**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’)

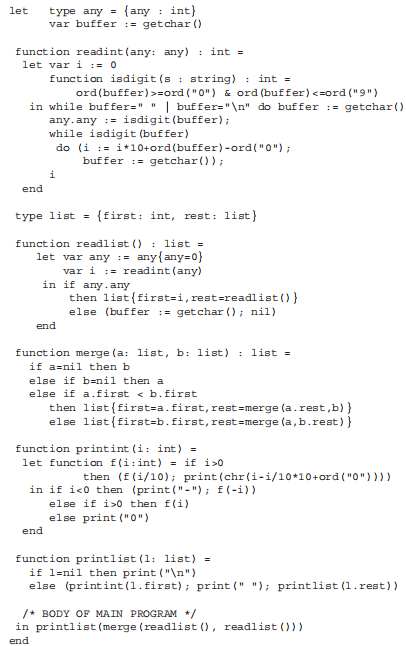
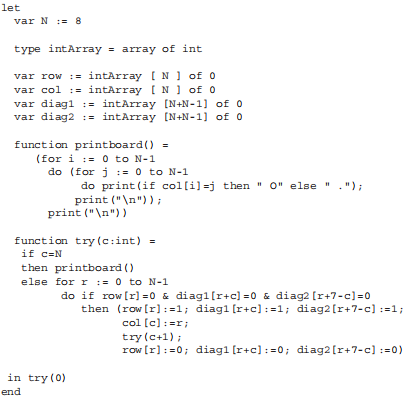
**数组**记录从 0 开始，循环有**break**操作（没continue）

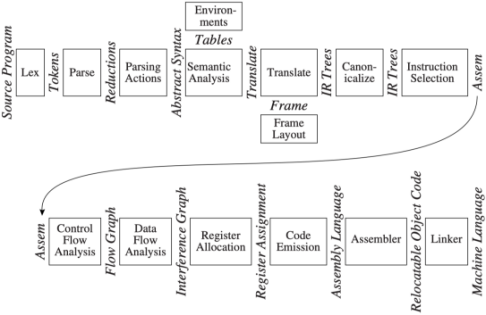
Var a := 1 、Var a: int := 1(**nil**只能用此方法，nil匹配所有类型，但类型需要明确)

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阶段语法分析，parse action阶段建立抽象tree

Frame layout就是栈空间的分配；Canonicalize简化、清除条件分支与副作用；code emission:替换寄存器为真实名称；translate阶段进行IR树的构建

**control flow**关注程序的控制结构流，**data flow**控制记录具体信息变量是否仍处liveness状态

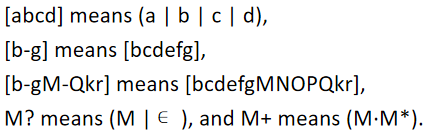
在“虎书”中，string类型在定义后不会再被改变

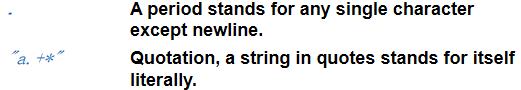
两阶段：front end：do analysis、back end：do synthesis

**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

注释、宏、预处理语句、空格、制表符、空行都**不属于**词法分析的token



Regular中“**空**”与**{“”}**等价;**优先**：Kleene closure >“连接” >“或”；空也可以省略不写，如（a|）

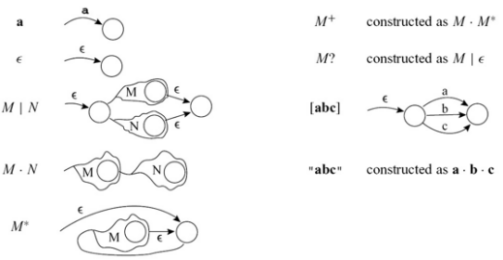
+ :至少有一个，？:一个或为空，. 表示 any single character except newline；

**("--"[a-z]\*"\n")|(" "|"\n"|"\t")+** tiger空格注释匹配（注释要以换行符结尾）

**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

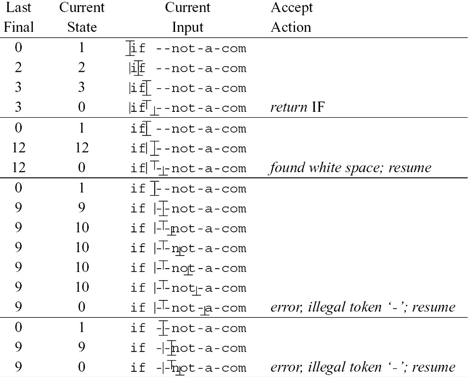
**当前位置**，**开始匹配位置**

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

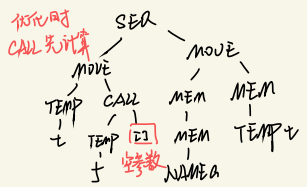
**Distinguishes**（可区分）状态，同一string不同接收结果

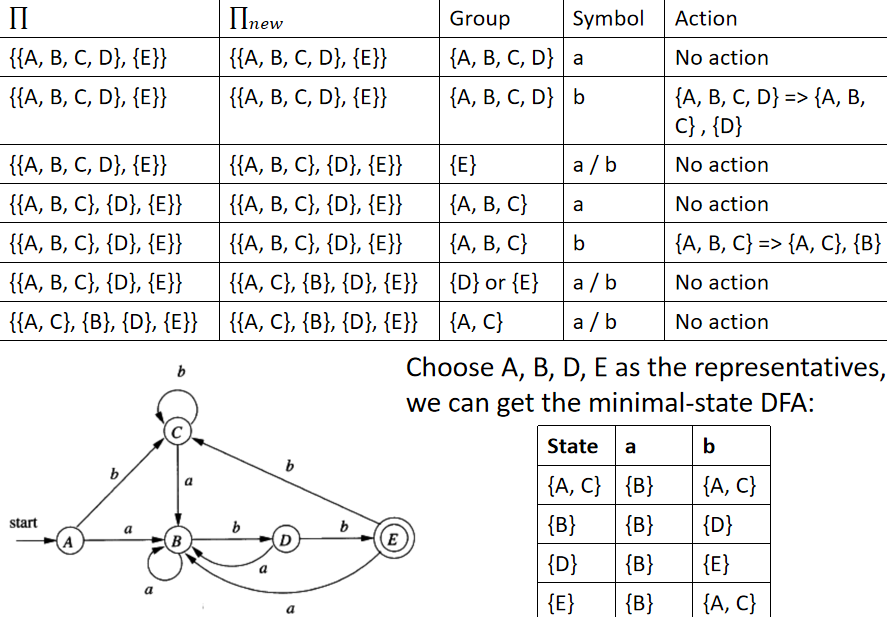
等价：两个状态发出的每一条edge指向相同的next state

transition matrix记录DFA（需添加最后state到rule映射）

**state等价（DFA化简）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）

在lexical的**RE与NFA的转换**过程中，final state需要额外标注终止时对映的token类型

IR tree优化，先计算并存储CALL结果



**Flex结构：**head表示ending state，tail表示start↑

%{ 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]；状态声明

%% { rules } %% 匹配token的正则表达式，表达式后{}中C语言可在规则匹配后执行

{ auxiliary routines} C语言执行函数（）

yylex()自动读取下一个token；yyleng记录当前token长度；yytext记录当前token；yyin指定输入源；yyout指定输出源；yylineno记录当前token对映行数（使用前需键入 %option yylineno，启用此保留字）；yylval对映Yacc中union定义的结构体类型，[yylval.结构 = xxx]

跨行注释匹配**（"/\*"[^\*]\*"\*"+([^\*/][^\*]\*"\*"+)\*"/"）**；[^\*] 表示任意非\*字符，同理 [^\*/] 表示任意非 \*、/ 字符

Derivation：从start symbol开始生成string（只有终结符）

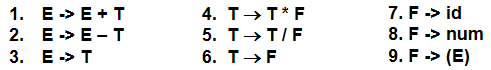
Derivation过程可以用parse tree 表示

Left-most derivation(每次都展开最左侧的非终结符)

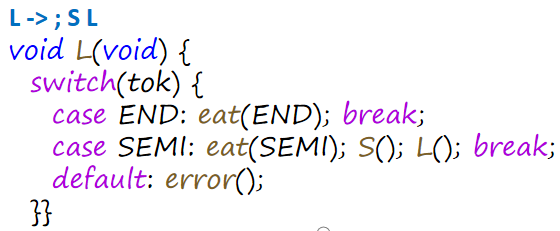
A **derivation** defines a parse tree；one **parse tree（**terminal在leaf上，non在内部节点**）**可能对映多个derivations

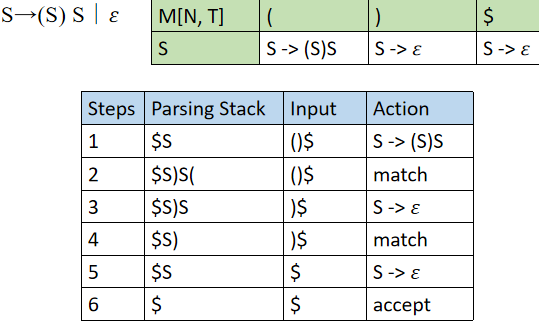
A **grammar is ambiguous（二义性）** 不同derive产生多棵parse tree（同string不止一个left/right derive [歧义]）

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

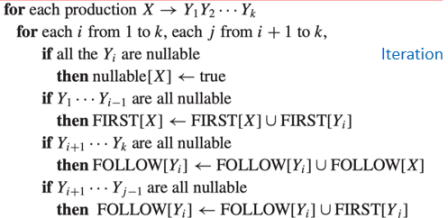
**递归下降（recursive descent parsing）**

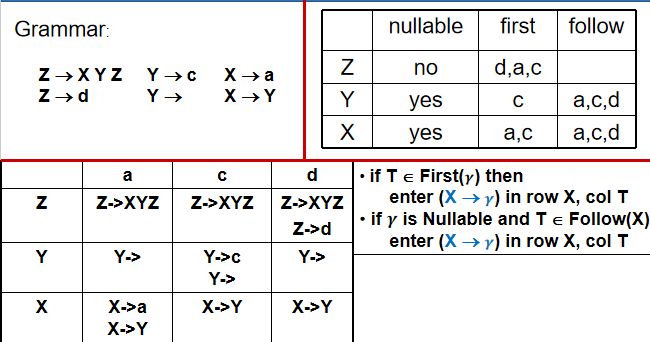
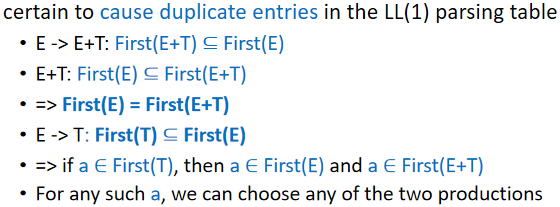
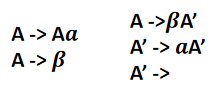
top-down parsing，can parse LL(1)[Left-to-right parse; Leftmost-derivation; 1 symbol look ahead]



**Parsing Stack**列需要从右向左看

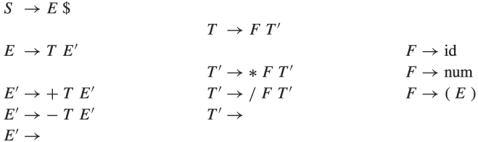
**FIRST(γ)** begin string derived from γ；**FOLLOW(X)** terminals immediately follow X.（注意nullable set，A -> B 则有follow(B) 包含follow(A) ↓第三个if-else）

**LL（1）Parse table**法则（注意Z那行，**完整性**）

1. **>XYZ也能产生first(d)** 左递归冲突证明↓

若在predictive parsing table中有**duplicate entries**（冗余信息），非LL(1)文法，不能用predictive parse

四则运算（含括号）修改为例



LL(1) 需要消除 left recursion（理由分析见上，策略见下）

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）

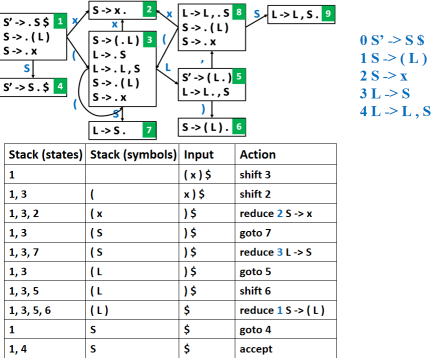
LR(k) Left-to-right parse、Rightmost derivation

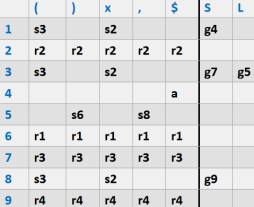
**bottom-up parsing**：将string归约到start symbol

有shift、reduce、goto、accept、error五种表现

State stack同symbol stack一样根据action被操作

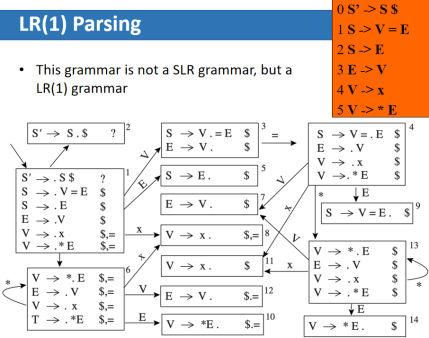
RHS：right hand side（LHS同理），表示产生式右侧

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

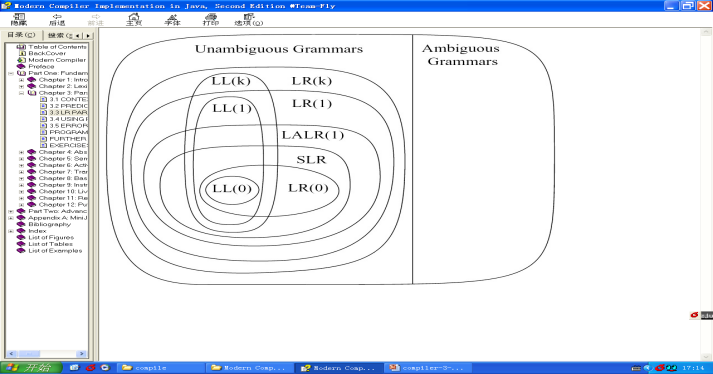
注意上图中states stack和symbol stack的同步变化

Table同一项既有reduce，又有shift -- reduce-shift冲突

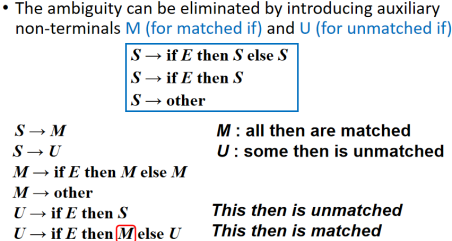
**SLR** parsing（simple LR parsing）：Put reduce actions only indicated by the FOLLOW set（文法左侧non-terminal的follow set），仍可能存在reduce-shift冲突

**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（特别注意，β为非总结符时要计算first set）. The look ahead token of LR(1) 比SLR更强

**LALR(1) parsing**: parsing table made by merging any two states whose items are identical except for look ahead sets in the LR(1) parsing table（保留LR(1)中信息并集）

****LALR可能存在reduce-reduce conflict

**IF-ELSE选择匹配问题（shift-reduce conflict）**



**YACC结构** LALR实现，处理**shift-reduce冲突默认shift，reduce-reduce看优先级** ↓%{definitions%} 同flex

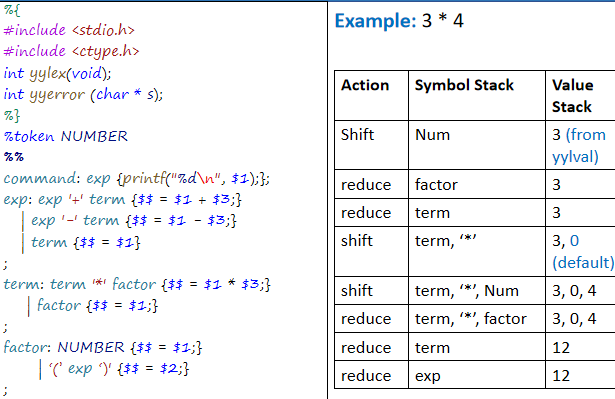
%union { double val; char op;} token可存储类型；若不定义union，则类型与YYSTYPE（INT）相同

%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）

%start Calc 起始字符（未定义则第一语法左则符号默认）

%% {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）

%left A（可以搭配%prec A使用，%prec A可视作占位符，A不会作为token被返回，自定义优先级）

The **priority of shifting** a token is given by this token；

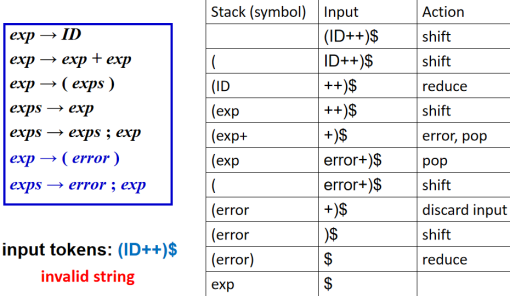
The **priority of a rule** is given by the last token occurring on the right-hand side of that rule

If the token and the rule have equal priority: %left favors reducing；%right favors shifting；%nonassoc yields an error action

{}中内容为匹配后执行（每个|之后都可有附加动作），**$$**表示左侧non-terminal，**$x**表示右侧第x个符号

yyparse为yacc相关调用函数(会自动调用yylex函数，返回token类型，并用yylval返回附加值，返回0表示token输入结束)，若语法分析成功则返回0，否则返回1

Yacc支持Embedded Actions，在rule内部额外引入“辅助操作”（额外的{} 辅助操作也被视为符号，即后续 $2 会因额外{}的引入转变为$3）

**Local error recovery**: 增加error的语法，在发生错误时，清空synchronizing(同步)token间stack中所有内容，并写入error符号(yacc使用)；注意语义失衡(**当前位置修改**)

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。(global可脱离错误发生位置)

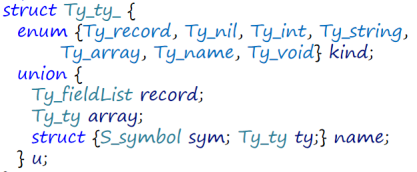
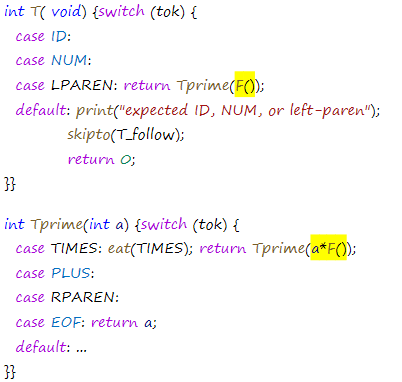
**Burke-Fisher Error Repair**：遇到error时，在symbol queue中的每一个位置尝试所有可能单词及操作，并从老栈开始进行修复，如果修复使parse能（在原位置后）连续进行3 - 4个单词，则认为修复成功

要求：Maintain the current stack and the old stack，Keep a queue of K tokens(两个stack之间原sequence)

Shift获得新单词时，需要push近当前栈，并把当前栈最底部的元素push至老栈中，并照常进行reduce动作

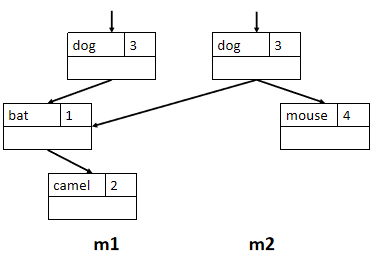
连乘的语义分析：

Concrete parse tree记录了具体的grammar rule reduce，具体语法树会饱含大量冗余信息，并极度依赖grammar的实现；**one leaf for each token of the input and one internal node for each grammar rule**



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**）将变量名与类型绑定

****Environment相加时，右侧的table会覆盖左侧的table（X + Y for tables is not the same as Y + X）

**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）（若复制整个hash table，内存消耗过大）

**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imperative 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））

仅用一张hash表实现，新作用域中变量产生时加入hash-bucket结构，作用域消失时更新删除hash-bucket结构中对映变量内容

**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

因String间的比较开销较大，考虑string转变为**symbol**类型进行比较；Symbol便于产生hash-key，且“比较”便捷（相等/大小比较）**L-value，**可被读和赋值的变量

**两个类型的symbol table**：type bindings for types；value bindings for variables and functions；从而类型a和变量a或者函数a可以同时存在（tiger实现见左图←）

[] 项可被重复0或1次；{} 项可被重复0或多次

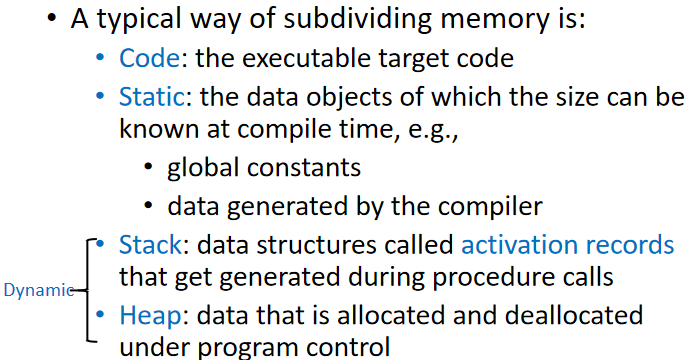
← Ty\_record, Ty\_array隐式记录对映object地址

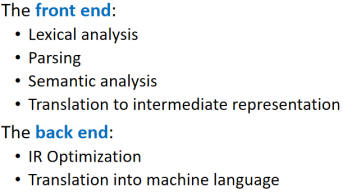
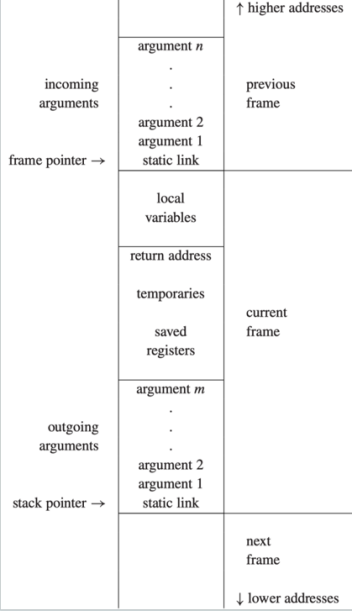
Tiger中不同**类型定义**占用空间不同，即使结构（甚至属性名、类型）一致也不能相互赋值

初始化Ty\_Nil类型表达式必须受到Ty\_Record类型约束

递归声明：type list = {first: int, rest: list}，可先使用S\_enter (tenv, list, **Ty\_Name**(list, NULL)) 占位（再照常定义覆盖）

一组相互递归的类型声明中每个循环必须至少含有一个record或array类型声明作为end（不会打开record检查）

处理函数时，第一遍遍历先记录所有函数名称、参数及返回值，body部分等第二遍时再创建新环境进行处理



如果一种语言既支持函数内函数定义，又支持返回值为函数，则其不能用stack来存储函数活动

函数调用时**参数需写入内存**的条件：the variable will be passed by reference（引用）；嵌套的内层函数可访问外层函数的变量；过大的变量，无法放入register中；Array数组变量 （**activation record也称为stack frame**）

**逃逸变量（escape variable）**：引用传递；地址被获取(如&)；内层函数获取外层函数变量(遍历整棵抽象语法树寻找并记录)

**Block Structure**:支持函数嵌套定义，且内层函数可以访问外层函数变量 （**针对enclose function↓**）

**Static link 静态链接：**每次调用函数，都会传递一指针(current **FP**)，指向**最近活跃**的函数帧栈(不一定是嵌套层)

**全局array**：display(数组)，在位置i存储一指针，指向静态嵌套深度为i的最近输入过程帧（frame）

**lambda lifting**：需要被内层函数使用的变量直接作为参数传递给内层函数

函数调用时，会发生“**shift of view**”，可能需额外代码克服此问题(callee视角register、stack位置同caller视角)

**Tiger**用frame.h分离源语言和机器栈帧(一些源语言不支持嵌套函数定义);使用translate.h分离类型检查和语义转换，translate模块管理static link

中间code使用利于将具体源语言与具体机器相隔离

IR is a kind of abstract machine language

Compiler具体前后端分割（图见前）**中间表示↓**

**三地址编码：**最多三个地址和一个operator(不能让同一寄存器又为源寄存器又为目标寄存器)；使用四元组形式表示，不足用empty or null占位（op放在最前，无标准）

**Expression(有返回值)**：CONST(i)立即数i；NAME(n)符号常量n，汇编语言标签；TEMP(t)寄存器t；BINOP(o, e1, e2)四则、逻辑运算、逻辑左右移、算术右移；MEM(e)地址为e的内容，左侧表写入，右侧表读出；CALL(f, l)调用函数f（常用NAME标签），输入参数列表l（explist引导）；ESEQ(s, e)先执行s的副作用，再计算e的值作为返回值

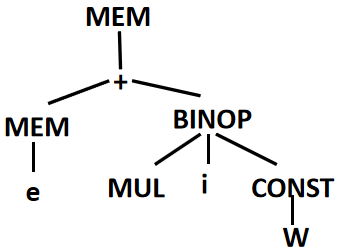
（处理ESEQ优化时，最顶端为ESEQ，其左子为SEQ序列（右子相连），右子为需返回的EXP表达式）

**Statement(无返回值)**：MOVE(TEMP t, e)计算e并把结果放入寄存器t中；MOVE(MEM(e1), e2)计算e2并把结果放入e1地址中；EXP(e)计算e并忽略返回值；JUMP(e, labs) 跳转至地址e或name标签位置；CJUMP(o, e1, e2, t, f)以条件o比较e1，e2的结果，并根据比较结果跳转 t or f；SEQ(s1, s2)连续执行两个statement；LABEL(n)将名称n的常数定义为当前机器代码地址

CJUMP，可能未知t or f的具体跳转地址，可建立对映t or f地址列表，待后续填充

Tiger中数组变量仅是地址，支持数组名之间的直接赋值

取数组元素a[i] 如下。数组名仅是scalar（标量）



Subscripting（下标） and Field Selection（字段选择）

tree language没有二元操作符，-n需要表示为 0 - n 的三元样式，一般以四元组表现(加符号)，不足用**NULL( \_ )**

**If语句有返回值**，为执行的then/else代码段的最后结果

调用函数时，需要将static link作为参数进行传递

tree language中没有SEQ和ESEQ操作，**CALL的parent只能是EXP 或MOVE操作**（一棵树一个CALL操作）

**canonical tree**(no SEQ or ESEQ，CALL父节点限制)

**置顶ESEQ**，注意不同statement间执行顺序不能改变

若statement提前会引起statement与expression的冲突

因存在可能的先后关系，我们保守使用交换；常数、空语句一定支持交换（具体规则见最后）

(tiger中)**Reorder函数**将参数传递链表指针指向的所有操作分离，提取所有statement并合并，消除ESEQ并保持statement语句和expression语句的相对顺序不变；若为seq语句且涉及常数操作，直接删去

不同**CALL连续调用**可能会覆盖register中需要使用的有效值，直接将CALL的返回值记录在temp t中即可

**消除SEQ，**反复以下规则，逐渐将seq后移，最后将一连串SEQ(s1, SEQ(s2, ..., SEQ(sn-1,sn)...))视作普通序列s1, s2, s3...（**CALL父节点只能是EXP或MOVE**）



**基本块(basic block)**内部无label标签和jump跳转，以LABEL开头，JUMP or CJUMP结尾（没有需额外添加，条件判断也算已拥有）return语句不用额外添加

**Control flow（控制流）**关注instruction在程序中运行顺序；忽略变量具体value，假设jump true or false地址均可能被跳转

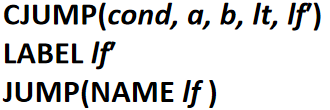
**Trace（迹）**一连串可连续执行的语句（可跨越branch）

一条trace一般无法覆盖整个程序的运行，希望使用尽可能少的trace覆盖整个程序（**默认选择跟f跳转**）

每条trace第一个block可任意选取(选未marked block)

**Trace Schedule**：CJUMP后紧紧跟随f地址的跳转目标

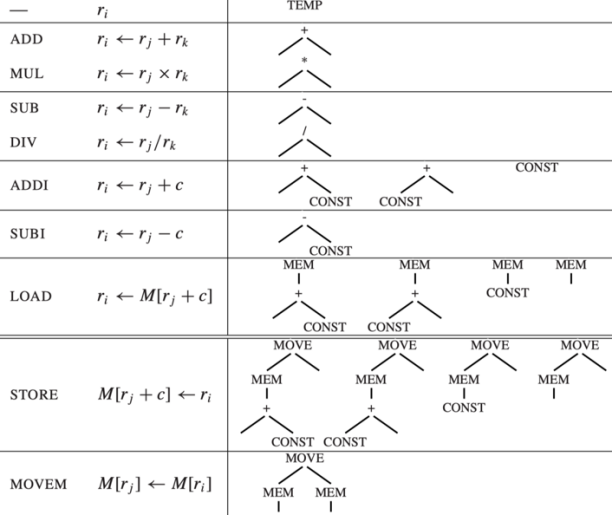
若只能跟随t跳转地址，则反转跳转条件并互换t / f跳转地址，使CJUMP后跟随f跳转地址；若CJUMP后没有跟随任何的t or f标签，则使用如下方法（添label）



频繁一同执行的序列块最好处于同一trace上

Tree language可被看成多部分的组合，称为**tree pattern**

Instruction selection：用最少的tree pattern覆盖整棵树（tree pattern间不能重叠），覆盖过程称为**tiling**

**Jouette architecture(**前6为expression，后2为statement)

**Optimum tiling**：整个tile过程代价最小

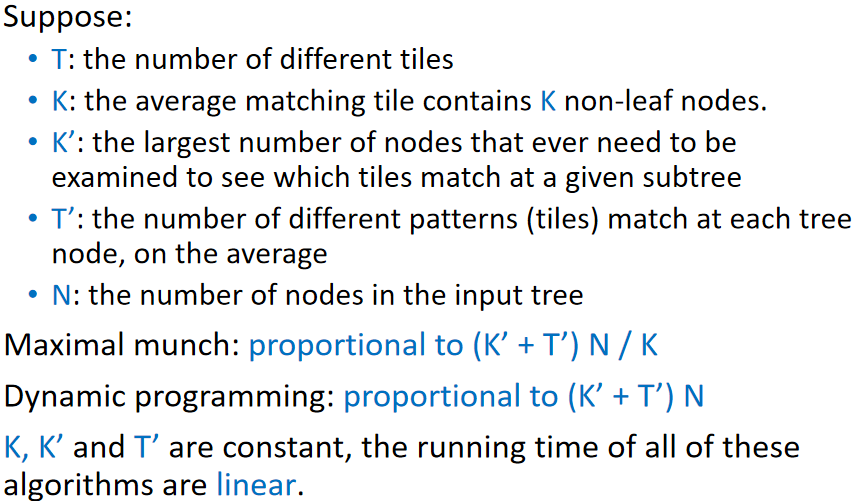
**Optimal tiling**: 没有两个tile可进行合并并收获更小Cost

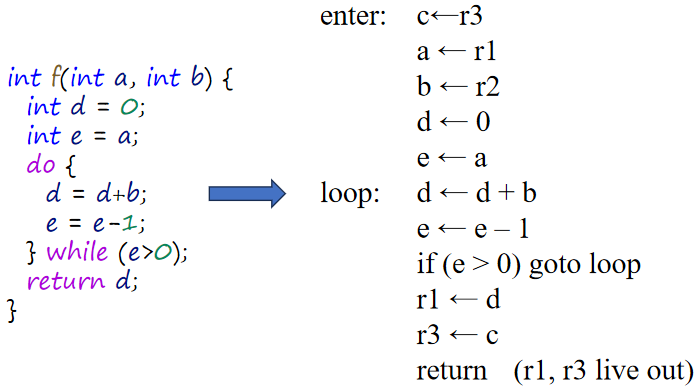
规定：不允许两个相对较小的patter value之和小于它们合并得到的较大pattern的value

Optimum一定是optimal，反之不一定( Mem[]可能+0↓)

**Maximal Munch算法**：从 root开始向下寻找，每次都使用最大tree pattern进行覆盖（同大小tree pattern随机选择）；在**生成具体instruction**时，一般从下至上、从左至右执行，最先完备指令（无需等待其他指令变量结果，顺序特别注意，const需要ADDI生成，ADDI、SUBI不同；乘除法无法直接操作立即数）最先生成（optimal）

**Dynamic programming方法**：Works bottom-up，可找到optimum解；为每一node添加cost，cost表示tile以node为root的sub-tree最小cost，叶子节点cost就是它本身

若address register和value register需区别，则tree pattern需要额外说明与跟踪



（其他图插入，**编程语言程序转化↑**）

A register for address，d register for data（jouette architecture可能需要细致划分不同寄存器）

时间分析（快速匹配，每次匹配只需要K’+T’，不用作乘）

T不同tile数量，K平均每个瓦片的非叶子节点数，K’子树为确定使用何种tile所需检查最大节点个数（近似于最大瓦片大小），T’为每一个树节点都可与多少种tile匹配；

每个tile占用K个节点的前提下，N个节点只需要遍历N/K个node（快速匹配基于switch结构）；在DP中，每个节点都要计算cost，所以开销为N

**CISC**：复杂指令集；**RISC**：精简指令集；在RISC中，optimum tiling和optimal tiling一般没有区别

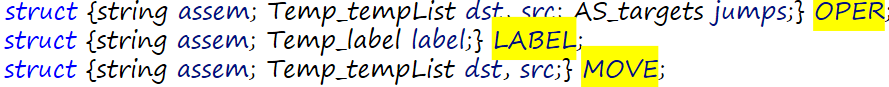
在tiger编译器中，自动丢弃计算得到的高位数据

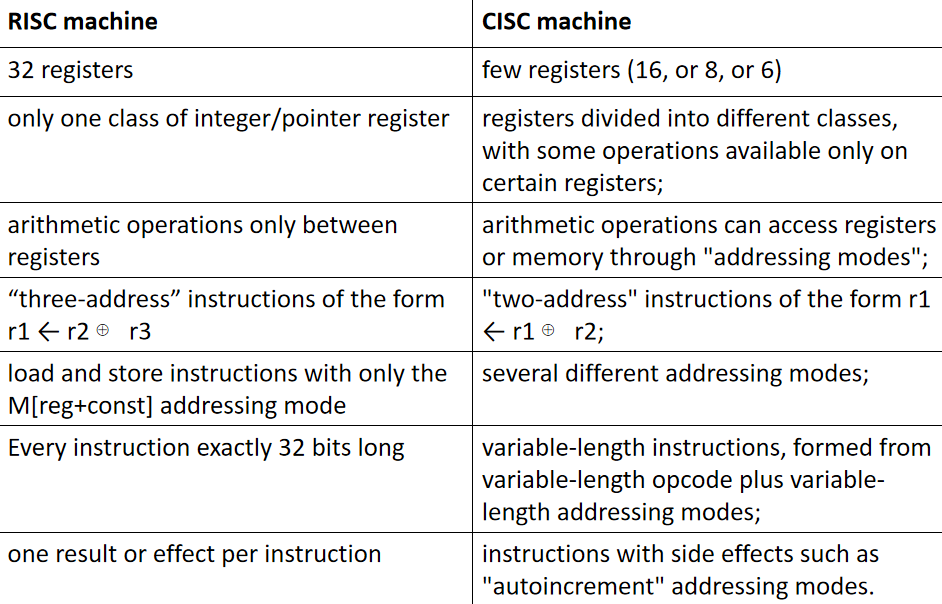
Two-address 指令中，第一源寄存器和目标寄存器一致（标注时忽略s0，只标识d0）

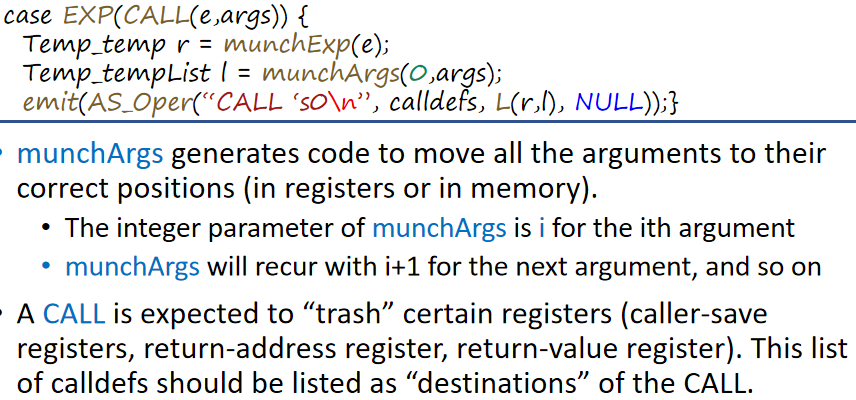
CISC中地址数据可直接与register数据相加，有更多addressing model（**两者都不会提升运行速度，仅减少寄存器污染使用，方便code**）

我们将在指令选择后进行寄存器的分配，所以在IR指令转换为machine时，我们仅分配abstract register

指令选择后再进行寄存器分配

tiger中间过程三种类型结构如

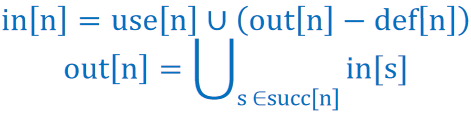
Procedure call：过程调用，无返回值；Function call：函数调用，有返回值

变量是**live**的如果其在下次使用之前不会再经过赋值

**Data flow构建**：Out-edges，flow去向；In-edges，flow来源；Pred[n]，all the predecessors of node n ；Succ[n]，the set of successors of node n；**Def(3/a)，**在节点3定义哪些变量，变量a在哪些节点定义；**Use(3/a)**，在节点3使用哪些变量，变量a在哪些节点使用

在**所有**in-edge上都活跃的node（live-in），在**所有**out-edge上都活跃的node（live-out）

递归计算**in/out set（完成后再迭代校验一遍）**



Out/inset可依赖bit array（大小固定）或sorted list实现，便于进行union操作（sorted list一般更快，更稀疏）

时间复杂度：N node，N变量，遍历需要考虑N node，每个node union操作要O(N)，总时间开销O(N2) ，元素最多2N2，in/out set，每个set最N个元素，共有N node，worst-time是O(N4)；实际，it between O(N) and O(N2)

该方法可以计算**least fixed point**（被所有其他解都包含）

**Dynamic liveness**：运行时是否经过node

**Static liveness**：control flow静态分析时是否经过node

动态活跃一定静态活跃

不同变量间**interference**（冲突），Out set中变量和当前语句被定义的变量a之间；MOVE语句中，仅添加MOVE左侧和live out set（近似于out set）间的冲突

即使是仅定义而未使用的变量，也需要进行interference计算（可能带来副作用）并放入register中

寄存器分配（NPC问题）

**Coloring近似算法 Build**建立冲突图，每个node代表一个临时寄存器（变量），Move指令变量间用虚线（不认为已冲突）；**Simplify**简化变量存储与于一个stack中；如果有顶点度小于K（寄存器数量），可直接从图中删除这些点；**Spill**如果simplify后仍留有度大于等于K的点，选择一些点，使其存储于stack中（不再与其他点interference）；算法结束后，如果有**actual spill**，需要**重写program**（每次取spill node需操作内存；多个actual spill**同时重写program**），重新执行算法（一般关注定义使用新变量区域即可，其余几乎不改动）；**Select**从stack中不断取出node，并进行color；简化程序

如果spill node的邻居确实拥有K种颜色，则是actual spill；否则，为该spill node上色（称为optimistic coloring）；**Coalescing（聚集）**，若move指令的target和source间没有实线相连，则两点间可合并；合并两点之后，原本可K color的冲突图可能无法被K color

**Briggs：**合并后度大于等于K的node邻居少于K个

**George：**原有邻居为共同邻居或邻居度本身小于K

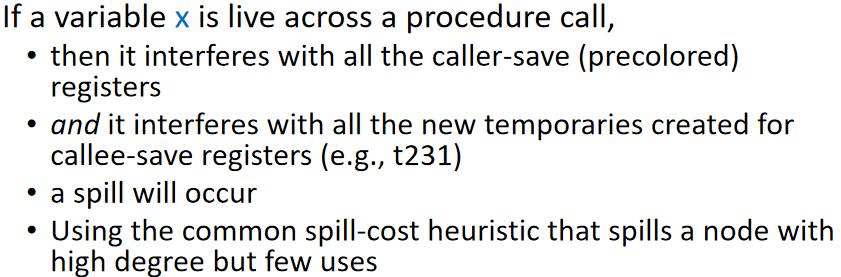
**考虑MOVE合并的color算法**：

**Build**不变；**simplify**中，与MOVE相关点不能simplify；

**coalesce**进行保守合并，和simplify往复进行（结点可能从MOVE有关变为MOVE无关）；**Freeze**如果simplify和coalesce都不能进行，将MOVE有关点转变为MOVE无关，继续simplify和coalesce过程；**Spill and select**，成功coalesce的多个node color同样颜色

**Precolored Nodes**：特定保留的register要precolor，需预着色的register之间相互interference(不能被simplify)

We should not spill precolored nodes to memory**冲突↓**



spill那些度数较高且不太被使用的变量**Spill priority** = （use + def）/ degree ；spill priority越低越先spill

Select完成后，将color的寄存器填入原程序中，并化简

Use/def使用及被赋值的次数

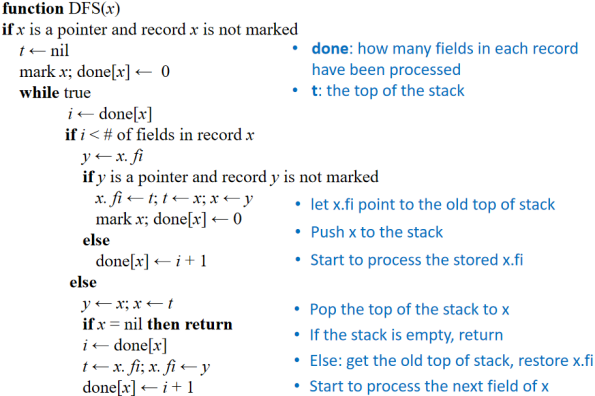
**Garbage:** 分配但不再使用的内存，Heap-allocated records如果不能被指针链reach则为垃圾（垃圾也可能被指向 --- 循环指针）

program variables是进行垃圾收集的root

**Mark-and-Sweep Collection**：使用DFS，从可访问pointer出发，遍历并mark所有可reach的memory；Mark完成后，遍历heap，清除没有mark的元素（可使用free-list收集）；遍历过程中需将已mark区域的mark清除

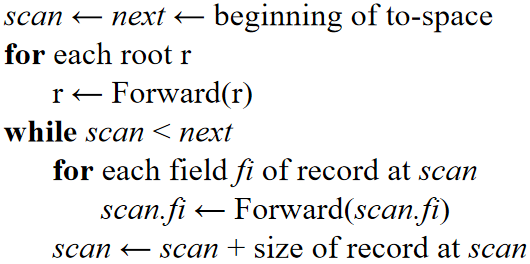
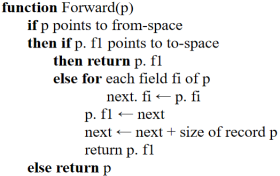
使用DFS，不停递归stack分配开销可能大于整个heap大小；考虑显示使用stack来实现DFS，但辅助的存储开销可能近似Heap大小；使用p**ointer Reversal**方法，更加节省空间开销（下图特别重要）

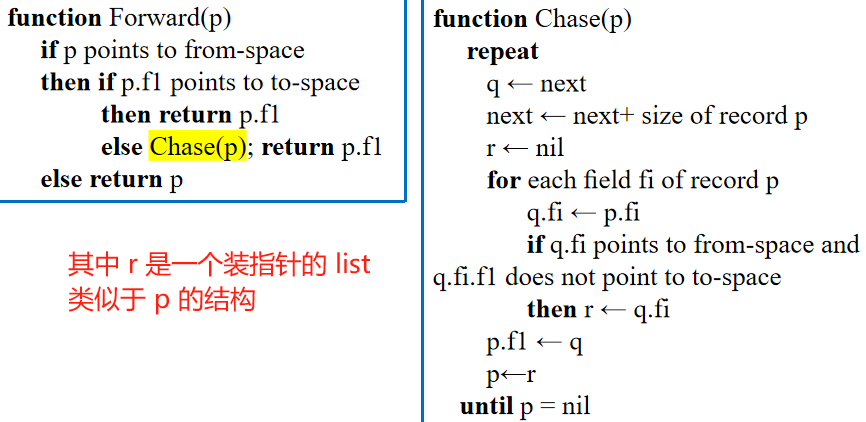
**Reference Counts(引用计数)**：记录records被多少其他pointer引用(将此值保存于对映record区域中)；当record reference count减少至0，需回收此record

缺陷：增加reference count会引入大量辅助代码；减少至0的递归不应立即进行，直到free-list中被再次分配时再触发，加速程序运行与管理；循环引用record无法被清除。优点：便于实现

**Copying collection**：heap分为二，from space，正常进行memory分配；to space，当from space中无法进行memory分配时，将from space的record搬移到to space；to space中数据连续存放

**Cheney’s algorithm**：使用BFS进行data的迁移（空间局部性差，使“DFS + **反转指针**”性能较差）

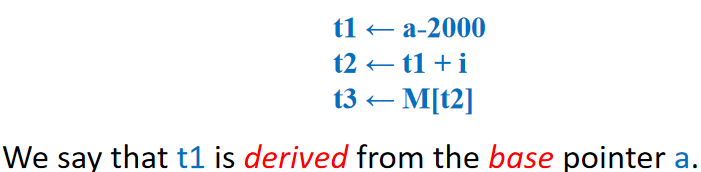
**Hybrid Algorithm**：混合使用BFS和DFS

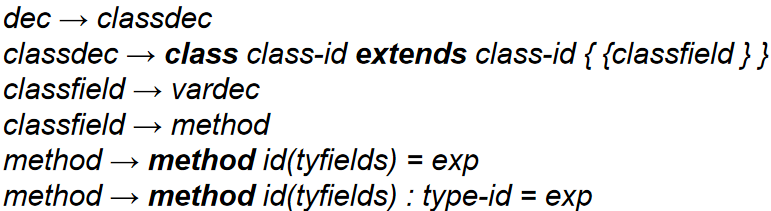


Tiger中，object的first word指向类描述符，指明当前object的total size与当前object中field location

使用**pointer map**记录存放指针为临时变量还是局部变量，并指出存放这些指针的区域(一般用位图实现)；pointer map有多张(程序不同的程序点)，以返回地址作索引，记录不同函数新建栈帧中变量指针，也需要记录base point

拓展tiger为面向对象语言，利于**模块封装（encapsulation）**和信息隐藏；**扩展（extension）和继承（inheritance）**



Tiger自带预先定义的class object（无任何属性和方法），任何类的方法自带self参数（not reversed word）

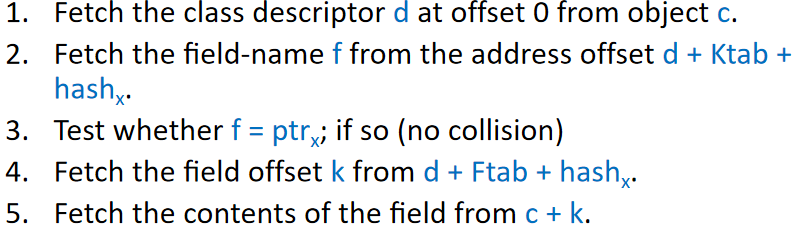
**单一继承**，若B继承A，将A的field放置于B的field之前，满足类对父类属性的访问（**prefixing**）；维护一个类描述符指向当前类支持的函数list以及父类

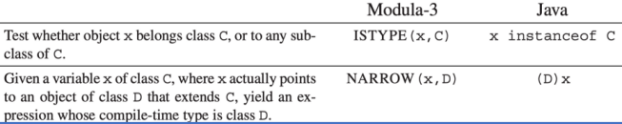
**静态函数**随parent层级向上寻找，直到首次找到；**动态函数**类似于field记录，在当前类前记录父类函数（指针）

**多继承**，使用图着色，分析不同class不同field同时出现可能性；根据着色情况对field进行不同offset排布（需全局记录各field的offset)；易导致类内大量empty slot,

借助类描述符，将field放置位置与offset映射，在类中，field就可直接连续存储；method类似field

图着色可在link-time完成，不支持动态增加class的语言

**Class descriptor实现**有Ftab（域位移表，真实存储数据）和Ktab(键值表，判别冲突情况)；hash冲突时，Ftab表可依赖Ktab的存储进行定位（两者记录偏移量一致）

NARROW强制类型转换（保护安全性前提下）

ISTYPE函数若层级向上寻找，效率低下；可显示维护display记录nested class关系；**Display数组**从0记录当前class顶层类类型（object），第j层记录自身类别，大于j的display索引均置0（在nested class关系中，display索引号一直保持不变）

**私有域和私有函数**，在对映类的symbol table中在field和method offset后添加boolean flag记录该对象是否为私有使用（类外元素不可获取、更新与调用）

不同类型的隐私与保护，定义class才可访问；定义class及其sub-class可访问；定义class同模块class可访问；定义class外class仅读，但可用method可写入；

**Loop**：有一组点，h能到每一个点，每一个点有指向h的边，没有外部边直接访问当前set中非h的点

A loop **entry node**：predecessor outside the loop；A loop **exit node：**successor outside the loop（本身为loop中node）

**Dominator支配者**：d支配n如果每一条从start node到n的路径都要通过d（每一个node都支配自己）

若所有入边支配集合都包括node A，则当前node的支配集合也包括node A

若a、b都支配n，则必存在a、b一者支配另一者

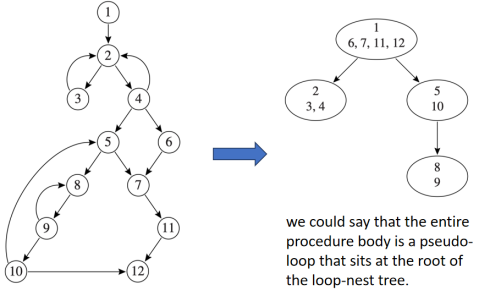
idom(n)，；n的支配节点且不是n，不支配任何n的支配者（离n最近的n支配者）

**Dominator Tree**，仅连接n和n的idom的（tree）图

**Back Edge**: n到支配n的点h的边

**natural loop of a back edge** n → h is the set of nodes x :h dominates x and a path from x to n not containing h

**Nested Loop**，若A、B以a、b为循环头，且a、b不等且b在A中，则B中node是A的真子集；loop B在A中（B is the inner loop）

**Loop-Nest tree**，循环包含树，叶子节点为原instruction最内层循坏；在树node中，第一行放置该loop的头结点，第二行放置loop中包含的节点；不同loop若header一致则合并；并列loop在图中node为并列关系，nest loop在图中为父子关系（将原图视作一棵“伪树”）

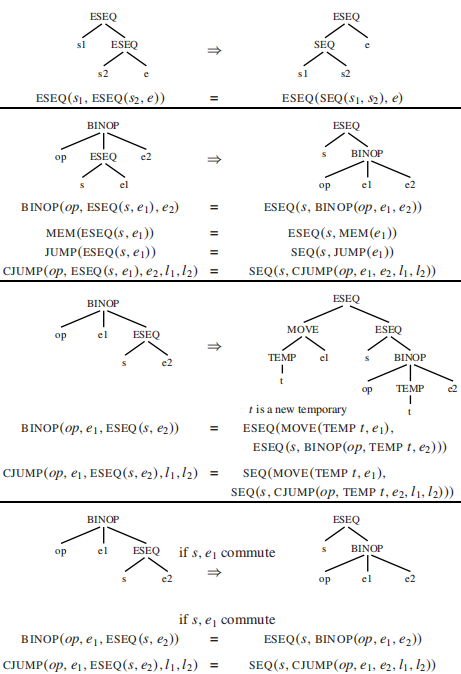
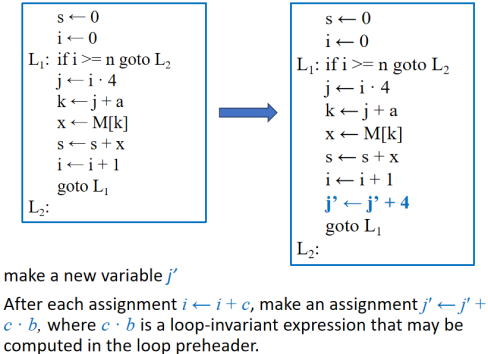
**Loop-Invariant Computations**循环不变计算，将循环内部分语句提至循环外进行而不影响循环一致性（提升效率）

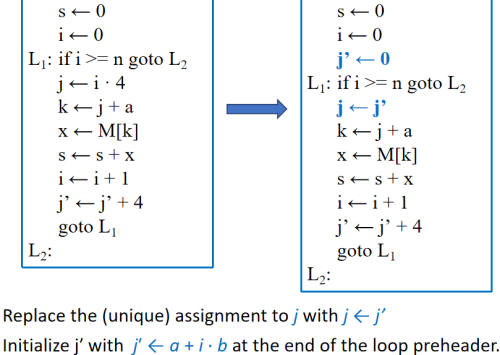
将**循环内操作**（t的计算指令d）**提前的限制**：d支配所有t存活的循环出口（会带来较大的限制）；循环中t仅定义一次；t进入循环时不是活跃的

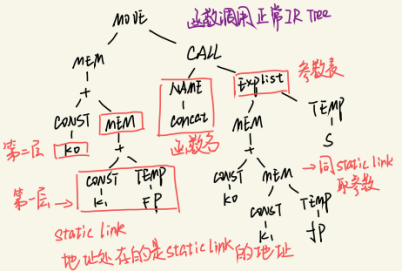
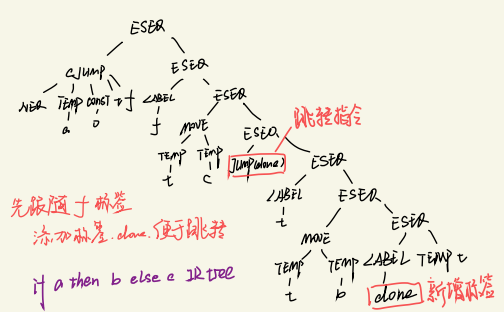
**Induction Variables**归纳变量

**Derivation induction variable**（导出归纳变量，有family族的概念，由哪个basic variable归纳获得）

**Linear induction variable**循环中每次迭代增加量均相同

**Strength reduction强度下降法**，乘法远较加法消耗大；可将导出变量基于basic变量将乘法更换为加法运算

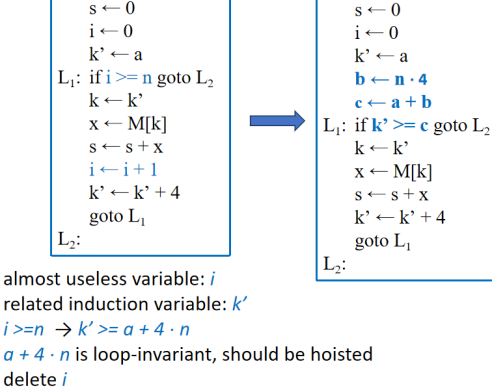
在上图中，可同理修改变量K；删除useless变量代码（仅定义未使用变量，在loop L所有出口均未使用的变量）



**Almost useless variable**同族归纳变量中有非useless变量，

仅定义时及循环条件比较时使用变量

下图继续简化其他类型冗余代码，获得相对简洁的结果

**Loop Unrolling**（循环展开）同时执行循环多个循环内容；Loop判断条件可能会发生调整，且可能会在loop的边界增加一些新条件判断（注意奇偶数循环次数执行）

跨过程使用变量用callee save寄存器，不跨过程用caller

CFG文法可以表示C等编程语言

返回地址应该由callee保存（call函数过程已覆盖ra）

不含连续baa正则表达式：a\*(b|ba|ca\*)\*

