

《软件分析与验证》

IMP 程序设计语言及其语义



贺飞

清华大学软件学院

2024 年 4 月 6 日

一阶理论的定义：

- 签名 Σ ，公理集 \mathcal{A}

一些常见的一阶理论：

- $\mathcal{T}_E, \mathcal{T}_{PA}, \mathcal{T}_{\mathbb{N}}, \mathcal{T}_{\mathbb{Z}}$

我们已经学习了命题逻辑、一阶逻辑和一阶理论；下面学习如何利用逻辑精确地定义和分析程序。

对于任何一种程序设计语言，我们关心的要素包括：语法、语义、证明系统。

IMP：一个简单的示例语言

- IMP 语言中包含了命令式程序设计语言最重要、最核心的语言要素。
- 本节课学习 IMP 语言最基本的语法和语义。后面随着学习的深入，还将在对 IMP 语言进行扩展，加入更多的语言要素。
- IMP 的证明系统将在下节课介绍。

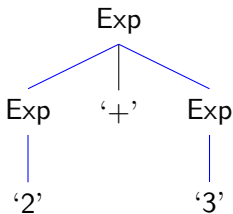
1. 语法

2. 语义

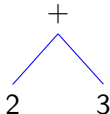
语法

本课程不讨论程序的具体语法 (concrete syntax)，主要关注程序的抽象语法 (abstract syntax)。

- 具体语法：从字符到程序的构成规则
- 抽象语法：只关注有语义的单词，是具体语法的抽象



(a) 编译树 (Parse Tree)



(b) 抽象语法树
(Abstract Syntax Tree)

对程序的具体语法感兴趣的同学，请参阅编译原理相关书籍。

语法范畴:

- 整数集 \mathbb{Z} , 用 a, b, c 表示整数。
- 变元集 Var , 用 x, y, z 表示变元。
- 算术表达式集 $AExp$, 用 e 表示算术表达式。
- 布尔表达式集 $BExp$, 用 p, q 表示布尔表达式。
- 语句集 $Stmt$, 用 st 表示语句。

简单起见, 这里主要讨论整数, 假定所有变量都是整型。

定义

IMP 的抽象语法递归定义如下：

$$\begin{aligned} e \in AExp &::= c \in \mathbb{Z} \mid x \in Var \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \\ p \in BExp &::= \mathbf{true} \mid \mathbf{false} \mid e_1 = e_2 \mid e_1 \leq e_2 \mid \neg p \mid p_1 \wedge p_2 \\ st \in Stmt &::= \mathbf{skip} \mid x := e \mid st_1; st_2 \\ &\quad \mid \mathbf{if} (p) \{st_1\} \mathbf{else} \{st_2\} \\ &\quad \mid \mathbf{while} (p) \{st\} \end{aligned}$$

IMP 包含命令式程序设计语言的核心特征：

- 赋值、分支、循环、顺序等
- 未来将扩展更多的数据结构（如数组）和更复杂的程序构造（如过程）

一阶逻辑	程序语言
项	算术表达式
原子公式	等式、关系式
公式	布尔表达式

语义

程序语义 (program semantics) 研究程序设计语言的含义，是以数学为工具，精确地定义和解释**程序所有可能的计算行为**的研究领域。

程序语义的描述方法有很多种，典型的有¹：

- **指称语义** (denotational semantics)，使程序的执行效果对应数学对象，更关心程序的执行效果而非过程；
- **操作语义** (operational semantics)，用抽象状态机描述程序执行产生的状态改变，适合描述程序的执行过程；
- **关系语义** (relational semantics)²，基于关系定义程序语言的语义，更适合自动程序验证；
- **公理语义** (axiomatic semantics)，将程序的语义性质表示为命题，采用数理逻辑的方法研究。

¹[https://en.wikipedia.org/wiki/Semantics_\(computer_science\)](https://en.wikipedia.org/wiki/Semantics_(computer_science))

²https://en.wikipedia.org/wiki/Kripke_semantics

```
while (!(y == 0)) {  
    if (y >= 0) {  
        y = y - 1;  
    } else {  
        y = y + 1;  
    }  
    x = x + 1;  
}
```

- 执行该程序之后， x 的值变为 x 的初始值和 y 的初始值的绝对值的和， y 的值变为 0。

注意：在程序的具体语法中，赋值运算符使用“=”，关系运算符使用“==”、“>=”等，逻辑运算符使用“!”、“&&”、“||”等。

状态 (state)³ 是一个从变元集到整数集的全函数

$$State : Var \rightarrow \mathbb{Z}$$

- 状态就是对程序变元的一组赋值
- 除非引起歧义，后面的讨论中将不区分状态和赋值
- 记 \mathcal{S} 为程序中所有状态的全集

例 (设程序 P 的变元集是 $\{x, y\}$)

- $s = \{x \mapsto 5, y \mapsto 7\}$ 是程序的一个状态 (赋值)，且 $s(x) = 5, s(y) = 7$ 。
- 如果我们总是按照先 x 再 y 的顺序确定状态赋值，上面的状态 s 可以简记为 $\langle 5, 7 \rangle$ 。
- $s' = s[x \mapsto 0]$ 是赋值 (状态) s 的一个变体，即 $s' = \langle 0, 7 \rangle$ 。

³一些文献中将程序状态也称为格局 (configuration)，定义为二元组 $\langle env, pc \rangle$ ，其中 env 为变量赋值，对应这里的状态； pc 为下一条将被执行的语句。

考虑下面的算术表达式：

$$(x + 2) * y$$

- 要确定该表达式的值，必须先确定变元 x 和 y 的赋值
- 例如，设 $x = 1, y = 3$ ，则原表达式的值为 $(1 + 2) * 3 = 9$
- 状态是对程序所有变元（包括 x 和 y ）的一组赋值
- 在给定了状态的情况下，算术表达式的值容易确定

算术表达式的语义函数 $\llbracket \cdot \rrbracket_s^A : AExp \rightarrow (\mathcal{S} \rightarrow \mathbb{Z})$ 。

定义 (算术表达式的语义)

算术表达式 e 在状态 s 下的值 (记作 $\llbracket e \rrbracket_s^A$) 递归定义如下:

- $\llbracket c \rrbracket_s^A = c$, 其中 c 为整数
- $\llbracket x \rrbracket_s^A = s(x)$, 其中 x 为变元
- $\llbracket e_1 + e_2 \rrbracket_s^A = \llbracket e_1 \rrbracket_s^A + \llbracket e_2 \rrbracket_s^A$
- $\llbracket e_1 - e_2 \rrbracket_s^A = \llbracket e_1 \rrbracket_s^A - \llbracket e_2 \rrbracket_s^A$
- $\llbracket e_1 * e_2 \rrbracket_s^A = \llbracket e_1 \rrbracket_s^A \times \llbracket e_2 \rrbracket_s^A$

注意:

- 上述公式中等号左边的“+”，“-”，“*”是语法符号，等号右边的“+”，“-”，“×”分别代表整数上的加法、减法和乘法运算。
- 在不引起歧义的情况下，常常将 $\llbracket e \rrbracket_s^A$ 简记为 $\llbracket e \rrbracket_s$ 。

例

求算术表达式 $(x + 2) * y$ 在状态 $s = \{x \mapsto 1, y \mapsto 3\}$ 下的值。

解

$$\begin{aligned} \llbracket (x + 2) * y \rrbracket_s^A &= \llbracket (x + 2) \rrbracket_s^A \times \llbracket y \rrbracket_s^A \\ &= (\llbracket x \rrbracket_s^A + \llbracket 2 \rrbracket_s^A) \times \llbracket y \rrbracket_s^A \\ &= (s(x) + 2) \times s(y) \\ &= (1 + 2) \times 3 \\ &= 9 \end{aligned}$$

考虑下面的布尔表达式：

$$(x > 2) \wedge (y \neq x)$$

- 要确定该布尔表达式是否成立，必须先确定变元 x 和 y 的赋值
- 例如，设 $x = 1, y = 3$ ，则原布尔表达式变为 $(1 > 2) \wedge (3 \neq 1)$ ，显然不成立

布尔表达式的语义函数 $\llbracket \cdot \rrbracket_s^B : BExp \rightarrow (\mathcal{S} \rightarrow \mathbb{B})$ 。

定义 (布尔表达式的语义)

布尔表达式 p 在状态 s 下的值 (记作 $\llbracket p \rrbracket_s^B$) 递归定义如下:

- $\llbracket \mathbf{true} \rrbracket_s^B = true, \llbracket \mathbf{false} \rrbracket_s^B = false$
- $\llbracket e_1 = e_2 \rrbracket_s^B = \begin{cases} true, & \text{如果 } \llbracket e_1 \rrbracket_s^A = \llbracket e_2 \rrbracket_s^A \\ false, & \text{如果 } \llbracket e_1 \rrbracket_s^A \neq \llbracket e_2 \rrbracket_s^A \end{cases}$
- $\llbracket e_1 \leq e_2 \rrbracket_s^B = \begin{cases} true, & \text{如果 } \llbracket e_1 \rrbracket_s^A \leq \llbracket e_2 \rrbracket_s^A \\ false, & \text{如果 } \llbracket e_1 \rrbracket_s^A > \llbracket e_2 \rrbracket_s^A \end{cases}$
- $\llbracket \neg p \rrbracket_s^B = \begin{cases} true, & \text{如果 } \llbracket p \rrbracket_s^B = false \\ false, & \text{如果 } \llbracket p \rrbracket_s^B = true \end{cases}$
- $\llbracket p_1 \wedge p_2 \rrbracket_s^B = \begin{cases} true, & \text{如果 } \llbracket p_1 \rrbracket_s^B = true \text{ 且 } \llbracket p_2 \rrbracket_s^B = true \\ false, & \text{如果 } \llbracket p_1 \rrbracket_s^B = false \text{ 或 } \llbracket p_2 \rrbracket_s^B = false \end{cases}$

在不引起歧义的情况下，常常将 $\llbracket e \rrbracket_s^B$ 简记为 $\llbracket e \rrbracket_s$ 。

根据上面的定义，有：

- $\llbracket e_1 = e_2 \rrbracket_s^B \Leftrightarrow (\llbracket e_1 \rrbracket_s^A = \llbracket e_2 \rrbracket_s^A)$
- $\llbracket e_1 \leq e_2 \rrbracket_s^B \Leftrightarrow (\llbracket e_1 \rrbracket_s^A \leq \llbracket e_2 \rrbracket_s^A)$
- $\llbracket \neg p \rrbracket_s^B \Leftrightarrow \neg \llbracket p \rrbracket_s^B$
- $\llbracket p_1 \wedge p_2 \rrbracket_s^B \Leftrightarrow \llbracket p_1 \rrbracket_s^B \wedge \llbracket p_2 \rrbracket_s^B$

如果布尔表达式 p 在状态 s 下的值为

- 真：就称状态 s 满足 p ，记作 $s \models p$ ；
- 假：就称状态 s 不满足 p ，记作 $s \not\models p$ 。

例

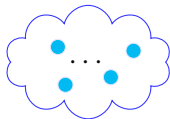
计算布尔表达式 $(x > 2) \wedge (y \neq x)$ 在状态 $s = \{x \mapsto 1, y \mapsto 3\}$ 的值。

解

$$\begin{aligned} \llbracket (x > 2) \wedge (y \neq x) \rrbracket_s^B &\Leftrightarrow \llbracket (x > 2) \rrbracket_s^B \wedge \llbracket (y \neq x) \rrbracket_s^B \\ &\Leftrightarrow (\llbracket x \rrbracket_s^A > \llbracket 2 \rrbracket_s^A) \wedge (\llbracket y \rrbracket_s^A \neq \llbracket x \rrbracket_s^A) \\ &\Leftrightarrow (s(x) > 2) \wedge (s(y) \neq s(x)) \\ &\Leftrightarrow (1 > 2) \wedge (3 \neq 1) \\ &\Leftrightarrow \text{false} \wedge \text{true} \\ &\Leftrightarrow \text{false} \end{aligned}$$

布尔表达式 p 可以用满足 p 的状态集合来刻画，即

$$\{s \mid s \models p\}$$



简单起见，以 $\{p\}$ 表示满足 p 的状态集合。

布尔表达式 p 在状态 s 下语义为真的另一种表述是 $s \in \{p\}$ 。

区别于算术表达式和布尔表达式，程序语句的语义无法在单个状态下确定。相反，程序语句的执行会导致程序状态的改变。

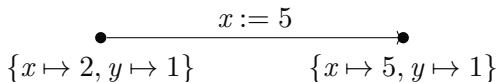
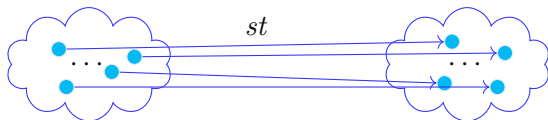


图: 语句执行导致程序状态的改变

- 显然，从不同状态出发执行该语句，可能得到不同的结束状态。
- 状态对 $(\{x \mapsto 2, y \mapsto 1\}, \{x \mapsto 5, y \mapsto 1\})$ 代表了该语句的一种执行情况。
- 如果能够找出该语句所有可能的执行情况，也就定义了该语句的语义。

基本思想：如果从状态 s 出发执行语句 st 能够得到状态 s' ，那么状态对 (s, s') 就代表了语句 st 的一种可能的执行情况。**语句 st 的语义**可以被解释为对应其所有执行情况的状态对，即

$$\llbracket st \rrbracket = \{(s, s') \mid \text{从 } s \text{ 出发执行 } st \text{ 可能会得到 } s'\}$$



此时，也称 s 为**前状态** (prestate)， s' 为**后状态** (poststate)。

显然， $\llbracket st \rrbracket \subseteq \mathcal{S} \times \mathcal{S}$ 是状态集合 \mathcal{S} 上的二元关系。

注意：我们分别用 $\llbracket \cdot \rrbracket^A$, $\llbracket \cdot \rrbracket^B$, $\llbracket \cdot \rrbracket$ 表示算术表达式、布尔表达式和程序语句的语义函数。

```
while (!(y == 0)) {  
    if (y >= 0) {  
        y = y - 1;  
    } else {  
        y = y + 1;  
    }  
    x = x + 1;  
}
```

- 执行该程序之后， x 的值变为 x 和 y 的初始值绝对值的和， y 的值变为 0。
- 该程序的状态全集： $\mathcal{S} = \mathbb{Z}\{x,y\}$

下面通过这个例子演示程序语句关系语义的定义

注意：在程序的具体语法中，赋值运算符使用“=”，关系运算符使用“==”、“>=”等，逻辑运算符使用“!”、“&&”、“||”等。

IMP 中的语句:

$$\begin{aligned} st \in Stmt ::= & \mathbf{skip} \mid x := e \mid st_1; st_2 \\ & \mid \mathbf{if} (p) \{st_1\} \mathbf{else} \{st_2\} \\ & \mid \mathbf{while} (p) \{st\} \end{aligned}$$

空语句的语义:

$$\llbracket \mathbf{skip} \rrbracket = \{(s, s) \mid s \in \mathcal{S}\}$$

即空语句对程序状态无影响。

赋值语句的语义:

$$\llbracket x := e \rrbracket = \{(s, s') \mid s' = s[x \mapsto \llbracket e \rrbracket_s^A]\}$$

即后状态 s' 中只有 x 的值发生改变, 且被修改为表达式 e 在前状态 s 下的值。

示例程序 P_{xy} :

$$\begin{aligned}\llbracket x := x + 1 \rrbracket &= \{(s, s') \mid s' = s[x \mapsto \llbracket x + 1 \rrbracket_s^A]\} \\ &= \{(s, s') \mid s'(x) = \llbracket x + 1 \rrbracket_s^A \text{ 且 } s'(y) = s(y)\} \\ &= \{(s, s') \mid s'(x) = s(x) + 1 \text{ 且 } s'(y) = s(y)\}\end{aligned}$$

分支语句的语义：

$$\llbracket \text{if } (p) \{st_1\} \text{ else } \{st_2\} \rrbracket = \left\{ (s, s') \mid \begin{array}{l} s \models p \text{ 且 } (s, s') \in \llbracket st_1 \rrbracket \\ \text{或 } s \not\models p \text{ 且 } (s, s') \in \llbracket st_2 \rrbracket \end{array} \right\}$$

即如果条件 p 在前状态 s 下成立，执行 st_1 ，否则执行 st_2 分支。

示例程序 P_{xy} ：

$$\begin{aligned} & \llbracket \text{if } (y \geq 0) \{y := y - 1\} \text{ else } \{y := y + 1\} \rrbracket \\ &= \left\{ (s, s') \mid \begin{array}{l} s \models y \geq 0 \text{ 且 } (s, s') \in \llbracket y := y - 1 \rrbracket \\ \text{或 } s \models y < 0 \text{ 且 } (s, s') \in \llbracket y := y + 1 \rrbracket \end{array} \right\} \\ &= \left\{ (s, s') \mid \begin{array}{l} \llbracket y \rrbracket_s^A \geq 0 \text{ 且 } s' = s[y \mapsto \llbracket y - 1 \rrbracket_s^A] \\ \text{或 } \llbracket y \rrbracket_s^A < 0 \text{ 且 } s' = s[y \mapsto \llbracket y + 1 \rrbracket_s^A] \end{array} \right\} \\ &= \left\{ (s, s') \mid \begin{array}{l} s(y) \geq 0 \text{ 且 } s'(y) = s(y) - 1 \text{ 且 } s'(x) = s(x) \\ \text{或 } s(y) < 0 \text{ 且 } s'(y) = s(y) + 1 \text{ 且 } s'(x) = s(x) \end{array} \right\} \end{aligned}$$

定义 (关系的组合)

设 R_1, R_2 为定义在同一个集合 X 上的两个二元关系, R_1 和 R_2 的组合关系 $R_1 \circ R_2$ 定义为:

$$R_1 \circ R_2 ::= \{(a, b) \mid \text{存在 } c \in X, \text{ 使得 } (a, c) \in R_1, (c, b) \in R_2\}$$

例

设 R_1, R_2 都是整数集上的“加一”关系, 即

$$R_i = \{(a, b) \mid b = a + 1\}, \text{ 其中 } i = 1, 2$$

则 $R_1 \circ R_2 = \{(a, b) \mid b = a + 2\}$ 。

顺序语句的语义：

$$\begin{aligned} \llbracket st_1; st_2 \rrbracket &= \llbracket st_1 \rrbracket \circ \llbracket st_2 \rrbracket \\ &= \{(s, s') \mid \text{存在 } s'' \text{ 使得 } (s, s'') \in \llbracket st_1 \rrbracket, (s'', s') \in \llbracket st_2 \rrbracket\} \end{aligned}$$

从前状态 s 出发执行 st_1 ，得到中间状态 s'' ，再从 s'' 出发执行 st_2 。

举例：

$$\begin{aligned} &\llbracket y := y - 1; x := x + 1 \rrbracket \\ &= \llbracket y := y - 1 \rrbracket \circ \llbracket x := x + 1 \rrbracket \\ &= \{(s, s') \mid \text{存在 } s'' \text{ 使得 } (s, s'') \in \llbracket y := y - 1 \rrbracket, (s'', s') \in \llbracket x := x + 1 \rrbracket\} \\ &= \{(s, s') \mid s'(x) = s(x) + 1, s'(y) = s(y) - 1\} \end{aligned}$$

示例程序 P_{xy} :

$$\begin{aligned}
 & \llbracket \text{if } (y \geq 0) \{y := y - 1\} \text{ else } \{y := y + 1\}; x := x + 1 \rrbracket \\
 = & \llbracket \text{if } (y \geq 0) \{y := y - 1\} \text{ else } \{y := y + 1\} \rrbracket \circ \llbracket x := x + 1 \rrbracket \\
 = & \left\{ (s, s'') \left| \begin{array}{l} s(y) \geq 0 \text{ 且 } s''(y) = s(y) - 1 \text{ 且 } s''(x) = s(x) \\ \text{或 } s(y) < 0 \text{ 且 } s''(y) = s(y) + 1 \text{ 且 } s''(x) = s(x) \end{array} \right. \right\} \\
 & \circ \{(s'', s') \mid s'(x) = s''(x) + 1, s'(y) = s''(y)\} \\
 = & \left\{ (s, s') \left| \begin{array}{l} \text{存在 } s'' \text{ 使得下面的 (1) 或 (2) 成立, 并且 (3) 成立} \\ (1) s(y) \geq 0 \text{ 且 } s''(y) = s(y) - 1 \text{ 且 } s''(x) = s(x) \\ (2) s(y) < 0 \text{ 且 } s''(y) = s(y) + 1 \text{ 且 } s''(x) = s(x) \\ (3) s'(x) = s''(x) + 1, s'(y) = s''(y) \end{array} \right. \right\} \\
 = & \left\{ (s, s') \left| \begin{array}{l} s(y) \geq 0 \text{ 且 } s'(y) = s(y) - 1 \text{ 且 } s'(x) = s(x) + 1 \\ \text{或 } s(y) < 0 \text{ 且 } s'(y) = s(y) + 1 \text{ 且 } s'(x) = s(x) + 1 \end{array} \right. \right\}
 \end{aligned}$$

循环语句的语义：

$\llbracket \text{while } (p) \{st\} \rrbracket$

$$= \left\{ (s, s') \left| \begin{array}{l} \text{存在一个整数 } n (n \geq 0) \text{ 和一组状态序列 } t_0, \dots, t_n, \\ \text{其中 } t_0 = s, t_n = s', \text{ 使得:} \\ (1) \text{ 最后的状态不满足循环条件, 即 } t_n \not\models p \\ (2) \text{ 对于所有满足 } 0 \leq i < n \text{ 的状态 } t_i: \\ \quad t_i \models p, \quad (t_i, t_{i+1}) \in \llbracket st \rrbracket \\ \quad \text{即 } t_i \text{ 满足循环条件, 并且从 } t_i \text{ 出发执行 } st \text{ 得到 } t_{i+1} \end{array} \right. \right\}$$

从状态 s 出发反复执行语句 st , 直至循环条件不成立。

注意：

- 条件 (1) 假设循环迭代 n 次之后会终止；如果循环不终止，则循环永远无法到达结束状态，对应的语义为空集。
- 当 $n = 0$ 时，不存在 $0 \leq i < n$ 的状态，对条件 (2) 无要求。

请思考下列语句的语义:

- $\llbracket \mathbf{while}(\mathbf{true}) \{st\} \rrbracket = \emptyset$
- $\llbracket \mathbf{while}(\mathbf{false}) \{st\} \rrbracket = \{(s, s) \mid s \in \mathcal{S}\}$
- $\llbracket x := x \rrbracket = \{(s, s) \mid s \in \mathcal{S}\}$

请思考示例程序 P_{xy} 的语义 (以 $body$ 代表循环体中的语句序列):

$$P_{xy} = \left\{ (s, s') \left| \begin{array}{l} \text{存在一个整数 } n \text{ 和一组状态序列} \\ t_0 = s, t_1, t_2, \dots, t_n = s', \text{ 使得:} \\ (1) t_n \not\models y \neq 0 \\ (2) t_i \models y \neq 0, (t_i, t_{i+1}) \in \llbracket body \rrbracket, 0 \leq i < n \end{array} \right. \right\} \\
 = \left\{ (s, s') \left| \begin{array}{l} \text{存在一个整数 } n \text{ 和一组状态序列} \\ t_0 = s, t_1, t_2, \dots, t_n = s', \text{ 使得:} \\ (1) t_n(y) = 0 \\ (2) t_i(y) \neq 0, (t_i, t_{i+1}) \in \llbracket body \rrbracket, 0 \leq i < n \end{array} \right. \right\}$$

循环语句的语义难以确定!

定义

如果对于任意状态 s 和 s' , $(s, s') \in \llbracket st_1 \rrbracket$ 当且仅当 $(s, s') \in \llbracket st_2 \rrbracket$, 则称语句 st_1 和 st_2 **语义等价** (semantically equivalent)。

例

证明语句 $x := x$ 和语句 **while (false) {st}** 语义等价。

证明.

$$\llbracket x := x \rrbracket = \llbracket \mathbf{while} \ (\mathbf{false}) \ \{st\} \rrbracket = \{(s, s) \mid s \in \mathcal{S}\} \quad \square$$

例

证明语句 $\mathbf{while} (p)\{st\}$ 和语句 $\mathbf{if} (p)\{st; \mathbf{while} (p)\{st\}\} \mathbf{else skip}$ 语义等价。

证明.

- \Rightarrow : 设 $(s, s') \in \llbracket \mathbf{while} (p)\{st\} \rrbracket$, 根据循环语句的语义, 存在一个整数 n 和一个状态序列满足其语义定义中的三个条件, 由此容易证明 $(s, s') \in \llbracket \mathbf{if} (p)\{st; \mathbf{while} (p)\{st\}\} \mathbf{else skip} \rrbracket$ 。
- \Leftarrow : 设 $(s, s') \in \llbracket \mathbf{if} (p)\{st; \mathbf{while} (p)\{st\}\} \mathbf{else skip} \rrbracket$, 分 $\llbracket p \rrbracket_s$ 为真和假两种情况, 分别证明 $(s, s') \in \llbracket \mathbf{while} (p)\{st\} \rrbracket$ 。



IMP 语法：

- 算术表达式、布尔表达式
- 程序语句

IMP 语义：

- 算术表达式和布尔表达式的语义
- 程序语句的语义

- 霍尔证明系统

谢谢!