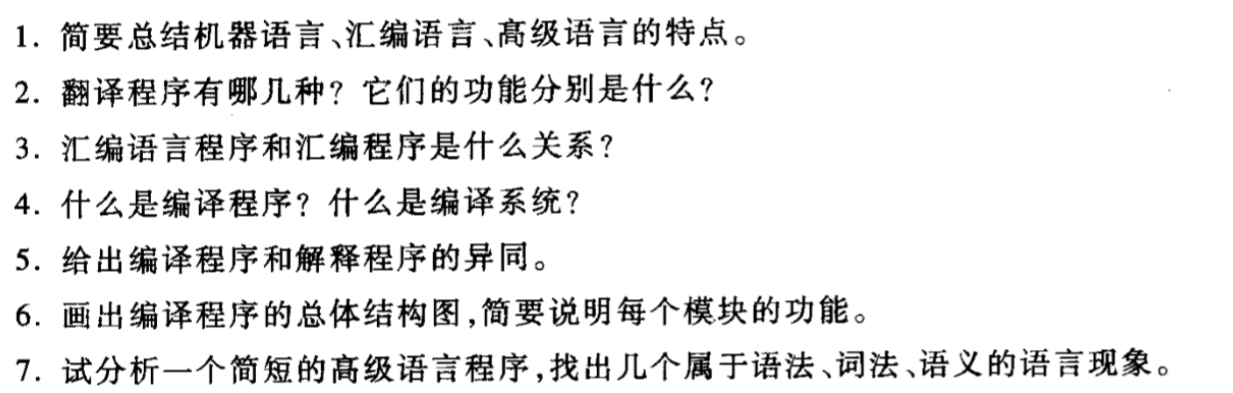
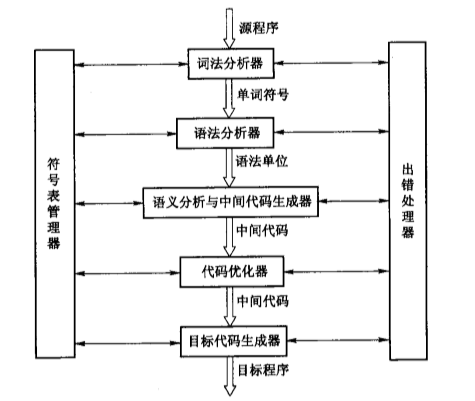
编译课本习题.md

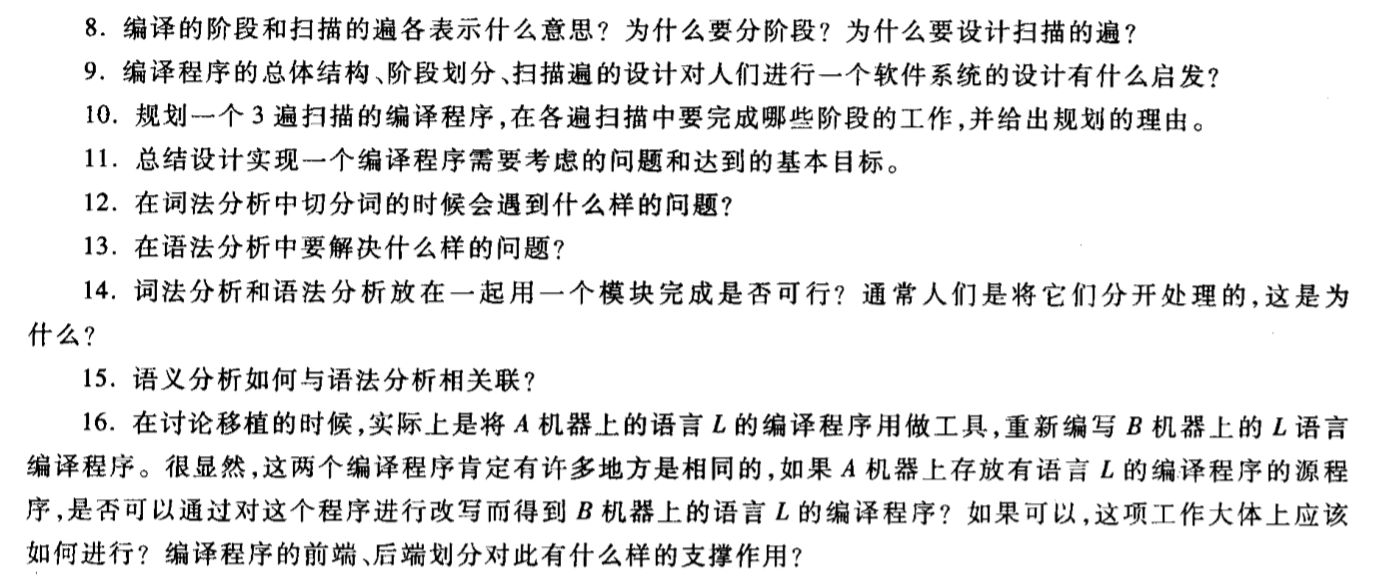
# 1. 概述



* + 机器语言: 0,1代码组成指令序列的集合. 编写效率低下, 容易出错
  + 汇编语言: 一系列助记符表示指令中的操作和操作数, 同时用符号表示程序用到的一系列数据. 受限于机器且仍对程序设计人员有严格要求
  + 高级语言: 以人们容易理解的形式表达要求和过程, 通常不受具体机型的限制, 接近人们的习惯, 可读性好,便于理解和维护

1. 翻译程序
   * 编译程序: 接受源程序, 输出与之等价的目标程序, 然后再对其进行相应处理后投入运行. 先完整的转换再处理,执行
   * 解释程序: 对每次读到的语句进行解释, 同时读取执行此语句需要的数据. 一边解释一边执行
   * 汇编语言程序: 汇编语言编写的源程序
   * 汇编程序: 源程序为**汇编语言程序**, 目标程序是机器语言程序时翻译程序称为汇编程序
   * 编译程序: 接受源程序, 输出与之等价的目标程序, 然后再对其进行相应处理后投入运行.
   * 编译系统: 编译程序和相应的运行系统
   * 异: 一个完整转换再执行, 一个边解释边执行
   * 同: 都是通过对组成程序的语句进行翻译完成的
2. 
3. //rust  
   let a = vec![1,2,3];  
   let b = a;  
   println!("{:#?}",a);

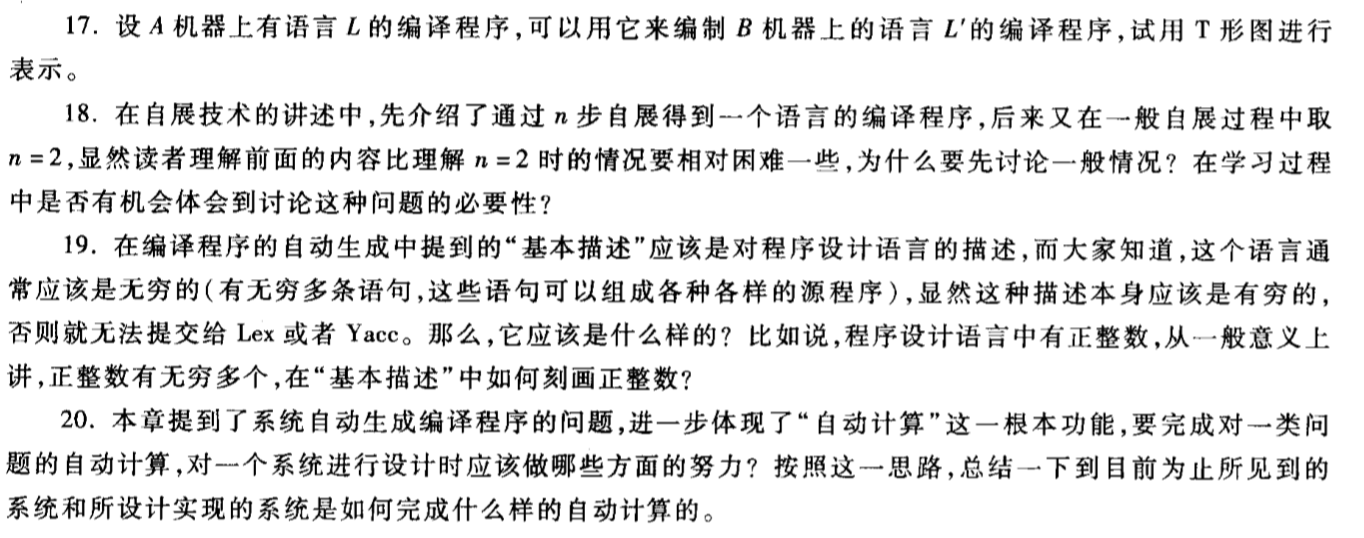
* 以上程序语法过关, 词法过关, 语义错误.

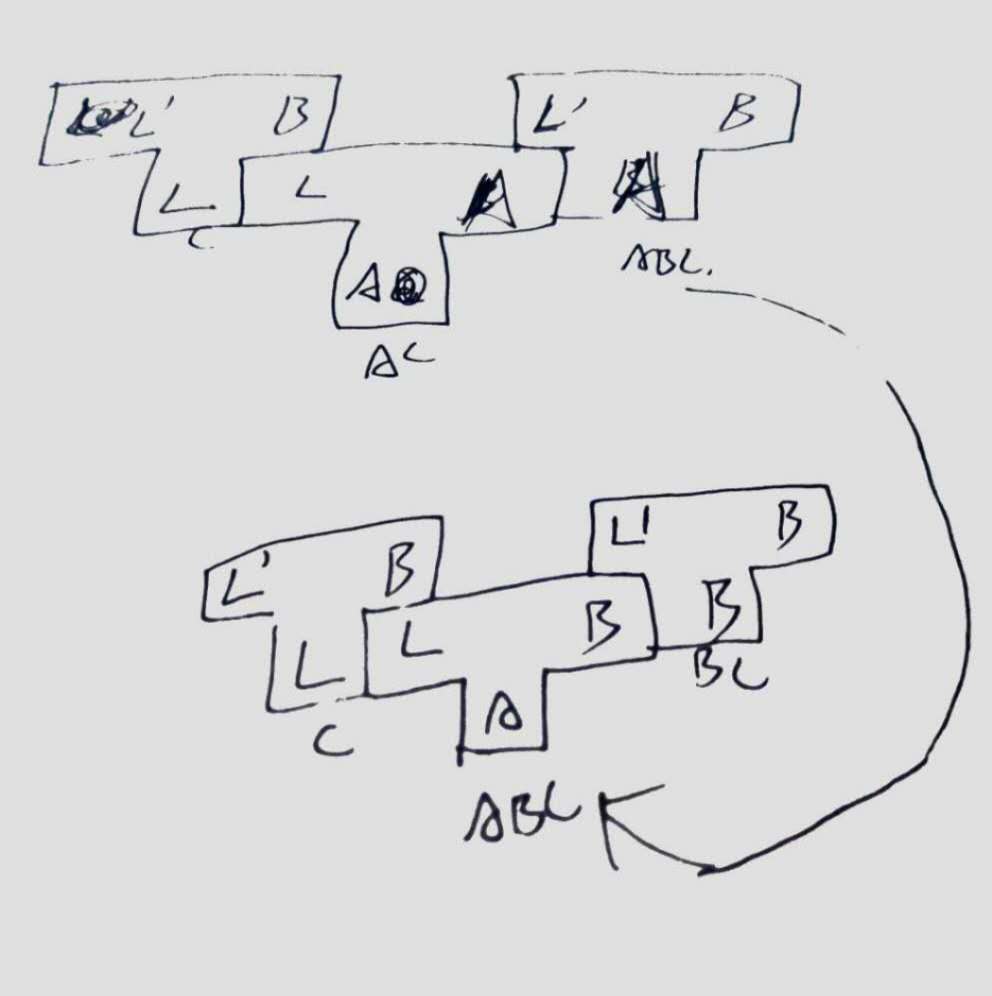


* + 阶段: 逻辑功能的划分
  + 分阶段: 划分逻辑功能
  + 遍: 编译程序对源程序和中间结果的完整扫描
  + 设计遍: 权衡编译系统效率与目标语言效率

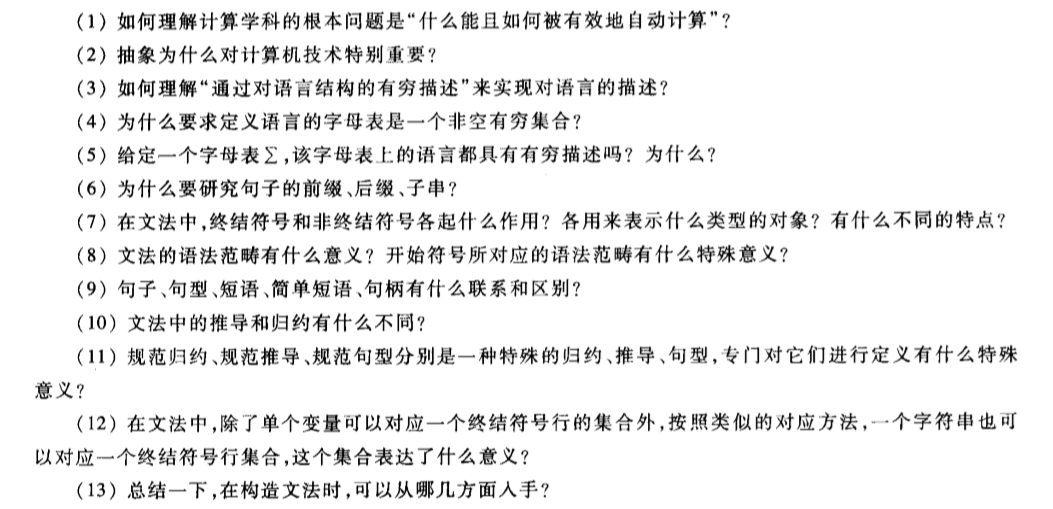
1. 可自顶向下的进行模块化设计, 并按照逻辑功能划分到不同的模块中, 并可有机组合这些模块.
   * 词法分析, 语法分析, 语义分析, 中间代码生成
   * 代码优化
   * 目标代码生成
   * 问题
     + 编译程序的效率
     + 优化
     + 先引用后定义
     + 内存资源消耗
   * 基本目标
     + 正确性
     + 完备性
     + 可维护
     + 可移植
2. 如何确定词的边界, 指出单词属于哪一类
3. 如何从输入单词序列构建出程序的结构不同层次的语法成分, 指出语法错误, 制导语义翻译.
4. 可行. 二者逻辑功能不同, 且较为独立.
5. 语法制导翻译
6. 可以, 在目标代码生成中加入B机器语言.

前端不变, 需要改写后端



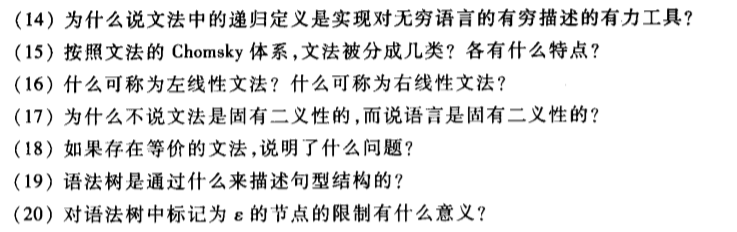
1. 
2. 一般情况范围更广不局限于单例计算. 比较少
   * 存在递归描述
   * 0|[1-9]+[0-9]\*
   * 可形式化描述这一类问题
   * 根据形式化描述完成自动计算

# 2. 高级语言及其文法



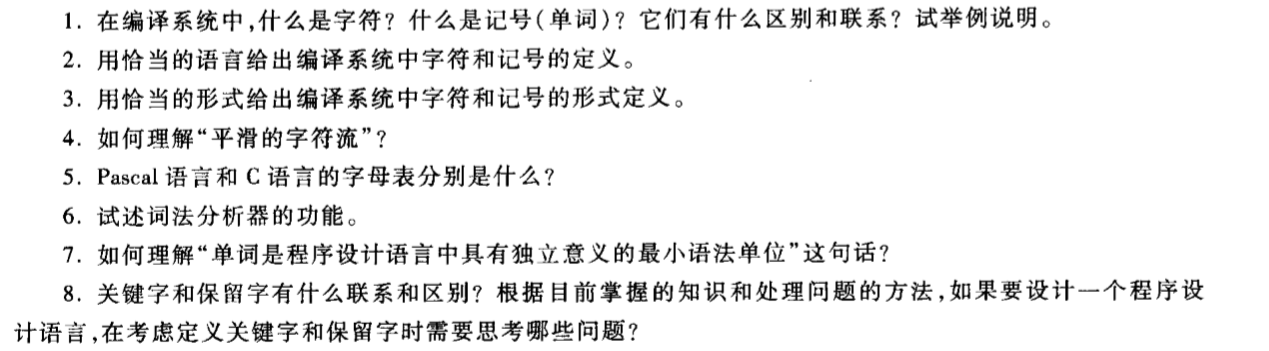
* + 什么能: 自动计算中什么是可判定的, 什么是不可判定的
  + 如何能: 具体的求解步骤

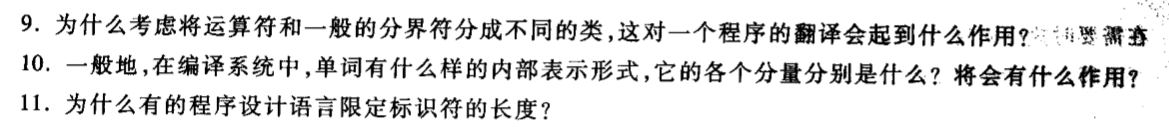
1. 抽象出一类问题, 进行形式化描述才能自动计算
2. 语言是无穷多的, 而描述是有穷的, 否则编译系统无法处理他们
   * 为空则无意义
   * 有穷是为了编译系统可处理
3. 不是, 随机生成的就不行
4. 有效的划分句子结构然后可以形式化的分析句子
   * 终结符号: 起结束作用, 表示语言的句子中出现的字符, 无法派生
     + 终结符号确定了这个文法产生的所有句子中出现字符的界限
   * 非终结符: 非终结确定一个语法范畴，包含了了一个符号的所有推导结果
   * 表示该非终结符能产生的语言
   * 表示该文法能产生的语言
   * 句子: ; 称为产生的一个句子 不含语法变量
   * 句型: , 如果, 称是产生的一个句型 可能含语法变量
     + 句子一定是句型
     + 句型不一定是句子
   * 短语 $\exist \alpha, \beta , \gamma \in (V\cup T)^\*, S\Rightarrow^\* \gamma A \beta, A\Rightarrow^+ \alpha$ 称是句型相对于变量的**短语**
   * 简单短语: (直接短语) , 称称是句型相对于变量的**直接短语**
   * 句柄: 的**句型的最左直接短语**称为句柄
   * 推导/派生: 产生式右部替换左部
   * 归约: 左部替换右部
   * 最左归约: 规范归约
   * 最右推导: 规范推导
   * 规范句型: 规范归约, 规范推导产生的句型
   * **句柄与规范归约有什么关系**? 规范归约=归约句柄
5. 由字符串可派生出的语言
6. 终结符, 非终结符, 产生式集合, 开始符号, 有无二义性



1. 递归可实现有限描述无穷
2. 4类
   1. 0型文法
   2. 1型文法: , 均有 成立
   3. 2型文法: , 均有 且成立
   4. 3型文法: , 均有如下形式 或者
   * 左线性文法:
   * 右线性文法:
   * 有些二义性文法是可以消除二义性的
   * 而语言二义性无法消除
   * 文法是语言的一个有穷描述
   * 文法等价表示生成的语言相同
   * 一种语言可有多种文法描述, 语言具有二义性
3. 叶结点与中间结点的排列关系以及派生/归约过程
4. 避免一颗语法树中出现不必要的标记为 的节点.

# 3. 词法分析



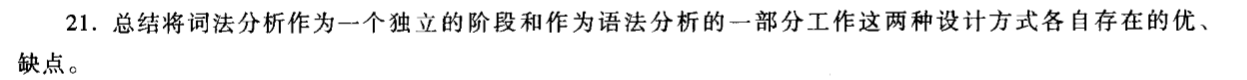


* + 字符: 源程序的最小单元
  + 单词: 独立意义的基本语法单位
  + 单词的属性值由字符构成, 但单词又含种别
  + 字符: 源程序的最小单元
  + 单词: 独立意义的基本语法单位
  + 单词是程序设计语言中具有独立意义的最小语法单位
  + 字符: 构成源程序的最小基本单元

1. 读取速度一致
2. ???
3. 读入表示源程序的字符流, 按照程序功能等价的要求, 将其转换为对应的单词序列, 并剔除其中的空格, 注释等不影响程序语义的字符.
4. 语法结构由独立的单词构成
   * 关键字: 多用来作为语句的标识
   * 保留字: 关键字在程序中只能用于规定意义的表示, 而不可以用于其他用途
   * 符合人类的思维习惯, 尽量无二义性
   * 二者意义不同, 运算符表示操作, 分界符为区分单词.
   * 简化操作, 明确语义
   * (种别, 属性值)
   * 种别表示相应单词所属的内部
   * 属性值表示相应单词的值
5. 当初内存较小, 且实现方便

fig:

* + 最坏情况下将可识别的单词长度增加
  + 平滑字符流, 即当超前搜索时, 避免等待新的读入而降低效率.

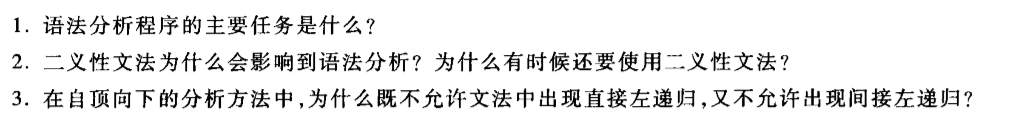


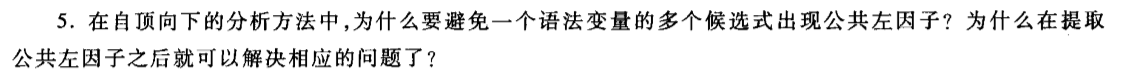
* + 独立的阶段
    - 优点: 简化编译器设计使编译程序结果更简洁, 清晰和条理, 提高编译器效率, 增强编译器的可移植性
    - 缺点: 遍数增加
  + 语法分析的一部分
    - 编译程序紧凑, 比较简单
    - 功能受限

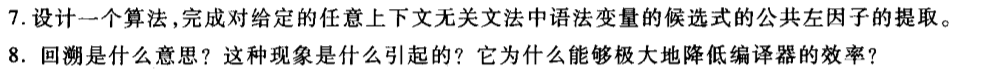
fig:

1. 根据正则表达式构造转换图的表格形式以及使用该表格的驱动程序.
   * 对识别规则用到的文件和变量进行说明
   * 词法的识别规则
   * 以文件的形式

# 4. 自顶向下语法分析

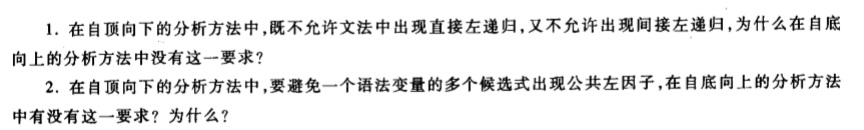




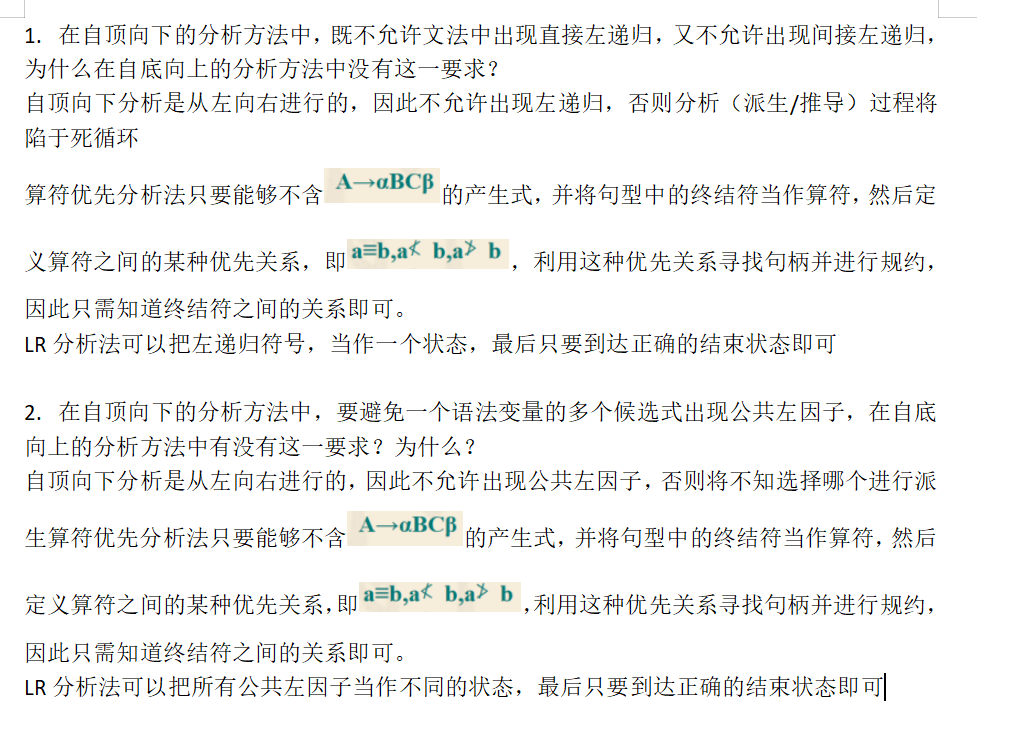


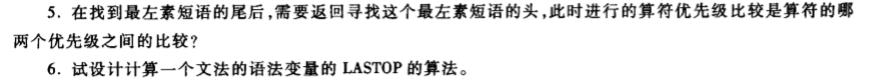
1. 分析出输入串的语法结构
   * 一个输入串对应至少两个语法结构, 无法确定唯一的语法结构.
   * 二义性文法形式简单, 语法树深度较低.
   * 直接左递归: 因为自顶向下每一步推导都选择A的某个候选式, 若有左递归则陷入无穷推导.
   * 间接左递归: 进行多次推导后, 仍陷入无穷推导
   * 为啥右递归🆗呢? 输入指针已经变化了,不会陷入无穷派生
   * 公共左因子: 无法保证分析的确定性, 因为有多个候选式满足条件
   * 提取后: 提取公共左因子后, 剩余部分一定是可区分的, 可保证分析的确定性
2. 无法确定下一个符号. 可保证分析的确定性.
   * 当前分支不正确, 返回到分支起始点,选择另一个分析继续分析.
   * 分析的不确定性, 多个满足条件的候选式
   * 最坏情况下需要遍历每一个分支才能完成分析
3. ...
   * 当前分支试探不成功, 就需要退回上一步推导, 看A是否还有其他的候选式
   * 多个候选式存在公共前缀
   * 最坏情况下需遍历所有候选式可能的派生

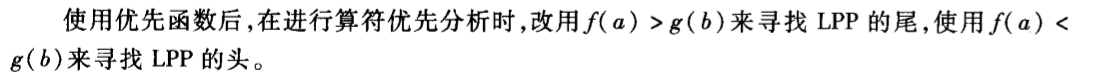
# 5. 自底向上分析



* 1. 自底向上分析寻找输入串的规范归约, 最终归约为文法的开始符号. 在左递归文法中仍可读取输入字符.
  2. 自底向上分析中, 如果当前句型的一个最左子串和某个产生式的右部相匹配, 则进行归约. 公共左因子并不影响这一过程.

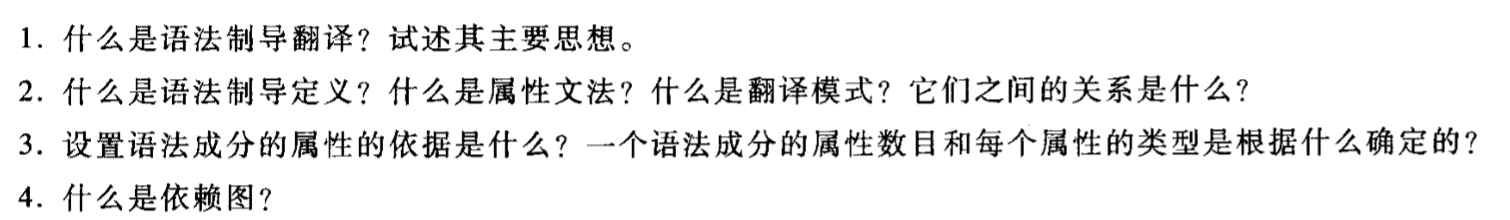


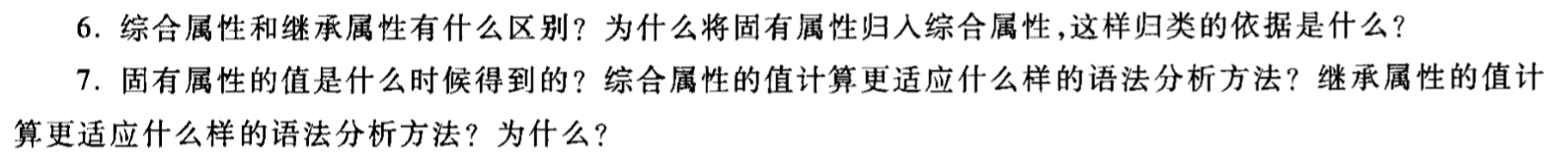


1. 

* fn insert(A,a){  
   if not F[A,a] {  
   F[A,a] = true  
   push(A,a)  
   }  
  }  
  fn LASTOP(G){  
   for (A,a) in V x T{  
   F[A,a] = false  
   }  
   for A->...a in P 或 A-> ...aC in P{  
   insert(A,a);  
   while 栈非空{  
   pop(B,a)  
   for A->...B in P{  
   insert(A,a)  
   }  
   }  
   }  
   for A in V{  
   LASTOP(A) = {}  
   }  
   for (A,a) in V x T{  
   if F[A,a]{  
   LASTOP(A) = LASTOP(A) ∪ {a}  
   }  
   }  
  }

# 6. 语法制导翻译与属性文法





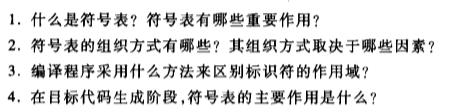
* + 语法制导翻译: 将静态检查和中间代码生成结合到语法分析中进行的技术
  + 主要思想: 在进行语法分析的同时完成相应的语义处理
  + 语法制导定义: 附带有**属性**和**语义规则**的上下文无关文法.
    - 即对应每一个产生式编制一个语义子程序, 当某个产生式获得匹配时就调用相应的语义子程序进行语义检查与翻译. <u>适合在完成归约的时候进行</u>
    - **属性是与文法符号相关联的语义信息**
    - 语义规则是产生式相关联的语义动作
  + 属性文法: 没有**副作用**的语法制导定义
  + 翻译模式: 将属性与文法符号相关联, 并将语义规则插入到产生式的右部来描述语言结构的翻译方案
  + 语法制导定义是各语法结构翻译方案的逻辑描述, 而翻译模式是翻译方案的一种实现形式
  + 首先根据语言结构的语义来设计该语言的语法制导定义, 再根据语法制导定义及其所采用的语法分析方法来设计合适的翻译模式
  + 依据: 属性代表与文法符号相关联的语义信息, 例如数字, 类型, 值, 内存地址和中间代码序列
  + 属性数目和每个属性类型: 基于语言结构的语义要求用到的数据结构

1. 依赖图其实就是一个有向图, 用于描述分析树中节点的属性和属性间的**相互依赖**关系, 称为分析树的依赖图.
   * 每个属性对应依赖图中的一个节点, 如果属性依赖于属性, 在图中
   * 综合属性: 节点属性值通过分析树中该节点或其子节点的属性值计算出来的
   * 继承属性: 节点属性值通过该节点, 该节点的兄弟节点或父节点的属性值计算出来的
   * p232
   * 固有属性: 通过词法分析直接得到的属性综合属性.
   * 依据: 固有属性通过词法分析直接得到, 不依赖兄弟或父节点
   * 固有属性的值: 词法分析时
   * 自底向上: 不依赖于父节点和兄弟节点
   * 自顶向下: 需要父节点或兄弟节点的信息

# 7. 中间代码表示

1. 在编译过程中为什么要生成中间代码?
   * 翻译的效率和代码的优化

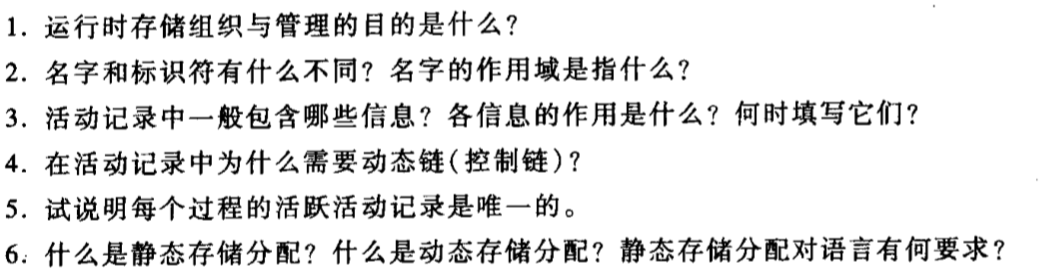
# 8. 符号表管理



* + 记录源程序中各种名字的类型和属性信息有关信息
  + 协助语义检查, 为名字分配地址
  + 线性表
  + 散列表
  + 插入, 查找的时间效率

1. 程序块结构的符号表
2. 为名字分配地址的主要依据

# 9. 运行时的储存组织



1. 负责处理诸如各种对象的储存空间的布局和分配问题, 目标程序所使用的变量访问机制, 过程之间的链接, 参数传递以及与操作系统,输入输出设备和其他程序的接口问题
   * 名字: 编译时的名字
   * 变量: 运行时该名字所代表的内存位置
   * 标识符: 字符串, 用于指示数据对象, 过程, 类或对象的入口
   * 名字的作用域: 在该区域x的引用均指向x的这一声明
   * 声明: 描述了事物的类型
   * 定义: 给出值
2. 活动记录: 过程的每个活动所需要的信息
   * 临时变量: 临时数据对象, 如表达式存放中间结果的临时变量
   * 局部数据: 相应过程的局部数据
   * 保存的机器状态: 保存进行过程调用前机器的状态信息
   * 访问链静态链: 访问存放在其他活动记录中的非局部数据
   * 控制链动态链: 指向调用者的活动记录
   * 返回值: 存放被调用过程返回给调用过程的值
   * 实在参数: 存放调用过程中提供给被调用过程的参数
3. 当过程调用结束返回时, 利用动态链可得到调用前的活动记录的起始地址
4. 若不唯一, 出现过程对应多个活跃活动记录, 无法确定使用哪一个活动记录
   * 静态:在编译时就能安排好目标程序运行时全部数据空间, 并能确定每个数据项的地址
   * 动态:编译时无法确定过程何时运行, 运行所需的储存空间的对象
   * 静态要求:
     + 数据对象的长度和在内存位置在编译时已知
     + 不允许过程的递归调用
     + 数据结构不能动态建立

# 10. 代码优化

fig:

