Theory of Programming Languages
程序设计语言理论

张昱

School of Computer Science and Technology
University of Science and Technology of China

September, 2009

课 程 简 介

- 计算机科学的理论
- 关于程序设计语言的研究
- 学习本课程的意义
- 参考书目及资源
- 课程要求

计算机科学的理论

- 数理逻辑
  - 一阶逻辑、高阶逻辑、公理集合论、递归论、模型论和证明论等，它也是现代数学的基础
- 计算理论
  - 可计算性和计算复杂性、算法、形式语言理论、自动机理论等
- 程序理论
  - 形式语义、类型论、λ演算、程序验证等等
- 数值计算

围绕程序设计语言的研究

- 语法

  形式语言和自动机理论,语法分析的实现技术

- 语义

  公理语义、操作语义、指称语义、属性文法

  形式描述技术还有：代数规范、范畴论

- 程序设计的范型

  命令式语言、函数式语言、逻辑程序设计语言、

  面向对象程序设计语言、并行程序设计语言

关于程序设计语言的研究

- 类型论与类型系统
  - 多态类型、子类型、存在类型、依赖类型等
- 程序验证
  - 程序正确性证明
- 程序分析技术
  - 数据流分析、控制流分析、模型检查、抽象解释
- 程序的自动生成技术
  - 程序变换

学习本课程的意义

- 学习掌握和程序设计语言有关的理论和技术
  - 形式语义学（操作语义、公理语义、指称语义、……）
    - 以数学为工具，利用符号和公式，精确地定义和解释计算机程序设计语言的语义，使语义形式化的学科.
  - 与程序行为和程序分析有关的推理技术
  - 一些语言及特征的案例研究

目标：寻找精确、抽象地描述程序行为的方法

- 精确(precise)：用数学工具来形式化和证明所感兴趣的性质
- 抽象(abstract)：清晰地讨论性质，又不陷入底层细节

学习本课程的意义

- 讨论程序理论和技术的实际应用
  - 深入理解语言的特征和它们的交互作用
  - 指导程序设计语言的设计和实现
  - 程序的自动生成
  - 程序分析
  - 软件安全
  - 程序验证

参考书目及资源

❖ 参考书

- 陈意云. 程序设计语言理论（2009年改版的书稿）.
  从 http://staff.ustc.edu.cn/~yiyun/下载
- Robert Harper. Practical Foundations for Programming Languages. Draft, 2008. (Operational Semantics)
- Benjamin C.Pierce. Types and Programming Languages. 2002. (Type theory)
  马世龙等译. 类型和程序设计语言.电子工业出版社，2005.
- John C. Mitchell. Foundations for Programming Languages. MIT Press, 1996. (Semantics,…)
  许满武等译. 程序设计语言理论基础.电子工业出版社，2006.

参考书目及资源

- 课程资源
  - UPENN
    - B.C.Pierce, CIS500: Software Foundations
  - UC Berkeley
    - George Necula, CS263:Design and Analysis of Programming Languages
    - http://www.cs.berkeley.edu/~adamc/itp/
  - Princeton
    - David Walker, COS441: Programming Languages

  ……

课程要求

- 课程主页
  - http://staff.ustc.edu.cn/~yuzhang/tpl
- 课程考核
  - 作业
  - 调研报告
  - 考试

第1章　引言

1.1　基本概念

1.2　等式、归约和语义

1.3　类型和类型系统

1.4　语言设计

1.5　归纳定义

1.1 基本概念—程序设计语言的建模

- 对程序设计语言进行数学分析：从语言的建模开始
  - 模型语言的设计：突出感兴趣的程序构造，忽略一些无关的细节
- 程序设计语言形式化为两部分
  - 能抓住该语言本质机制的一个非常小的核心演算：
    λ演算(lambda calculus) (1930s, Alonzo Church & Stephen Cole Kleene)
    1）源于可计算理论，奠定语言中函数定义和命名约定的基本机制
    2）既可看成一种简单的语言(用于描述计算)，又可看成一种数学对象（可证明）
  - 一组导出部分：通过把它们翻译成 λ 演算来理解
- 用类型化λ演算(typed lambda calculus)的框架来研究程序设计语言的各种概念

1.1 基本概念- λ表示法-1

- λ表示法的主要特征
  - λ抽象(abstraction): 用于定义函数
  - λ应用(application): 使用所定义的函数

  用λ表示法写出的表达式叫做 λ 表达式或 λ 项
- 举例

  类型化 λ 演算（自然数类型上的几个例子）
  - 恒等函数: λx:nat.x （Id(x:nat) = x）
    无须给函数命名
  - 后继函数: λx:nat.x+1
  - 常函数: λx:nat.10

  无类型 λ 演算 λx.x

1.1 基本概念- λ表示法-2

- λ项$\lambda x{:}\sigma.\ M$ 和谓词演算公式$\forall x{:}A.\ \phi$
  - λ是一个约束算子，x是一个占位符，称为约束变元，可以重新命名 λ 约束变元而不改变表达式的含义
  - 在$\lambda x{:}\sigma.\ x+y$中，x的出现是约束的，y的出现是自由的
  - 不含自由变元的表达式称为闭表达式,或闭项
- 用项的并置来表示函数应用
  - $(\lambda x{:}nat.\ x)\ 5$
  - $(\lambda x{:}nat.\ x)\ 5 = 5$

1.1 基本概念- λ表示法-3

- λ表示法中有两个约定
  - 函数应用是左结合的：MNP 应看成 (MN)P
  - 每个λ的约束范围尽可能地大，一直到表达式的结束或碰到不能配对的右括号为止
- 一个例子

$$
\begin{aligned}
&(\lambda x.(\lambda y.\lambda z.(x+y)+z)3)4\ \ 5\\
=\ &(\lambda x.\lambda z.(x+3)+z\ )4\ \ 5\\
=\ &(\lambda z.(4+3)+z\ )\ \ 5\\
=\ &(4+3)+5\\
=\ &12
\end{aligned}
$$

1.1 基本概念- λ表示法-4

- 例
  - λx:σ. MN 解释为λx:σ. (MN)，而不是(λx:σ. M)N
  - λx:σ. λy:τ. MN 是 λx:σ. (λy:τ. (MN)) 的简写
  - (((λx:σ. (λy:τ. (λz:ρ. M)))N)P)Q 可以简写为
    (λx:σ. λy:τ. λz:ρ. M)NPQ

1.2 等式、归约和语义

- 形式语义学(Formal Semantics)
  - 以数学为工具，利用符号和公式，精确地定义和解释计算机程序设计语言的语义，使语义形式化的学科.

1.2.1 公理语义(Axiomatic Semantics)

推导表达式之间等式的形式系统

1.2.2 操作语义(Operational Semantics)

证明系统

将等式确定为有向规则的推理，称为归约（符号求值）

1.2.3 指称语义(Denotational Semantics)

称为模型。一个模型是一组集合，每种类型一个集合，每个良类型的表达式可解释为相应集合中的一个元素

1.2.1 公理语义

- 1969,Hoare, An Axiomatic Basis for Computer Programming
  - 语言的数学理论，提供证明程序性质的逻辑基础
    - 语法规则：确定什么是合式公式(well-formed fomula)
    - 公理：是不加证明地被接受的基本定理
    - 推理规则：从已确定的定理演绎新定理的机理
    - 基本的逻辑系统，如，带有等式的一阶谓词演算
  - 对证明程序正确性有用
  - 示例
    - 整数运算
    - 程序执行

1.2.1 公理语义-整数运算

❖整数运算公理

$$
\begin{array}{ll}
A1 & x + y = y + x \\
A2 & x \times y = y \times x \\
A3 & (x + y) + z = x + (y + z) \\
A4 & (x \times y) \times z = x \times (y \times z) \\
A5 & x \times (y + z) = x \times y + x \times z \\
A6 & y \leq x \supset (x - y) + y = x \\
A7 & x + 0 = x \\
A8 & x \times 0 = 0 \\
A9 & x \times 1 = x \\
 & \ldots\ldots
\end{array}
$$

❖演绎

定理

$$y \leq r \supset r + y \times q = (r - y) + y \times (1 + q)$$

证明

$$
\begin{array}{lll}
 & (r - y) + y \times (1 + q) & \\
= & (r - y) + (y \times 1 + y \times q) & (A5) \\
= & (r - y) + (y + y \times q) & (A9) \\
= & ((r - y) + y) + y \times q) & (A3) \\
= & r + y \times q \ \text{ provided } y \leq r & (A6)
\end{array}
$$

1.2.1 公理语义-程序执行-1

- 公式：P {Q} R （{P} Q {R}） P和R都是一阶公式

  如果前条件(断言)P在程序Q执行前的状态成立，则执行Q后将得到满足后条件(断言)R的状态。

  部分正确性断言：如果P在Q执行前为真，那么，如果Q的执行终止，则终止在使R为真的某个状态。

  终止性断言：如果P在Q执行前为真，那么Q将终止在使R为真的某个状态。

- 推理规则的表示

$$\frac{f_0, f_1, \ldots f_n}{f_0}$$

前提(premise)

推论(conclusion)

1.2.1 公理语义-程序执行-2

- 赋值公理 $\vdash P_0\{x := f\}P$

  其中x是变量，f是表达式，P_0可以通过用f代换P中的每一个x而得到

$$\vdash y > 8\{x := y + 4\}x > 12$$

- 推理规则
  - D1: Rules of Consequence

$$\frac{P\{Q\}R,\ R \rightarrow S}{P\{Q\}S} \qquad \frac{P\{Q\}R,\ S \rightarrow P}{S\{Q\}R}$$

  - D2: Rule of Composition

$$\frac{P\{Q1\}R1,\ R1\{Q2\}R}{P\{Q1;Q2\}R}$$

1.2.1 公理语义-程序执行-3

- 推理规则
  - D3: Rules of Iteration

$$\frac{P \ \& \ B\{S\} \ P}{P \ \{\text{while } B \text{ do } S\} \ \neg B \ \& \ P}$$

- 例子

```
1 PROCEDURE FACT ( N:INTEGER; VAR Y:INTEGER);
2 VAR X: INTEGER;
3 BEGIN
4 X := 0;
5 Y := 1;
6 ASSERT ( Y = X! & X ≤ N )
7 WHILE X < N DO BEGIN
8 X := X + 1;
9 Y := Y * X
10 END
11 END;
ENTRY: N ≥ 0
EXIT: Y = N!
```

1.2.1 公理语义

❖ 一个等式公理系统

- **代换**：[N/x]M 表示M中的自由变元x用N代换的结果
  注意:N中的自由变元不能代换到M中后成为约束变元
- **约束变元改名公理**

$$\lambda x{:}\sigma.\ M = \lambda y{:}\sigma.\ [\,y/x\,]M,\ M\text{中无自由出现的}y \qquad (\alpha)$$

例如， $\lambda x{:}\sigma.x+y = \lambda z{:}\sigma.z+y$

- **等价公理**：计算函数应用就是在函数体中用实在变元代替形式变元

$$(\lambda x{:}\sigma.\ M)\ N = [N/x]\ M \qquad (\beta)_{eq}$$

- **同余性规则**：相等的函数应用于相等的变元产生相等的结果

$$\frac{M_1 = M_2,\ N_1 = N_2}{M_1 N_1 = M_2 N_2}$$

1.2.2 操作语义-1

- 1964, P.J.Landin,The mechanical evaluation of expressions
  - SECD (Stack, Environment, Code, Dump) machine
    - 定义一个抽象机,给出在抽象机上的执行规则
      - 抽象机的大状态 (st, s, c)

        st: 栈区(工作区)

        s: 环境区(数据)

        c: 控制区(程序)
      - 状态转移规则

        例: $(x_1*x_2)+1$的求值

        在x_1,x_2值为2和3时

        符号”/”用于分割存放的信息

$$
\begin{aligned}
&(st,s,((x_1 * x_2)+1)/c)\\
\overset{1}{\Rightarrow}\ &(st,s,(x_1 * x_2)/1/+/c)\\
\overset{1}{\Rightarrow}\ &(st,s,x_1/x_2/*/1/+/c)\\
\overset{3}{\Rightarrow}\ &(2/st,s,x_2/*/1/+/c)\\
\overset{3}{\Rightarrow}\ &(3/2/st,s,*/1/+/c)\\
\overset{4}{\Rightarrow}\ &(6/st,s,1/+/c)\\
\overset{2}{\Rightarrow}\ &(1/6:st,s,+/c)\\
\overset{4}{\Rightarrow}\ &(7/st,s,c)
\end{aligned}
$$

1.2.2 操作语义-2

❖操作语义(Operational Semantics)

- 是演绎出最终结果的证明系统
  或者说是通过一系列步骤变换一个表达式的证明系统
- 由等式公理的单向形式给出了归约规则
  β归约

$$(\lambda x{:}\sigma.\ M)\ N \rightarrow_{\beta} [N/x]\ M \qquad (\beta)_{red}$$

例如， $(\lambda x{:}nat.\ x{+}4)\ 4 \rightarrow 4{+}4$

β归约是定义在 α 等价上的，即β 归约的结果不是唯一确定的，但是归约产生的任何两个项仅在约束变元的名字上有区别。

- 对实现编译器或解释器有用

1.2.3 指称语义

- 源于Christopher Strachey 和 Dana Scott 在1960年代的工作

  又称为不动点语义(fixed-point semantics),Scott-Strachey语义

  - 先确定**指称物**(多为数学对象，如整数、集合、函数)，然后给出语言各种构造到指称物的语义映射，该映射满足
    - 每个语言构造的每个实例都有对应的指称
    - 复合语言构造的实例的指称只依赖于它的子构造的指称
  - 类型化λ演算的指称语义
    - 每个类型表达式对应到一个集合，称为该类型的值集
    - 类型σ 的项解释为其值集上的一个元素
    - 类型$\sigma \to \tau$ 的值集是函数集合，项$\lambda x{:}\sigma.\ M$解释为一个数学函数
  - 无类型化λ演算可以从类型化λ演算中派生

1.3 类型和类型系统-1

❖静态检查和动态检查

- 编译器必须检查源程序是否满足源语言在语法和语义两个方面的约定。这种检查叫做**静态检查**（以区别在目标程序运行时的**动态检查**）。
- 静态检查的例子
  - 类型检查
  - 控制流检查
  - 惟一性检查
  - 关联名字检查

1.3 类型和类型系统-2

❖ 执行错误和安全语言

- 程序运行时出现的错误称为**执行错误(execution error)**
  - **trapped error**:引起计算立即停止
  - **untrapped error**:当时未引起注意，之后可能引发难以预见的行为

  禁止错误(forbidden error): 对任何一种语言，可以指定所有可能执行错误集合的一个子集 (所有不会被捕获的错误，再加上部分会被捕获的错误)
- 一个程序是**良行为的(well behaved)**，如果它在运行时不可能引起不会被捕获的错误。
- 所有合法程序都是良行为的语言叫做**安全语言(safe language)**

1.3 类型和类型系统-3

❖类型论

- 避免集合论悖论而建立起来的数学理论，主要研究集合的分层、分类方法。
- 在程序语言理论中，指类型系统的设计、分析和研究

❖类型和类型系统

- **类型**：具有共同特征的事物所形成的种类，一个类型是一群有某些公共性质的值。
- **类型化语言(typed language)**：变量都被给定类型的语言
- **无类型语言(untyped language)**：语言不限制变量值的范围，即没有类型或者仅有一个包含所有值的泛类型

1.3 类型和类型系统-4

❖类型和类型系统

- **类型系统(type system)**
  - 是语言的一个组成部分
  - 由一组**定型规则(typing rule)**构成：给语言的各种构造（程序、语句、表达式等）指派类型

设计目的：防止程序运行时出现禁止错误，以保证其运行是良行为的

- 若语言的类型是语法的一部分，那么该语言是**显式类型化的(explicit typing)**，否则是**隐式类型化的(implicit typing)**。

1.3 类型和类型系统-5

❖类型和类型系统

- **类型检查(type checking)**：根据定型规则来确定程序中各语法构造的类型
  - 能够通过类型检查的程序称为**良类型的(well typed)**程序，否则是**不良类型的(ill typed)**程序
- 如果良类型程序一定是良行为的，则称该语言是**类型可靠的（type sound）**。
- 类型可靠的语言一定是安全语言。

1.3 类型和类型系统-6

- 类型和类型系统
  - **静态类型化语言(statically typed language)**：数据类型在编译期间检查。
    动态(dynamically)类型化语言：数据类型在运行期间才被检查。
  - 类型系统的研究也需要形式化的方法。
- 两种研究类型系统的重要分支
  - 类型系统在程序设计语言中的应用。
  - 通过**Curry-Howard对应(correspondence)**将多种“纯类型化λ演算”和不同逻辑联系在一起。

1.3 类型和类型系统-7

❖类型化语言的优点

- 开发时的实惠
  - 可以较早发现错误
  - 类型信息具有文档作用（比程序注解精确，比形式规范容易理解）
- 编译时的实惠
  - 程序模块可以相互独立地编译
- 运行时的实惠
  - 更有效的空间安排和访问方式，提高了目标代码的运行效率

1.3 类型和类型系统-8

❖类型系统的其他应用

- 计算机和网络安全
  - 静态类型化：是Java的安全模型的核心，是携带证明代码（PCC，Proof-carrying code）的一个关键授权技术
  - ……
- 许多程序分析工具使用类型检查或类型推断算法
- 类型系统用来表示逻辑命题和证明
  - 基于类型理论的证明辅助器，如 coq等
- 基于DTD，XML Schema的查询和处理XML的新语言
- ……

1.4 语言设计

- 一个好语言所具有的特征
  - 语法(syntax)和语义(semantics)简单(simplicity)
  - 可读性(readability)高
  - 安全性(safety)高
  - 支持对大型系统的编程
  - 执行和编译高效(efficiency)

C.A.R. Hoare. Hints on programming language design. Stanford University. Technical Report: CS-TR-73-403. 1973.

1.5 归纳法(Induction)

归纳证明在程序设计语言理论中是普适的证明方法。

归纳法是一种用有限的方式写出一个无限的证明的方法。

1.5.1 自然数归纳法(Natural Number Induction)

1.5.2 结构归纳法(Structural Induction)

1.5.3 良基归纳法(Well-formed Induction)

1.5.1 自然数归纳法-1

❖ 自然数上的一般归纳原理（形式1）

假设P是自然数上的一个性质，则如果：

- P(0)成立 --基本情形base case
- 且对所有的自然数k，P(k)蕴涵P(k+1)

--归纳步骤induction step

则P(n)对所有自然数n成立.

自然演绎法(natural deduction)中的规则记法

$$\frac{[P(k)] \quad P(0) \quad P(k+1)}{P(n)}$$

归纳假设
induction hypothesis

k不能出现在P(k+1)的任何假设中

1.5.1 自然数归纳法-2

- 自然数上的一般归纳原理（形式1）

定理1.1：每个自然数要么是偶数，要么是奇数.

证明：要归纳的性质是： 自然数n是偶数或n是奇数，

下面对n进行归纳来证明。

基本情形：0是偶数 ➔ 0是偶数或奇数

归纳步骤：假设k是偶数或奇数，证明k+1是偶数或奇数

1) 如果k是偶数，则k+1是奇数

2) 如果k是奇数，则k+1是偶数

得证。

1.5.1 自然数归纳法-3

❖ 自然数上的完全归纳原理（形式2）

假设P是自然数上的一个性质，则：

如果对每个自然数k，假定P(i)对所有的自然数i (i<k) 成立，我们能证明P(k)成立，

则P(n)对所有n成立.

$$\frac{[\forall i < k.P(i)] \quad P(k)}{P(n)}$$

k不能出现在前提条件的其他假设中

1.5.1 自然数归纳法-4

- 自然数上的完全归纳原理（形式2）

定理1.2：所有 $n \geq 2$ 的自然数都可以写成素数的乘积

$$n = p_1 \ldots p_k$$

证明：对n进行完全归纳来证明。

1) 如果n是素数，则结果是显然的，且k=1.
2) 如果n不是素数，则它可以被某个满足1<m<n的自然数m整除。由于m<n且n/m<n，我们可以使用两次完全归纳法的归纳假设，把这两个数写成素数的乘积

$$m = p_1 \ldots p_k \quad \text{且} \quad n/m = q_1 \ldots q_l$$

从而 $n = m \times (n/m) = p_1 \ldots p_k \; q_1 \ldots q_l$

得证。

1.5.1 自然数归纳法-5

❖字典序归纳原理

假设P是自然数序对上的一个性质，则：

如果对每个自然数序对(m,n)，

假定P(m',n')对所有的(m',n') < (m, n)成立，

我们能证明P(m,n)成立，

则P(m,n)对所有的m, n成立.

1.5.2 结构归纳法

结构归纳法是自然数归纳法在其他数据类型上的推广。

❖ 结构归纳(形式1)

假设P是某个文法产生的任意表达式e上的一个性质，则

基本情形：对每个原子表达式e，证明P(e)为真

归纳步骤：对直接子表达式为$e_1,\ldots,e_k$的任何复合表达式e，证明如果$P(e_i)$ (i=1,…,k)都为真，则P(e)也为真。

❖ 结构归纳(形式2)

假设P是某个文法产生的任意表达式e上的一个性质，则

基本情形：对每个原子表达式e，证明P(e)为真

归纳步骤：对任何表达式e的任何子表达式e'，证明如果P(e')都为真，则P(e)也为真。

1.5.2 结构归纳法-表-1

❖关于表的结构归纳法

假设P是元素类型为τ的表(类型为τ list)上的一个性质,则

基本情形: P([ ]), [ ]表示空表.

归纳步骤: 对类型为τ的所有元素y 以及类型为τ list的表ys都有P(ys)蕴涵P(y::ys)

则P(xs)对所有类型为τ list的表xs成立.

$$\frac{P([]) \quad \begin{matrix}[P(ys)] \\ P(y :: ys)\end{matrix}}{P(xs)}$$

y和ys不能出现在P(y::ys)的其他假设中

这一规则也可以通过对表的长度应用数学归纳法来证明.

1.5.2 结构归纳法-表-2

❖ 关于表的结构归纳法

定理1.3：所有的表都不等于自己的表尾　　$\forall x.x :: xs \neq xs$

证明：对表**xs**应用结构归纳

基本情形：$\forall x.[x] \neq [\,]$，根据表相等的定义显然成立.两个表是相等的当且仅当它们具有相同的长度且对应的元素也相等。

归纳步骤：假定归纳假设 $\forall x.x :: ys \neq ys$ 成立，并对于任意的**y**和**ys**证明 $\forall x.x :: (y :: ys) \neq y :: ys$

根据表相等的定义，只要证明两边的表尾不等就够了：也就是证明 $y :: ys \neq ys$ 。这就是归纳假设，只要将其中受全称量词约束的**x**换成**y**即可。

这个定理并不适用于无穷表，因为**[1,1,...]**等于它自己的表尾。

1.5.2 结构归纳法-表-3

❖ 关于表的结构归纳法

定理1.4：对于所有的表**xs**和**ys**，有 $len(xs@ys) = len\ xs + len\ ys$

(@表示连接运算)

证明：对表**xs**应用结构归纳

基本情形：$len([\,]@ys) = len[\,] + len\ ys$是成立的.因为

$$\begin{array}{lcll} len([\,]@ys) & = & len\ ys & [@] \\ & = & 0 + len\ ys & [arith] \\ & = & len[\,] + len\ ys & [len] \end{array}$$

归纳步骤：假定$len(xs@ys) = len\ xs + len\ ys$

证明对于所有**x**和**xs**,有 $len((x :: xs)@ys) = len(x :: xs) + len\ ys$

这是成立的，因为

$$\begin{array}{lcll} len((x :: xs)@ys) & = & len(x :: (xs@ys)) & [@] \\ & = & 1 + len\ (xs@ys) & [len] \\ & = & 1 + (len\ xs + len\ ys) & [inductive\ hypothesis] \\ & = & (1 + len\ xs) + len\ ys & [associative] \\ & = & len(x :: xs) + len\ ys) & [len] \end{array}$$

1.5.2 结构归纳法-树

❖关于树的结构归纳法

假设P是类型为 τ tree 的树上的一个性质，则：

- 基本情形：P(empty)
- 归纳步骤：对于所有类型为 τ 的元素x 以及类型为 τ tree的树t_1和t_2，都有

  $P(t_1)$ 和$P(t_2)$ 蕴涵 $P(node(x,t_1,t_2))$.

$$\frac{P(\text{empty}) \quad [P(t_1), P(t_2)]}{P(node(x,t_1,t_2))}$$
$$P(t)$$

x、t_1和t_2不能出现在$P(node(x, t_1, t_2))$的其他假设中

结构归纳法很适合结构递归(structual recursion)函数.

1.5.2 结构归纳法-证明上的归纳-1

❖证明结构上的归纳

本质上与表达式结构上的归纳一样

回顾：Hilbert风格的证明系统

- **公理**：是一个公式，它被定义成可证明的。
- **推理规则**：若某一组公式可证，则另一个公式也可证。

$\frac{A_1 \ldots A_n}{B}$ 如果$A_1,\ldots,A_n$ (前提)都可证，那么B (结论)也可证。

- **证明**：是一个结构化的对象，它由公式来构造，这些公式遵循由一组公理和推理规则确定的约束。

$\frac{A_1 \ldots A_n}{B}$ 把公式$A_1,\ldots,A_n$的证明合起来就形成公式B的证明.

1.5.2 结构归纳法-证明上的归纳-2

❖ 证明结构上的归纳

本质上与表达式结构上的归纳一样

回顾：Hilbert风格的证明系统

➢ 例如，

相等性的自反公理　e=e　(ref)

相等性的传递规则

$$\frac{e_1 = e_2 \quad e_2 = e_3}{e_1 = e_3} \quad \text{(trans)}$$

假设有1个2 + 6 = 8的证明，以及1个$8 = 2^3$的证明，则可以组合这2个证明得到对等式$2 + 6 = 2^3$的证明。

1.5.2 结构归纳法-证明上的归纳-3

- 形式地，一个证明可以定义为一个公式序列，该序列中的每个公式都是公理或者是由先前的公式通过一条推理规则得到的结论。
- 用自然数归纳法讨论证明的性质

  基于上述对证明的定义，可以对证明长度（即公式序列的长度）进行自然数归纳来讨论证明的性质。
- 用树的结构归纳法讨论证明的性质

  把证明看成树，证明所用公理看成叶节点，所用的推理规则看成树的内部节点

  由对公式$A_1, \ldots, A_n$的证明来构造对公式 B的证明。

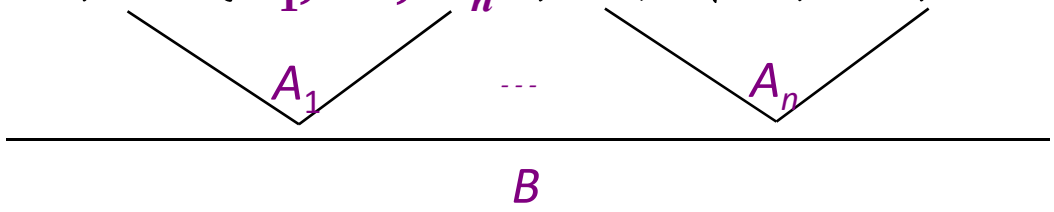

图1.1 证明树示意图

1.5.2 结构归纳法-证明上的归纳-3

在证明上的结构归纳：在某个证明系统中，为了证明对每个证明π，P(π)为真，只要完成下面两步就足够了：

- 基本情形：对该证明系统中的每个公理，证明P成立
- 归纳情形：假定对证明$\pi_1,\ldots,\pi_k$，P都成立，证明P(π)也成立。π是结束于使用一个推理规则，并且是从证明$\pi_1,\ldots,\pi_k$延伸出来的一个证明。

例1.1 表达式小于等于关系的一个简单证明系统

$e ::= 0 \mid 1 \mid v \mid e + e \mid e * e$

公理： $e \le e$ *(ref)*　　$0 \le e$ *(0 ≤)*

规则：

$$\frac{e \le e' \quad e' \le e''}{e \le e''}\ (trans) \qquad \frac{e_1 \le e_2 \quad e_3 \le e_4}{e_1 + e_3 \le e_2 + e_4}\ (+mon) \qquad \frac{e_1 \le e_2 \quad e_3 \le e_4}{e_1 * e_3 \le e_2 * e_4}\ (*mon)$$

$\frac{e \le e' \quad e' \le e''}{e \le e''}$

1.5.2 结构归纳法-证明上的归纳-4

证明系统的可靠性(*soundness*)：在公式的某种特定解释下，每个可证的公式都为真。

例1.1 表达式小于等于关系的一个简单证明系统

证明系统对≤可靠，即证明下面的性质对任何证明π都成立

P(π)≙ 如果π是$e \le e'$的一个证明，那么e的值≤ e'的值，不管其中的变量怎样取值。

证明：基础情形. 为所有的公理证明该性质。不管e中的变量怎样取值，e都是解释到某个自然数n，因此总有$n \le n$和$0 \le n$。

归纳情形. 假定可以证明$e_1 \le e_2$和$e_3 \le e_4$。任意选择$e_1, \ldots, e_4$中变量的值，并把这4个表达式的值叫做$n_1, \ldots, n_4$。由归纳假设，有$n_1 \le n_2$和$n_3 \le n_4$。很容易看出$n_1 + n_3 \le n_2 + n_4$，同样有$n_1 \times n_3 \le n_2 \times n_4$。该推理可用于变量所有可能的取值，因此该性质对终止于$(+mon)$和$(\times mon)$的证明都成立。……

1.5.3 良基归纳法-1

- 良基关系(well-founded relation)
  - 集合A的良基关系是A上的一个二元关系$\prec$，它具有这样的性质：A上不存在无穷递减序列 $\ldots \prec a_2 \prec a_1 \prec a_0$.
- 示例
  - 自然数上的“小于”关系是良基的
  - 整数上的“小于”关系不是良基的
- 良基关系的一些特点
  - 良基关系不一定有传递性

    例如，在自然数上，如果$j=i+1$，则$i \prec j$，该关系是良基的，但不具有传递性。
  - 良基关系都是非自反的，即对任何$a \in A$，$a \prec a$不成立；否则会出现无穷递减序列 $\ldots \prec a \prec a \prec a$.

1.5.3 良基归纳法-2

引理1.1 如果$\prec$是集合A上的二元关系，那么$\prec$是良基的当且仅当A的每个非空子集都有一个极小元.

证明 令$B \subseteq A$是任意非空子集。用反证法证明B有极小元

- 如果B无极小元，那么对每个$a \in B$，可以找到某个$a' \in B$使得$a' \prec a$。这样，可以从任意的$a_0 \in B$开始，构造一个无穷递减序列 $\ldots \prec a_2 \prec a_1 \prec a_0$
- 反过来，假定每个子集有一个极小元，那么不可能存在 $\ldots \prec a_2 \prec a_1 \prec a_0$，因为该序列给出了无极小元的集合$\{a_0, a_1, a_2, \ldots\}$

1.5.3 良基归纳法-3

❖ **良基归纳** 令$\prec$是集合A上的一个良基关系，令P是A上的某个性质。如果对所有的$b \prec a$ $(a,b \in A)$，有P(b)蕴涵P(a)

$(\forall a.(\forall b.(b \prec a \supset P(b)) \supset P(a))$，那么，对所有的$a \in A$,P(a)为真.

$$\frac{[\forall b \prec a.P(b)]}{P(a)} \qquad P(x)$$

a不能出现在前提条件的其他假设中

证明(反证法) 假定$\forall a.(\forall b.(b \prec a \supset P(b)) \supset P(a))$

如果存在某个$x \in A$使得P(x)不成立，那么集合

$$B = \{a \in A | \neg P(a)\}$$

非空。

由引理1.1，B一定有极小元$a \in B$。但是，对所有的$b \prec a$，P(b)一定成立（否则$b \in A$，从而a不是B的极小元），这就和假定$\forall a.(\forall b.(b \prec a \supset P(b)) \supset P(a))$矛盾。

1.5.3 良基归纳法-4

❖ 良基归纳 令$\prec$是集合A上的一个良基关系，令P是A上的某个性质。如果对所有的$b \prec a$ $(a,b \in A)$，有P(b)蕴涵P(a)

$(\forall a.(\forall b.(b \prec a \supset P(b)) \supset P(a))$，那么，对所有的$a \in A$,P(a)为真.

如何理解 $\forall a.(\forall b.(b \prec a \supset P(b)) \supset P(a))$?

- 对某些a, 不存在b，使得$b \prec a$，则

  $\forall b.(b \prec a \supset P(b)) \supset P(a)$等价于 P(a)

  （因为$\forall b{:}\Phi.P(b)$为真，其中Φ表示空集）

- 对另一些a,存在b，使得$b \prec a$，则

  $\forall b.(b \prec a \supset P(b)) \supset P(a)$的证明是从$\forall b.(b \prec a \supset P(b))$为真来推导P(a)为真.

1.5.3 良基归纳法-5

- **良基递归** 令$\prec$是类型τ上的一个良基关系。如果**f**是以**x**为形参的函数，并且**f**仅当**y** $\prec$ **x**时才递归调用**f(y)**，那么**f(x)**对于所有**x**都是可以终止的。

  f是通过在 $\prec$上的**良基递归(well-founded recursion)**来定义的.

  f(x)能够终止是因为$\prec$没有无穷递减序列。

作业

程序设计语言理论(修改版)，2009.

1.3　1.5　1.6

Thanks!