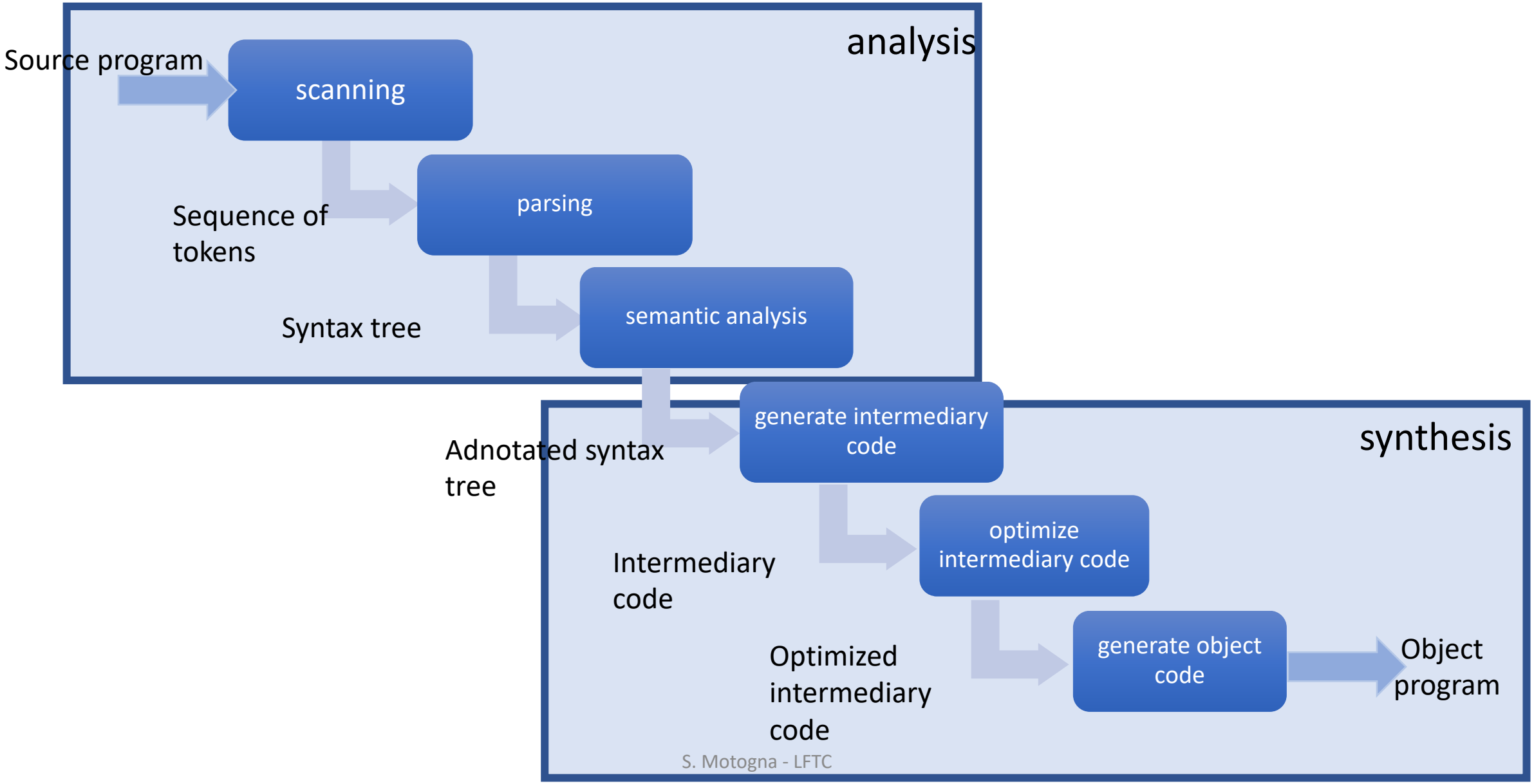
Course 10

Structure of compiler



Semantic analysis

- Attach meanings to syntactical constructions of a program
- What:
  - Identifiers -> values / how to be evaluated
  - Statements -> how to be executed
  - Declaration -> determine space to be allocated and location to be stored
- Examples:
  - Type checkings
  - Verify properties
- How:
  - **Attribute grammars**
  - Manual methods

Semantic analysis – Attribute grammars

- Parsing – result: syntax tree (ST)
- Simplification: abstract syntax tree (AST)
- Annotated abstract syntax tree (AAST)
  - Attach semantic info in tree nodes

Attribute grammar

- Syntactical constructions (nonterminals) and tokens (terminals) – attributes

$$\forall X \in N \cup \Sigma: A(X)$$

- Productions – rules to compute/ evaluate attributes

$$\forall p \in P: R(p)$$

Definition

AG = (G,A,R) is called ***attribute grammar*** where:

- G = (N,Σ,P,S) is a context free grammar
- A = {A(X) | X ∈N U Σ} – is a finite set of attributes
- R = {R(p) | p ∈P} – is a finite set of rules to compute/evaluate attributes

Example 1

- G = ({N,B},{0,1}, P, N}

  P: N -> NB

  N -> B

  B -> 0

  B -> 1

$N_1.v = 2^* N_2.v + B.v$

N.v = B.v

B.v = 0

B.v = 1

Attribute – value of number = **v**

- **Synthetized attribute: A(lhp) depends on rhp**
- **Inherited attribute: A(rhp) depends on lhp**

Evaluate attributes

- Traverse the tree: can be an infinite cycle

- Special classes of AG:
  - L-attribute grammars
  - S-attribute grammars

Example 2 (L-attribute grammar)

Decl -> DeclTip ListId

ListId -> Id

ListId -> ListId, Id

ListId.type = DeclTip.type
Id.type = ListId.type
$ListId_2$.type = $ListId_1$.type
Id.type = $ListId_1$.type

Attribute – type

int i,j

Example 3 (S-attribute grammar)

ListDecl -> ListDecl; Decl

ListDecl -> Decl

Decl -> Type ListId

Type -> int

Type -> long

ListId -> Id

ListId -> ListId, Id

$ListDecl_1.dim = ListDecl_2.dim + Decl.dim$

$ListDecl.dim = Decl.dim$

$Decl.dim = Type.dim * ListId.no$

$Type.dim = 4$

$Type.dim = 8$

$ListId.nr = 1$

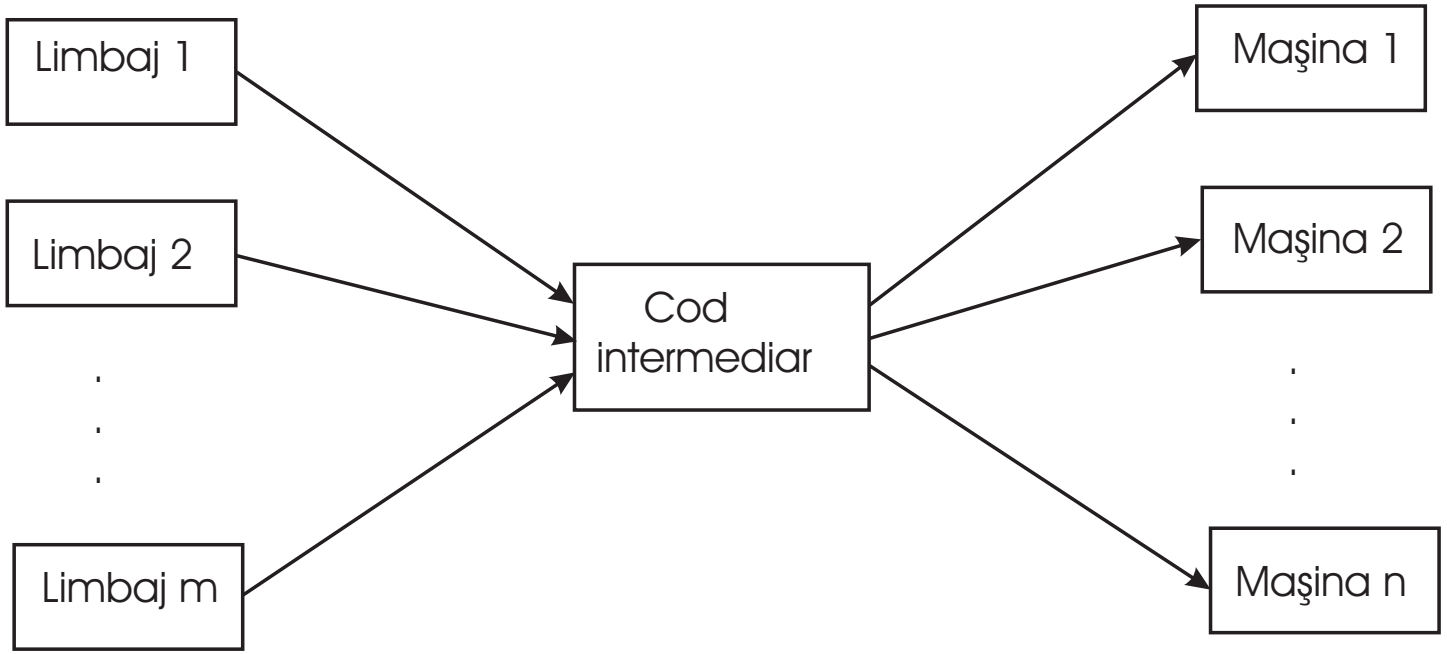
$ListId_1.nr = ListId_2.nr + 1$

Attributes – dim + no – **for which symbols**

Proposed problems (HW):

1) Define an attribute grammar for arithmetic expressions
2) Define an attribute grammar for logical expressions
3) Define an attribute grammar for if statement

Generate intermediary code



Forms of intermediary code

- Java bytecode – source language: Java
  – machine language (dif. platforms) JVM
- MSIL (Microsoft Intermediate Language)
  – source language: C#, VB, etc.
  – machine language (dif. platforms) Windows
- GNU RTL (Register Transfer Language)
  – source language: C, C++, Pascal, Fortran etc.
  – machine language (dif. platforms)

Representations of intermediary code

- Annotated tree: intermediary code is generated in semantic analysis
- Polish postfix form:
  - No parenthesis
  - Operators appear in the order of execution
  - Ex.: MSIL

Exp = a + b * c	ppf = abc*+
Exp = a * b + c	ppf = ab*c+
Exp = a * (b + c)	ppf = abc+*

- 3 address code

3 address code

= sequence of simple format statements, close to object code, with the following general form:

< result >=< arg1 >< op >< arg2 >

Represented as:

- Quadruples
- Triples
- Indirected Triples

- Quadruples:

< op > < arg1 > < arg2 > < result >

- Triples:

< op > < arg1 > < arg2 >

(considered that the triple is storing the result)

- Indirect triples

Special cases:

1. Expressions with unary operator: **< result >=< op >< arg2 >**
2. Assignment of the form **a := b** => the 3 addresss code is **a = b** (no operatorand no 2nd argument)
3. Unconditional jump: statement is **goto L**, where L is the label of a 3 address code
4. Conditional jump: **if c goto L**: if **c** is evaluated to **true** then unconditional jump to statement labeled with L, else (if c is evaluated to false), execute the next statement
5. Function call p(x1, x2, ..., xn) – sequence of statements: **param x1, param x2 , param xn, call p, n**
6. Indexed variables: < arg1 >,< arg2 >,< result > can be array elements of the form **a[i]**
7. Pointer, references: **&x, *x**

Example 1: b*b–4*a*c

op	arg1	arg2	rez
*	b	b	t1
*	4	a	t2
*	t2	c	t3
-	t1	t3	t4

nr	op	arg1	arg2
(1)	*	b	b
(2)	*	4	a
(3)	*	(2)	c
(4)	-	(1)	(3)

If (a<2) {a=b} else {a=b+1}