《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

2022 年 3 月 20 日

上节课：一阶理论

一阶理论的定义：

- 签名 Σ，公理集 $\mathcal{A}$

一些常见的一阶理论：

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

本节课的内容

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

IMP：一个简单的示例语言

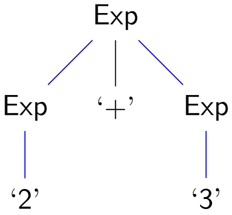
- 语法
- 语义
- 证明系统（下节课介绍）

语法

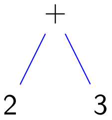
抽象语法

不考虑程序的具体语法（concrete syntax），而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。

IMP 的抽象语法

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）

程序语法与逻辑语法的对比

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对应的所有可能的计算的学科。

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；
- 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。
- 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。

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

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

示例程序 P_{xy}


```
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]是从变元集到整数集的全函数

$$State: \quad Var \to \mathbb{Z}$$

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

例 (程序 P_{xy} 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 x, y 的顺序确定赋值，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]一些文献中程序状态又称格局（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 为下一条将被执行语句。

可用一阶逻辑公式 φ 表示程序状态集合：

- φ 定义在程序变量之上，即 $FVar(\varphi) \subseteq Var$
- φ 中可以出现常元符号、函数符号和谓词符号；它们的解释 $\mathcal{M}$ 由对应的一阶理论（如整数算术理论）定义

以 $\{\varphi\}$ 表示满足 φ 的所有状态的集合，即：

$$\{\varphi\} ::= \{s \mid [\![\varphi]\!]_{\mathcal{M},s} = true\}$$

其中，$[\![\varphi]\!]_{\mathcal{M},s}$ 中的 $\mathcal{M}$ 由一阶理论给定，常简记为 $[\![\varphi]\!]_s$。

例 (程序 P_{xy} 的变元集是 $\{x, y\}$)

- $x = 5 \wedge y = 7$ 对应的状态集合为 $\{\langle 5, 7\rangle\}$
- $x = 5$ 对应的状态集合为 $\{\dots, \langle 5, -1\rangle, \langle 5, 0\rangle, \langle 5, 1\rangle, \dots\}$
- $\exists k.(x = 2 * k \wedge y = 2 * k + 1)$ 对应的状态集合为 $\{\dots, \langle -2, -1\rangle, \langle 0, 1\rangle, \langle 2, 3\rangle, \dots\}$
- $\top$ 对应全集 $\mathcal{S} = \mathbb{Z}^{Var}$，$\perp$ 对应空集 $\emptyset$

定义 (算术表达式的语义)

算术表达式 e 在状态 s 下的值 $[\![e]\!]_s$ 递归定义如下：

- $[\![c]\!]_s = c$, 其中 c 为整数
- $[\![x]\!]_s = s(x)$，其中 x 为变元
- $[\![e_1 + e_2]\!]_s = [\![e_1]\!]_s + [\![e_2]\!]_s$
- $[\![e_1 - e_2]\!]_s = [\![e_1]\!]_s - [\![e_2]\!]_s$
- $[\![e_1 * e_2]\!]_s = [\![e_1]\!]_s \times [\![e_2]\!]_s$

注意：上述公式中等号左边的“+”，“-”，“*”是语法符号，等号右边的“+”，“-”，“×”是整数算术理论的函数符号（分别被解释为整数加法、整数减法和整数乘法）。

例

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

解

$$
\begin{aligned}
[\![(x+2)*y]\!]_s &= [\![(x+2)]\!]_s \times [\![y]\!]_s \\
&= ([\![x]\!]_s + [\![2]\!]_s) \times [\![y]\!]_s \\
&= (s(x)+2) \times s(y) \\
&= (1+2) \times 3 \\
&= 9
\end{aligned}
$$

布尔表达式的语义

定义 (布尔表达式的语义)

布尔表达式 p 在状态 s 下的值 $[\![p]\!]_s$ 递归定义如下：

- $[\![\mathbf{true}]\!]_s = \mathit{true}$
- $[\![\mathbf{false}]\!]_s = \mathit{false}$
- $[\![e_1 = e_2]\!]_s = \mathit{true}$ 当且仅当 $[\![e_1]\!]_s = [\![e_2]\!]_s$
- $[\![e_1 \le e_2]\!]_s = \mathit{true}$ 当且仅当 $[\![e_1]\!]_s \le [\![e_2]\!]_s$
- $[\![\neg p]\!]_s = \mathit{true}$ 当且仅当 $[\![p]\!]_s = \mathit{false}$
- $[\![p_1 \wedge p_2]\!]_s = \mathit{true}$ 当且仅当 $[\![p_1]\!]_s$ 和 $[\![p_2]\!]_s$ 都为真

布尔表达式都是一阶逻辑公式，因此每个布尔表达式对应程序的一个状态集合。
布尔表达式 p 在状态 s 下语义为真的另一种表述是 $s \in \{p\}$。

示例

例

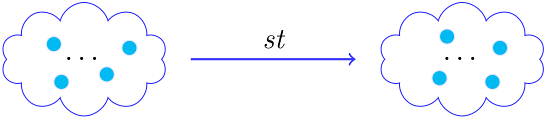
计算布尔表达式 $(x > 2) \wedge (y \neq x)$ 在状态 $s = \{x \mapsto 1, y \mapsto 3\}$ 的值。

解

$$
\begin{aligned}
[\![(x > 2) \wedge (y \neq x)]\!]_s &\Leftrightarrow [\![(x > 2)]\!]_s \wedge [\![(y \neq x)]\!]_s \\
&\Leftrightarrow ([\![x]\!]_s > [\![2]\!]_s) \wedge ([\![y]\!]_s \neq [\![x]\!]_s) \\
&\Leftrightarrow (s(x) > 2) \wedge (s(y) \neq s(x)) \\
&\Leftrightarrow (1 > 2) \wedge (3 \neq 1) \\
&\Leftrightarrow \mathit{false} \wedge \mathit{true} \\
&\Leftrightarrow \mathit{false}
\end{aligned}
$$

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

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



此时，也称 s 为前状态（prestate），s' 为后状态（poststate）。

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

注意：这里的语句 st 可以是一条简单语句，也可以是由多条语句通过 “;” 组合而成的语句序列。

回顾：示例程序 P_{xy}


```
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。
- 该程序的状态全集：$\mathcal{S} = \mathbb{Z}^{Var}$
- 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：

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

---

skip 语句的语义：

$$
[\![\textbf{skip}]\!] = \{(s, s) \mid s \in \mathcal{S}\}
$$

即 **skip** 语句对程序状态无影响。

赋值语句的语义：

$$[\![x := e]\!] = \{(s, s') \mid s' = s[x \mapsto [\![e]\!]_s]\}$$

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

示例程序 P_{xy}：

$$\begin{aligned}[\![x := x+1]\!] =& \{(s', s'') \mid s'' = s'[x \mapsto [\![x+1]\!]_{s'}]\} \\ =& \{(s', s'') \mid s''(x) = [\![x+1]\!]_{s'} \text{ 且 } s''(y) = s'(y)\} \\ =& \{(s', s'') \mid s''(x) = s'(x) + 1, \text{ 且 } s''(y) = s'(y)\}\end{aligned}$$

分支语句的语义：

$$[\![\textbf{if } (p)\{st_1\} \textbf{ else } \{st_2\}]\!] = \left\{ (s, s') \;\middle|\; \begin{array}{r} [\![p]\!]_s = \mathit{true} \text{ 且 } (s, s') \in [\![st_1]\!] \\ \text{或 } [\![p]\!]_s = \mathit{false} \text{ 且 } (s, s') \in [\![st_2]\!] \end{array} \right\}$$

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}：

$$\begin{aligned}
& [\![\textbf{if } (y \geq 0)\ \{y := y - 1\} \textbf{ else } \{y := y + 1\}]\!] \\
= & \left\{ (s, s') \;\middle|\; \begin{array}{r} [\![y \geq 0]\!]_s = \mathit{true} \text{ 且 } (s, s') \in [\![y := y - 1]\!] \\ \text{或 } [\![y \geq 0]\!]_s = \mathit{false} \text{ 且 } (s, s') \in [\![y := y + 1]\!] \end{array} \right\} \\
= & \left\{ (s, s') \;\middle|\; \begin{array}{r} [\![y]\!]_s \geq 0 \text{ 且 } s' = s[y \mapsto [\![y - 1]\!]_s] \\ \text{或 } [\![y]\!]_s < 0 \text{ 且 } s' = s[y \mapsto [\![y + 1]\!]_s] \end{array} \right\} \\
= & \left\{ (s, s') \;\middle|\; \begin{array}{r} s(y) \geq 0 \text{ 且 } s'(y) = s(y) - 1 \text{ 且 } s'(x) = s(x) \\ \text{或 } s(y) < 0 \text{ 且 } s'(y) = s(y) + 1 \text{ 且 } s'(x) = s(x) \end{array} \right\}
\end{aligned}$$

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
$R_i = \{(a, b) \mid b = a + 1\}$，其中 $i = 1, 2$，则
$R_1 \circ R_2 = \{(a, b) \mid b = a + 2\}$。

顺序语句的语义：

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

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。

示例程序 P_{xy}**：**

$$
\begin{aligned}
&[\![\textbf{if } (y \geq 0)\ \{y := y - 1\} \textbf{ else } \{y := y + 1\};\ x := x + 1]\!] \\
=&[\![\textbf{if } (y \geq 0)\ \{y := y - 1\} \textbf{ else } \{y := y + 1\}]\!] \circ [\![x := x + 1]\!] \\
=&\left\{ (s, s') \;\middle|\; \begin{array}{r} s(y) \geq 0 \text{ 且 } s'(y) = s(y) - 1 \text{ 且 } s'(x) = s(x) \\ \text{或 } s(y) < 0 \text{ 且 } s'(y) = s(y) + 1 \text{ 且 } s'(x) = s(x) \end{array} \right\} \\
&\circ \{(s', s'') \mid s''(x) = s'(x) + 1, s''(y) = s'(y)\} \\
=&\left\{ (s, s'') \;\middle|\; \begin{array}{r} s(y) \geq 0 \text{ 且 } s''(y) = s(y) - 1 \text{ 且 } s''(x) = s(x) + 1 \\ \text{或 } s(y) < 0 \text{ 且 } s''(y) = s(y) + 1 \text{ 且 } s''(x) = s(x) + 1 \end{array} \right\}
\end{aligned}
$$

循环语句的语义：

$$
\begin{aligned}
&[\![\textbf{while}\ (p)\ \{st\}]\!] \\
&= \left\{ (s, s') \;\middle|\; \begin{array}{l} \text{存在一个整数 } n \text{ 和一组状态序列 } t_0 = s, t_1, t_2, \ldots, \\ t_n = s', \text{使得对任意的} 0 \le i < n: \\ \text{(1) 循环条件都成立，即 } [\![p]\!]_{t_i} = \mathit{true} \\ \text{(2) 从 } t_i \text{ 出发执行 } st \text{ 得到 } t_{i+1}, \text{即 } (t_i, t_{i+1}) \in [\![st]\!] \\ \text{(3) 最后不满足循环条件，即 } [\![p]\!]_{t_n} = \mathit{false} \end{array} \right\}
\end{aligned}
$$

从前状态 s 出发反复执行语句 st，直至循环条件不成立。
注意：非终止循环永远无法到达任何结束状态，对应的语义为空集。

请思考下列语句的语义:

- $[\![\textbf{while}\ (\textbf{true})\ \{st\}]\!] = \emptyset$
- $[\![\textbf{while}\ (\textbf{false})\ \{st\}]\!] = \{(s, s) \mid s \in \mathcal{S}\}$
- $[\![x := x]\!] = \{(s, s) \mid s \in \mathcal{S}\}$

请思考示例程序 P_{xy} 的语义:

$$P_{xy} = \left\{ (s, s') \left| \begin{array}{l} \text{存在一个整数 } n \text{ 和一组状态序列} \\ t_0 = s, t_1, t_2, \ldots, t_n = s', \text{使得:} \\ (1)\, [\![y \neq 0]\!]_{t_i} = \mathit{true}, 0 \leq i < n \\ (2)\, (t_i, t_{i+1}) \in [\![\mathit{body}]\!], 0 \leq i < n \\ (3)\, [\![y \neq 0]\!]_{t_n} = \mathit{false} \end{array} \right. \right\}$$

$$= \left\{ (s, s') \left| \begin{array}{l} \text{存在一个整数 } n \text{ 和一组状态序列} \\ t_0 = s, t_1, t_2, \ldots, t_n = s', \text{使得:} \\ (1)\, t_i(y) \neq 0, 0 \leq i < n \\ (2)\, (t_i, t_{i+1}) \in [\![\mathit{body}]\!], 0 \leq i < n \\ (3)\, t_n(y) = 0 \end{array} \right. \right\}$$

循环语句的语义难以确定!

定义

如果对于任意状态 s 和 s'，$(s, s') \in [\![st_1]\!]$ 当且仅当 $(s, s') \in [\![st_2]\!]$，则称语句 st_1 和 st_2 是语义等价（semantically equivalent）的。

例

证明语句 $x := x$ 和语句 $\textbf{while}\ (\textbf{false})\ \{st\}$ 是语义等价的。

证明.

$[\![x := x]\!] = [\![\textbf{while}\ (\textbf{false})\ \{st\}]\!] = \{(s, s) \mid s \in \mathcal{S}\}$ $\square$

例

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

证明.

$\rightarrow$: 设 $(s, s') \in [\![\textbf{while}\ (p)\{st\}]\!]$，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语义定义中的三个条件，由此容易证明 $(s, s') \in [\![\textbf{if}\ (p)\{st; \textbf{while}\ (p)\{st\}\}\ \textbf{else skip}]\!]$。

$\leftarrow$: 设 $(s, s') \in [\![\textbf{if}\ (p)\{st; \textbf{while}\ (p)\{st\}\}\ \textbf{else skip}]\!]$，分 $[\![p]\!]_s$ 为真和假两种情况，分别证明 $(s, s') \in [\![\textbf{while}\ (p)\{st\}]\!]$。

$\square$

总结：IMP 程序设计语言及其语义

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

下节课内容

- 霍尔证明系统

谢谢!