形式语言与编译：课件定义整理

0. 编译与课程基础

出处：XJTU 第0章课件、抽取文本、XJTU 总复习

编译程序

编译程序是一个程序，它以某种源语言写成的源程序为输入，将其翻译成功能等价的目标语言程序，并在翻译过程中报告错误信息。

源语言、目标语言、实现语言

- 源语言：源程序所使用的语言。
- 目标语言：编译结果所使用的语言，可以是汇编、机器语言，也可以是另一种高级语言。
- 实现语言：编译器自身所使用的编程语言。

源程序、目标程序

- 源程序：作为编译器输入的程序。
- 目标程序：编译器输出的、与源程序功能等价的程序。

前端与后端

- 前端：编译器中与目标机无关、面向源语言的部分，通常包括词法分析、语法分析、语义分析和中间代码生成。
- 后端：编译器中与目标机有关的部分，通常包括目标代码生成、运行时环境处理和部分优化。

中间表示

中间表示是源程序经过前端分析后得到的、介于源语言和目标语言之间的程序表示形式。课件中常见形式包括三地址代码、四元式、抽象语法树和 DAG。

编译过程

编译过程通常包括词法分析、语法分析、语义分析、中间代码生成、代码优化和代码生成等阶段。

词法分析

词法分析是按照语言的词法规则扫描源程序字符流，识别单词，并把它们表示成内部形式 `TOKEN` 流的过程。

语法分析

语法分析是依据语言的语法规则，把 `TOKEN` 序列组织为语法单位及其结构关系，并判断输入是否构成语法正确程序的过程。

语义分析

语义分析是对语法分析识别出的语法范畴分析其含义、检查语义合法性，并为中间代码生成准备信息的过程。

语义性质

语义性质是程序语言规格中关于含义和合法性的约束，例如变量必须先声明后使用、类型必须匹配、函数调用参数必须符合声明等。

程序合法性与正确性

程序合法性指程序满足词法、语法、语义规则；程序正确性指程序满足用户期望或规格说明。合法程序不一定正确。

1. 形式语言基本概念

出处：XJTU 第1章课件、抽取文本、XJTU 总复习

符号

符号是用于表示对象的基本记号，可以是字母、数字、特殊字符或它们在某一形式系统中的抽象标记。

字母表

字母表是符号的有穷集合，常记为 Σ。

符号串

符号串是由某个字母表中的符号组成的有限长序列。

空串

长度为 0 的符号串称为空串，记为 ε。

串长

串 w 的长度是其中符号的个数，记为 $|w|$。特别地，$|\varepsilon| = 0$。

字母表闭包

Σ^* 表示字母表 Σ 上所有有限长符号串的集合，包括空串 ε。

语言

字母表 Σ 上的语言是 Σ^* 的一个子集，即由 Σ 上若干有限长符号串组成的集合。

判定性问题

判定性问题是答案为“是”或“否”的问题。形式语言中常把问题转化为成员关系判定：给定 w，判断 $w \in L$ 是否成立。

形式语言

形式语言是某个字母表上有限长符号串的集合。

形式文法

形式文法是描述形式语言的一种机制：从开始符号出发，通过不断应用产生式规则生成符号串集合。

自动机

自动机是抽象机器模型，通过状态迁移消耗输入串，并根据结束时状态或配置判断输入串是否被接受。

识别器

识别器是用于判定输入串是否属于某语言的自动机或算法。

Chomsky 体系

Chomsky 体系是按照产生式受限程度划分文法和语言表达能力的分类体系：

- 0 型文法：短语结构文法，对应递归可枚举语言和图灵机。
- 1 型文法：上下文有关文法。
- 2 型文法：上下文无关文法，对应上下文无关语言和 PDA。
- 3 型文法：正则文法，对应正则语言和 DFA/NFA。

2. 正则语言、DFA、NFA 与 RE

出处：XJTU 第2章 DFA 课件、DFA 抽取文本、XJTU 第2章 NFA 课件、NFA 抽取文本、XJTU 第3章 RE 课件、RE 抽取文本

有穷自动机

有穷自动机是具有有限状态集合的自动机，通过读取输入符号并发生状态迁移来识别语言。

DFA

确定型有穷自动机 `DFA A` 是五元组：

```
A = (Q, Σ, υ, q0, F)
```

其中 `Q` 是有穷状态集，`Σ` 是输入字母表，`υ: Q × Σ -> Q` 是状态转移函数，`q0 ∈ Q` 是初始状态，`F ⊆ Q` 是接受状态集。

完全 DFA

完全 DFA 是转移函数对任意 `q ∈ Q` 和 `a ∈ Σ` 都有定义的 DFA。

简化 DFA

简化 DFA 允许某些转移无定义；课件中可用 `⊥` 表示转移无定义或自动机突然死亡。

DFA 接受串

DFA `A=(Q,Σ,υ,q0,F)` 接受串 `w`，当且仅当从 `q0` 出发沿 `w` 标记的路径读完全部输入后到达某个接受状态。

DFA 的扩展转移函数

DFA 的扩展转移函数 `δ̃` 定义为：

```
δ̃(q, ε) = q
δ̃(q, wa) = δ(δ̃(q, w), a)
```

它表示从状态 `q` 出发读入整个串 `w` 后到达的状态。

DFA 的语言

DFA `A` 定义的语言为：

```
L(A) = { w ∈ Σ* | δ̃(q0, w) ∈ F }
```

正则语言

语言 `L` 是正则语言，当且仅当存在某个 DFA `A` 使得 `L = L(A)`。

可达状态

状态 `q` 是可达的，当且仅当存在串 `w ∈ Σ*`，使得 `q = δ̃(q0, w)`。

不可达状态

状态 `q` 是不可达的，当且仅当不存在从初始状态 `q0` 到 `q` 的路径。

能到达接受状态的状态

状态 `q` 能到达接受状态，当且仅当存在 `p ∈ F` 和 `w ∈ Σ*`，使得 `p = δ̃(q, w)`。

无用状态

对语言识别没有贡献的状态称为无用状态，典型包括不可达状态和不能到达接受状态的状态。

NFA

非确定型有穷自动机 `NFA A` 是五元组：

```
A = (Q, Σ, δ, q0, F)
```

与 DFA 的区别是转移函数为：

```
δ: Q × Σ -> 2^Q
```

即同一状态读同一符号可以转移到多个状态，也可以无转移。

线索

对于输入串，线索是自动机中一条以初始状态为起点、其路径标记正好消耗输入串某个前缀的状态转移路径。

成功线索

成功线索是始端为初始状态、标记为整个输入串、末端为接受状态的线索。

活动状态集

NFA 读入输入串某个前缀后，所有可能线索末端状态组成的集合称为活动状态集。

NFA 的扩展转移函数

NFA 的扩展转移函数 $\tilde{\upsilon}$ 定义为：

$$\tilde{\upsilon}(q, \varepsilon) = \{q\}$$
$$\tilde{\upsilon}(q, wa) = \bigcup_{p \in \tilde{\upsilon}(q,w)} \upsilon(p,a)$$

它表示从状态 q 出发读入串 w 后所有可能到达的状态集合。

NFA 的语言

NFA A 定义的语言为：

$$L(A) = \{ w \in \Sigma^* \mid \tilde{\upsilon}(q0, w) \cap F \neq \varnothing \}$$

ε-转移

ε-转移是不消耗输入符号即可发生的状态转移。

ε-NFA

含有 ε-转移的 NFA 称为 ε-NFA。其转移函数通常写为：

$$\upsilon: Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$$

ε-闭包

状态 q 的 ε-闭包 $\omega(q)$ 是 q 自身以及从 q 出发经过零次或多次 ε-转移能到达的所有状态集合。

状态集合的 ε-闭包

状态集合 S 的 ε-闭包定义为：

```
ω(S) = ∪_{q ∈ S} ω(q)
```

ε-闭集

若状态集合 S 满足 $S = \omega(S)$，则称 S 是 ε-闭集。

ε-NFA 的扩展转移函数

ε-NFA 的扩展转移函数定义为：

```
υ̃(q, ε) = ω(q)
υ̃(q, xa) = ω(∪_{p ∈ υ̃(q,x)} υ(p,a))
```

子集构造法

子集构造法是把 NFA 转换为等价 DFA 的方法：把 NFA 的状态集合的子集作为 DFA 的状态，DFA 的转移由 NFA 的活动状态集更新规则给出。

DFA/NFA/ε-NFA 等价

DFA、NFA 和 ε-NFA 在定义语言能力上等价：它们定义的语言类都是正则语言。

原子语言

正则表达式构造中的原子语言包括：

```
∅, {ε}, {a} 其中 a ∈ Σ
```

语言连接

语言 $L1$ 与 $L2$ 的连接定义为：

```
L1L2 = { st | s ∈ L1, t ∈ L2 }
```

语言幂

语言 L 的 n 次幂定义为 n 个 L 的连接：

```
L^0 = {ε}
L^n = L L ... L  (n 个 L)
```

Kleene 闭包

语言 L 的闭包定义为：

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots$$

正则表达式

字母表 Σ 上的正则表达式由以下规则定义：

- 基础：$\varnothing$、ε、任意 $a \in \Sigma$ 都是正则表达式。
- 归纳：若 R 和 S 是正则表达式，则 $R+S$、RS、R^*、(R) 都是正则表达式。

正则表达式定义的语言

正则表达式 R 的语言记为 $L(R)$，递归定义为：

$$
\begin{aligned}
&L(\varnothing) = \varnothing \\
&L(\varepsilon) = \{\varepsilon\} \\
&L(a) = \{a\} \\
&L(RS) = L(R)L(S) \\
&L(R+S) = L(R) \cup L(S) \\
&L(R^*) = L(R)^*
\end{aligned}
$$

Kleene 定理

一个语言能由正则表达式定义，当且仅当它能由有穷自动机定义。

GNFA

通用 NFA 是弧上标记可为正则表达式的 NFA，通常要求只有一个接受状态、没有射入初始状态的弧、没有从接受状态射出的弧。

3. 词法分析相关定义

出处：XJTU 第4章课件、抽取文本、HIT 编译原理第3讲、HIT 抽取文本

词法单元

词法单元是词法分析器识别出的单词类别，如关键字、标识符、常数、界符、运算符等。

Token

Token 是词法单元的内部表示，通常表示为：

```
<种属，值>
```

一符一种

一符一种是指每个具体符号或关键字单独作为一个记号类别，例如 `+`、`++`、`while` 各有自己的类别。

全体一种

全体一种是指一类词法单元共用一个记号类别，并用值区分具体实例，例如所有标识符共用 `ID`，所有整数共用 `NUM`。

σ-DFA

$\sigma\text{-DFA}(A, \mathcal{L})$ 是正则语言簇 $\mathcal{L}$ 上的多语言联合 DFA，用于同时识别多个词法单元语言，并为接受状态标注对应词法类别。

事实优先级

事实优先级是词法语言有序集合中的优先级规则：下标较小者优先级较高，常用于解决关键字与标识符等重叠匹配问题。

前缀最大化原则

前缀最大化原则是词法分析中的最长匹配原则：从当前位置开始，词法分析器尽可能读入更长前缀；当继续读入导致失败时，回退到最近一次接受状态。

超前搜索

超前搜索是为实现前缀最大化而读入超过当前 token 末尾的字符，并在失败后回退的机制。

有效串

有效串是源程序经过预处理后可供词法分析器识别的字符序列。

词法分析器

词法分析器是将源程序字符流转换为 token 流的程序，通常以正则表达式和 DFA 为理论基础。

4. 上下文无关文法与上下文无关语言

出处：XJTU 第5章课件、抽取文本、HIT 形式语言 ch5-s1、HIT 形式语言 ch5-s4

CFG

上下文无关文法 `CFG G` 是四元组：

```
G = (V, T, P, S)
```

其中 `V` 是变元集合，`T` 是终结符集合，`P` 是产生式集合，$S \in V$ 是开始符号。

变元

变元也称非终结符，是文法中可以继续被产生式替换的符号。

终结符

终结符是文法中不能继续被替换、最终出现在句子中的符号。

产生式

产生式是形如 `A -> α` 的规则，其中 `A` 是变元，`α` 是由变元和终结符组成的符号串。

候选式

同一产生式左部对应的每个右部称为该变元的一个候选式。例如 `A -> α | β` 中，`α` 和 `β` 都是 `A` 的候选式。

子文法

子文法是从原文法中选取部分符号和产生式形成的文法，通常用于文法修剪或局部分析。

直接推导

若文法中有产生式 `A -> γ`，则：

```
αAβ => αγβ
```

称为一步直接推导。

推导

推导是零步或多步直接推导的序列，记为 `=>*`。

最左推导

每一步都替换当前句型中最左变元的推导称为最左推导，记为 `=>*_lm`。

最右推导

每一步都替换当前句型中最右变元的推导称为最右推导，记为 `=>*_rm`。

句型

若 `S =>* α`，则 `α` 是文法的句型。句型中可以含有变元和终结符。

句子

只含终结符的句型称为句子。若 `S =>* w` 且 `w ∈ T*`，则 `w` 是文法的句子。

文法的语言

CFG `G=(V,T,P,S)` 定义的语言为：

```
L(G) = { w ∈ T* | S =>* w }
```

上下文无关语言

若存在某个 CFG `G` 使得 `L = L(G)`，则称 `L` 是上下文无关语言，简称 CFL。

归约

归约是推导的逆过程：把当前符号串中某个候选式的出现替换为对应产生式左部变元。

直接归约

若有产生式 `A -> γ`，则把 `αγβ` 中的 `γ` 替换为 `A` 得到 `αAβ`，称为一步直接归约。

可归约串

当前符号串中与某个产生式右部候选式相同、可被替换为产生式左部的子串称为可归约串。

语法树

语法树是表示一个句子如何由文法开始符号推导出来的树形结构：根为开始符号或某变元，内部结点为变元，叶子为终结符或 `ε`，每个内部结点及其孩子对应一条产生式。

产物

语法树从左到右连接所有叶子结点得到的符号串称为该语法树的产物。

短语

句型中由某棵子树产生的子串称为该句型相对于该语法树的短语。

直接短语

由高度为 1 的子树产生的短语称为直接短语，即某个产生式右部在句型中的直接出现。

句柄

在规范归约中，应被下一步归约的直接短语称为句柄。直观地说，句柄是最右推导逆过程中的“下一刀”。

规范推导

最右推导常称为规范推导。

规范归约

规范归约是规范推导的逆过程，即按句柄逐步归约的过程。

歧义文法

一个 CFG 是歧义的，当且仅当它的语言中存在某个句子至少有两棵不同语法树。

文法歧义性的等价刻画

对 CFG 的句子 `w`，以下等价：

- `w` 有两棵不同语法树。
- `w` 有两个不同最左推导。
- `w` 有两个不同最右推导。

歧义性的常见来源

歧义性常来自运算符优先级不明确、结合性不明确、悬空 `else` 等。

固有歧义语言

若一个 CFL 的所有 CFG 都是歧义文法，则称该语言是固有歧义语言。

有用符号

若存在：

```
S =>* αXβ =>* w,  w ∈ T*
```

则符号 `X` 是有用符号。

无用符号

不是有用符号的文法符号称为无用符号。

可达符号

若存在：

```
S =>* αXβ
```

则符号 `X` 是可达符号。

有产出符号

若存在：

```
X =>* w,  w ∈ T*
```

则符号 `X` 是有产出符号。

ε-产生式

形如 `A -> ε` 的产生式称为 ε-产生式。

可空变元

若 `A =>* ε`，则变元 `A` 称为可空变元。

单位产生式

形如 `A -> B` 的产生式称为单位产生式，其中 `A`、`B` 均为变元。

单位对

若 `A` 可以经零步或多步单位产生式推出 `B`，则 `(A,B)` 称为单位对。

直接左递归

若文法中存在形如：

```
A -> Aα
```

的产生式，则称其具有直接左递归。

左递归

若存在某个变元 `A` 和串 `α`，使得：

```
A =>+ Aα
```

则称文法具有左递归。

间接左递归

经过两步或多步推导才出现的左递归称为间接左递归。

左公因子

若同一变元的多个候选式具有共同前缀，则该共同前缀称为左公因子。

5. PDA 与 CFL

出处：XJTU 第7章课件、抽取文本、HIT 形式语言 ch6-s1、ch6-s2、ch6-s4

PDA

下推自动机 `PDA P` 是七元组：

```
P = (Q, Σ, Γ, δ, q0, Z0, F)
```

其中 `Q` 是状态集，`Σ` 是输入字母表，`Γ` 是栈符号集，`δ` 是转移函数，`q0` 是初始状态，`Z0` 是初始栈符号，`F` 是接受状态集。

PDA 转移函数

PDA 的转移函数为：

```
δ: Q × (Σ ∪ {ε}) × (Γ ∪ {ε}) -> 2^(Q × Γ*)
```

若 `(p, γ) ∈ δ(q, a, X)`，表示 PDA 在状态 `q`，读入 `a`，弹出栈顶 `X`，转到状态 `p`，并把 `γ` 压入栈。

PDA 瞬时描述

PDA 的瞬时描述 `ID` 是三元组：

```
(q, w, β)
```

其中 `q` 是当前状态，`w` 是剩余输入串，`β` 是栈内容，栈顶在左端。

PDA 直接移动

若 `(p, α) ∈ δ(q, a, X)`，则：

```
(q, aw, Xβ) ⊢ (p, w, αβ)
```

称为一步直接移动。

PDA 移动

`⊢*` 表示零步或多步直接移动，具有自反性和传递性。

终态接受

PDA 以终态方式接受语言：

```
L(P) = { w | (q0,w,Z0) ⊢* (f,ε,α), f ∈ F }
```

空栈接受

PDA 以空栈方式接受语言：

```
N(P) = { w | (q0,w,Z0) ⊢* (q,ε,ε), q ∈ Q }
```

PDA 两种接受方式等价

对 PDA 而言，终态接受与空栈接受在定义语言能力上等价。

DPDA

确定型下推自动机是满足确定性限制的 PDA：在任意状态、输入符号/ε 和栈顶符号/ε 条件下，至多有一个可选转移，并且不能同时出现相互冲突的读输入转移与 ε 转移。

DCFL

由 DPDA 接受的上下文无关语言称为确定型上下文无关语言，简称 DCFL。

前缀性质

语言 `L` 有前缀性质，当且仅当 `L` 中不存在两个不同字符串 `x` 和 `y`，使得 `x` 是 `y` 的前缀。

6. LL(1) 分析

出处：XJTU 第6章课件、抽取文本、HIT 编译原理第4讲、HIT 抽取文本

自上而下分析

自上而下分析是从文法开始符号出发，试图构造输入串的推导，通常对应语法树的先根扩展。

递归下降分析

递归下降分析是为每个非终结符写一个过程，通过过程调用模拟自上而下推导的语法分析方法。

LL(k) 文法

LL(k) 文法是能从左到右扫描输入、产生最左推导，并且每步只需向前看 `k` 个输入符号即可唯一选择产生式的文法。

LL(1) 文法

LL(1) 文法是 `k=1` 的 LL(k) 文法，即每步只需一个向前看符号即可确定使用哪个候选式。

FIRST 集

符号串 `α` 的 FIRST 集是 `α` 能推导出的终结符串的首符号集合；若 `α =>* ε`，则 `ε ∈ FIRST(α)`。

FOLLOW 集

变元 `A` 的 FOLLOW 集定义为：

```
FOLLOW(A) = { a | S =>* α A a β, a ∈ T }
```

若 `A` 可能出现在句型末尾，则结束符 `#` 属于 `FOLLOW(A)`。

SELECT 集

产生式 `A -> β` 的 SELECT 集是可以选择该产生式进行推导时的输入符号集合：

```
若 ε ∉ FIRST(β), SELECT(A->β)=FIRST(β)
若 ε ∈ FIRST(β), SELECT(A->β)=(FIRST(β)-{ε}) ∪ FOLLOW(A)
```

预测分析表

预测分析表 `M` 是 LL(1) 分析器根据当前变元和当前输入符号选择产生式的表。

LL(1) 条件

同一变元的各候选式 SELECT 集两两不交，则该文法满足 LL(1) 条件。

LL(1) 修剪

LL(1) 修剪是通过消除左递归、提取左公因子、划分子文法和去除无用符号等方法，把文法调整为适合 LL(1) 分析的形式。

7. 自下而上、LR 与 SLR(1) 分析

出处：XJTU 第8章课件、抽取文本、HIT 编译原理第5讲、第6讲、第7讲

自下而上分析

自下而上分析是从输入串开始不断归约，直到归约为开始符号；成功过程对应自底向上构建语法树。

移进-归约分析

移进-归约分析使用符号栈：把输入符号移进栈，当栈顶形成某个候选式时，将其归约为产生式左部变元。

移进

移进是把当前输入符号压入分析栈，并读取下一个输入符号的动作。

归约

归约是在分析栈顶发现某个候选式后，弹出该候选式并压入对应左部变元的动作。

SR 冲突

移进-归约冲突是指分析器在某状态和输入符号下，既可以移进又可以归约。

RR 冲突

归约-归约冲突是指分析器在某状态和输入符号下，有多个可选归约动作。

右句型

能由开始符号经过最右推导得到的句型称为右句型。

活前缀

活前缀是某个右句型的前缀，并且不超过该右句型的句柄右端。它是 LR 分析栈中可能出现的符号串。

文法项目

文法项目是在产生式右部某位置加入圆点得到的形式，例如：

```
A -> α . β
```

圆点表示当前识别到的位置。

初始项目

增广文法中开始产生式 `S' -> . S` 对应的项目称为初始项目。

完全项目

圆点位于候选式末尾的项目称为完全项目，例如：

```
A -> α .
```

它表示可以归约。

移进项目

圆点后面还有符号的项目称为移进项目，例如：

```
A -> α . X β
```

它表示还可继续识别符号 `X`。

有效项目

项目 `A -> β . γ` 对活前缀 `αβ` 有效，当且仅当存在推导使得 `A -> βγ` 正在被识别，且 `β` 已经出现在活前缀末端。

LR(0) 项目集

LR(0) 项目集是若干 LR(0) 项目的集合，表示 LR 分析器在某一状态下的所有可能识别情形。

闭包

项目集闭包是指：若项目集中有 `A -> α . B β`，则把 `B` 的所有产生式对应的项目 `B -> . γ` 加入项目集，直到不再变化。

GOTO 函数

`GOTO(I, X)` 表示项目集 `I` 在识别文法符号 `X` 后转移到的项目集。

LR(0) 规范簇

从初始项目集出发，通过闭包和 GOTO 构造得到的全部 LR(0) 项目集组成 LR(0) 规范簇。

itemDFA

itemDFA 是识别活前缀的 DFA，其状态是 LR(0) 项目集，转移由 GOTO 函数给出。

LR(0) 文法

若基于 LR(0) 项目集构造的分析表没有移进-归约冲突和归约-归约冲突，则该文法是 LR(0) 文法。

SLR(1) 文法

SLR(1) 文法是在 LR(0) 项目集基础上，使用 FOLLOW 集限制归约动作后分析表无冲突的文法。

LR(0) 分析表

LR(0) 分析表由 ACTION 和 GOTO 两部分组成，用于指导移进、归约、接受和出错。

SLR(1) 分析表

SLR(1) 分析表是在 LR(0) 项目集基础上，只在 FOLLOW(A) 中的输入符号上为完全项目 `A -> α .` 填写归约动作得到的分析表。

8. 属性文法与语法制导翻译

出处：XJTU 第9章课件、抽取文本、HIT 编译原理第8讲、第9讲、第10讲

属性

属性是附着在文法符号上的语义信息。

属性名

属性名是某类属性的名称，例如 `type`、`place`、`code`、`val`、`tc`、`fc`。

属性值

属性值是属性在某个语法树结点上的具体取值。

属性方程

属性方程是定义属性值如何由其他属性值计算得到的语义规则。

属性文法

属性文法是在 CFG 的产生式上配备属性和属性方程，用于描述语义分析或翻译行为的文法。

综合属性

综合属性是由当前结点的子结点属性计算得到，并向语法树上方传递的属性。

继承属性

继承属性是由父结点或左兄弟结点等上下文信息计算得到，并向语法树下方或横向传递的属性。

属性求值

属性求值是在语法树或分析过程中按照属性依赖关系计算属性值的过程。

带注释语法树

带注释语法树是在语法树结点上标出属性值的语法树。

S-属性文法

只含综合属性的属性文法称为 S-属性文法。

L-属性文法

若任意继承属性只依赖于父结点的属性以及该符号左兄弟的属性，而不依赖右兄弟属性，则称为 L-属性文法。

语法制导翻译

语法制导翻译是伴随语法分析进行属性求值，并以属性计算的副作用生成中间代码或其他语义结果的翻译方法。

LR 制导属性求值

LR 制导属性求值是在自底向上的 LR/SLR 分析归约时计算属性值，常适用于 S-属性文法。

LL 制导属性求值

LL 制导属性求值是在自上而下分析过程中计算属性值，常适用于 L-属性文法。

属性栈

属性栈是在 LR 制导属性求值中与分析栈联动的栈，用于保存文法符号的属性值。

桩变量

桩变量是为在分析过程中暂存、传递或转化属性值而引入的辅助变量。

9. 符号表与声明语义

出处：XJTU 第9章课件、抽取文本、XJTU 第10章课件、第10章抽取文本、HIT 编译原理第11讲

符号表

符号表是编译过程中记录源程序中各名字的属性和语义信息的数据结构，供语义分析、代码生成等阶段使用。

符号表表头

符号表表头记录符号表整体信息，如外层符号表指针 `outer`、宽度 `width`、参数个数 `argc`、参数表 `arglist`、返回类型 `rtype`、嵌套层数 `level`、代码 `code` 等。

符号表登记项

符号表登记项记录一个名字及其语义信息，通常包括 `name`、`type` 和类型特有域。

名字

名字是程序中的标识符，如变量名、函数名、数组名、标号名等。

类型

类型是名字或表达式的语义类别，如 `INT`、`FLO`、`ARRAY`、`FUNC`、`ARRPTT`、`FUNPTT` 等。

类型特有信息

类型特有信息是与某种类型相关的附加信息，例如数组的元素类型、维数、维长、基地址，函数的参数表、返回类型和局部符号表指针等。

bind

`bind(name, type)` 在当前符号表中登记名字及其类型；若已存在同名登记项则报错。

lookup

`lookup(name)` 在符号表中查找名字并返回登记项或 `UNBOUND`；也可用于查询或设置登记项的某个域。

newtab

`newtab()` 创建并初始化一个新的符号表。

update

`update[]` 用于访问或更新当前符号表表头中的域。

作用域

作用域是名字可被合法引用的程序区域。

静态作用域

静态作用域是由程序的声明嵌套结构决定的作用域规则。

声明宿主

若函数 `g` 声明在函数 `f` 内，则 `f` 是 `g` 的声明宿主，也称直接外层。

引用宿主

若函数 `f` 调用函数 `g`，则 `f` 是 `g` 的引用宿主。

数组类型

数组类型用 ARRAY 表示，通常包含元素类型、维数、各维维长和存储基址等信息。

数组原型

数组作为形参时可声明为数组原型，表示实参传入的是数组存储区的引用或首址。

函数类型

n 参数函数的类型表达式可写为：

```
D1 × D2 × ... × Dn -> R
```

其中 Dk 是第 k 个参数类型，R 是返回类型。

函数原型

函数作为形参时可声明为函数原型，表示实参传入的是函数入口和相关环境信息。

类型检查

类型检查是检查声明和使用是否满足类型语义规则的过程，包括类型匹配、参数匹配、数组下标约束等。

10. 中间代码与翻译属性

出处：XJTU 第10章课件、抽取文本、HIT 编译原理第11讲、第12讲、第13讲、第14讲

三地址代码

三地址代码是每条指令最多含有三个地址的中间代码形式，例如：

```
t1 = x + y
IF x < y THEN l1 ELSE l2
```

四元式

四元式是三地址代码的一种记录形式：

```
(op, arg1, arg2, result)
```

抽象语法树

抽象语法树是省略部分具体语法细节后得到的树形中间表示，通常内部结点为操作符，叶子为操作数。

DAG

DAG 是有向无环图，用于表示表达式时，每个子表达式对应一个结点，公共子表达式可由多个父结点共享。

place 属性

`place` 属性通常表示表达式或名字的计算结果存放位置，例如变量名或临时变量。

code 属性

`code` 属性用于保存某语法单位翻译后产生的中间代码序列。

tc 属性

布尔表达式的 `tc` 属性表示条件为真时跳转目标标号的集合。

fc 属性

布尔表达式的 `fc` 属性表示条件为假时跳转目标标号的集合。

newvar

`newvar()` 用于生成新的临时变量。

newlabel

`newlabel()` 用于生成新的标号。

gen

`gen[...]` 根据参数生成一段三地址代码或代码列表。

emit

`emit` 根据参数生成指令并追加到代码区，课件中区分是否对参数求值的宏/函数形式。

merge_code

`merge_code` 用于把多个代码列表按顺序合并。

结构化代码块

结构化代码块是只有一个入口和一个出口的代码块。

标号

标号是代码中可作为跳转目标的位置名字。

goto 语句

`goto` 语句是直接跳转到某个标号的控制语句。

11. 运行时环境

出处：XJTU 第11章课件、抽取文本、HIT 编译原理第15讲、HIT 抽取文本

运行时环境

运行时环境是程序执行时用于管理内存、过程调用、参数传递、返回、非局部名访问等机制的系统结构。

内存映像

内存映像是程序运行时各存储区域的组织图，包括代码区、静态区、栈区、堆区等。

代码区

代码区存放可执行代码。

静态区

静态区存放全局或静态生命周期的数据。

栈区

栈区用于保存过程调用期间的活动记录，支持后进先出的调用/返回。

堆区

堆区用于管理动态分配对象。

过程活动

过程体的一次执行称为该过程的一个活动。

活动记录

活动记录是保存一个过程活动所需信息的连续存储区，也称栈帧。

栈帧

栈帧是活动记录在运行栈中的具体表现，通常包括参数区、链接区、返回地址和局部区。

参数区

参数区保存调用过程传递给被调过程的实参信息。

链接区

链接区保存过程调用和返回所需的链接信息，如访问链、控制链等。

局部区

局部区保存被调过程的局部变量和临时变量。

访问链

访问链是指向静态外层过程最新活动记录的指针，用于访问非局部名字。

控制链

控制链是指向调用者活动记录的指针，用于过程返回时恢复调用者栈帧。

返回地址

返回地址是被调过程结束后控制流应回到的代码位置。

栈快照

栈快照是程序运行到某一时刻时栈内容的静态描述。

栈指针

栈指针 `sp` 指向当前运行栈顶部或下一个可用位置，具体含义依实现约定。

帧指针

帧指针 `fp` 指向当前活动记录中的固定位置，用于稳定访问参数、链接和局部变量。

调用序列

调用序列是主调过程在调用被调过程前后执行的代码段，用于构建参数区、链接区、转入被调过程、接收返回值并释放调用相关空间。

返回序列

返回序列是处理 `RETURN` 指令的代码段，用于保存返回值并跳转到函数尾声。

函数序言

函数序言是添加在函数入口处的代码段，用于保存必要寄存器、处理返回地址、把参数单元内容赋给形参变量并构建局部区。

函数尾声

函数尾声是添加在函数出口处的代码段，用于恢复寄存器、释放局部区、取出返回地址并转移控制流。

传值

传值是调用时把实参的值拷贝到形参单元中，被调过程修改形参不影响原实参。

传地址

传地址是调用时把实参地址传给形参，被调过程通过地址访问或修改原实参。

得结果

得结果是一种参数传递机制：调用结束时将形参单元内容拷贝回对应实参单元。

传名

传名类似宏展开，把实参看作表达式文本替换到被调过程中的形参出现处。

非局部名

非局部名是在当前过程之外、但处于静态作用域可见范围内声明的名字。

非局部名寻址

非局部名寻址是通过访问链或显示表找到声明该名字的外层活动记录，再按偏移量访问该名字存储单元的过程。

显示表

显示表是访问链的优化结构，用一个向量保存各静态层当前活动记录指针，以便快速访问非局部名字。

静态层数

静态层数是过程在声明嵌套结构中的层级编号。

数组形参

数组形参通常传递数组存储区地址或首址，课件中常用 `ARRPTT` 表示数组原型参数类型。

函数形参

函数形参通常传递函数入口地址及其定义环境信息，课件中常用 `FUNPTT` 表示函数原型参数类型。

12. 图灵机与可判定性补充

> XJTU 绪论课件提到图灵机作为抽象计算模型；HIT 形式语言课件给出更完整形式定义。若考试只覆盖 XJTU 总复习中列出的图灵机基础，掌握本节核心定义即可。

出处：XJTU 第0章课件、XJTU 第1章课件、HIT 形式语言 ch8-s1、ch8-s2

图灵机

图灵机是由有限控制器、无限纸带和读写头组成的抽象计算模型。它根据当前状态和当前带符号，改写符号、移动读写头并改变状态。

图灵机形式定义

常见图灵机 `M` 是七元组：

```
M = (Q, Σ, Γ, δ, q0, B, F)
```

其中 `Q` 是状态集，`Σ` 是输入字母表，`Γ` 是带字母表，`δ` 是转移函数，`q0` 是初始状态，`B` 是空白符，`F` 是接受状态集。

通用图灵机

通用图灵机是能够读入任意图灵机编码及其输入，并模拟该图灵机运行的图灵机。

递归可枚举语言

若语言 `L` 被某个图灵机接受，则 `L` 是递归可枚举语言。

递归语言

若存在一个对所有输入都停机的图灵机接受语言 L，则 L 是递归语言。

可判定问题

如果一个问题对应的语言是递归语言，则该问题是可判定问题。

不可判定问题

如果一个问题对应的语言不是递归语言，则该问题是不可判定问题。