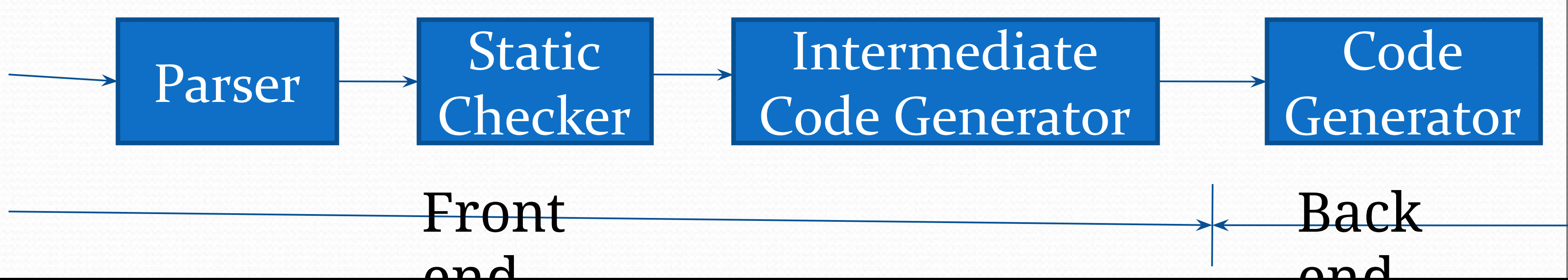
Compiler course

Chapter 6

Intermediate Code Generation

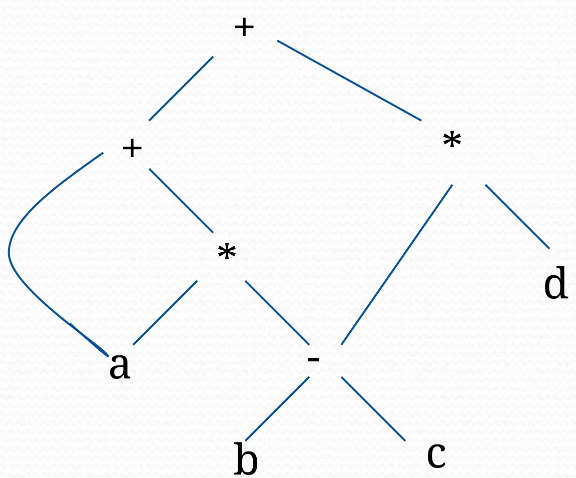
Introduction

- Intermediate code is the interface between front end and back end in a compiler
- In this chapter we study intermediate representations, static type checking and intermediate code generation



Variants of syntax trees

- It is sometimes beneficial to crate a DAG instead of tree for Expressions.
- This way we can easily show the common sub-expressions and then use that knowledge during code generation
- Example: a+a*(b-c)+(b-c)*d

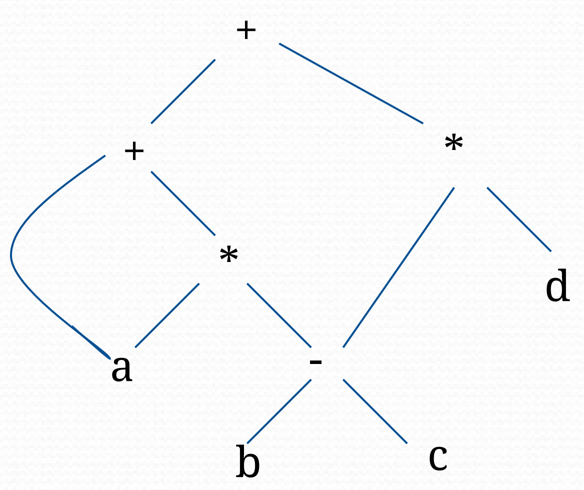


SDD for creating DAG's

Production	Semantic Rules
1) E -> E1+T	E.node= new Node('+', E1.node,T.node)
2) E -> E1-T	E.node= new Node('-', E1.node,T.node)
3) E -> T	E.node = T.node
4) T -> (E)	T.node = E.node
5) T -> id	T.node = new Leaf(id, id.entry)
6) T -> num	T.node = new Leaf(num, num.val)

Three address code

- In a three address code there is at most one operator at the right side of an instruction
- Example:



t1 = b – c
t2 = a * t1
t3 = a +
t2
t4 = t1 *
d
t5 = t3 +
t4

Forms of three address instructions

- x = y op z
- x = op y
- x = y
- goto L
- if x goto L and ifFalse x goto L
- if x relop y goto L
- Procedure calls using:
  - param x
  - call p,n
  - y = call p,n
- x = y[i] and x[i] = y
- x = &y and x = *y and *x =y

Example

- do i = i+1; while (a[i] < v);

```
L:  t1 = i + 1
    i = t1
    t2 = i * 8
    t3 = a[t2]
    if t3 < v goto L
```

Symbolic labels

```
100:    t1 = i + 1
101:    i = t1
102:    t2 = i * 8
103:    t3 = a[t2]
104:    if t3 < v goto
100
```

Position numbers

Data structures for three address codes

- Quadruples
  - Has four fields: op, arg1, arg2 and result
- Triples
  - Temporaries are not used and instead references to instructions are made
- Indirect triples
  - In addition to triples we use a list of pointers to triples

Example

- b * minus c + b * minus c

Three address code

t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
a = t5

Quadruples

op	arg1	arg2	result
minus	c		t1
*	b	t1	t2
minus	c		t3
*	b	t3	t4
+	t2	t4	t5
=	t5		a

Triples

	op	arg1	arg2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)

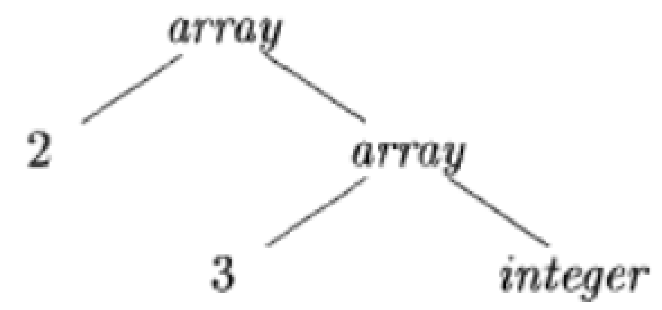
Indirect Triples

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

	op	arg1	arg2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)

Type Expressions

Example: int[2][3]
array(2,array(3,integer))



- A basic type is a type expression
- A type name is a type expression
- A type expression can be formed by applying the array type constructor to a number and a type expression.
- A record is a data structure with named field
- A type expression can be formed by using the type constructor → for function types
- If s and t are type expressions, then their Cartesian product s**t* is a type expression
- Type expressions may contain variables whose values are type expressions

Type Equivalence

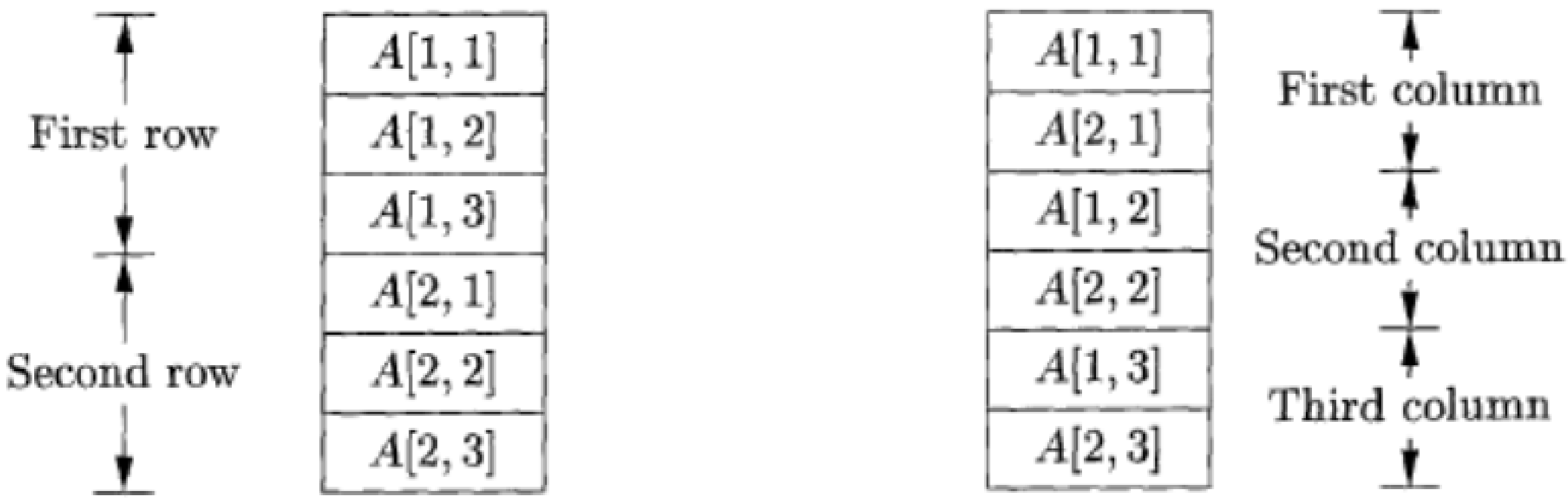
- They are the same basic type.
- They are formed by applying the same constructor to structurally equivalent types.
- One is a type name that denotes the other.

Declarations

- Refer Section 8.2 From book(Aho,Ullman,sethi)

Addressing Array Elements
(From section 8.3 Aho)

- Layouts for a two-dimensional array:



(a) Row Major (b) Column Major

8.4 Boolean Expressions

Control Flow

boolean expressions are often used to:

- *Alter the flow of control.*
- *Compute logical values.*

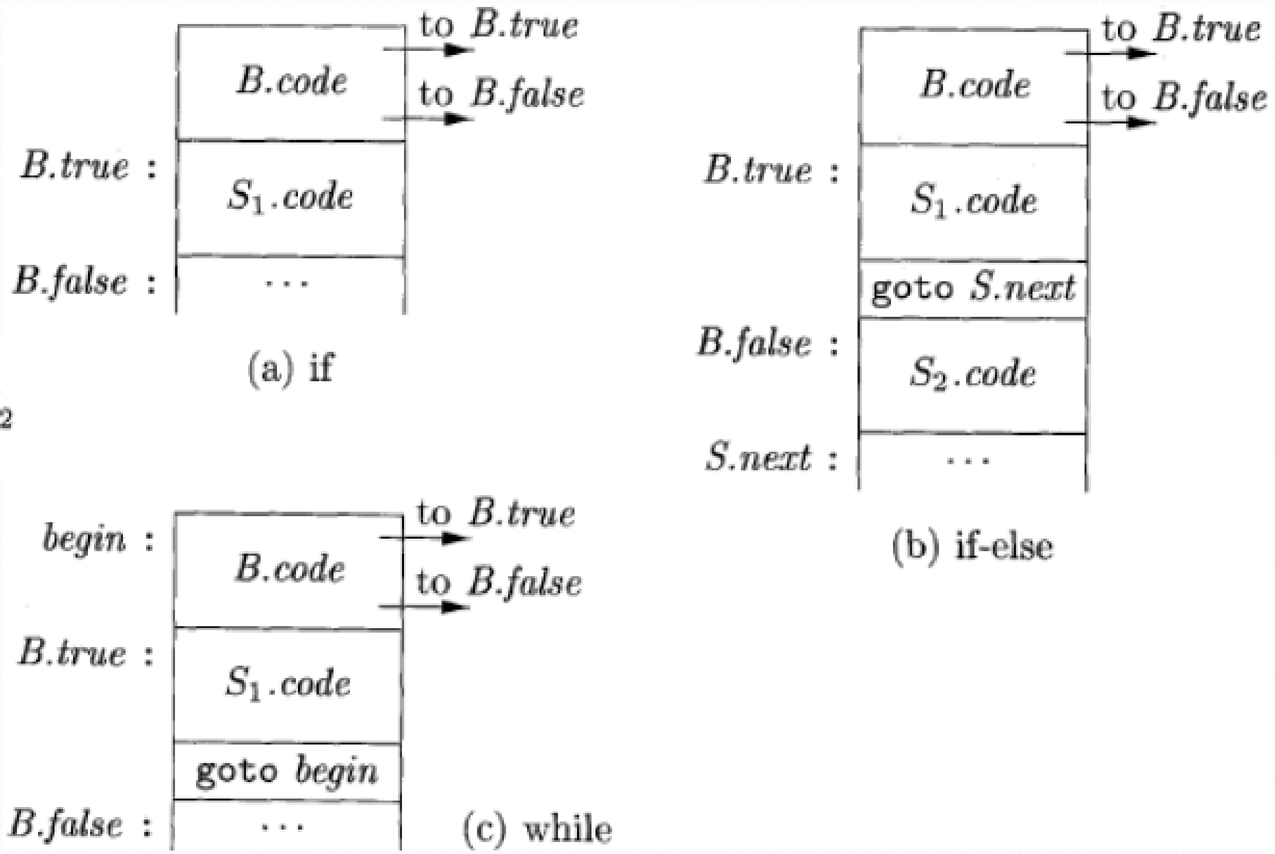
Short-Circuit Code

- `if ( x < 100 || x > 200 && x != y ) x = 0;`

-
```
        if x < 100 goto L2
        ifFalse x > 200 goto L1
        ifFalse x != y goto L1
L2:     x = 0
L1:
```

Flow-of-Control Statements

$$
\begin{array}{lcl}
S & \rightarrow & \textbf{if} \; (\; B \;) \; S_1 \\
S & \rightarrow & \textbf{if} \; (\; B \;) \; S_1 \; \textbf{else} \; S_2 \\
S & \rightarrow & \textbf{while} \; (\; B \;) \; S_1
\end{array}
$$



Syntax-directed definition

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow \textbf{assign}$	$S.code = \textbf{assign}.code$
$S \rightarrow \textbf{if}\ (\ B\ )\ S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow \textbf{if}\ (\ B\ )\ S_1\ \textbf{else}\ S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen(\texttt{'goto'}\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow \textbf{while}\ (\ B\ )\ S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen(\texttt{'goto'}\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

Generating three-address code for booleans

PRODUCTION	SEMANTIC RULES
$B \rightarrow B_1 \mid\mid B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$
$B \rightarrow B_1 \;\&\&\; B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$
$B \rightarrow \; ! \; B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1 \;\mathbf{rel}\; E_2$	$B.code = E_1.code \mid\mid E_2.code$ $\mid\mid gen(\texttt{'if'}\; E_1.addr\; \mathbf{rel}.op\; E_2.addr\; \texttt{'goto'}\; B.true)$ $\mid\mid gen(\texttt{'goto'}\; B.false)$
$B \rightarrow \mathbf{true}$	$B.code = gen(\texttt{'goto'}\; B.true)$
$B \rightarrow \mathbf{false}$	$B.code = gen(\texttt{'goto'}\; B.false)$

translation of a simple if-statement

- `if( x < 100 || x > 200 && x != y ) x = 0;`

-
```
        if x < 100 goto L2
        goto L3
L3:     if x > 200 goto L4
        goto L1
L4:     if x != y goto L2
        goto L1
L2:     x = 0
L1:
```

Readings

- Refer Chapter 8 Intermediate Code generation from the book Aho,Ullman,Sethi.