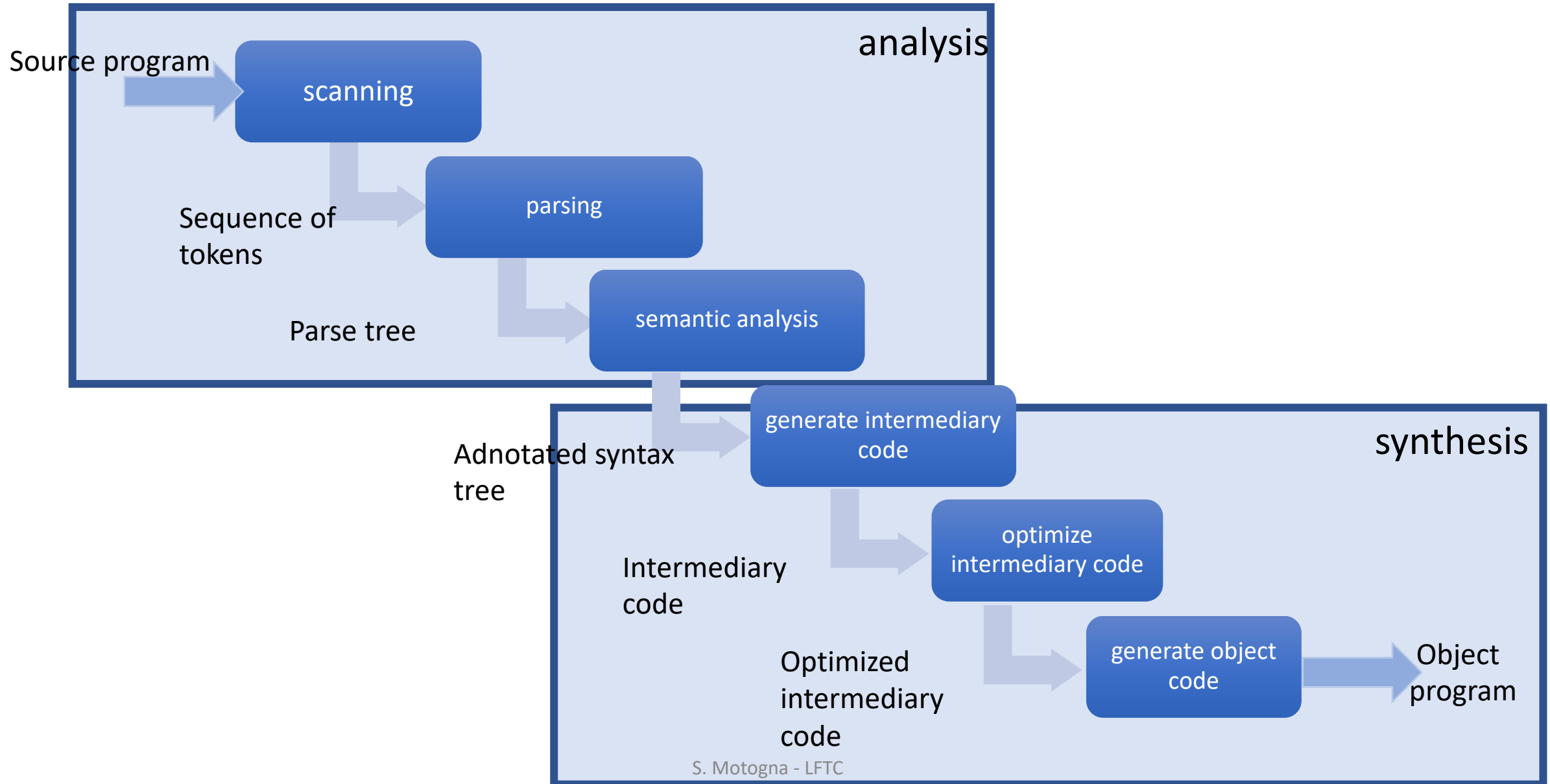


Course 12

Structure of compiler



Generate object code

= translate intermediary code statements into statements of object code (machine language)

- Depend on “machine”: architecture and OS

Computer with accumulator

- A **stack machine** consists of:
- a stack for storing and manipulating values (store subexpressions and results)
- Accumulator – to execute operation
- 2 types of statements:
 - move and copy values in and from head of stack to accumulator
 - Operations on stack head, functioning as follows: operands are popped from stack, execute operation and then put the result in stack

Example: $4 * (5+1)$

Code	acc	stack
$\text{acc} \leftarrow 4$	4	$\langle \rangle$
push acc	4	$\langle 4 \rangle$
$\text{acc} \leftarrow 5$	5	$\langle 4 \rangle$
push acc	5	$\langle 5, 4 \rangle$
$\text{acc} \leftarrow 1$	1	$\langle 5, 4 \rangle$
$\text{acc} \leftarrow \text{acc} + \text{head}$	6	$\langle 5, 4 \rangle$
pop	6	$\langle 4 \rangle$
$\text{acc} \leftarrow \text{acc} * \text{head}$	24	$\langle 4 \rangle$
pop	24	$\langle \rangle$

Computer with registers

- Registers +
- Memory
- Instructions:
 - LOAD v, R – load value v in register R
 - STORE R, v – put value v from register R in memory
 - ADD $R1, R2$ – add to the value from register $R1$, value from register $R2$ and store the result in $R1$ (initial value is lost!)

2 aspects:

- Register allocation – way in which variables are stored and manipulated;
- Instruction selection – way and order in which the intermediary code statements are mapped to machine instructions

Remarks:

1. A register can be available or occupied =>

$\text{VAR}(R)$ = set of variables whose values are stored in register R

2. For every variable, the place (register, stack or memory) in which the current value of the value exists=>

$\text{MEM}(x)$ = set of locations in which the value of variable x exists (will be stored in Symbol Table)

Example: $F := A * B - (C + B) * (A * B)$

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) $T1 = A * B$			
(2) $T2 = C + B$			
(3) $T3 = T2 * T1$			
(4) $F := T1 - T3$			

Example: $F := A * B - (C + B) * (A * B)$

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) $T1 = A * B$	LOAD A, R0 MUL R0, B	VAR(R0) = {A} VAR(R0) = {T1}	MEM(T1) = {R0}
(2) $T2 = C + B$			
(3) $T3 = T2 * T1$			
(4) $F := T1 - T3$			

Example: $F := A * B - (C + B) * (A * B)$

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) T1 = A * B	LOAD A, R0 MUL R0, B	VAR(R0) = {T1}	MEM(T1) = {R0}
(2) T2 = C + B	LOAD C, R1 ADD R1, B	VAR(R1) = {T2}	MEM(T2) = {R1}
(3) T3 = T2 * T1			
(4) F := T1 - T3			

Example: $F := A * B - (C + B) * (A * B)$

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) $T1 = A * B$	LOAD A, R0 MUL R0, B	VAR(R0) = {T1}	MEM(T1) = {R0}
(2) $T2 = C + B$	LOAD C, R1 ADD R1, B	VAR(R1) = {T2}	MEM(T2) = {R1}
(3) $T3 = T2 * T1$	MUL R1,R0	VAR(R1) = {T3}	MEM(T2) = {} MEM(T3) = {R1}
(4) $F := T1 - T3$			

Example: $F := A * B - (C + B) * (A * B)$

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) $T1 = A * B$	LOAD A, R0 MUL R0, B	VAR(R0) = {T1}	MEM(T1) = {R0}
(2) $T2 = C + B$	LOAD C, R1 ADD R1, B	VAR(R1) = {T2}	MEM(T2) = {R1}
(3) $T3 = T2 * T1$	MUL R1, R0	VAR(R1) = {T3}	MEM(T2) = {} MEM(T3) = {R1}
(4) $F := T1 - T3$	SUB R0, R1 STORE R0, F	VAR(R0) = {F} VAR(R1) = {}	MEM(T1) = {} MEM(F) = {R0, F}

More about Register Allocation

- Registers – **limited resource**
- Registers – perform operations / computations
- Variables **much more** than registers

IDEA: assigning a large number of variables to a reduced number of registers

Live variables

- Determine the number of variables that are live (used)

Example:

$a = b + c$

$d = a + e$

$e = a + c$

	op	op1	op2	rez
1	+	b	c	a
2	+	a	e	d
3	+	a	c	e

	1	2	3
a	x	x	x
b	x		
c	x	x	x
d		x	
e		x	x

Graph coloring allocation (Chaitin a.o. 1982)

- Graph:
 - nodes = live variables that should be allocated to registers
 - edges = live ranges simultaneously live

Register allocation = graph coloring: colors (registers) are assigned to the nodes such that two nodes connected by an edge do not receive the same color

Disadvantage:

- NP complete problem

Linear scan allocation (Poletto a.o., 1999)

- determine all live range, represented as an interval
- intervals are traversed chronologically
- greedy algorithm

Advantage: speed – code is generated faster (speed in code generation)

Disadvantage: generated code is slower (NO speed in code execution)

Instruction selection

Example: $F := A * B - (C + B) * (A * B)$

Intermediary code	Object code	VAR	MEM
		VAR(R0) = {} VAR(R1) = {}	
(1) $T1 = A * B$	LOAD A, R0 MUL R0, B	VAR(R0) = {T1}	MEM(T1) = {R0}
(2) $T2 = C + B$	LOAD C, R1 ADD R1, B	VAR(R1) = {T2}	MEM(T2) = {R1}
(3) $T3 = T2 * T1$	MUL R1, R0 MUL R0, R1	VAR(R1) = {T3}	MEM(T2) = {} MEM(T3) = {R1}
(4) $F := T1 - T3$	LOAD T1, R1		

Decide which register to use for an instruction

Turing Machines

Alan Turing

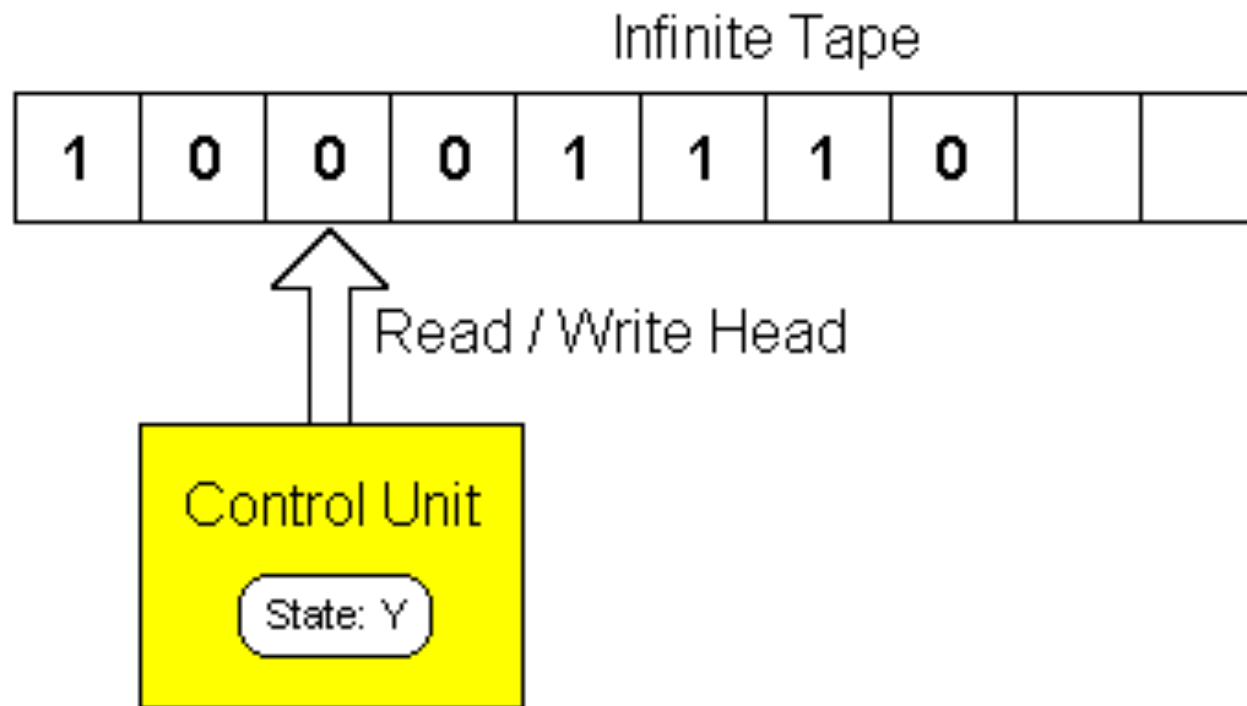
- Enigma (criptography)
- Turing test
- Turing machine (1937)



Turing Machine

- Mathematical model for computation
- Abstract machine
- Can simulate any algorithm

Turing Machine



- Input band (infinite)
- Reading head
- Control Unit: states
- Transitions / moves

Turing machine – definition

7-tuple $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$ where:

- Q – finite set of states
- Γ - alphabet (finite set of band symbols)
- $b \in \Gamma$ - blank (symbol)
- $\Sigma \subseteq \Gamma \setminus \{b\}$ – input alphabet
- $\delta : (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ – transition function
- $q_0 \in Q$ – initial state
- $F \subseteq Q$ – set of final states

L = left
R = right

Example – palindrome over $\{0,1\}$

- 001100, 00100, 101101 a.s.o. accepted
- 00110, 1011 a.s.o. not accepted

001100

Example – palindrome over $\{0,1\}$

	0	1	b
q_0	(p_1, b, R)	(p_2, b, R)	(q_f, b, R)
p_1	$(p_1, 0, R)$	$(p_1, 1, R)$	(q_1, b, L)
p_2	$(p_2, 0, R)$	$(p_2, 1, R)$	(q_2, b, L)
q_1	(q_r, b, L)		(q_f, b, R)
q_2		(q_r, b, L)	(q_f, b, R)
q_r	$(q_r, 0, L)$	$(q_r, 1, L)$	(q_0, b, R)
q_f			

Delete 0 in left side;
search 0 in right side

Delete 1 in left side;
search 1 in right side

On right is 0 or 1?

Shift right

q_1 and q_2 – process 0 and
1 on the right

q_f – final state

0110

0	1	1	0	
	1	1	0	
	1	1	0	
	1	1	0	
	1	1	0	
	1	1	0	
	1	1	0	

	1	1		
	1	1		
	1	1		
	1	1		
	1	1		
		1		

...

$(q_0, \underline{0}11\underline{0}) \mid - (p_1, \underline{1}1\underline{0}) \mid - (p_1, 1\underline{1}\underline{0})$

$\mid - (p_1, 11\underline{0}) \mid - (p_1, 11\underline{0}\underline{b}) \mid - (q_1, 11\underline{0})$

$\mid - (q_r, 1\underline{1}) \mid - (q_r, \underline{1}1) \mid - (q_r, \underline{b}11)$

$\mid - (q_0, \underline{1}1) \mid - \dots$

	0	1	b
q_0	(p_1, b, R)	(p_2, b, R)	(q_f, b, R)
p_1	$(p_1, 0, R)$	$(p_1, 1, R)$	(q_1, b, L)
p_2	$(p_2, 0, R)$	$(p_2, 1, R)$	(q_2, b, L)
q_1	(q_r, b, L)		(q_f, b, R)
q_2		(q_r, b, L)	(q_f, b, R)
q_r	$(q_r, 0, L)$	$(q_r, 1, L)$	(q_0, b, R)
q_f			

<https://turingmachinesimulator.com>