

# 霍尔逻辑

## 1 霍尔三元组

对于任意程序状态  $s_1$  与  $s_2$ ，如果  $s_1$  满足性质  $P$  并且  $(s_1, s_2) \in \llbracket c \rrbracket$ ，那么  $s_2$  满足性质  $Q$ 。这一性质写作  $\{P\}c\{Q\}$ ，称为霍尔三元组， $P$  称为前条件， $Q$  称为后条件。

```
{ x == 0 }
y = 0;
while (y < 6) do {
  x = x + y;
  y = y + 1
}
{ x == 1 + 2 + 3 + 4 + 5 }
```

## 2 顺序执行规则、空语句规则与条件分支语句规则

下面是顺序执行规则：

如果  $\{P\}c_1\{Q\}$  并且  $\{Q\}c_2\{R\}$ ，那么  $\{P\}c_1; c_2\{R\}$ 。

下面是一个顺序执行规则的例子。我们想要证明，下面程序确实交换了变量  $x$  与  $y$  的值。

```
{ x == m && y == n }
t = x;
x = y;
y = t
{ x == n && y == m }
```

首先，下面的霍尔三元组显然为真。

```
{ x == m && y == n }
t = x
{ t == m && y == n }
```

```
{ t == m && y == n }
x = y
{ t == m && x == n }
```

```
{ t == m && x == n }
y = t
{ x == n && y == m }
```

由其中的第二第三条可以得到：

这里不是已经用了顺序执行性质了吗？

```

{ t == m && y == n }
x = y;
y = t
{ x == n && y == m }

```

最后再用顺序执行规则和第一条霍尔三元组可知：

```

{ x == m && y == n }
t = x;
x = y;
y = t
{ x == n && y == m }

```

下面是空语句规则：

$$\{P\} \text{ skip } \{P\}$$

下面是条件分支语句规则：

如果  $\{P \&\& e\} c_1 \{Q\}$  并且  $\{P \&\& !e\} c_2 \{Q\}$ ，那么

$$\{P\} \text{ if } (e) \text{ then } \{ c_1 \} \text{ else } \{ c_2 \} \{Q\}$$

### 3 While 语句规则与循环不变量

While 语句规则

如果  $\{ P \&\& e \} c \{ P \}$ ，那么

$$\{ P \} \text{ while } (e) \text{ do } \{ c \} \{ P \&\& !e \}$$

其中，我们一般会称  $P$  为循环不变量。

例如，要证明下面霍尔三元组

```

{ x == 0 }
while (x < 10) do
{
    x = x + 1
}
{ x == 10 }

```

就只需证明：

```

{ x <= 10 }
while (x < 10) do
{
    x = x + 1
}
{ x <= 10 && ! (x < 10) }

```

这又可以被规约为：

```

{ x <= 10 && x < 10 }
x = x + 1
{ x <= 10 }

```

习题 1. 如何证明：

```

{ x >= 0 }
while (0 < x) do {
    x = x - 1
}
{ x == 0 }

```

下面通过一个例子分析如何找循环不变量。

假如  $m$  与  $n$  是给定正整数，如何证明：

```

{ x == m && y == 0 }
while (! (x < n)) do {
    x = x - n;
    y = y + 1
}
{ n * y + x == m && 0 <= x < n }

```

考虑  $n$  的值为 3、 $m$  的值为 10 的具体情况。

```

{ x == 10 && y == 0 }
x = x - 3;
y = y + 1
{ x == 7 && y == 1 }

```

```

{ x == 7 && y == 1 }
x = x - 3;
y = y + 1
{ x == 4 && y == 2 }

```

```

{ x == 4 && y == 2 }
x = x - 3;
y = y + 1
{ x == 1 && y == 3 }

```

循环不变量  $P$  应当使得下面断言能推出  $P$ ：

```

x == 10 && y == 0 ||
x == 7 && y == 1 ||
x == 4 && y == 2 ||
x == 1 && y == 3

```

否则以下两个条件总有一个不成立：

- 循环的前条件能推出  $P$ ；
- $\{ P \ \&\& \text{循环条件} \}$  循环体  $\{ P \}$ 。

另一方面，循环不变量  $P$  也不能太弱，否则

$P \ \&\& \text{!循环条件}$

不足以推出循环整体的后条件。

结合这两点考虑，我们在刚才的例子中可以取循环不变量  $P$  为：

```

x == 10 && y == 0 ||
x == 7 && y == 1 ||
x == 4 && y == 2 ||
x == 1 && y == 3

```

不难检查，它满足下面三条性质：

- 循环前条件  $x == 10 \ \&\& \ y == 0$  能推出  $P$
- 循环体能保持循环不变量

```

{ P && ! (x < 3) }
x = x - 3; y = y + 1
{ P }

```

- $P \ \&\& \ ! \ (x < 3)$  能推出循环后条件  $3 * y + x == 10 \ \&\& \ 0 \leq x < 3$ 。

当然，循环不变量也可以是弱一些的断言，例如： $3 * y + x == 10 \ \&\& \ 0 \leq x$

构造循环不变量的一般思路如下：

- 循环的前条件要能推出循环不变量
- 循环不变量不能太强，至少要保证程序运行中每次循环体执行结束后的程序状态应当满足循环不变量，否则循环体不满足霍尔三元组：

$$\{ P \ \&\& \ e \} c \{ P \};$$

- 循环不变量不能太弱，否则  $P \ \&\& \ !e$  不足以推出循环的后条件；
- 通常情况下，可以考虑选择一个循环不变量，使得满足这个循环不变量的程序状态恰好是所有循环体执行结束之后的程序状态。

**习题 2.** 假如  $m$  与  $n$  是给定正整数，如何证明：

```

{ x == m }
while ( ! (x < n) ) do {
  x = x - n
}
{ exists y'. n * y' + x = m && 0 <= x < n }

```

**习题 3.** 假如  $m$  与  $n$  是给定正整数，如何证明：

```

{ x == m && y == n }
while ( ! ( ! (x < 0) && ! (0 < x) ) ) do {
  y = y - 1;
  x = x - 1
}
{ y == n - m }

```

**习题 4.** 假如  $m$  是给定正整数，如何证明：

```

{ x == m && i == res == 0 }           res <= i * x && i <= m && x = m
while ( i < x ) do {
  res = res + x;
  i = i + 1
}
{ res == m * m }

```

## 4 变量赋值语句规则（正向）与最强后条件

如何严格定义最佳后条件？

- $\{P\}c\{Q\}$  成立；
- 如果  $\{P\}c\{Q'\}$  成立，那么  $Q$  能推出  $Q'$ ；

亦称为最强后条件：strongest postcondition。

习题 5. 对于前条件  $0 \leq y$  与程序  $x = y$ ，它们的最强后条件是什么？

习题 6. 对于前条件  $n * y + x == m \ \&\& \ 0 \leq x \ \&\& \ n \leq x$  与程序  $x = x - n$ ，它们的最强后条件是什么？

习题 7. 对于前条件  $n * y + x + n == m \ \&\& \ 0 \leq x$  与程序  $y = y + 1$ ，它们的最强后条件是什么？

习题 8. 对于前条件  $x == n \ \&\& \ y == m$  与程序  $x = x + y$ ，它们的最强后条件是什么？

变量赋值规则（正向）：

$$\{P\} x = e \{ \exists x'. e[x \mapsto x'] = x \ \&\& \ P[x \mapsto x'] \}$$

这里，我们用  $P[x \mapsto x']$  表示将断言  $P$  中出现的每一个  $x$  都替换成为  $x'$ 。例如：

- $(x \geq 0)[x \mapsto x']$  表示  $x' \geq 0$ ，
- $(\exists k. x = a_k)[x \mapsto x']$  表示  $\exists k. x' = a_k$ ，
- $(x + y)[x \mapsto x']$  表示  $x' + y$ 。

**x'是老的值**

请注意，此处  $(x + y)$  是一个 SimpleWhile 语言的表达式（变量加变量），将其中的  $x$  替换为  $x'$  之后得到的  $(x' + y)$  还是一个程序表达式（常量加变量），它的求值结果是  $x' + y$ ，这里的加号是数学上的加号。

- $y[x \mapsto x']$  表示  $y$ 。
- $(x + 1)[x \mapsto x']$  表示  $x' + 1$ 。

例如

```
{ x == m && y == n }  
x = x + y  
{ exists x'. x' + y == x && x' == m && y == n }
```

```
{ x == m && y == n }  
temp = x  
{ exists temp'. x == temp && x == m && y == n }
```

**x == y == n && temp == m**

习题 9. 对于前条件  $x == temp == m \ \&\& \ y == n$  与程序  $x = y$ ，它们的最强后条件是什么？

## 5 变量赋值语句规则（反向）与最弱前条件

后条件  $Q$  与程序  $c$  的最弱前条件  $P$  是指满足下面条件的断言  $P$ 。

- $\{P\}c\{Q\}$  成立；
- 如果  $\{P'\}c\{Q\}$  成立，那么  $P'$  能推出  $P$ ；

例如，后条件  $temp == m \ \&\& \ y == n$  与程序  $temp = x$  的一个最弱前条件是：  $x == m \ \&\& \ y == n$ 。

下面我们写出变量赋值规则（反向）：

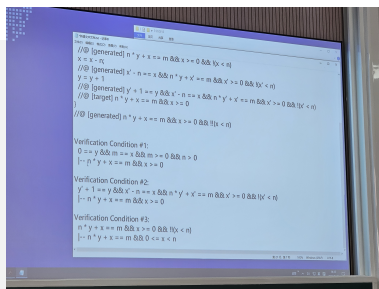
那么  $\{ P[x \mapsto e] \} x = e \{ P \}$

## 6 可靠性证明

## 7 符号执行与验证条件生成

符号执行的例子：

```
//@ require true
//@ ensure x == 10
x = 0;
//@ [generated] 0 == x && true
//@ inv x <= 10    循环不变量
while (x < 10) do
{
    //@ [generated] x <= 10 && x < 10
    x = x + 1    省去了exists
    //@ [generated] x' + 1 == x && x' <= 10 && x' < 10    注意这里的值也要跟着变为x'
    //@ [target] x <= 10
}
//@ [generated] x <= 10 && !(x < 10)    这一条还要推出ensure (整个程序的后条件)
```



生成的验证条件

这个过程称为符号执行

```
0 == x && true |-- x <= 10    从一个断言 (命题) 得到另一个命题，像符号的执行
```

```
x' + 1 == x && x' <= 10 && x' < 10 |-- x <= 10
```

```
x <= 10 && !(x < 10) |-- x == 10
```

反向符号执行

```
//@ require true
//@ ensure x == 10
//@ [generated] 0 <= 10    反向符号执行顺序是上到下还是下到上？
```

```
x = 0;
//@ inv x <= 10
while (x < 10) do
{
    //@ [target] x <= 10 && x < 10
    //@ [generated] x + 1 <= 10    赋值语句的最弱前条件
    x = x + 1
    //@ [generated] x <= 10    这条为啥是generated? 从下面那个target进入循环，
                                自动generated? 由循环不变量生成
}
//@ [target] x <= 10 && !(x < 10)

simplewhile上是数值语言，不用内存
```

生成的验证条件 都是Target 推出 generated

```
true |-- 0 <= 10
```

```
x <= 10 && x < 10 |-- x + 1 <= 10
```

```
x <= 10 && !(x < 10) |-- x == 10
```