程序设计语言

胡振江, 张伟

2022年12月2日

Outline

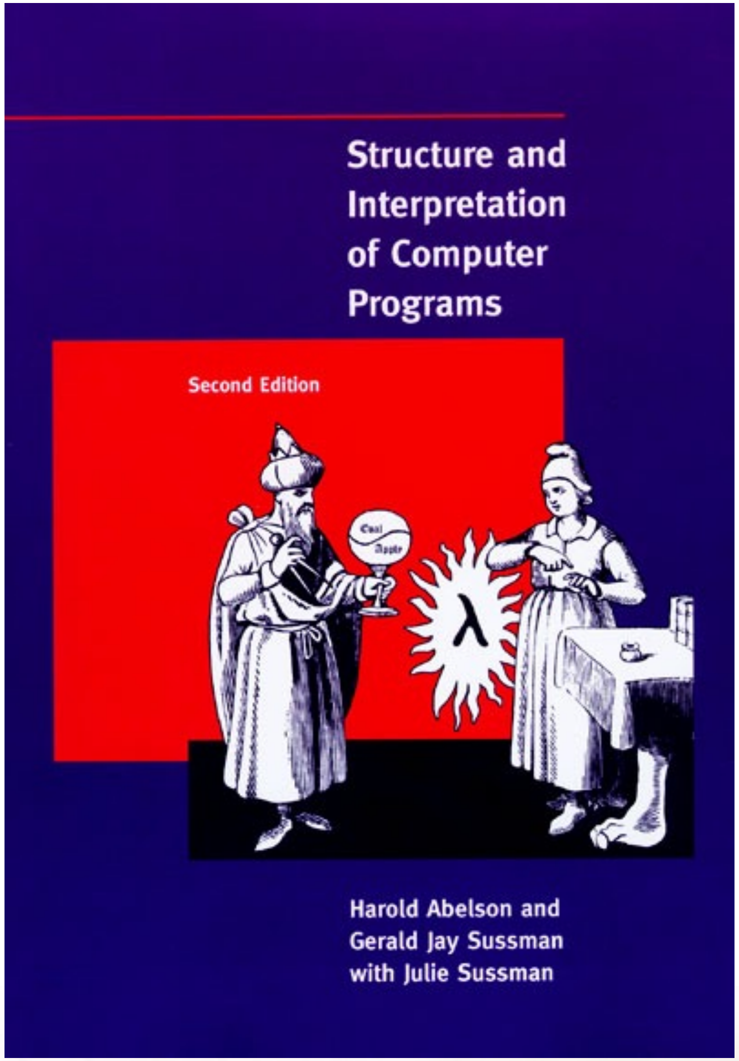
- 为什么要研究程序设计语言?
- 程序设计语言的基本概念
  - 程序设计语言与计算
  - 高级/低级程序设计语言
  - 编程范式
  - 语言的实现
- 语言的定义
  - 文法
  - 语义（操作语义，指称语义，公理化语义）

Outline

- 为什么要研究程序设计语言?
- 程序设计语言的基本概念
  - 程序设计语言与计算
  - 高级/低级程序设计语言
  - 编程范式
  - 语言的实现
- 语言的定义
  - 文法
  - 语义（操作语义，指称语义，公理化语义）

计算机科学 vs 程序设计语言

- "... the technology for coping with large-scale computer systems merges with the technology for building new computer languages, and computer science itself becomes no more (and no less) than the discipline of constructing appropriate descriptive languages"



学习程序设计语言的目的

- 帮助你理解程序设计语言的结构和设计原理
- 增强你描述、分析、利用程序设计语言的特征的能力
- 帮助你学习新的语言

Outline

- 为什么要研究程序设计语言?
- **程序设计语言的基本概念**
  - 程序设计语言与计算
  - 高级/低级程序设计语言
  - 编程范式
  - 语言的实现
- 语言的定义
  - 文法
  - 语义（操作语义，指称语义，公理化语义）

程序设计语言是什么?

- 程序设计语言是一个标记(notations)，用于
  - 描述计算
  - 组织计算
  - 推理计算

- 程序设计语言的设计者应该
  - 使人易于理解计算
  - 使机器易于高效实现

程序设计语言

程序设计语言的目的是为了使机器易用

- 一个语言是：
  - 高级的，如果它不依赖于特定的机器
    - 低级的
  - 通用的，如果可以应用于不同领域的问题
    - 领域特定的

机器语言：无智能

- 下面是对于冯诺伊曼机器的一个代码片段

0 0 0 0 0 0 1 0 1 0 1 1 1 1 0 0 1 0 1 0

0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 0 0

0 0 0 0 0 0 1 1 0 0 1 1 1 0 1 0 1 0 0 0

这的代码对人无智能，但是机器能够理解

汇编语言：低级

- 汇编语言：机器语言的一个变种

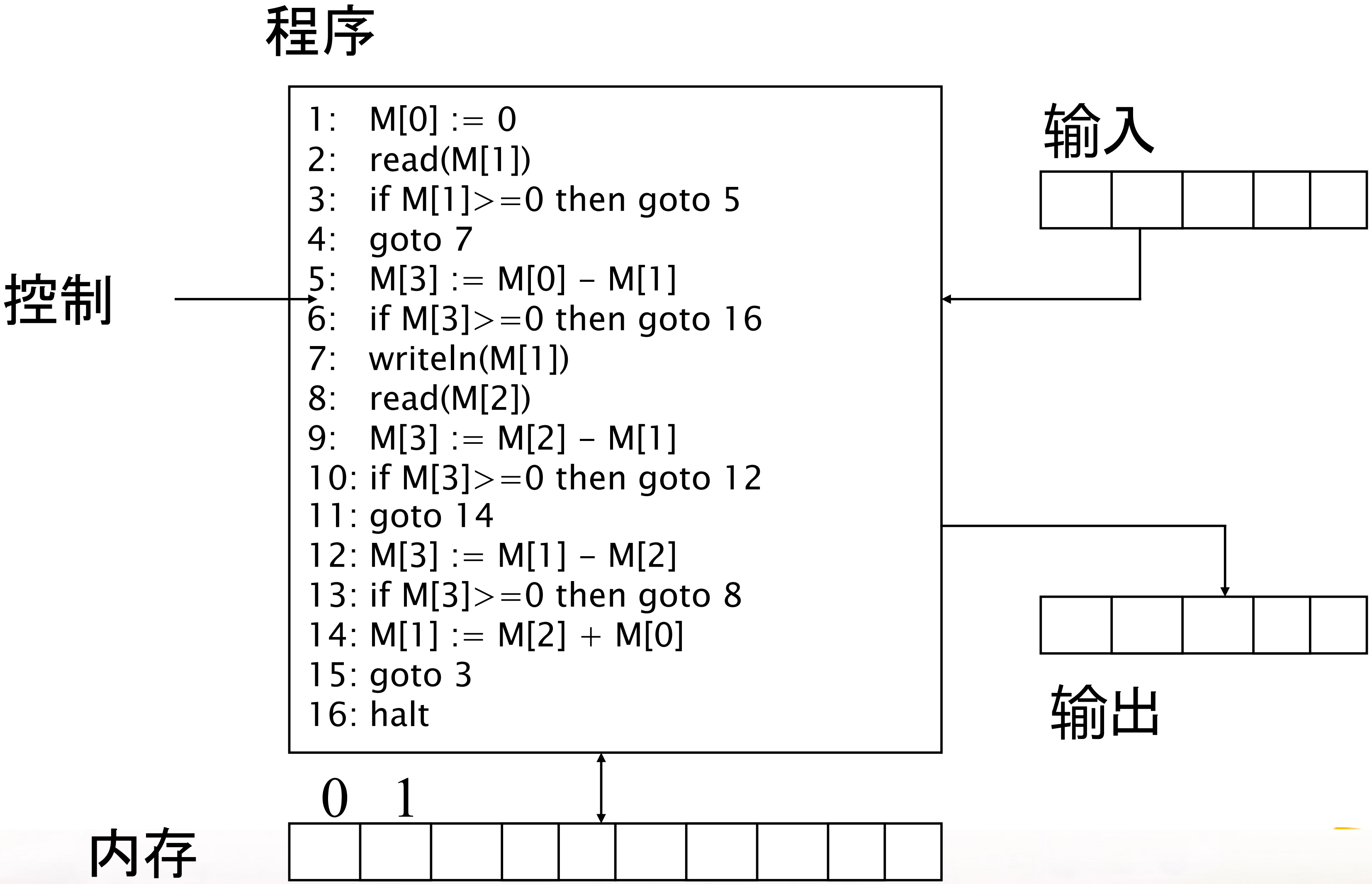
名字

↑

机器指令，值，地址

汇编语言的每个指令容易读懂

随机存取机 (Random Access Machine)



高级程序设计语言的利点

- 易读的描述
- 不依赖于特定的机器（可移植）
- 配备有用的库函数
- 自动一致性检查

编程范式（programming paradigm）

- 新的语言不是轻易引入的：
  - 语言的设计
  - 语言的实现
  - 语言的教学
  - 语言的支撑
- 语言：编程范式
  - 命令式程序设计语言
  - 函数式程序设计语言
  - 逻辑式程序设计语言
  - 面向目标的程序设计语言

命令式编程

计算被描述为状态的改变

- 例子
  - Fortran (1958)：面向科学计算
  - Algol 族：通用语言
  - Pascal (1971)：面向教学
  - C (1972)：面向系统软件开发（例如，Unix）

函数式编程

计算被描述为函数

- 例子
  - Lisp (1958)：面向符号处理
  - Scheme：用于研究和教学
  - ML：具有类型的严格计算语言
  - Haskell：具有类型的延迟计算语言

逻辑式编程

计算被描述为证明

- Prolog (1972): 开始是为了自然语言处理而设计

Every psychiatrist is a person.
Every person he analyzes is sick.
Jacques is a psychiatrist in Marseille.

Is Jacques a person?
Where is Jacques?
Is Jacques sick?

面向对象的编程

计算被描述为对象上的操作

- 例子
  - Simula (1967): 用于模拟
  - C++: C + 对象
  - Smalltalk: 受Lisp的影响

语言的实现

- 编译

源程序

机器代码

机器

- 解释

源程序

解释器

机器

编译 vs 解释

- 编译一般比解释更高效
  - 利用程序的静态性质
- 解释比编译更灵活
  - 利用程序的动态性质
  - 适合于程序调试

Outline

- 为什么要研究程序设计语言?
- 程序设计语言的基本概念
  - 程序设计语言与计算
  - 高级/低级程序设计语言
  - 编程范式
  - 语言的实现
- 语言的定义
  - 文法
  - 语义（操作语义，指称语义，公理化语义）

语言描述

- 语法：描述程序的结构（容易）

D D / D D / D D D D

- 语义：描述语言的含义（难）

0 1 / 0 2 / 2 0 0 1

具体文法

- 语言设计从定义具体语法(concrete syntax)开始

$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$

⇓

(-b + sqrt(b*b - 4.0*a*c)) / (2.0 * a)

前缀表示

- 表达式的前缀表示
  - 常数 或 变量 => 本身
  - 操作子作用到表达式 E1 和 E2 => op E1 E2

```
* + 20 30 60 = * 50 60 = 3000
* 20 + 30 60 = * 20 90 = 1800
```

- 利点
  - 易于从左到右解释表达式

后缀表示

- 表达式的后缀表示
  - 常数 或 变量 => 本身
  - 操作子作用到表达式 E1 和 E2 => E1 E2 op

20 30 + 60 * = 50 60 * = 3000
20 30 60 + * = 20 90 * = 1800

- 利点
  - 易于利用堆栈计算表达式

中缀表示

- 表达式的中缀表示
  - 常数 或 变量 => 本身
  - 操作子作用到表达式 E1 和 E2 => E1 op E2

$$20 + 30 * 60 = 50 * 60 = 3000$$
$$4 - 2 - 1 = 2 - 1 = 1$$

- 利点
  - 易于人们对于表达式的理解

Mixfix 描述

- 符号或保留字与表达式的元素交叉放置

```
if a>b then a else b
```

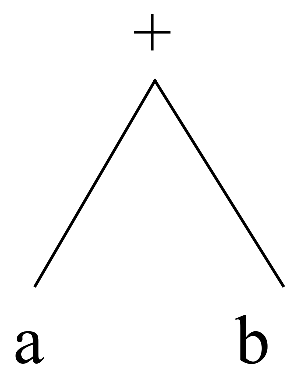
抽象语法树

- 描述每个语言的有意义的结构（不依赖于具体的描述）.

+ a b

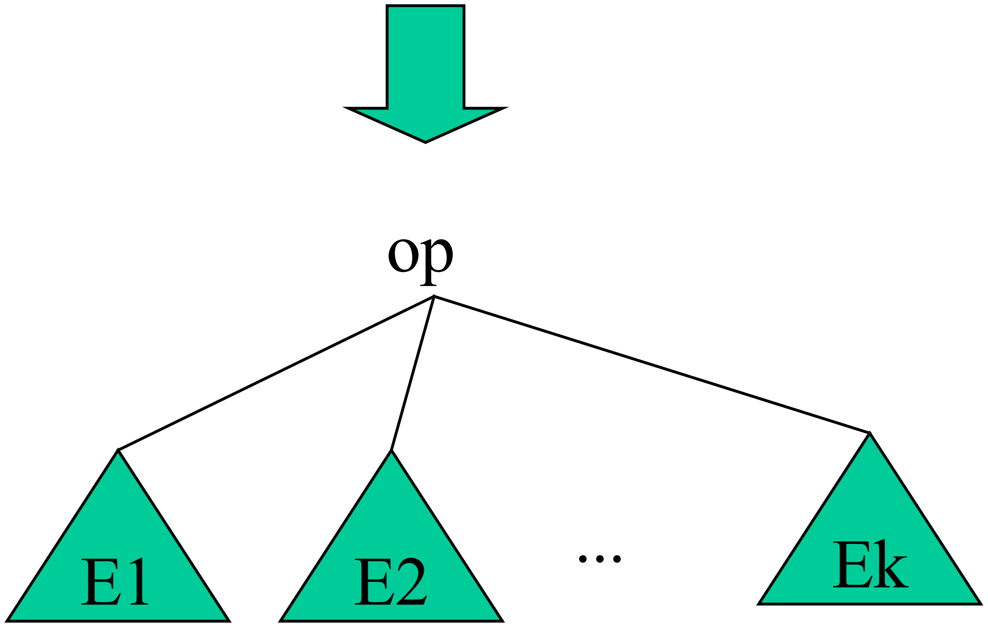
a + b

a b +



表达式的树状描述

op E1 E2 … Ek



上下文无关文法

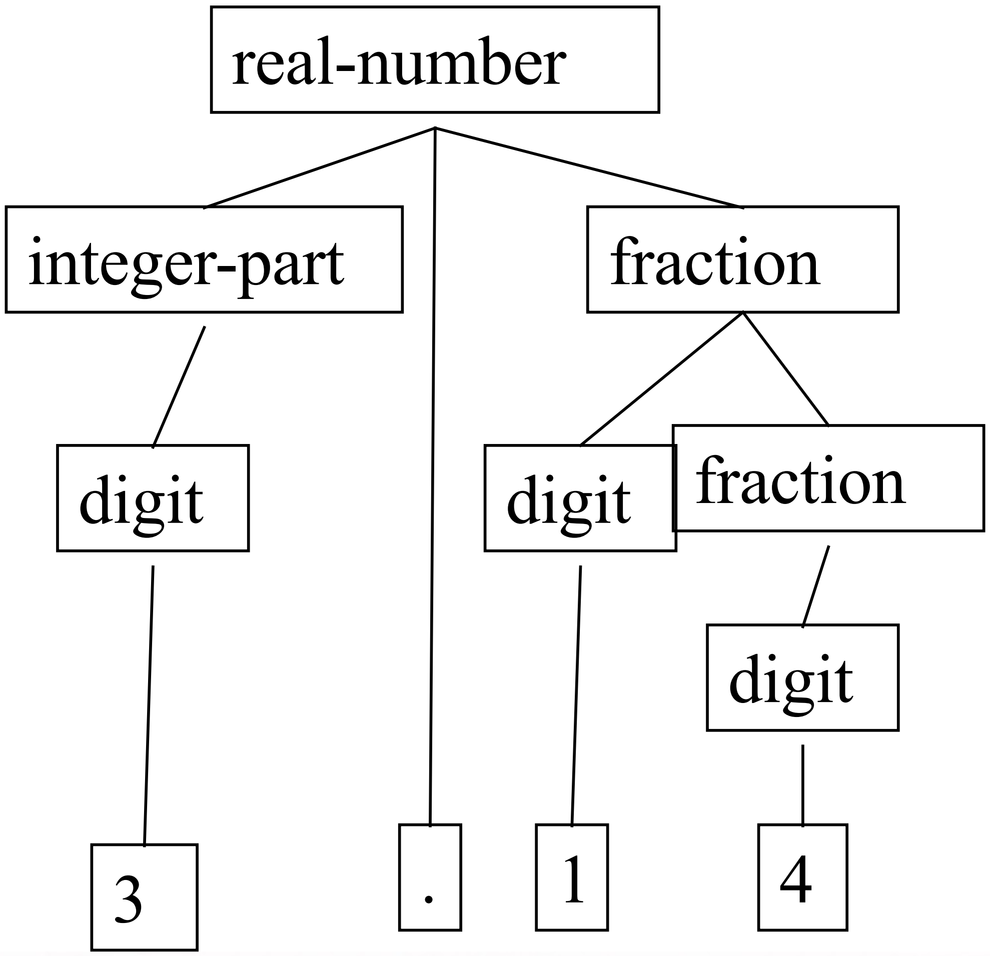
- 终端符集合
  - 表示语言的原子标志符
- 非终端符集合
  - 表示语言的结构
- 一组生成规则
  - 描述语言结构的生成关系
- 一个初始非终端符
  - 表示语言的主结构

BNF: Backus-Naur Form

<real-number> ::= <integer-part> . <fraction>
<integer-part> ::= <digit> | <integer-part> <digit>
<fraction> ::= <digit> | <digit> <fraction>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

BNF for real number

解析树 (parse tree)



- **root**: labeled with the staring non-terminal
- **non-leaf node**: labeled with a nonterminal
- **parent-children**: corresponds to production rules

解析过程

派生过程

real-number
=> integer-part . fraction
=> integer-part digit . fraction
=> digit digit . fraction
=> 2 digit . fraction
=> 2 1 . fraction
=> 2 1 . digit fraction
=> 2 1 . 8 fraction
=> 2 1 . 8 digit
=> 2 1 . 8 9

语法二义性

- 一个文字序列对应于两种不同的解析树

E ::= E - E | 0 | 1

1 - 0 - 1

S ::= if E then S
    | if E then S else S

if E1 then if E2 then S1else S2

Outline

- 为什么要研究程序设计语言?
- 程序设计语言的基本概念
  - 程序设计语言与计算
  - 高级/低级程序设计语言
  - 编程范式
  - 语言的实现
- 语言的定义
  - 文法
  - **语义（操作语义，指称语义，公理化语义）**

作用

- 程序设计语言的语义是描述文法与计算模型之间的关系

语义的三种定义方法

- 操作语义 (Operational Semantics)
  - 定义程序如何（在某个抽象机上）运行
  - 适合于语言实现
- 指称语义 (Denotational Semantics)
  - 将语言映射到一些数学系统里
  - 适合于程序推理
- 公理化语义 (Axiomatic semantics)
  - 根据公理和推理规则定义程序含义
  - 适合于程序验证

操作语义

- 一个语言的操作语义包括两个部分：
  - 语言的抽象文法

$$
\begin{aligned}
N ::= {} & 0 \\
& \mid S(N) \\
& \mid (N + N) \\
& \mid (N \times N)
\end{aligned}
$$

  - 基于重写规则定义计算关系

$$
\begin{aligned}
& I: N \rightarrow N \\
& I[\,(\, n + 0\,)\,] \rightarrow n \\
& I[\,(\, m + S(n)\,)\,] \rightarrow S(\, I[\,(m+n\,)\,]\,) \\
& I[\,(\, n \times 0\,)\,] \rightarrow 0 \\
& I[\,(\, m \times S(n))\,] \rightarrow I[\,((\, m \times n) + m)\,]
\end{aligned}
$$

北京大學 PEKING UNIVERSITY

例：简单的语言 IMP

The syntax of **IMP**defined by BNF (Backus-Naur form).

- **For Aexp:** $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$
- **For Bexp:** $b ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
- **For Com:**
  $c ::= \mathbf{skip} \mid X := a \mid c_0; c_1 \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c$

```
x := 1;
s := 0;
While x < 10 do (
  if x = 5 then skip else x + 1;
  s := s + x
)
```

例：简单的语言 IMP

- The set of states consists of functions $\sigma : \mathbf{Loc} \to \mathbf{N}$.
- A configuration is a pair $\langle a, \sigma \rangle$, where a is an arithmetic expression and σ a state.
- An evaluation relation between pairs and numbers $\langle a, \sigma \rangle \to n$

例：简单的语言 IMP

算术表达式的计算

Evaluation of numbers $\langle n, \sigma \rangle \to n$

Evaluation of locations $\langle X, \sigma \rangle \to \sigma(X)$

Evaluation of sums

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the sum of } n_0 \text{ and } n_1}{\langle a_0 + a_1, \sigma \rangle \to n}$$

Evaluation of subtractions

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the result of subtracting } n_1 \text{ from } n_0}{\langle a_0 - a_1, \sigma \rangle \to n}$$

Evaluation of products

$$\frac{\langle a_0, \sigma \rangle \to n_0 \qquad \langle a_1, \sigma \rangle \to n_1 \qquad n \text{ is the product of } n_0 \text{ and } n_1}{\langle a_0 \times a_1, \sigma \rangle \to n}$$

例：简单的语言 IMP

逻辑表达式的计算

$$\langle \mathbf{true}, \sigma\rangle \to \mathbf{true} \qquad \langle \mathbf{false}, \sigma\rangle \to \mathbf{false}$$

$$\frac{\langle a_0, \sigma\rangle \to n \quad \langle a_1, \sigma\rangle \to n}{\langle a_0 = a_1, \sigma\rangle \to \mathbf{true}} \qquad \frac{\langle a_0, \sigma\rangle \to n \quad \langle a_1, \sigma\rangle \to m \quad n \not\equiv m}{\langle a_0 = a_1, \sigma\rangle \to \mathbf{false}}$$

$$\frac{\langle a_0, \sigma\rangle \to n \quad \langle a_1, \sigma\rangle \to m \quad \text{if } n \text{ is less than or equal to } m}{\langle a_0 \le a_1, \sigma\rangle \to \mathbf{true}}$$

$$\frac{\langle a_0, \sigma\rangle \to n \quad \langle a_1, \sigma\rangle \to m \quad \text{if } n \text{ is not less than or equal to } m}{\langle a_0 \le a_1, \sigma\rangle \to \mathbf{false}}$$

$$\frac{\langle b, \sigma\rangle \to \mathbf{true}}{\langle \neg b, \sigma\rangle \to \mathbf{false}} \qquad \frac{\langle b, \sigma\rangle \to \mathbf{false}}{\langle \neg b, \sigma\rangle \to \mathbf{true}}$$

$$\frac{\langle b_0, \sigma\rangle \to t_0 \quad \langle b_1, \sigma\rangle \to t_1 \quad \text{if } t \text{ is } \mathbf{true} \text{ iff } t_0 \equiv t_1 \equiv \mathbf{true}}{\langle b_0 \wedge b_1, \sigma\rangle \to t}$$

$$\frac{\langle b_0, \sigma\rangle \to t_0 \quad \langle b_1, \sigma\rangle \to t_1 \quad \text{if } t \text{ is } \mathbf{false} \text{ iff } t_0 \equiv t_1 \equiv \mathbf{false}}{\langle b_0 \vee b_1, \sigma\rangle \to t}$$

例：简单的语言 IMP

命令的计算

Atomic commands

$$\langle \mathbf{skip}, \sigma \rangle \to \sigma \qquad \frac{\langle a, \sigma \rangle \to m}{\langle X := a, \sigma \rangle \to \sigma[m/X]}$$

Sequencing

$$\frac{\langle c_0, \sigma \rangle \to \sigma'' \qquad \langle c_1, \sigma'' \rangle \to \sigma'}{\langle c_0; c_1, \sigma \rangle \to \sigma'}$$

Conditionals

$$\frac{\langle b, \sigma \rangle \to \mathbf{true} \qquad \langle c_0, \sigma \rangle \to \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \to \sigma'} \qquad \frac{\langle b, \sigma \rangle \to \mathbf{false} \qquad \langle c_1, \sigma \rangle \to \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \to \sigma'}$$

While-loops

$$\frac{\langle b, \sigma \rangle \to \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \to \sigma}$$

$$\frac{\langle b, \sigma \rangle \to \mathbf{true} \qquad \langle c, \sigma \rangle \to \sigma'' \qquad \langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'' \rangle \to \sigma'}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \to \sigma'}$$

练习

To see the semantics just defined is a big step semantics, consider the following program:

$$
\begin{aligned}
Factorial \quad \equiv \quad & Y := 1; \\
& \textbf{while } X > 1 \textbf{ do} \\
& \quad \{Y := Y \times X;\ X := X - 1\}; \\
& Z := Y
\end{aligned}
$$

Let σ be a state with $\sigma(X) = 3$, what's the state σ' such that $\langle Factorial, \sigma\rangle \to \sigma'$? Construct the derivation tree.

指称语义

- 指称语义的定义包括
  1. 语言的抽象语法

$$
\begin{aligned}
N ::= &\ 0 \\
&|\ S(N) \\
&|\ (N + N) \\
&|\ (N \times N)
\end{aligned}
$$

  2. 数学模型

Nat (the natural numbers (0, 1, ...)

$+ : Nat \rightarrow Nat \rightarrow Nat$

$* : Nat \rightarrow Nat \rightarrow Nat$

指称语义

- 指称语义的定义包括
  3. 将语言的抽象语法映射到数学模型

$$D : N \rightarrow Nat$$

$$D[0] = 0$$
$$D[S(n)] = D[n] + 1$$
$$D[(\ n + n\ )] = D[m] + D[n]$$
$$D[(\ m \times n)] = D[m] * D[n]$$

where m,n in N

IMP的指称语义（1/3）

Define the semantic function $\mathcal{A}$: Cmd $\rightarrow (\Sigma \rightarrow \mathbf{N})$

$$
\begin{aligned}
\mathcal{A}[\![n]\!] &= \{(\sigma, n) \mid \sigma \in \Sigma\} \\
\mathcal{A}[\![X]\!] &= \{(\sigma, \sigma(X)) \mid \sigma \in \Sigma\} \\
\mathcal{A}[\![a_0 + a_1]\!] &= \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\} \\
\mathcal{A}[\![a_0 - a_1]\!] &= \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\} \\
\mathcal{A}[\![a_0 \times a_1]\!] &= \{(\sigma, n_0 \times n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\}
\end{aligned}
$$

The "+" on the left-hand side represents syntactic sign in **IMP** whereas the sign on the right represents sum on numbers. Similarly for "-", "$\times$".

北京大学 PEKING UNIVERSITY

IMP的指称语义（2/3）

Define the semantic function $\mathcal{B} : \mathbf{Bexp} \to (\Sigma \to \mathbf{T})$

$$
\begin{aligned}
\mathcal{B}[\![\mathbf{true}]\!] &= \{(\sigma, \mathbf{true}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![\mathbf{false}]\!] &= \{(\sigma, \mathbf{false}) \mid \sigma \in \Sigma\} \\
\mathcal{B}[\![a_0 = a_1]\!] &= \{(\sigma, \mathbf{true}) \mid \sigma \in \Sigma \ \ \& \ \ \mathcal{A}[\![a_0]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma\} \cup \\
&\quad\ \{(\sigma, \mathbf{false}) \mid \sigma \in \Sigma \ \ \& \ \ \mathcal{A}[\![a_0]\!]\sigma \neq \mathcal{A}[\![a_1]\!]\sigma\} \cup \\
\mathcal{B}[\![\neg b]\!] &= \{(\sigma, \neg_T t) \mid \sigma \in \Sigma \ \ \& \ \ (\sigma, t) \in \mathcal{B}[\![b]\!]\} \\
\mathcal{B}[\![b_0 \wedge b_1]\!] &= \{(\sigma, t_0 \wedge_T t_1) \mid \sigma \in \Sigma \ \ \& \ \ (\sigma, t_0) \in \mathcal{B}[\![b_0]\!] \ \ \& \ \ (\sigma, t_1) \in \mathcal{B}[\![b_1]\!]\} \\
&\cdots
\end{aligned}
$$

The sign "$\wedge_T$" is the conjunction operation on truth values.

IMP的指称语义（3/3）

Define the compositional semantic function $\mathcal{C} : \mathbf{Aexp} \to (\Sigma \to \Sigma)$

$$
\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!] &= \{(\sigma,\sigma) \mid \sigma \in \Sigma\} \\
\mathcal{C}[\![X := a]\!] &= \{(\sigma,\sigma[n/X]) \mid \sigma \in \Sigma \ \ \& \ \ n = \mathcal{A}[\![a]\!]\sigma\} \\
\mathcal{C}[\![c_0; c_1]\!] &= \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!] \\
\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!] &= \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \ \ \& \ \ (\sigma,\sigma') \in \mathcal{C}[\![c_0]\!]\} \cup \\
&\quad\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false} \ \ \& \ \ (\sigma,\sigma') \in \mathcal{C}[\![c_1]\!]\} \\
\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!] &= fix(\Gamma)
\end{aligned}
$$

where

$$
\begin{aligned}
\Gamma(\varphi) = \ & \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{true} \ \ \& \ \ (\sigma,\sigma') \in \varphi \circ \mathcal{C}[\![c]\!]\} \cup \\
& \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!]\sigma = \mathbf{false}\}
\end{aligned}
$$

公理化语义

- 公理化语义的定义包括：
  - 抽象文法

```
C ::= skip
    |  x := E
    |  C1 ; C2
    |  if B then C1 else C2 endif
    |  while B do C done
```

  - 公理
  - 推理规则

Hoare Triple Form

- {P} C {Q}
  - C：命令
  - P：前置条件
  - Q：后置条件

如果C在满足P的状态下开始执行，而且停止，
那么C一定停止于满足Q的状态。

公理

- 空操作公理

$$\frac{}{\{P\}\ \textbf{skip}\ \{P\}}$$

- 代入公理

$$\frac{}{\{P[E/x]\}\ x := E\ \{P\}}$$

例：

$$\{x+1=43\}\ y := x+1\ \{y=43\}$$

$$\{x+1\leq N\}\ x := x+1\ \{x\leq N\}$$

推理规则

$$\frac{\{P\}\ S\ \{Q\}\ ,\ \ \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$

$$\frac{\{B \wedge P\}\ S\ \{Q\}\ ,\ \ \{\neg B \wedge P\}\ T\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{else}\ T\ \textbf{endif}\ \{Q\}}$$

$$\frac{\{P \wedge B\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\neg B \wedge P\}}$$

$$\frac{P' \rightarrow\ P\ ,\ \{P\}\ S\ \{Q\}\ ,\ Q \rightarrow\ Q'}{\{P'\ \}\ S\ \{Q'\}}$$

证明例

{ x=x0 and y=y0 }

t := x;

x := y;

y := t

{ x=y0 and y=x0 }

{ x=x0 and y=y0 }

t := x;

x := y;

{ x=y0 and t=x0 }

y := t

{ x=y0 and y=x0 }

小结

- 为什么要研究程序设计语言？
- 程序设计语言的基本概念
  - 程序设计语言与计算
  - 高级/低级程序设计语言
  - 编程范式
  - 语言的实现
- 语言的定义
  - 文法
  - 语义（操作语义，指称语义，公理化语义）

随堂测试

程序语言随堂测试

微信扫码或长按识别，填写内容