Principles of Compiler Construction

Prof. Wen-jun LI
School of Computer Science and Engineering
lnslwj@mail.sysu.edu.cn

语义分析、中间代码生成是同时进行的

Lecture 9. Semantic Analysis and Intermediate Code Generation

1. Introduction
2. Types and Declarations
3. Assignments and Expressions 数组
4. Type Checking
5. Boolean Expressions 短路求值
6. Backpatching and Flow-of-Control Statements

override（重写，覆盖） （1）方法名、参数、返回值相同。 （2）子类方法不能缩小父类方法的访问权限。 （3）子类方法不能抛出比父类方法更多的异常(但子类方法可以不抛出异常)。 （4）存在于父类和子类之间。 （5）方法被定义为final不能被重写。 （6）被覆盖的方法不能为private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。
overload（重载，过载） （1）参数类型、个数、顺序至少有一个不相同。 （2）不能重载只有返回值不同的方法名。 （3）针对于一个类而言。 （4）不能通过访问权限、返回类型、抛出的异常进行重载； （5）方法的异常类型和数目不会对重载造成影响；

1. Introduction

- Review
  - Front end vs. back end
    - $m \times n$: m front ends and n back ends.
  - Interface between front ends and back ends
    - Intermediate representation
    - Why IR? Extendability and optimization.
  - Semantic (static) analysis
    - The most common analysis
      - Type checking
    - Other static checking
      - Unreachable code
      - Use of uninitialized variables
      - etc.

控制流永远达不到的代码，有的没有这个检查

不同语言不同

多态：
接口继承
overload、override？

dynamic：运行时做
static：编译时做

Static Checking

- Semantic analysis also focuses on the well-formness of source code
  - Due to the expressiveness power of Context-Free Grammars.
  - For example,
    - Number matching of actual parameters.
      - Context sensitive requirements cannot be specified using a context free grammar.
    - **break** statement must be in a loop or **switch**.
      - Requires a complicated and unnatural context free grammar.

一般前端是使用现有工具

Intermediate Representation

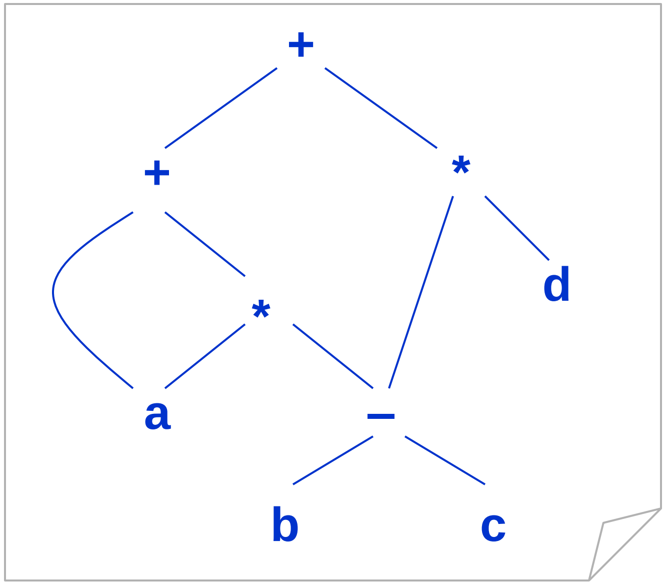
- High level intermediate representations
  抽象语法树
  - AST and DAG 有向无环图　二者本质相同
  - Suitable for tasks like static type checking
- Low level intermediate representations
  - 3-address code: x = y **op** z 三地址码：支持一、二元运算
  - Suitable for machine-dependent tasks, such as register allocation and instruction selection.
- IR choice/design are application specific
  - C language is commonly used (AT&T Bell Lab Advanced C++)

Three-Address Code

- Compiler-generated temporary variables
  - $x + y * z$
  - $t_1 = y * z$
    $t_2 = x + t_1$
- An example
  - $t_1 = b - c$
    $t_2 = a * t_1$
    $t_3 = a + t_2$
    $t_4 = t_1 * d$
    $t_5 = t_3 + t_4$

三地址码

DAG



高级语言和低级语言的区别在于表达式
代码优化后难以推算中间变量的个数

Addresses

- Addresses in 3-address code
  - Name (variables in source code)
    - May be implemented as a pointer or reference to its entry in the symbol table.
  - Constant
    - Type conversions must be considered.
  - Compiler-generated temporary
    - Useful for optimization.
    - Register allocation.

Instructions

- Common 3-address instructions
  - $x = y$ *op* z // arithmetic and logical
    $x =$ *op* y // negation and conversion
    $x = y$ // copy
  - **goto** L // unconditional jump
    if x **goto** L // conditional jump
    ifFalse x **goto** L // conditional jump
    if x *op* y **goto** L // relational operation
  - **param** x_1 // parameter passing
    param x_2
    ...
    param x_n
    call p, n // procedure call
    y = **call** p, n // function call
    return y // return a value

Instructions (cont')

- Common 3-address instructions
  - x = y[i] // indexed copy, i is the offset
    x[i] = y
  - x = &y // address and pointer assignment
    x = *y
    *x = y

?

Three-Address Code: Example

- Source code
  - **do** i = i + 1;
    while (a[i] < v);
- Translation to 3-address code (symbolic labels)
  - L: t_1 = i + 1
    i = t_1
    t_2 = i * 8
    t_3 = a[t_2]
    if t_3 < v **goto** L
- Another translation form (position numbers)
  - 100: t_1 = i + 1
    101: i = t_1
    102: t_2 = i * 8
    103: t_3 = a[t_2]
    104: **if** t_3 < v **goto** 100

Implementations of Three-Address Code

- Quadruples (quads) 四元
  - Pros and cons?
- Triples
  - Pros and cons?
- Indirect triples
  - Pros and cons?

Space consuming
Flexibility to optimizations

1) Quadruples

- Source code
  - $a = b * - c + b * - c$
- Three-address code
  - t_1 = **minus** c
    $t_2 = b * t_1$
    t_3 = **minus** c
    $t_4 = b * t_3$
    $t_5 = t_2 + t_4$
    $a = t_5$
- Quads

四元表达式

	op	arg_1	arg_2	result
0	**minus**	c		t_1
1	*	b	t_1	t_2
2	**minus**	c		t_3
3	*	b	t_3	t_4
4	+	t_2	t_4	t_5
5	=	t_5		a
…	…			

2) Triples

- Three-address code
  - t_1 = **minus** c
    t_2 = b * t_1
    t_3 = **minus** c
    t_4 = b * t_3
    t_5 = t_2 + t_4
    a = t_5

	op	arg_1	arg_2
0	**minus**	c	
1	*	b	(0)
2	**minus**	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)
…	…		

省空间（可忽略），不利于优化

3) Indirect Triples

- Three-address code
  - t_1 = **minus** c
    t_2 = b * t_1
    t_3 = **minus** c
    t_4 = b * t_3
    t_5 = t_2 + t_4
    a = t_5

这个是三元的抽象表示
四元式是具体实现

35	(0)
36	(1)
37	(2)
38	(3)
39	(4)
40	(5)
…	…

	op	arg_1	arg_2
0	**minus**	c	
1	*	b	(0)
2	**minus**	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)
…	…		

In Java, array of instruction objects

2. Types and Declarations

- Declaration
  - Literals: implicitly 3.14f
  - Variables: explicitly
  - Other names: explicitly 显式的声明
- Type checking in strong-typing languages
  - Type compatibility
  - Type inference
  - Implicit type conversion
  - Resolving overloading operators

Simplified Grammar

- Declare only one name at a time

$$
\begin{aligned}
D &\rightarrow T\ \mathbf{id}\ \mathbf{;}\ D \mid \varepsilon \\
T &\rightarrow B\ C \mid \mathbf{record}\ \mathbf{\{}\ D\ \mathbf{\}} \\
B &\rightarrow \mathbf{int} \mid \mathbf{double} \\
C &\rightarrow \mathbf{[}\ \mathbf{num}\ \mathbf{]}\ C \mid \varepsilon
\end{aligned}
$$

Translation of Type Declarations

- Computing types and their widths

$T \rightarrow B$	{ t = B.type; w = B.width }
C	{ T.type = C.type; T.width = C.width }
$B \rightarrow$ **int**	{ B.type = INTEGER; B.width = 4 }
$B \rightarrow$ **double**	{ B.type = DOUBLE; B.width = 8 }
$C \rightarrow$ **[num]** C_1	{ C.type = array(**num**.value, C_1.type); C.width = **num**.value $\times$ C_1.width }
$C \rightarrow \varepsilon$	{ C.type = t; C.width = w }

translation scheme

Type expression

Just try it: **int[2][3]**
What is T.type and T.width?

Translation of Type Declarations (cont')

- Computing relative addresses

P → { offset = 0 }
偏移量
D

D → T **id ;** { top.put(**id**.lexeme, T.type, offset);
offset += T.width }
D_1

D → ε

top denotes the current symbol table

Embedded actions can be removed with markers

Another Example

- Enter types and their widths

当前作用域的table

P →		{ offset = 0 }
	D	
D →	D **;** D	
D →	**id :** T	{ table.enter(id.name, T.type, offset); offset += T.width }
T →	**integer**	{ T.type = INTEGER; T.width = 4 }
T →	**real**	{ T.type = REAL; T.width = 8 }
T →	**array [num] of** T_1	{ T.type = array(**num**.value, T_1.type); T.width = **num**.value × T_1.width }
T →	**^** T_1	{ T.type = pointer(T_1.type); T.width = 4 }

Just try it: **k: array [5] of ^real**
What are the side effects?

DBv1

?

Another Example (cont')

新block， push new table

- Declarations in nested procedures

P → D

{ tableStack.push(**new** Table(**null**));
offsetStack.push(0) }

{ addWidth(tableStack.top(), offsetStack.top());
tableStack.pop();
offsetStack.pop() }

D → D ; D

D → **proc id ;** { tableStack.push(**new** Table(tableStack.top()));
offsetStack.push(0) }

声明

记录长度

D_1 ; S

语句

{ addWidth(tableStack.top(), offsetStack.top());
t = tableStack.top(); tableStack.pop();
offsetStack.pop();
tableStack.top().enter(**id**.name, t) }

D → **id** : T { tableStack.top().enter(**id**.name, T.type, offsetStack.top());
offsetStack.top() += T.width }

作用域问题

Another Example (cont')

- Field names in records

$T \rightarrow$ **record** { tableStack.push(**new** Table(**null**));
offsetStack.push(0) }
D **end** { T.type = record(tableStack.top());
T.width = offsetStack.top();
tableStack.pop();
offsetStack.pop() }

以下内容未仔细复习

3. Assignments and Expressions

- Intermediate code generation
  - Code concatenation
    - gen(...)　生成字符串
    - ||
    - No side effects
  - Incremental generation 增量
    - DBv1: emit(...)
    - DBv2: overloading gen(...)
    - Side effects

Translation of Expressions

- Code concatenation (syntax-directed definition)

	Productions	Semantic Rules
1	$S \rightarrow$ **id = ** E **;**	S.code = E.code || gen(top.get(**id**.lexeme) '=' E.addr)
2	$E \rightarrow E_1 + E_2$	E.addr = **new** Temp(); E.code = E_1.code || E_2.code || gen(E.addr '=' E_1.addr '+' E_2.addr)
3	$E \rightarrow$ **–** E_1	E.addr = **new** Temp(); E.code = E_1.code || gen(E.addr '=' '**minus**' E_1.addr)
4	$E \rightarrow$ **(** E_1 **)**	E.addr = E_1.addr; E.code = E_1.code
5	$E \rightarrow$ **id**	E.addr = top.get(**id**.lexeme); E.code = ' '

SDD

Translation of Expressions (cont')

- Incremental translation (translation scheme)

$S \rightarrow$ **id** **=** E **;**	{ gen(top.get(**id**.lexeme) '=' E.addr) }
$E \rightarrow E_1$ **+** E_2	{ E.addr = **new** Temp(); gen(E.addr '=' E_1.addr '+' E_2.addr) }
$E \rightarrow$ **–** E_1	{ E.addr = **new** Temp(); gen(E.addr '=' '**minus**' E_1.addr) }
$E \rightarrow$ **(** E_1 **)**	{ E.addr = E_1.addr }
$E \rightarrow$ **id**	{ E.addr = top.get(**id**.lexeme) }

Another Example

- Declared variables

找id，找不到就到外层找

$S \rightarrow$	**id := E ;**	{ p = symbolTable.lookup(**id**.name); **if** (p == **null**) **throw new** SomeException(); emit(p '=' E.place) }
$E \rightarrow$	E_1 **+** E_2	{ E.place = **new** Temp(); emit(E.place '=' E_1.place '+' E_2.place) }
$E \rightarrow$	**–** E_1	{ E.place = **new** Temp(); emit(E.place '=' '**minus**' E_1.place) }
$E \rightarrow$	**(** E_1 **)**	{ E.place = E_1.place }
$E \rightarrow$	**id**	{ p = symbolTable.lookup(**id**.name); **if** (p == **null**) **throw new** SomeException(); E.place = p }

Addressing Array Elements

- 2-dimensional array layout
  - Row major vs. column major

1st row	A[1, 1]
	A[1, 2]
	A[1, 3]
2nd row	A[2, 1]
	A[2, 2]
	A[2, 3]

1st column	A[1, 1]
	A[2, 1]
2nd column	A[1, 2]
	A[2, 2]
3rd column	A[1, 3]
	A[2, 3]

Addressing Array Elements

- Relative address of array elements
  - $A[i]$
    - $base + (i - low) \times w$
    - $i \times w + (base - low \times w)$

Pascal语言下标从low开始

Constant for optimization

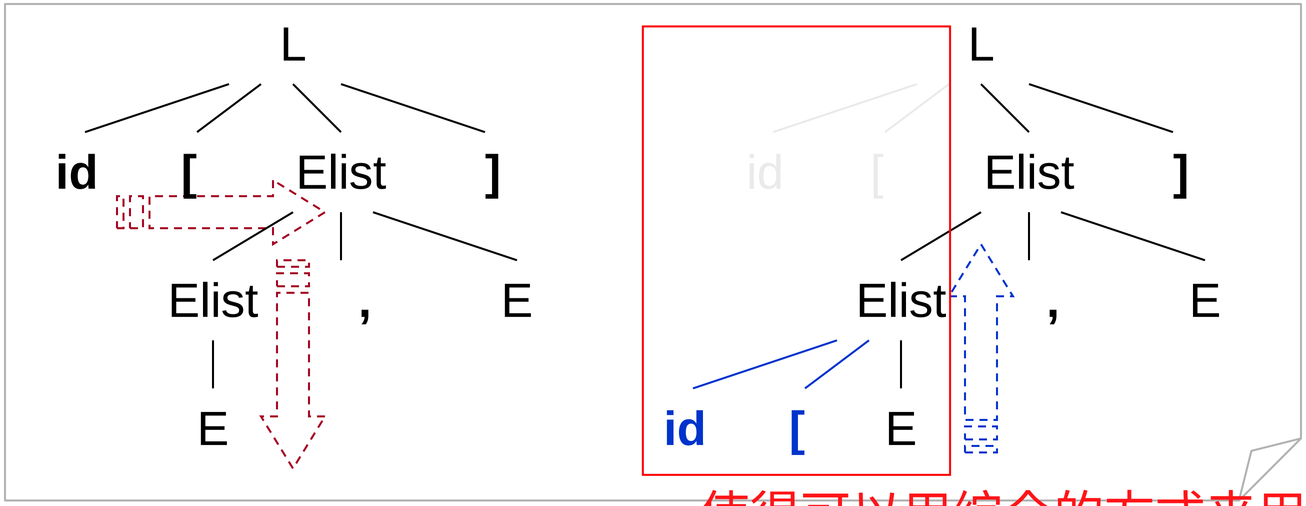
  - $A[i_1, i_2]$
    - $base + ((i_1 - low_1) \times n_2 + i_2 - low_2) \times w$
    - $((i_1 \times n_2) + i_2) \times w + (base - (low_1 \times n_2 + low_2) \times w)$
  - $A[i_1, i_2, \ldots, i_k]$
    - $((\ldots((i_1 \times n_2 + i_2) \times n_3 + i_3)\ldots) \times n_k + i_k) \times w +$
      $base - ((\ldots((low_1 \times n_2 + low_2) \times n_3 + low_3)\ldots) \times n_k + low_k) \times w$

Addressing Tips

- For each increment of a new dimension, addressing is calculated recursively, e.g. from k to k + 1
  - For variable part V: $V \times n_{k+1} + i_{k+1}$
  - For constant part C: $C \times n_{k+1} + low_{k+1}$

Grammar for Array References

- Array references in Pascal: a[2, 3]
  - L → **id [** Elist **]** | **id**
  - Elist → Elist **,** E | E
- Grammar transformation (why?)
  - L → Elist **]** | **id**
  - Elist → Elist **,** E | **id [** E



使得可以用综合的方式来用

Translation Scheme

表示是id

- Addressing array elements in Pascal

emit：做赋值运算

(1) S → L **:=** E { **if** (L.offset == **null**) emit(L.place '=' E.place)
else emit(L.place '[' L.offset ']' '=' E.place) }

(2) E → E_1 **+** E_2 { E.place = **new** Temp();
emit(E.place '=' E_1.place '+' E_2.place) }

(3) E → **(** E_1 **)** { E.place = E_1.place }

(4) E → L { **if** (L.offset == **null**) E.place = L.place
else {
E.place = **new** Temp();
emit(E.place '=' L.place '[' L.offset ']')
} }

(5) L → Elist **]** { L.place = **new** Temp();
emit(L.place '=' constant(Elist.array));
L.offset = **new** Temp();
emit(L.offset '=' Elist.place '*' width(Elist.array) }

(6) L → **id** { L.place = **id**.place;
L.offset = **null** }

L is a simple id (if L.offset is null)
or an array reference

L.place = base – C * w
L.offset = V * w

Translation Scheme (cont')

- Addressing array elements in Pascal (cont')

(7) Elist → $Elist_1$ **,** E { t = **new** Temp();
m = $Elist_1$.ndim + 1;
emit(t '=' $Elist_1$.place '*' limit($Elist_1$.array, m));
emit(t '+=' E.place);
Elist.array = $Elist_1$.array;
Elist.place = t;
Elist.ndim = m }

(8) Elist → **id [** E { Elist.array = **id**.place;
Elist.place = E.place;
Elist.ndim = 1 }

limit a,b 后缀两个参数的时候（/*参数必须是一个整数常量*/），其中a是指记录开始的偏移量,b是指从第a+1条开始，取b条记录。(这里计数就是从id=1开始的没有从0开始)

Elist.array = base
Elist.place = V
Elist.ndim = dimensions

Another Translation Scheme

- Array references in C/C++: a[2][3]
  - For all n, $low_n = 0$
  - Addressing formula
    - A[i]
      - $base + i \times w$
    - $A[i_1][i_2]$
      - $base + i_1 \times w_1 + i_2 \times w_2$
      - w_1 is the width of a row
      - w_2 is the width of an element in a row
    - $A[i_1][i_2]\ldots[i_k]$
      - $base + i_1 \times w_1 + i_2 \times w_2 + \ldots + i_k \times w_k$

Java does NOT use
row-major storage for arrays

Another Translation Scheme (cont')

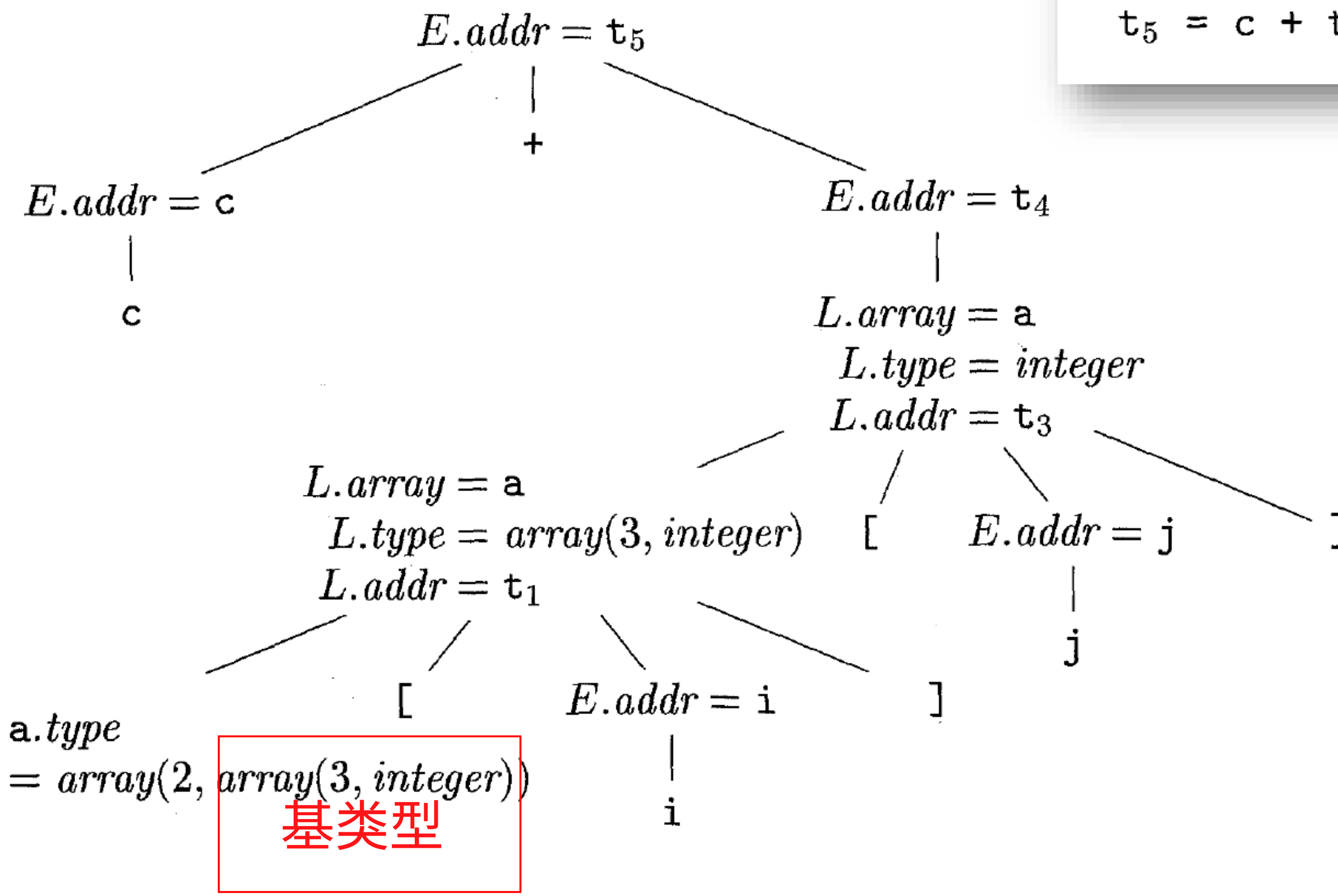
- Translation scheme

S	→	**id = E ;**	{ gen(top.get(**id**.lexeme) '=' E.addr) }
S	→	L **=** E **;**	{ gen(L.array.base '[' L.addr ']' '=' E.addr) }
E	→	E_1 **+** E_2	{ E.addr = **new** Temp(); gen(E.addr '=' E_1.addr '+' E_2.addr) }
E	→	**id**	{ E.addr = top.get(**id**.lexeme) }
E	→	L	{ E.addr = **new** Temp(); gen(E.addr '=' L.array.base '[' L.addr ']') }
L	→	**id [** E **]**	{ L.array = top.get(**id**.lexeme); L.type = L.array.type.element; L.addr = **new** Temp(); gen(L.addr '=' E.addr '*' L.type.width) }
L	→	L_1 **[** E **]**	{ L.array = L_1.array; L.type = L_1.type.element; t = **new** Temp(); L.addr = **new** Temp(); gen(t '=' E.addr '*' L.type.width); 这一维的长度 gen(L.addr '=' L_1.addr '+' t) }

L only for array reference
E.addr = E.place
L.array.base = L.place
L.addr = L.offset

DBv2

```
t1 = i * 12
t2 = j * 4
t3 = t1 + t2
t4 = a [ t3 ]
t5 = c + t4
```



4. Type Checking

- Strong typing vs. weak typing
  - Strongness is relative
- Type definitions
  - Primitive types: enumeration of constant
  - Composite types: type expressions
  - Type of functions: signatures
    - **if** f has type $s \rightarrow t$ **and** x has type s **then** expression f(x) has type t

Translation Scheme: An Example

- Type checking, inference and implicit casting

```
E → E₁ * E₂ { E.place := new Temp();
              if (E₁.type == TK_INT && E₂.type == TK_INT) {
                  emit(E.place '=' E₁.place '*int' E₂.place);
                  E.type = TK_INT;
              } elsif (E₁.type == TK_REAL && E₂.type == TK_REAL) {
                  emit(E.place '=' E₁.place '*real' E₂.place);
                  E.type = TK_REAL;
              } elsif (E₁.type == TK_INT && E₂.type == TK_REAL) {
                  t := new Temp();
                  emit(t '=' 'int2real' E₁.place);
                  emit(E.place '=' t '*real' E₂.place);
                  E.type = TK_REAL;
              } elsif (...) { ... }
            }
```

5. Boolean Expressions

- Boolean expressions are used in
  - Flows of control
  - Computing logical values

Computing Logical Values

- **a < b** equals to **if (a < b) then 1 else 0**

$E \rightarrow E_1$ **or** E_2 { E.place = **new** Temp();
emit(E.place '=' E_1.place '**or**' E_2.place) }

$E \rightarrow E_1$ **and** E_2 { E.place = **new** Temp();
emit(E.place '=' E_1.place '**and**' E_2.place) }

$E \rightarrow$ **not** E_1 { E.place = **new** Temp();
emit(E.place '=' '**not**' E_1.place) }

$E \rightarrow$ **(** E_1 **)** { E.place = E_1.place }

$E \rightarrow$ **id_1 relop id_2** { E.place = **new** Temp();
emit('**if**' **id_1**.place **relop**.op **id_2**.place '**goto**' currentStmt+3);
emit(E.place '=' '**0**');
emit('**goto**' currentStmt+2);
emit(E.place '=' '**1**') }

关系运算

$E \rightarrow$ **true** { E.place = **new** Temp();
emit(E.place '=' '**1**') }

$E \rightarrow$ **false** { E.place = **new** Temp();
emit(E.place '=' '**0**') }

DBv1

Computing Logical Values: An Example

- Source code
  - a < b **or** c < d **and** e < f
- Intermediate code

100:	**if** a < b **goto** 103	108:	**if** e < f **goto** 111
101:	t_1 = 0	109:	t_3 = 0
102:	**goto** 104	110:	**goto** 112
103:	t_1 = 1	111:	t_3 = 1
104:	**if** c < d **goto** 107	112:	t_4 = t_2 **and** t_3
105:	t_2 = 0	113:	t_5 = t_1 **or** t_4
106:	**goto** 108	114:	…
107:	t_2 = 1		

考试需要写出过程

建树（分析树 注释语法树）--DFS遍历树--生成动作顺序--执行动作--生成指令
action记得用虚线连接

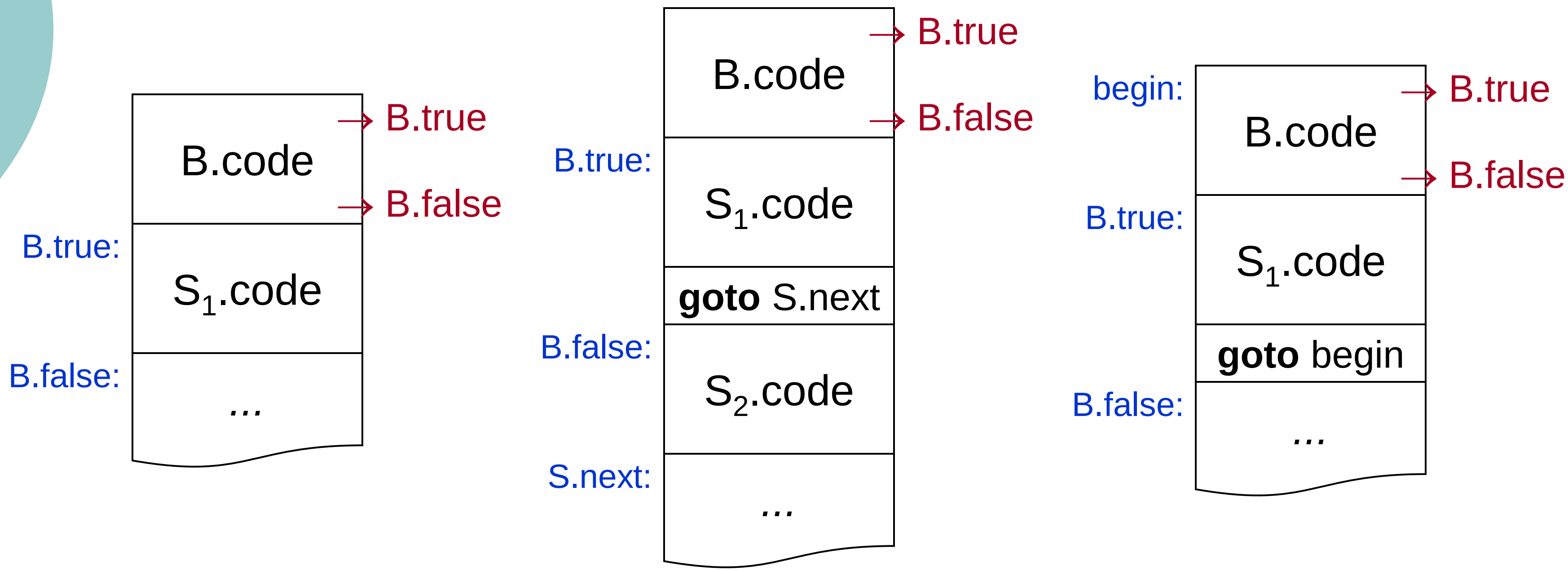
Short-Circuit Evaluation

- Flow-of-Control Statements
  - $S \rightarrow \textbf{if} \; \textbf{(} \; B \; \textbf{)} \; S_1$
  - $S \rightarrow \textbf{if} \; \textbf{(} \; B \; \textbf{)} \; S_1 \; \textbf{else} \; S_2$
  - $S \rightarrow \textbf{while} \; \textbf{(} \; B \; \textbf{)} \; S_1$

短路求值

- Short-circuit evaluation for **&&** and **||**
  - For higher evaluation efficiency
  - And …

Generated Code Illustration



if **if-else** **while**

Syntax-Directed Definition for Flow-of-Control Statements

Where does **S.next** come from?

Productions	Semantic Rules
P → S	S.next = **new** Label(); P.code = S.code || label(S.next)
S → **assign**	S.code = **assign**.code
S → S_1 S_2	S_1.next = **new** Label(); S_2.next = S.next; S.code = S_1.code || label(S_1.next) || S_2.code
S → **if (** B **)** S_1	B.true = **new** Label(); B.false = S_1.next = S.next; S.code = B.code || label(B.true) || S_1.code
S → **if (** B **)** S_1 **else** S_2	B.true = **new** Label(); B.false = **new** Label(); S_1.next = S_2.next = S.next; S.code = B.code || label(B.true) || S_1.code || gen('**goto**' S.next) || label(B.false) || S_2.code
S → **while (** B **)** S_1	begin = **new** Label(); B.true = **new** Label(); B.false = S.next; S_1.next = begin; S.code = label(begin) || B.code || label(B.true) || S_1.code || gen('**goto**' begin)

Avoid redundant **goto**s

Syntax-Directed Definition for Booleans

Productions	Semantic Rules
$B \rightarrow B_1$ **||** B_2	B_1.true = B.true; B_1.false = **new** Label(); B_2.true = B.true; B_2.false = B.false; B.code = B_1.code || label(B_1.false) || B_2.code
$B \rightarrow B_1$ **&&** B_2	B_1.true = **new** Label(); B_1.false = B.false; B_2.true = B.true; B_2.false = B.false; B.code = B_1.code || label(B_1.true) || B_2.code
$B \rightarrow$ **!** B_1	B_1.true = B.false; B_1.false = B.true; B.code = B_1.code
$B \rightarrow E_1$ **relop** E_2	B.code = E_1.code || E_2.code || gen('**if**' E_1.addr **relop**.op E_2.addr '**goto**' B.true) || gen('**goto**' B.false)
$B \rightarrow$ **true**	B.code = gen('**goto**' B.true)
$B \rightarrow$ **false**	B.code = gen('**goto**' B.false)

Short-Circuit Evaluation

不足：label需要回填

Syntax-Directed Translation: An Example

- Source code
  - **if** (x < 100 || x > 200 && x != y) x = 0
- Intermediate code

	if x < 100 **goto** L_2
	goto L_3
L_3:	**if** x > 200 **goto** L_4
	goto L_1
L_4:	**if** x != y **goto** L_2
	goto L_1
L_2:	x = 0
L_1:	…

6. Backpatching and Flow-of-Control Statements

- In SDD for Flow-of-Control Statements
  - Where does **S.next** come from?
  - Only after all intermediate code are generated, can **S.next** be computed.
- In SDD for Booleans
  - Where do **B.true** and **B.false** come from?
  - Must be provided by the context of the boolean expressions.
  - The context depends on the result of **S.next**.

Design Motivation and Solution

- Motivation
  - One-pass code generation
- Solution
  - Using backpatching 回填技术、一次性完成填入，解决上面的问题
- It is a general approach to dealing with initial values which must be computed at the end.

Backpatching for Boolean Expressions

- Translation scheme

$B \rightarrow B_1$ **||** $M\ B_2$	{ backpatch(B_1.falseList, M.instruction); B.trueList = merge(B_1.trueList, B_2.trueList); B.falseList = B_2.falseList; }
$B \rightarrow B_1$ **&&** $M\ B_2$	{ backpatch(B_1.trueList, M.instruction); B.trueList = B_2.trueList; B.falseList = merge(B_1.falseList, B_2.falseList); }
$B \rightarrow$ **!** B_1	{ B.trueList = B_1.falseList; B.falseList = B_1.trueList; }
$B \rightarrow$ **(** B_1 **)**	{ B.trueList = B_1.trueList; B.falseList = B_1.falseList; }
$B \rightarrow E_1$ **relop** E_2	{ B.trueList = **new** List(nextInstruction); B.falseList = **new** List(nextInstruction + 1); emit('**if**' E_1.addr relop.op E_2.addr '**goto** ___'); emit('**goto** ___'); }
$B \rightarrow$ **true**	{ B.trueList = **new** List(nextInstruction); emit('**goto** ___'); }
$B \rightarrow$ **false**	{ B.falseList = **new** List(nextInstruction); emit('**goto** ___'); }
$M \rightarrow \varepsilon$	{ M.instruction = nextInstruction; }

Backpatching for Flow-of-Control Statements

- Translation scheme

$S \rightarrow$ **if (** B **)** M S_1 { backpatch(B.trueList, M.instruction);
S.nextList = merge(B.falseList, S_1.nextList); }
指向同一个地方

$S \rightarrow$ **if (** B **)** M_1 S_1 N **else** M_2 S_2
{ backpatch(B.trueList, M_1.instruction);
backpatch(B.falseList, M_2.instruction);
S.nextList = merge(S_1.nextList, N.nextList, S_2.nextList); }

$S \rightarrow$ **while** M_1 **(** B **)** M_2 S_1
{ backpatch(B.trueList, M_2.instruction);
backpatch(S_1.nextList, M_1.instruction);
S.nextList = B.falseList;
emit('**goto**' M_1.instruction); }

$S \rightarrow$ **{** L **}** { S.nextList = L.nextList; }

$S \rightarrow$ A **;** { S.nextList = **new** List(); // Assignment or Atom }

$M \rightarrow \varepsilon$ { M.instruction = nextInstruction; }

$N \rightarrow \varepsilon$ { N.nextList = **new** List(nextInstruction);
生成goto语句
emit('**goto** __'); }

$L \rightarrow L_1$ M S { backpatch(L_1.nextList, M.instruction);
L.nextList = S.nextList; }

$L \rightarrow S$ { L.nextList = S.nextList; }

Exercise 9.1

- Let **A** be declared as a [2..4] x [1..5] array of integers and each integer occupy 4 bytes. What is the translation result of input token string: **x := A[3, 2]**?
  - Tips: use the translation scheme for Pascal.

1.parse tree
2.挂上动作
3.属性值写不写根据题目要求
4.执行完动作就得到最终的结果

DBv1, pp.484-485

Exercise 9.2

- Let **a** be declared as a 5 x 6 array of integers and each integer occupy 4 bytes. What is the translation result of input token string: **i = a[3][2]**?
  - Tips: use the translation scheme for C/C++.

DBv2, pp.383

Exercise 9.3

- What is the translation result of input token string: **x < 100 || x > 200 && x != y**?
  - Tips: use the translation scheme for boolean expressions with backpatching.
  - Suppose that the start position of the generated code is 100.

DBv2, pp.411-416

Further Reading

- Dragon Book, 2nd Edition (DBv2)
  - Comprehensive Reading:
    - Section 6.2 on introduction to intermediate representations.
    - Section 6.5 on type checking.
    - Section 6.3, 6.4, 6.6 and 6.7 on translations of various program constructs.
  - Skip Reading:
    - Section 6.1 on AST and DAG.
    - Section 6.8 and 6.9 on translations of switches and procedures.

Enjoy the Course!