在ACM上一辈子都用不到的内容

编译的各个阶段

1. 词法分析
2. 语法分析
3. 语义动作
4. 语义分析
5. 帧栈布局
6. 翻译
7. 规范化
8. 指令选择
9. 控制流分析
10. 数据流分析
11. 寄存器分配
12. 代码流出

本文将只涉及前2点的部分内容，作为潜在的可能对竞赛有用的工具
甚至可能用在字符串题(做梦)

词法分析

单词

单词的类型: `ID` (标识符), `NUM`(整数), `REAL`(小数), `IF`, `COMMA`, ...
而注释，预处理命令，宏，空白符不视为单词

词法分析器读入源程序，返回一个单词流，报告每个单词的类型
这些单词中一些（如标识符，数）有**语义值**，因此词法分析器也会附上这些信息

正则表达式

基本部分:

符号: 对于字母 `a`, 表达式 `a` 表示字符串 “a” (废话)
可选: 对于表达式 `M`, `N`, `M|N` 表示属于表达式 `M` 或 `N` 的字符串
连结: 符号 `·`, 表示连起两个字符串 (严格的定义懒得写了)
ϵ: 空串 **重复** kleene 闭包。 `(a|b)* == {“a”, “b”, “aa”, “ab”, “ba”, “bb”, …}`

正则表达式还有一些简写，如`[]` `?` `+` 等，但是他们并不影响表达式的描述能力

使用正则表达式可以指明语言的单词。对于每种单词，提供一段代码来报告单词的类型(和附加语义)

为了消除二义性，使用了规则**最长匹配** 和 **优先规则**。 为了压缩篇幅，这里不描述。

有限状态自动机

实现正则表达式匹配：

自动机包含一个**状态集合** 和状态转移的**边**
自动机吃入字符，根据吃入的字符选择边转移状态
如果当前状态是终态，则接受该字符串
如果自动机找不到字符对应的边，则拒绝接受字符串(error();)
为了保证最长匹配，需要记录上一次遇到的终态的位置

非确定状态自动机

NFA(非确定状态自动机)可能有ϵ的边
它允许不输入字符的情况下转换状态

正则更容易转换为NFA
这里只描述连接的转换方法:
若 M 可以建成自动机 $\rightarrow$M
则 M* 建成 $\rightarrow _ \epsilon N \rightarrow M \rightarrow _ \epsilon N$ (指回)，其中N是一个点

确定状态自动机

DFA(确定状态自动机)不包含ϵ的边，因此更容易实现

为了从NFA转换到DFA 有如下定义 **ϵ闭包**: `closure(S)` 是从 S(一个集合)中状态出发，只经过ϵ边(不接受任何字符)，便可到达的状态的集合
`closure(S)` 可以通过递归算出，直到不能再经过递归找到新的点为止
NFA从一个状态集合 S1 吃入一个字符 'c' 后可到达另一个集合 S2
实现可以是：遍历 S1，对每个 S1 中状态尝试转移 c，达到状态集合 S1A
再求`closure(S1A)` 得到 S2

有个方案可以求等价的最小自动机

语法分析

上下文无关文法

推导

语法分析树

二义性

递归下降的语法分析

预测分析 LL(1)文法

LR分析

LR(0)

LR(1)

例题(?)

|UOJ:未来程序・改