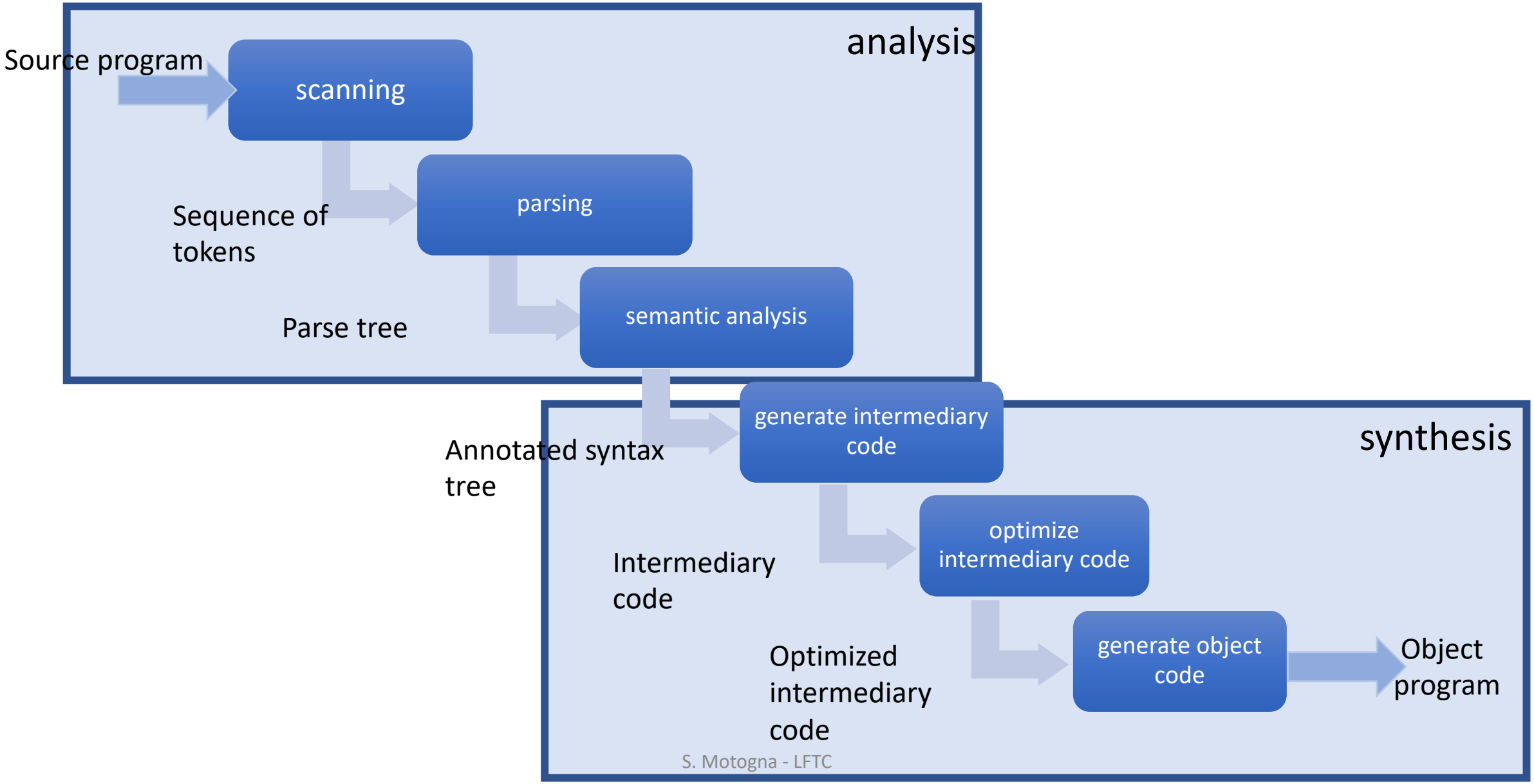
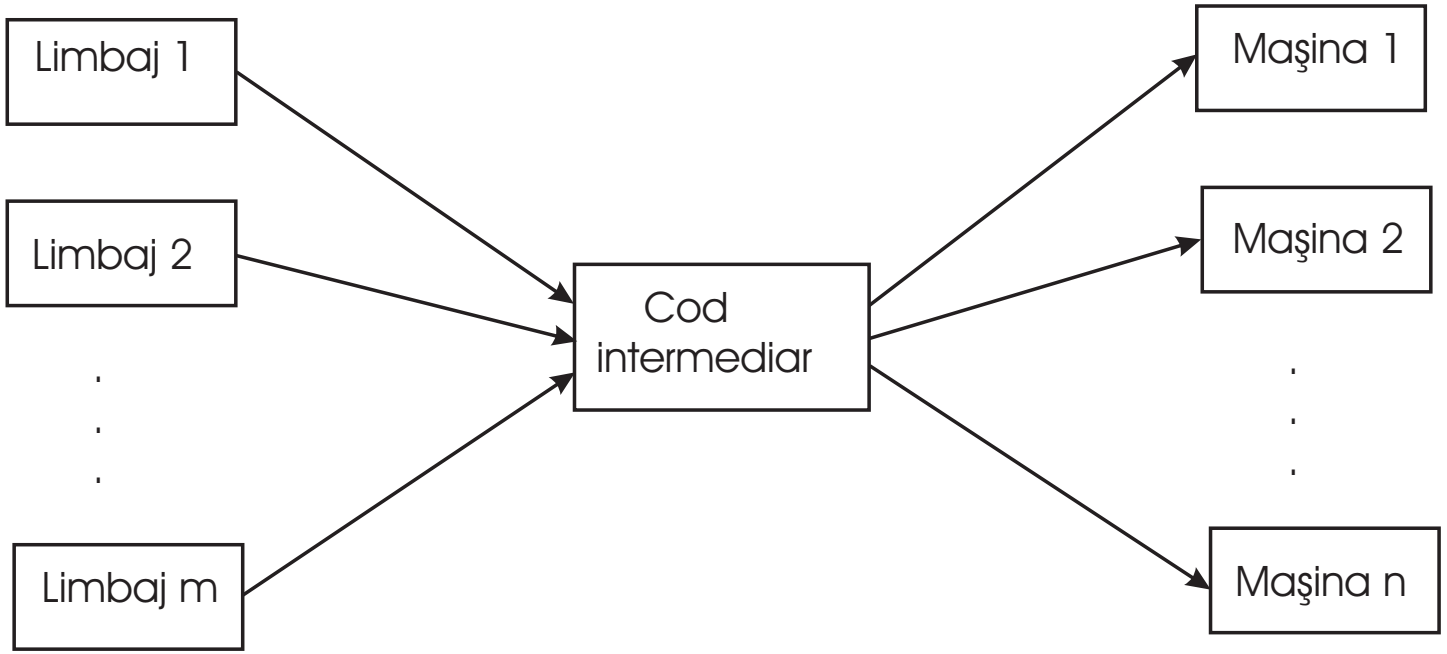
Course 12

Structure of compiler



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)

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: 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)

Example 2

If (a<2) then a=b else a=b*b

Optimize intermediary code

- Local optimizations:
  - Perform computation at compile time – constant values
  - Eliminate redundant computations
  - Eliminate inaccessible code – if…then...else...

- Loop optimizations:
  - Factorization of loop invariants
  - Reduce the power of operations

Eliminate redundant computations

Example:

D:=D+C*B
A:=D+C*B
C:=D+C*B

(1)	*	C	B
(2)	+	D	(1)
(3)	:=	(2)	D
(4)	*	C	B
(5)	+	D	(4)
(6)	:=	(5)	A
(7)	*	C	B
(8)	+	D	(7)
(9)	:=	(8)	C

Determine redundant operations

- Operation (j) is redudant to operation (i) with i<j if the 2 operations are identical and if the operands in (j) did not change in any operation between (i+1) and (j-1)
- Algorithm [Aho]

Factorization of loop invariants

What is a loop invariant?

```
for(i=0, i<=n,i++)
    { x=y+z;
    a[i]=i*x}
```

```
x=y+z;
for(i=0, i<=n,i++)
    { a[i]=i*x}
```

Challenge

```
V1:
P = a[0]
For i=1 to n
  P = P + a[i]*v^i
```

```
V2:
P = a[0]
Q=v
For i=1 to n
  P = P + a[i]*Q
  Q = Q*v
```

Consider n, and a[i] i=0,n the coefficients of a polynomial P.

Given v, write an algorithm that computes the value of P(v)

3 solutions

```
V3
P=a[n]
For i=1 to n
  P = P*v + a[n-i]
```

$$P(x) = a[n]*x^n+ \dots + a[1]*x + a[0] = (a[n]*x^{(n-1)}+ \dots + a[1])*x + a[0]$$

Reduce the power of operations

```
for(i=k, i<=n,i++)
      { t=i*v;
      . . .}
```

```
t1=k*v;
for(i=k, i<=n,i++)
      { t=t1;
      t1=t1+v;...}
```