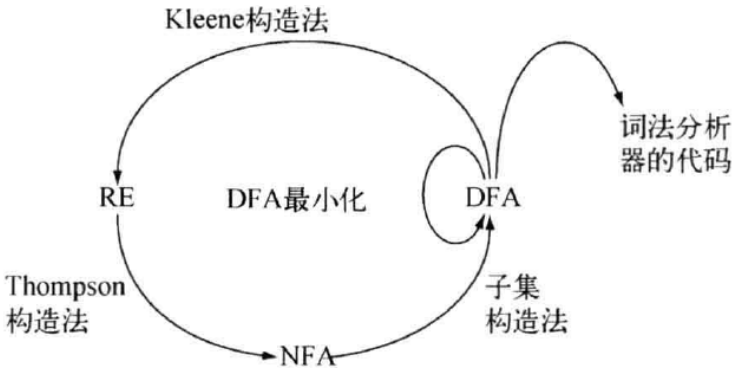
Compilers

- Outline
  - Lexer
    - ANTLR4
    - RE → Lexer



    - Regular Expression
      - look around
        - look ahead `?=`
        - look backward `?<=`
        - e.g. `(?<=<[hH]([1-6])>).*?(?=<\/[hH]\1>)`
      - conditional
        - `?(NO)`
        - e.g. `(<a[^>]+>)?(<img[^>]+>)(?(1)<\/a>)`
    - Automata
      - NFA（Nondeterministic Finite Automata）
      - DFA（Deterministic Finite Automata）
      - 转换表
    - **Thompson构造法** P_{100}

      给定字母表 Σ, Σ 上的正则表达式**由且仅由**以下规则定义:
      (1) ϵ 是正则表达式;
      (2) $\forall a \in \Sigma$, a 是正则表达式;
      (3) 如果 s 是正则表达式, 则 (s) 是正则表达式;
      (4) 如果 s 与 t 是正则表达式, 则 $s|t$, st, s^* 也是正则表达式。

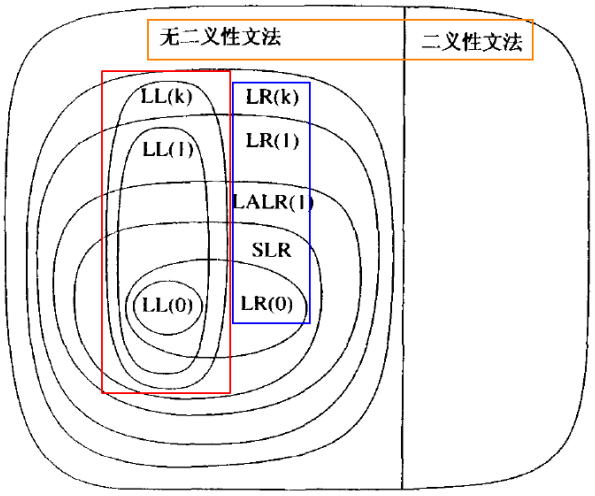
      RE → NFA
      - 按结构归纳
    - **子集构造法（Subset/Powerset Construction）** P_{97}

      NFA → DFA

        - 闭包 $f-\text{closure}(T)$
            - $T \implies f(T) \implies f(f(T)) \implies \cdots$
            - 直到找到 x 使得 $f(x)=x$，x 称为不动点
        - $\epsilon-\text{closure}(s)$
        - $\epsilon-\text{closure}(T)$
        - $\text{move}(T,a)$
    - 等价状态划分算法 P_{115}

      DFA最小化
        - 划分而非合并：$s \nsim t \iff \exists a \in \Sigma.(s \xrightarrow{a} s') \wedge (t \xrightarrow{a} t') \wedge (s' \nsim t')$
        - 接受状态与非接受状态必定不等价：$\Pi = \{F, S \backslash F\}$
    - Kleene闭包

      DFA → RE
- Parser
    - 二义性
        - 由结合性带来
        - 由优先级带来
    - ANTLR4
        - Listener Pattern
        - Visitor Pattern
    - Context-Free Grammar (CFG)
        - $A \in N \to \alpha \in (T \cup U)^*$
        - CSG (Context-Sensitive Grammar)
        - 推导
            - 经过一步推导出：$E \implies -E$
            - 经过零步或多步推导出：$E \overset{*}{\implies} -(\mathbf{id}+E)$
            - 经过一步或多步推导出：$E \overset{+}{\implies} -(\mathbf{id}+E)$
            - 最左推导、最右推导
        - 句型：$S \overset{*}{\implies} \alpha$，且 $\alpha \in (T \cup U)^*$
        - 句子：$S \overset{*}{\implies} w$，且 $w \in T^*$
        - 语言：$L(G)=\{w \mid S \overset{*}{\implies} w\}$
        - 表达能力强弱：regular < context-free < context-sensitive < recursively enumerable
        - Pumping Lemma for Regular Languages
    - 文法集合关系

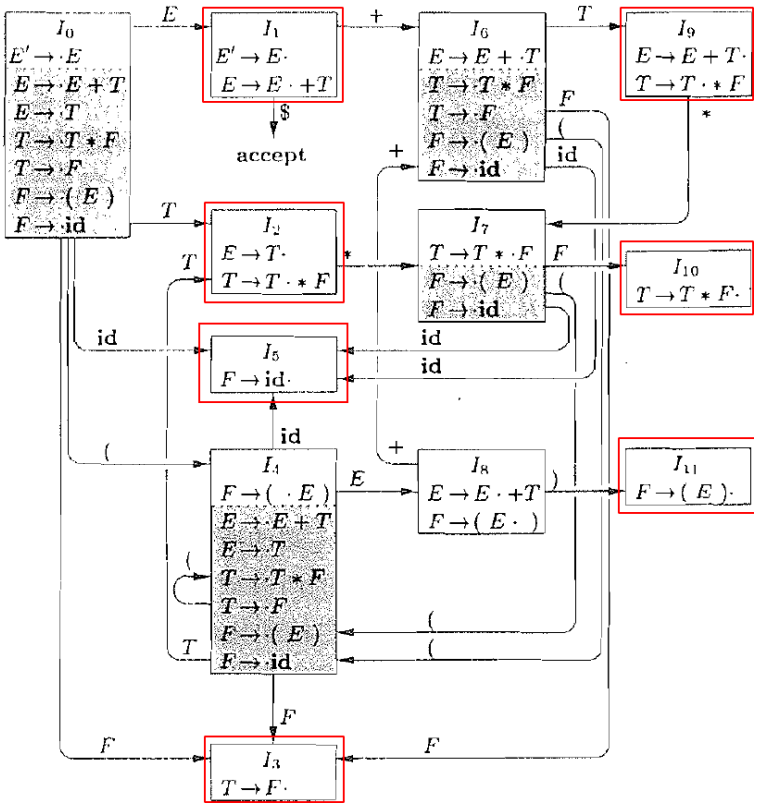


- LL Grammar
  - $\mathrm{FIRST}(\alpha) = \{t \in T \cup \{\epsilon\} \mid \alpha \overset{*}{\Longrightarrow} t\beta \vee \alpha \overset{*}{\Longrightarrow} \epsilon\}$
  - $\mathrm{FOLLOW}(\alpha) = \{t \in T \cup \{\ \$\ \} \mid \exists s.S \overset{*}{\Longrightarrow} s \overset{\Delta}{=\!=} \beta A t \gamma\}$
  - 预测分析表
    - 在表格 $[A, t]$ 中填入 $A \to \alpha$ 的条件
      - $t \in \mathrm{FIRST}(\alpha)$
      - $\epsilon \in \mathrm{FIRST}(\alpha) \wedge t \in \mathrm{FOLLOW}(\alpha)$
  - $\mathrm{LL}(1)$

    从左向右扫描输入，构建最左推导，只需向前看1个符号便可确定使用哪条产生式
    - 预测分析表无冲突
    - 直接左递归、间接左递归
    - 提取左公因子
  - Adaptive $\mathrm{LL}(*)$
    - **优先级上升算法**

      解决直接左递归与优先级问题
      - 左结合，优先级加1
      - 右结合，优先级不变
- LR Grammar
  - 增广文法

    加入产生式
  - 句柄（handle）
    - 项、项集、项集族
  - 句柄识别有穷自动机



- 初始状态：$\mathrm{CLOSURE}(\{[E' \to \cdot E]\})$
- 状态转移：$J = \mathrm{GOTO}(I, X) = \mathrm{CLOSURE}(\{[A \to \alpha X \cdot B] \mid [A \to \alpha \cdot X B] \in I\})$
- 接受状态：$F = \{I \in C \mid \exists [A \to \alpha \cdot] \in I\}$
- 在栈上操作

  状态机中的点指示了栈顶

- LR(0)

  从左向右扫描输入，构建**反向**最右推导，规约时无需向前看

  - LR(0)项
  - LR(0)分析表

sn	移入输入符号，并进入**状态** n
rk	使用k **号产生式**进行归约
gn	转换到**状态** n
acc	成功接受，结束
空白	错误

    函数拆分成 表（针对**终结符**）和 表（针对**非终结符**）

    - $\mathrm{GOTO}(I_i, a) = I_j \wedge a \in T \implies \mathrm{ACTION}[i, a] \leftarrow sj$
    - $\mathrm{GOTO}(I_i, a) = I_j \wedge A \in N \implies \mathrm{GOTO}[i, A] \leftarrow gj$
    - $[k : A \to \alpha \cdot] \in I_i \wedge A \neq S' \implies \forall t \in T \cup \{\,\$\,\}, \mathrm{ACTION}[i, t] \leftarrow rk$
    - $[S' \to S \cdot] \in I_i \implies \mathrm{ACTION}[i, \,\$\,] \leftarrow acc$

        - $\mathrm{LR}(0)$分析表无冲突
    - $\mathrm{SLR}(1)$

      Simple LR

        - 可能的冲突
            - 移入/规约
            - 规约/规约
        - 规约改为 $[k:A\to\alpha\cdot]\in I_i\wedge A\neq S'\implies\forall t\in\mathrm{FOLLOW}(A),\mathrm{ACTION}[i,t]\leftarrow rk$
            - 消除移入/规约冲突
        - 考虑结合性与优先级可消解冲突
        - $\mathrm{SLR}(1)$ 分析表无冲突
    - $\mathrm{LR}(1)$
        - $\mathrm{LR}(1)$项
            - $[A\to\alpha\cdot\beta,\alpha]\qquad(a\in T\cup\{\,\$\,\})$
            - $\forall b\in\mathrm{FIRST}(\beta a).[B\to\cdot\gamma,b]\in I$
        - 初始状态：$\mathrm{CLOSURE}(\{[E'\to\cdot E,\ \$\ ]\})$
        - 状态转移：$J=\mathrm{GOTO}(I,X)=\mathrm{CLOSURE}(\{[A\to\alpha X\cdot B,a]\mid[A\to\alpha\cdot XB,a]\in I\})$
        - $\mathrm{LR}(1)$分析表
            - $[k:A\to\alpha\cdot,a]\in I_i\wedge A\neq S'\implies\forall t\in T\cup\{\,\$\,\},\mathrm{ACTION}[i,a]\leftarrow rk$
            - $[S'\to S\cdot,\ \$\ ]\in I_i\implies\mathrm{ACTION}[i,\ \$\ ]\leftarrow acc$
        - $\mathrm{LR}(1)$ 分析表无冲突
    - $\mathrm{LALR}(1)$

      Look-ahead LR

        - 合并具有相同核心 $\mathrm{LR}(0)$ 项的状态
        - 不会引入移入/规约冲突
        - 可能引入规约/规约冲突
    - 除 $\mathrm{LR}(0)$ 外的各种 LR 类文法对应的语言等价
- Semantic Analysis
    - 符号表
        - DSL（Domain-Specific Language）通常只有单作用域
        - GPL（General Programming Language）通常需要嵌套作用域
    - Attribute Grammar
        - 为CFG赋予语义

- SDD（Syntax-Directed Definition）

  一个CFG和属性及规则的结合

  - 唯一确定了语法分析树上每个非终结符节点 N 的属性值
  - 未规定计算属性值的方式和顺序
- 注释（annotated）语法分析树

  显示各个属性值
- 依赖图
- 综合属性（Synthesized Attribute）

  只能通过 的子节点或 本身的属性来定义

  - S属性定义

    每个属性都是综合属性的SDD
  - 信息流自底向上
- 继承属性（Inherited Attribute）

  只能通过 的父节点、 本身和 的兄弟节点上的属性来定义

  - 信息流从左向右、从上到下
- L属性定义

  如果一个 SDD 的每个属性
  (1) 要么是综合属性,

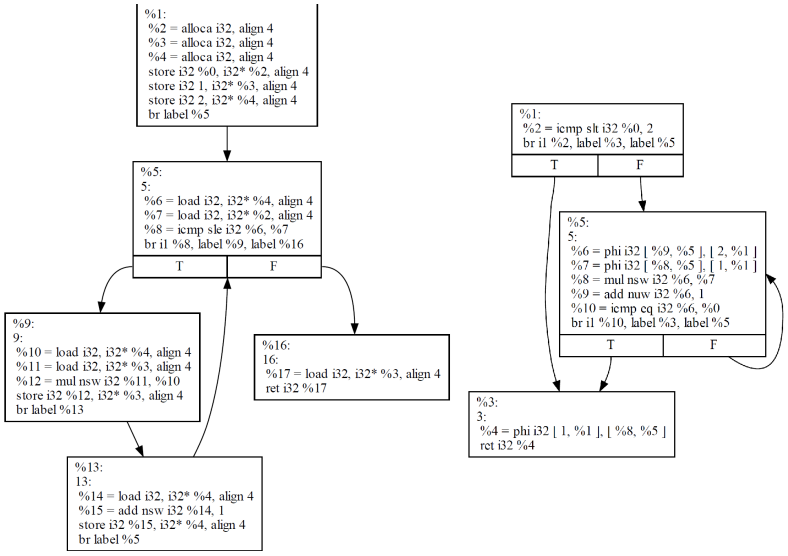
  (2) 要么是继承属性, 但是它的规则满足如下限制:
  对于产生式 $A \to X_1 X_2 \ldots X_n$ 及其对应规则定义的继承属性 $X_i.a$,
  则这个规则只能使用
  (a) 和**产生式头** A关联的**继承**属性;
  (b) 位于X_i **左边**的文法符号实例 X_1、X_2、...、X_{i-1} 相关的**继承**属性或**综合**属性;
  (c) 和**这个** X_i **的实例本身**相关的继承属性或综合属性, 但是在由这个 X_i 的全部属性组成的依赖图中**不存在环**。

  则它是 L 属性定义。
- SDT（Syntax-Directed Translation Scheme）

  在产生式中嵌入**语义动作**的CFG

  - LL语法分析中，从左到右处理各个 X_i 符号，对每个 X_i 先计算继承属性，后计算综合属性

- Intermediate Representation
  - clang生成LLVM IR：`clang -S -emit-llvm -fno-discard-value-names -g0`
  - TAC（Three Address Code）
  - SSA（Static Single Assignment）
  - CFG（Control Flow Graph）



以基本块为节点，仅从出点跳出，从入点跳入

- ϕ 指令：`phi [fast-math-flags] <ty> [<val0>, <label0>], ...`
- GEP指令：`getelementptr <base-type>, <base-type>*, <base-addr>, [<ty> <index>]+`
- 表达式的翻译
- 控制流语句的翻译 P_{258}, P_{260}
  - 地址回填（backpatching）技术 P_{263}
    - $\texttt{makelist}(i)$
    - $\texttt{merge}(p_1, p_2)$
    - $\texttt{backpatch}(p, i)$

- Target Code Generation
  - **RISC-V**
    - register conventions

Name	ABI Mnemonic	Meaning	Preserved across calls?
x0	zero	Zero	— (Immutable)
x1	ra	Return address	No
x2	sp	Stack pointer	Yes
x3	gp	Global pointer	— (Unallocatable)
x4	tp	Thread pointer	— (Unallocatable)
x5 - x7	t0 - t2	Temporary registers	No
x8 - x9	s0 - s1	Callee-saved registers	Yes
x10 - x17	a0 - a7	Argument registers	No
x18 - x27	s2 - s11	Callee-saved registers	Yes
x28 - x31	t3 - t6	Temporary registers	No

  - Instruction Selection
    - Selection DAG
    - 二义性：偏向于执行较大的规约
    - 规约/规约冲突时，优先选择较长的规约
    - 移入/规约冲突时，优先选择移入动作

- 瓦片覆盖（tiling）
  - 最大吞进（maximal munch）算法
  - 动态规划思想计算最优覆盖方案

- Stack Allocation
  - 活跃（live）

    对于给定的变量 x, 考虑从其一个定义点 p 到使用点 q 的路径 l。
    对于该路径 l 上的任意点 r, 如果 r 和 q 之间没有对变量 x 的其它定义,
    则称 x 在程序点 r 上是活跃的。

    - 活跃分析

      运用**不动点**求所有语句的和

      - $\mathrm{LIVEOUT(s)} = \bigcup_{p \in \mathrm{succ}(s)} \mathrm{LIVEIN}(p)$
      - $\mathrm{LIVEIN(s)} = \left(\mathrm{LIVEOUT(s)} \backslash \mathtt{def}(\mathrm{s}) \cup \mathtt{use}(\mathrm{s})\right)$

    - 活跃区间
  - 线性扫描分配算法
    - 占用、释放、溢出
    - 通常溢出剩余活跃区间更大的