阶）

大部分此前定义的小步关系可以被沿用。但由于引入了 break 和 continue，直接将循环按照分支+循环的组合来计算的做法不太行了。我们需考虑怎样设计抽象机器的运算规则，以支持新的两种指令和更复杂的循环。

有两种方法：一种是直接对循环语句的计算做一些调整，另一种在此基础上还用了**控制栈（Control Stack）**。

修改循环

将循环表示为形式

$$c_1; \mathrm{While}\ b\ \mathrm{Do}\ c\ \mathrm{EndWhile}$$

c_1 表示这次迭代中循环体待执行的语句。那么未被执行的循环就表示为 $\mathrm{skip; While}\ b\ \mathrm{Do}\ c\ \mathrm{EndWhile}$。使用了这种改良形式就可以很好的描述 break 和 continue 的行为。

如果要开启一个循环，就使用下面的小步关系。

$$(\mathrm{skip; While}\ b\ \mathrm{Do}\ c\ \mathrm{EndWhile}, st) \to (\mathrm{If}\ b\ \mathrm{Then}\ c\ \mathrm{Else\ break\ EndIf; While}\ b\ \mathrm{Do}\ c\ \mathrm{EndWhile}, st)$$

要推进循环体执行，可以仿照串行执行定义循环体执行的小步语义。

对于 break，我们可以通过在语法树上递归的方式判定一串语句是否以 break 为开头，对于 continue 类似。那么，如果 c_1 以 break 开头，则以下小步关系成立。

$$(c_1; \mathrm{While}\ b\ \mathrm{Do}\ c\ \mathrm{EndWhile}, st) \to (\mathrm{skip}, st)$$

也就是跳出了循环。

如果 c_1 以 continue 开头，则以下小步关系成立。

$$(c_1; \mathrm{While}\ b\ \mathrm{Do}\ c\ \mathrm{EndWhile}, st) \to (\mathrm{If}\ b\ \mathrm{Then}\ c\ \mathrm{Else\ break\ EndIf; While}\ b\ \mathrm{Do}\ c\ \mathrm{EndWhile}, st)$$

也就是跳出了这次迭代，进入新一轮迭代。

控制栈

控制栈是一个栈，里面的元素为（FILL IN HERE）

小结

引入了这两种控制语句之后，就可以利用相似的想法引入函数调用的语句。这样语言就更加丰富、乃至接近现代语言（如 C）了。