**左递归**只会对**自定向下（如LL1）**的语法产生二义性

符合文法的string可能对应不同的parsing tree（如四则运算的加、乘顺序）

一个**unambiguous grammar**只对应一棵parse tree

**LALR1**文法只有reduce-reduce conflict（当对映LR1文法没有reduce-shift conflict）

Finding the next handle is the man task of a LR parser（**handler句柄**，为生成式右侧可归约的串）

Parse tree不能反映一个string的产生顺序（derivation，如等高parse tree左右可任意选择生成先后）

Left-recursion常常暗含了**left associate（左结合）**

**YACC**使用的文法是LALR1

**Left factoring**（LL1文法中提取左公因子的操作）

viable prefix(es)**可行前缀**，句柄中任意不超句柄长度的子串

LL1 table有match、generate、accept的概念

Parser(解析器，进行**文法分析(**通常指语法分析))生成 syntax tree（parser tree的一种，更加简洁，冗余信息更少），但parser tree即可反映derivation的顺序

An **LR(1) parser** can detect errors **earlier** than an **LR(0) parser**.

**semantic analysis**语义分析，接受输入：抽象syntax tree

作文法分析时可以先看看能不能**化简**

**Follow集合**计算时特别注意**空集**，follow列填写parsing table时注意是写**产生null的对应文法**

**Tiger语言**

变量定义与C一致，换行符输出与C一致(‘\n’)

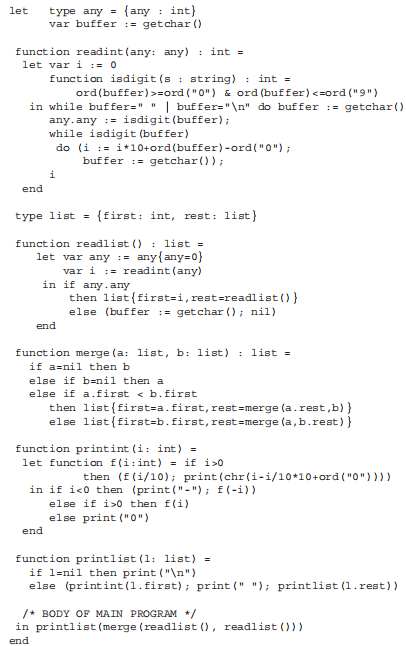
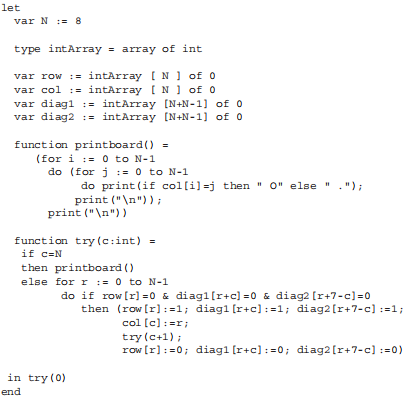
**数组**记录从 0 开始，循环有**break**操作（没continue）

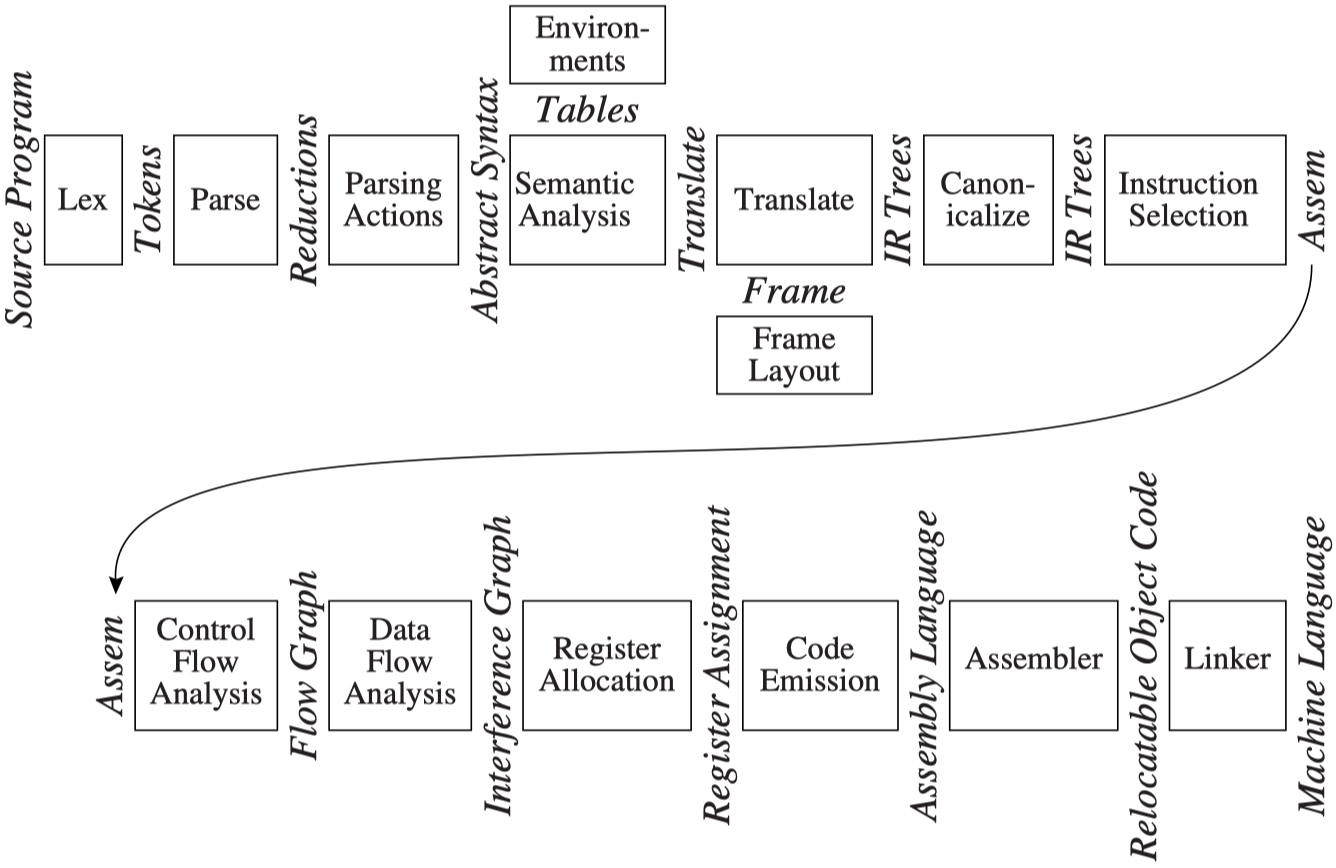
Var a := 1 、Var a: int := 1(**nil**只能用此方法，nil匹配所有类型，但类型需要明确)

Function abc(a:int[**参数**]):int[**返回值**] = 函数体

**Let**用于定义内、**In**表示实际的运算内容、**End**与let匹配使用，表示let块结束

不等号 **<>** 相等号 **=** 并 **&** 或 **|**

**没有连等**操作、**While** exp1 do exp2（根据for推导）

标准函数：**print()**输出字符串、**getchar()**获得一个字符，文件尾返回空、**ord(string):int**返回第一个字符ASCII码、**chr(int):string**根据ASCII码转变为字符串（超值报错）、**size(string):int**获得字符串个数、**substring(string,int,int):string**获得制定字符串从（第二个参数）开始长（第三个参数）的子串、**concat(string,string):string**获得串联字符串、**exit(int)**以指定状态码退出、**not(int)**判断是不是0

Source language and target language

Phases：一个或多个模块、interfaces：模块间接口

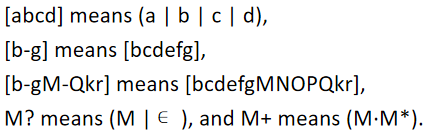
Parse阶段只是分析，parse action阶段建立抽象tree

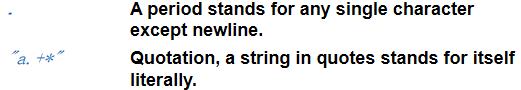
Frame layout就是栈空间的分配；Canonicalize简化、清楚条件分支；code emission:替换寄存器为真实名称

两阶段：front end：do analysis、back end：do synthesis

**Lex**分析：接受输入流、生成token流、解决空格注释

Non\_token：注释、空格、**预处理指令**

**Lex**对**大小写敏感**，**\_**被当作字母



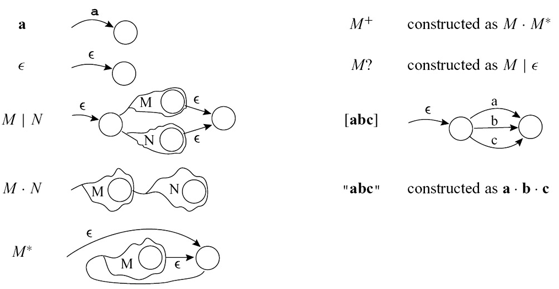
Regular中“**空**”与**{“”}**等价

**("--"[a-z]\*"\n")|(" "|"\n"|"\t")+** tiger空格注释匹配

**Longest match**（token最长匹配）、**rule priority**（同长情况下根据rule优先级匹配）原则

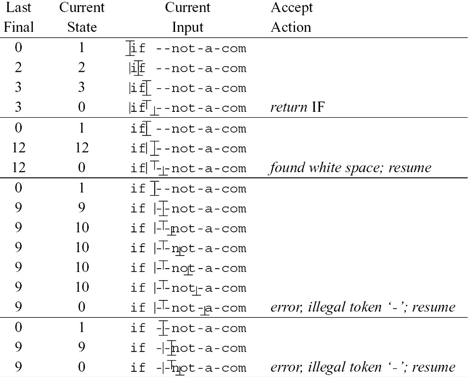
Lex进行token匹配时，需要maintain **last匹配**string

**当前位置**，**开始匹配位置**

Thompson’s Construction（regular->NFA）**基本单元**：

Distinguishes（可区分）状态，同一string不同结果

状态等价--两个状态发出的每一条edge指向相同的next state

计算state等价的算法：不停的划分类（只有类中state edge转移与其他类中state不同时，划分新类）

**Flex结构：**

%{ definitions %} C语言库及辅助函数定义

此处可以添加等价定义如：digit [0-9]

%% { rules } %% 匹配token的正则表达式

[yytext表示返回的token、yyleng记录长度]

{ auxiliary routines} C语言执行函数（可融入生成部分）

Parse阶段接受token流，输出abstract syntax

Derivation：从start symbol开始生成string

Derivation的过程可以用parse tree 表示

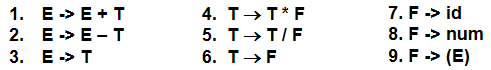
Left-most derivation(每次都展开最左侧的非终结符)

A **derivation** defines a parse tree

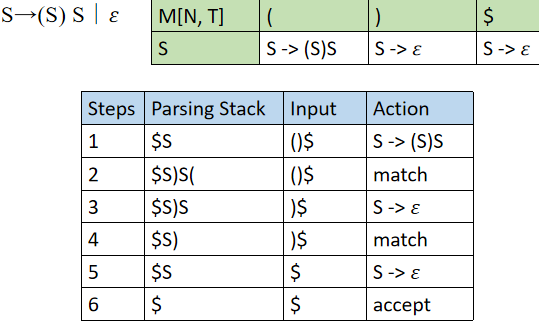
one **parse tree** may have many derivations

A **grammar is ambiguous（二义性）** if it can derive a string with two different parse trees

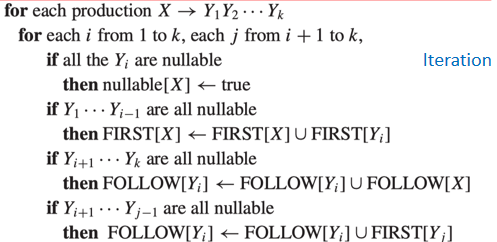
四则运算的left-association和**优先级**(precedence)

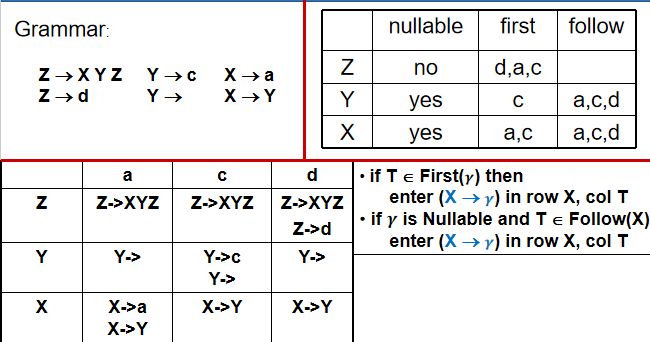
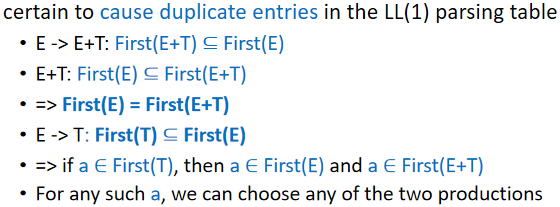
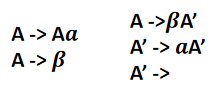
递归下降（recursive descent parsing）

top-down parsing，can parse LL(1)[Left-to-right parse; Leftmost-derivation; 1 symbol look ahead]

**Parsing Stack**列需要从右向左看

**FIRST(γ)** is the set of terminals that can begin strings derived from γ；**OLLOW(X)** is the set of terminals that can immediately follow X.

**LL（1）Parse table**法则（注意Z那行，**完整性**）

**Z->XYZ也能产生first(d)**

LL(1) 需要消除 left recursion

左递归修改策略

Left factoring提取左公因子也可以

Recover from error：inserting（可能无法终止）、deleting（skip token直到合法，一定会停止--EOF）

LR(k) Left-to-right parse、Rightmost derivation

**bottom-up parsing**：reduce the string to the start symbol

有shift、reduce、goto、accept、error五种表现

State stack同symbol stack一样根据action被操作

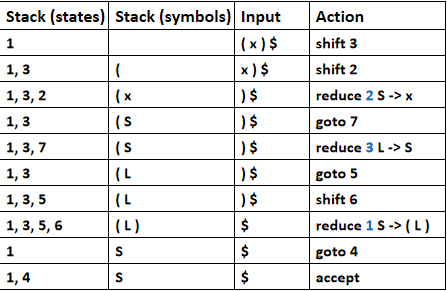
RHS：right hand side（LHS同理），表示产生式右侧

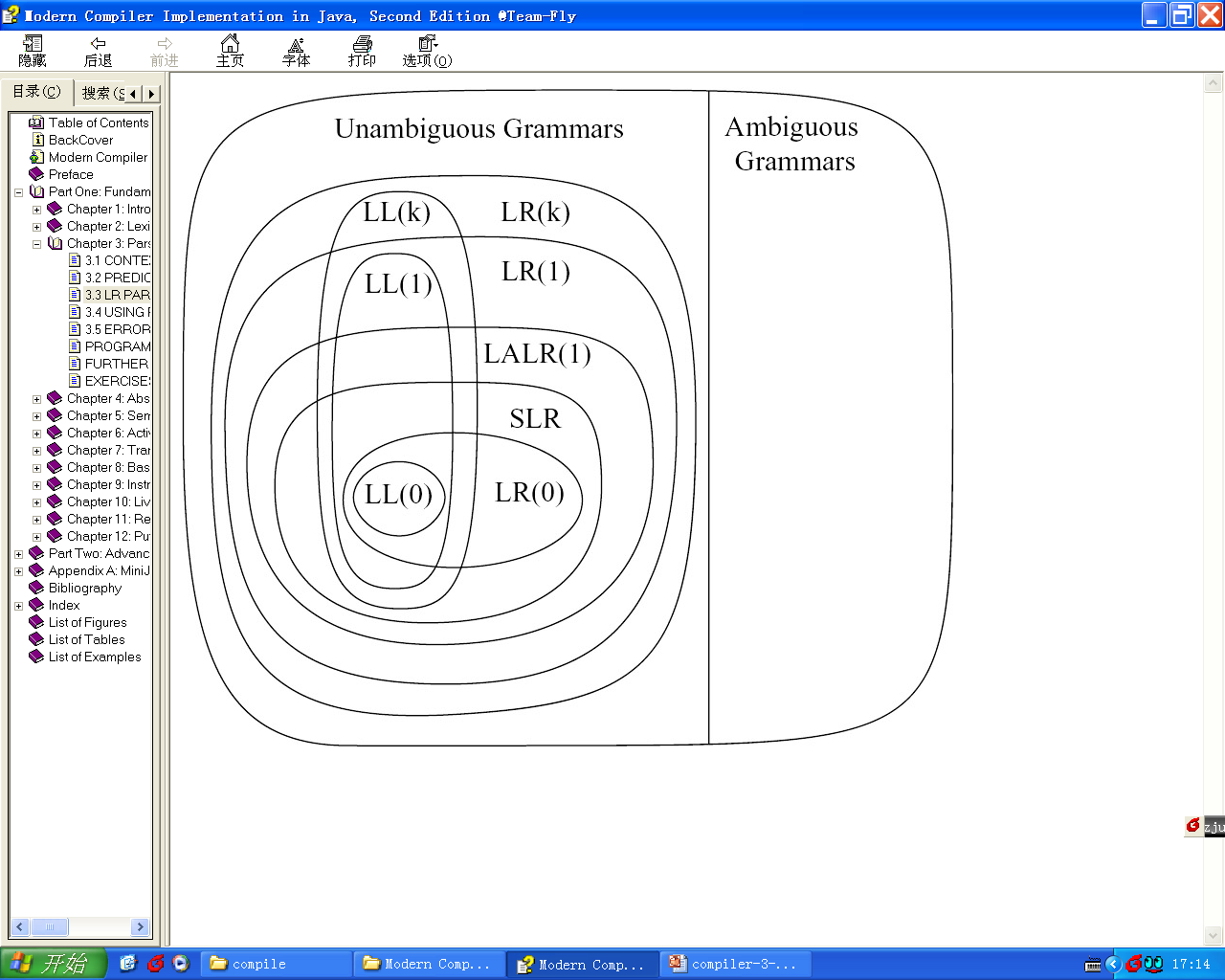
**LR(0)** parsing；state：S’-> . S$（再打开S，不停迭代）

**SLR** parsing（simple LR parsing）：Put reduce actions into the table only indicated by the FOLLOW set（非全部符号）

**LR(1)** parsing: An item (A → α.β, x) indicates that the sequence α is on top of the (symbol) stack, and at the head of the input is a string derivable from βx.

**LALR(1)** parsing: the parsing table is made by merging any two states whose items are identical except for look ahead sets in the LR(1) parsing table.

LALR可能存在reduce-reduce conflict

**YACC**结构

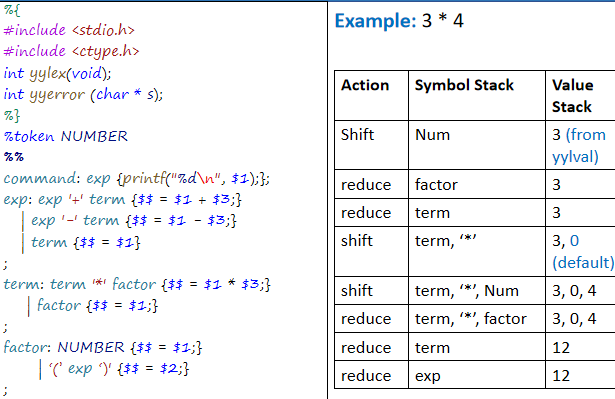
%{definitions%} 同flex

%union { double val; char op;} token可存储类型（或）

%token <op> 定义可能的token（terminal，non-terminal自动被声明为token，<>对映于union中类型）

%% {rules} %% 语法操作（可用 {} 来声明action code）

{auxiliary routines} 其余辅助函数error、main函数等

**yylex**会返回token或0，**yylval**会返回token语义值

Yacc resolves **shift reduce conﬂicts** by shifting、**reduce reduce conﬂicts** by use rule appears earlier in grammar

%nonassoc EQ NEQ（没有结合性，不能连续运算）

%left PLUS MINUS（left定义左结合）

%left TIMES DIV

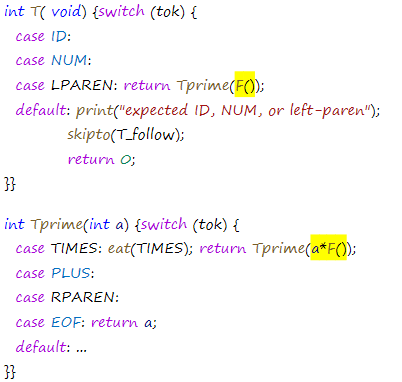
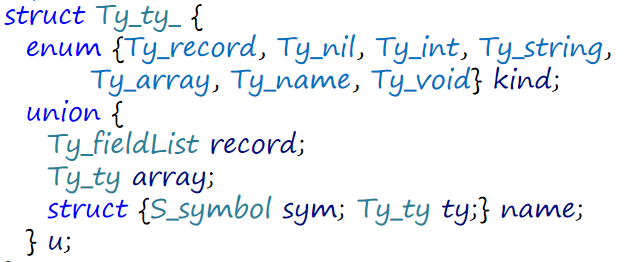
%right EXP（从上至下优先级不断提升）

%prec abc（定义label，在特定位置使用改变优先级）

同优先级左结合先reduce右结合先shift

Local error recovery: 增加一个转换到error的语法，在发生错误时，清空synchronizing（同步） token之间的stack中所有内容，并写入error符号

Global error repair: 找到将源字符串转换为语法正确的字符串的最小插入和删除集，即使插入和删除不在LL或LR解析器首先报告错误的位置。

Burke-Fisher Error Repair：在解析器报告错误点前不早于K token的每个点尝试插入、删除或替换每个token。

一般来说，如果修复将解析器带到它最初卡住的地方之后4个标记，即表示修复已足够好。

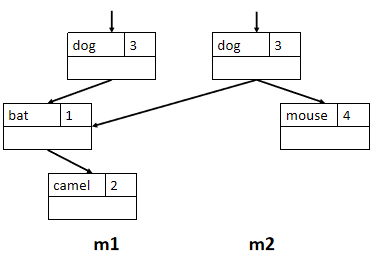
要求：Maintain two parse stacks: the current stack and the old stack，Keep a queue of K tokens(两个stack之间)

连乘的语义分析：

Concrete parse tree记录了具体的grammar rule reduce

Abstract syntax tree即具体语法树的化简（yacc生成）

**Symbol table**：mapping identifiers to their **types** and **locations**（**scope**）

**Functional style**函数式：旧环境不发生变化（仍可被访问），常使用二叉树结构（如果我们在树深度d处添加新节点，只需创建d个新节点即可--对应root到新节点的父节点）

**Imperative style**命令式：旧环境发生变化以转变为新环境（多为散列表结构）

**Tiger使用命令式symbol table**，维护type table和value table（变量和函数（分别记录出参数与返回值类型））

**destructive-update**环境，所以使用begin\_scope和end\_scope作为划分符，配合额外stack mark变量

[] 表示方括号内包含的项可被重复 0 次或 1 次

 {} 表示花括号内包含的项可被重复 0 次或多次

Tiger中不同**类型定义**占用空间不同，即使结构一致也不能相互赋值

初始化Ty\_Nil类型表达式必须受到Ty\_Record类型约束

处理函数类型时，第一遍仅处理（记录）函数头，第二遍再详细处理函数体

一组相互递归的类型声明中，每个cycle都必须依赖一个记录或数组声明。