Compiler Design

Samit Biswas

samit@cs.iiests.ac.in

Department of Computer Science and Technology,
Indian Institute of Engineering Science and Technology, Shibpur

July 23, 2019

Introduction to Compiler Design

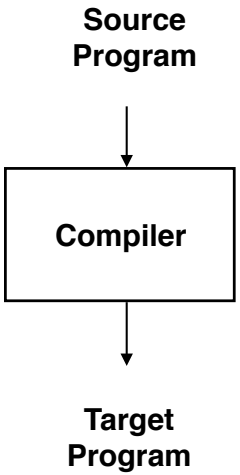
The Phases of a Compiler

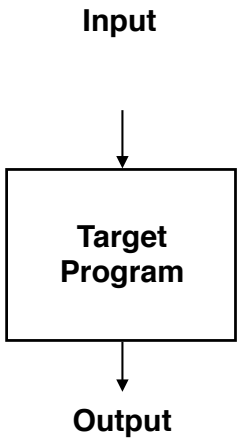
Introduction

Before a program can be run, it first must be translated into a form (Target Program) in which it can be executed by a computer.

Introduction

Before a program can be run, it first must be translated into a form (Target Program) in which it can be executed by a computer.



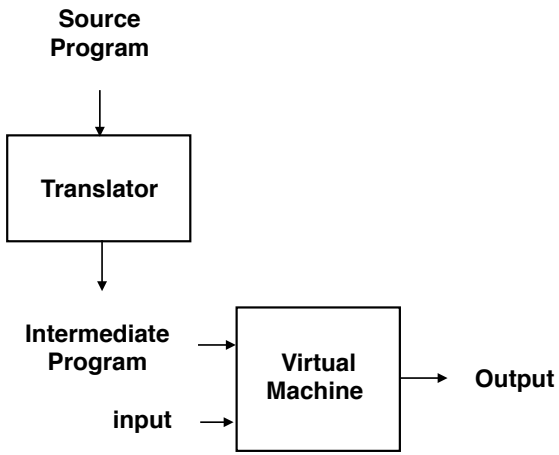


interpreter

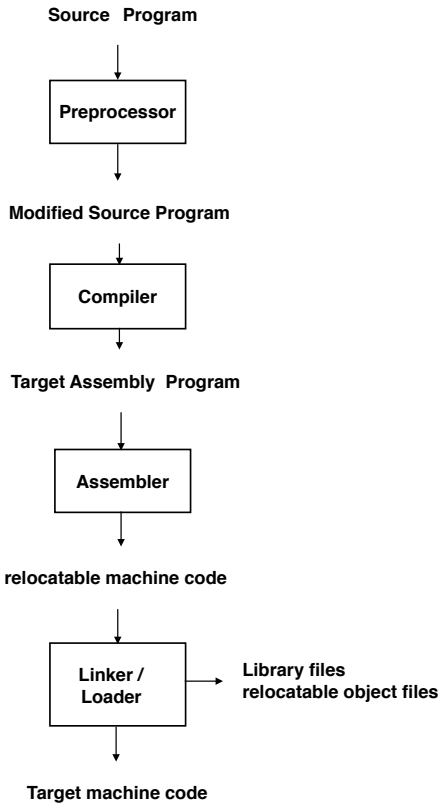
- An *interpreter* is another common kind of language processor. Instead of producing a target program as a translation, an interpreter appears to directly execute the operations specified in the source program on inputs supplied by the user.

Bytecode & Virtual Machine

- A Java source program may first be compiled into an intermediate form called bytecodes. The bytecodes are then interpreted by a virtual machine. A benefit of this arrangement is that bytecodes compiled on one machine can be interpreted on another machine, perhaps across a network.



A typical Language Processing System



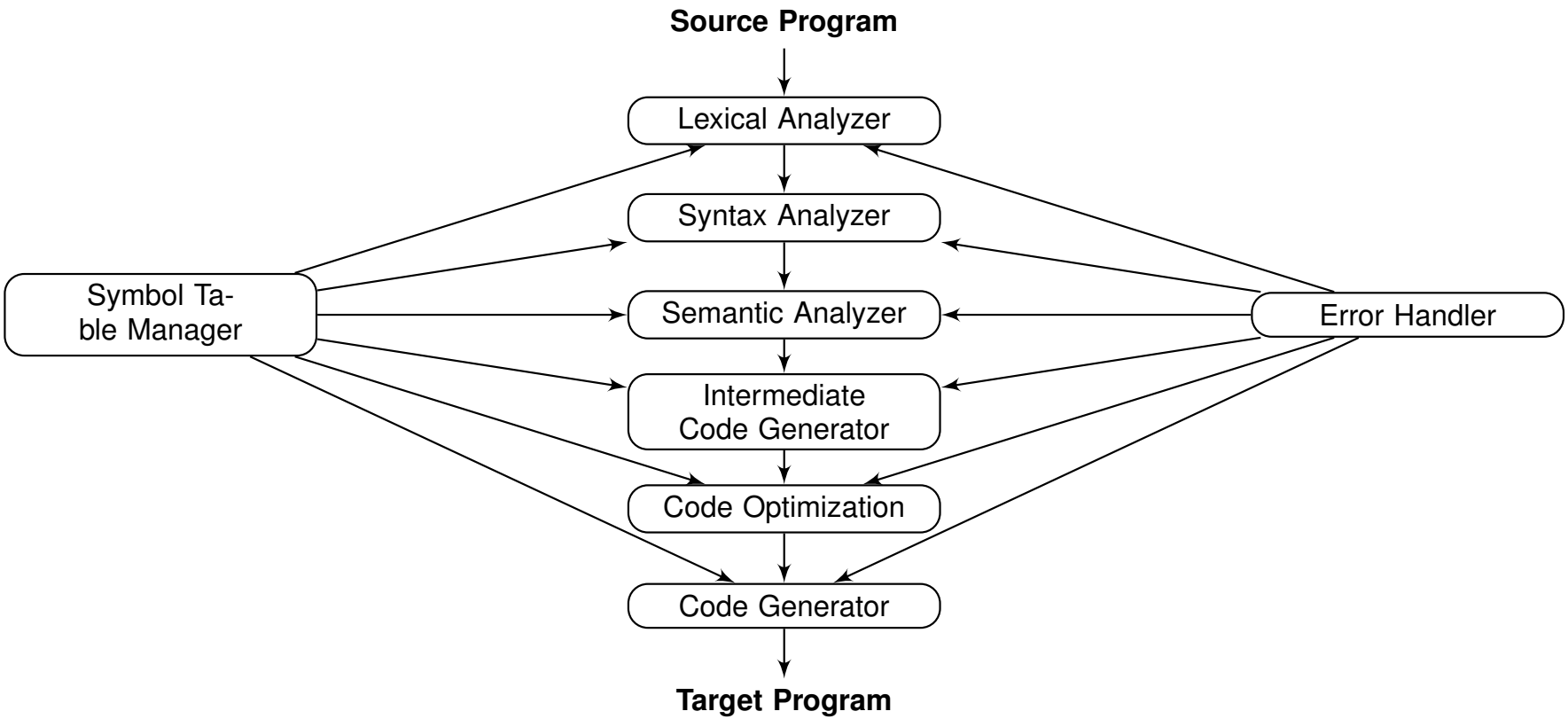
Analysis - Synthesis Model of Compilation

There are two parts of compilation -

- Analysis - It breaks up the source program into constituent pieces and creates an *intermediate representation* of the source program.If the analysis part detects that the source program is either syntactically ill formed or semantically unsound, then it must provide informative messages, so the user can take corrective action.
- Synthesis - It constructs the desired target program from the intermediate representation and the information in the symbol table.

The Phases of a Compiler

Conceptually, a compiler operates in phases, each of which translates the source program from one representation to another.

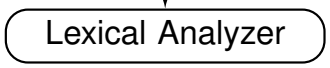


Translation of an Assignment Statement

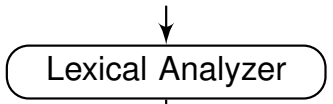
- Lexical Analyser takes the source program as input and produces a long string of tokens.
- Syntax Analyser takes an out of lexical analyser and produce a large tree.
- Semantic Analyser takes an output of Syntax analyser and produces another tree.
- Similarly Intermediate code generator takes a tree as an input produced by Semantic analyser and produces Intermediate code.

$$position = initial + rate * 60$$

$$position = initial + rate * 60$$

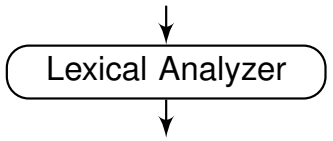


$$position = initial + rate * 60$$



$$id_1 = id_2 + id_3 * 60$$

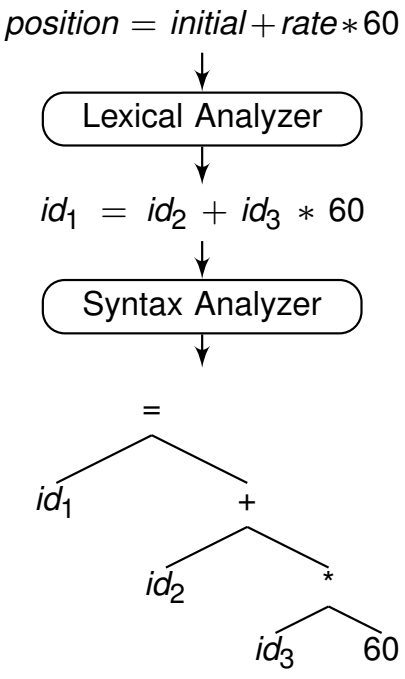
$position = initial + rate * 60$



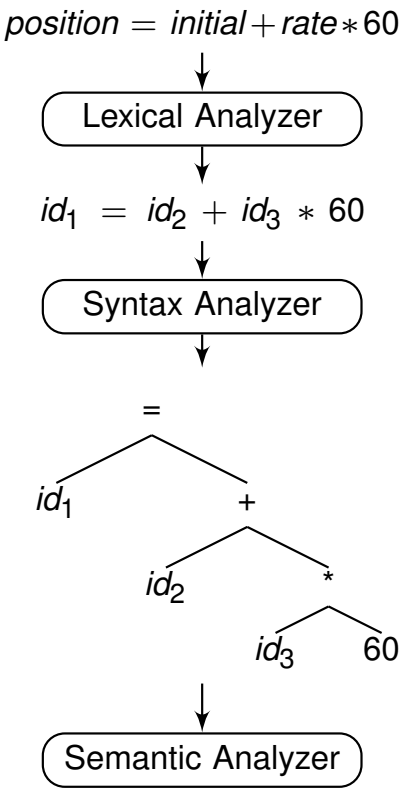
$id_1 = id_2 + id_3 * 60$

SPACE FOR
SYMBOL TABLE

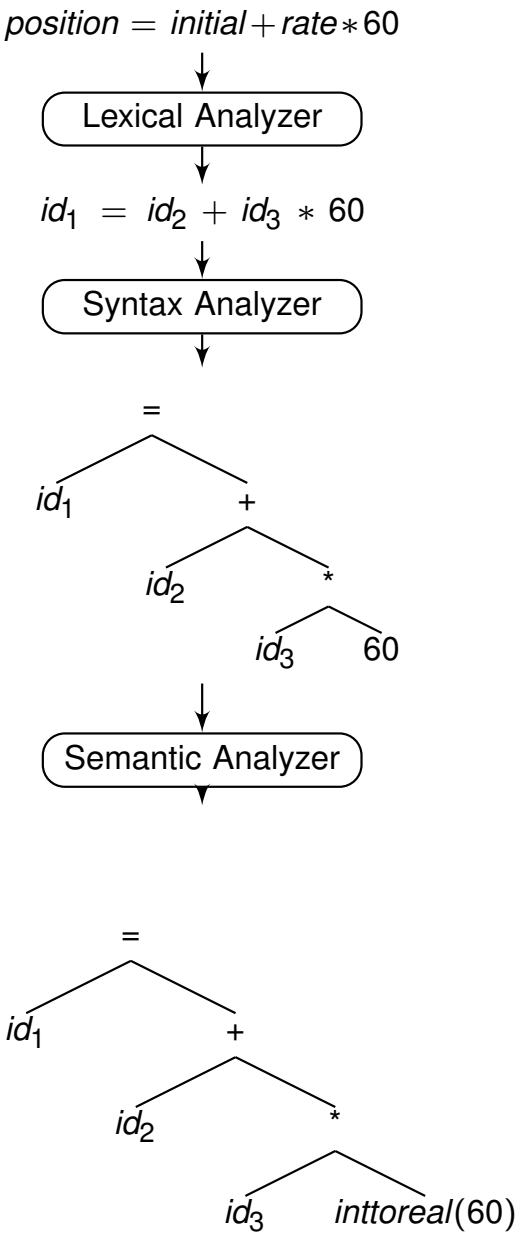
SPACE FOR
SYMBOL TABLE



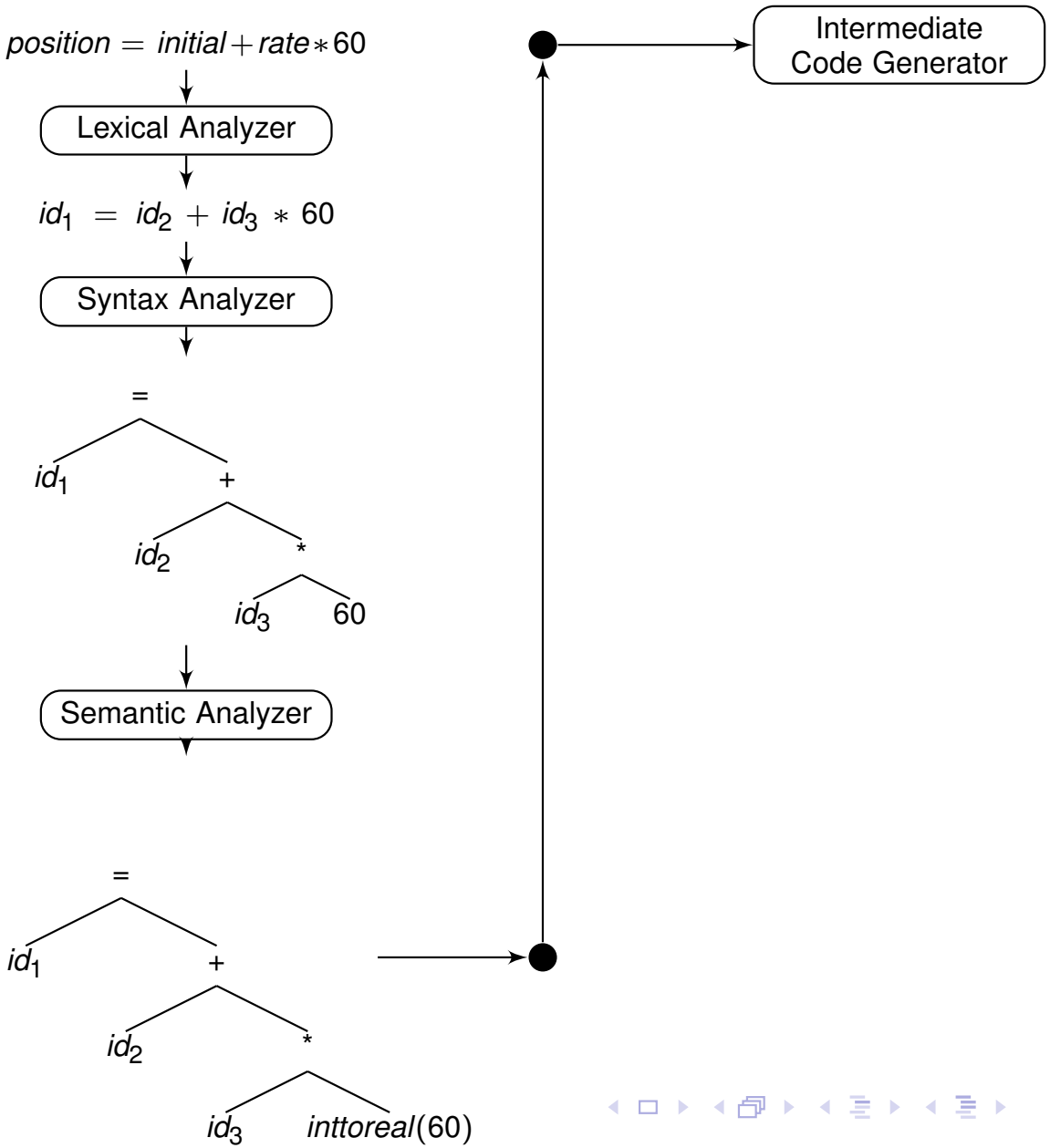
SPACE FOR
SYMBOL TABLE



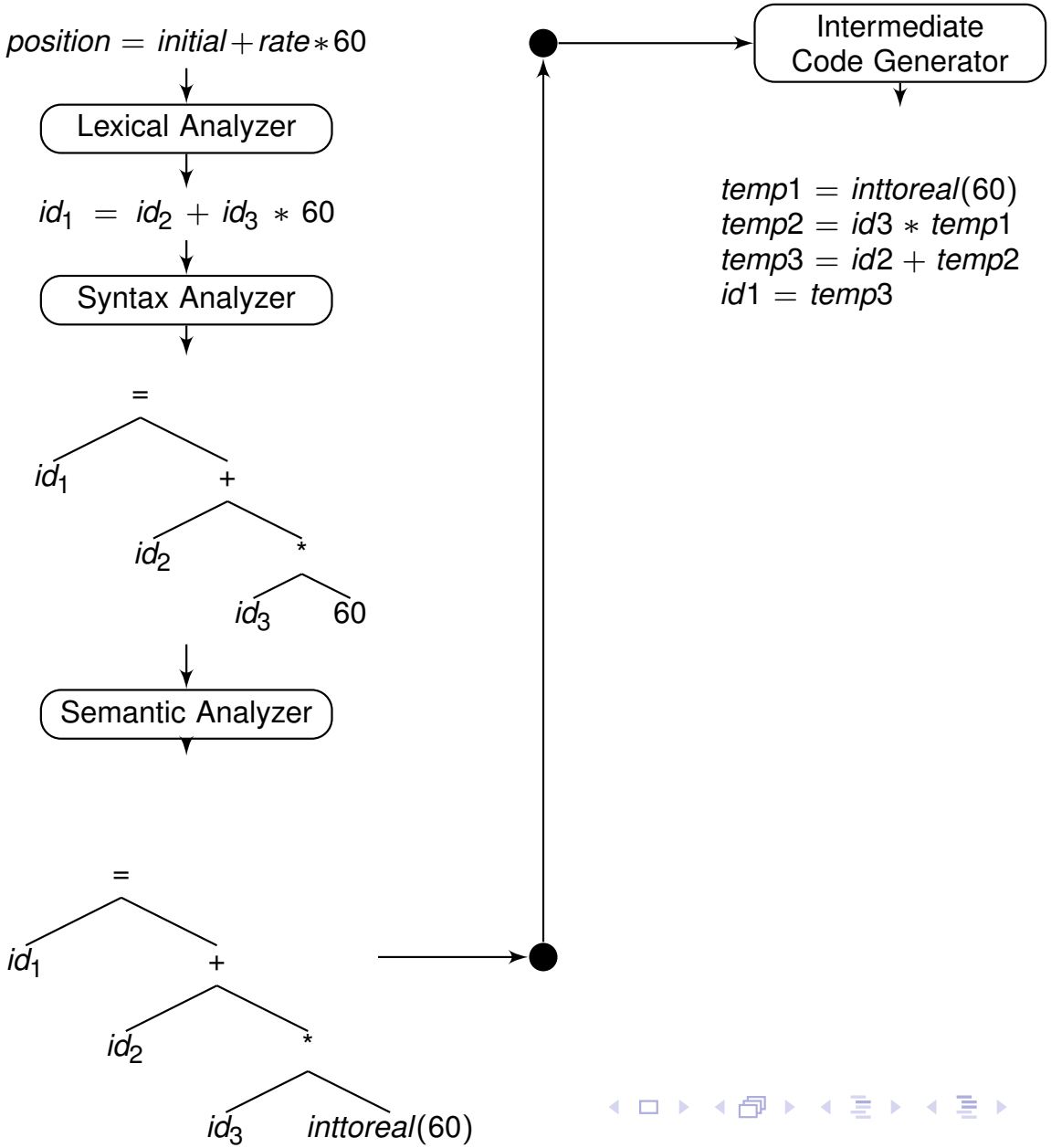
SPACE FOR
SYMBOL TABLE



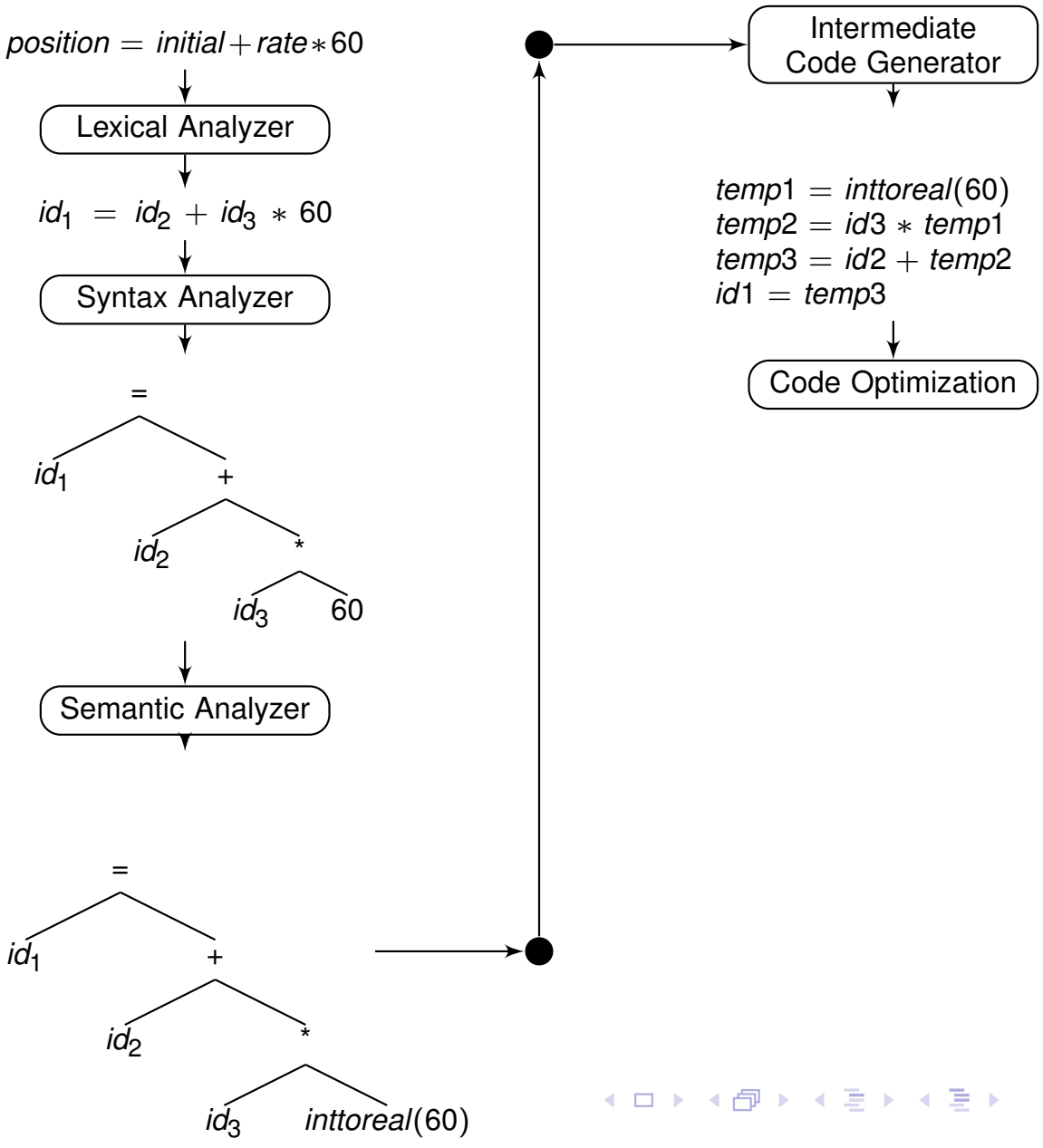
SPACE FOR
SYMBOL TABLE



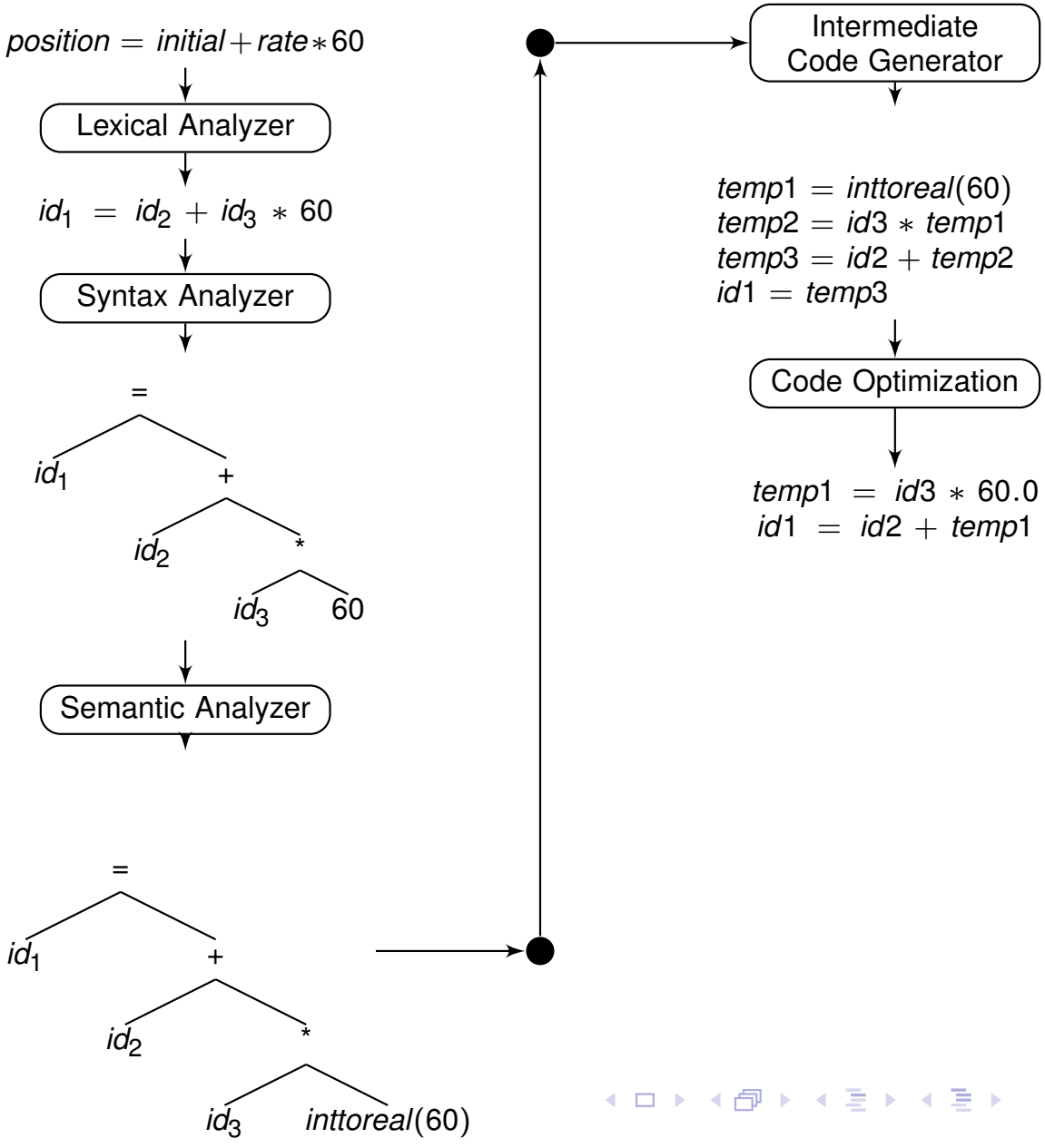
SPACE FOR
SYMBOL TABLE



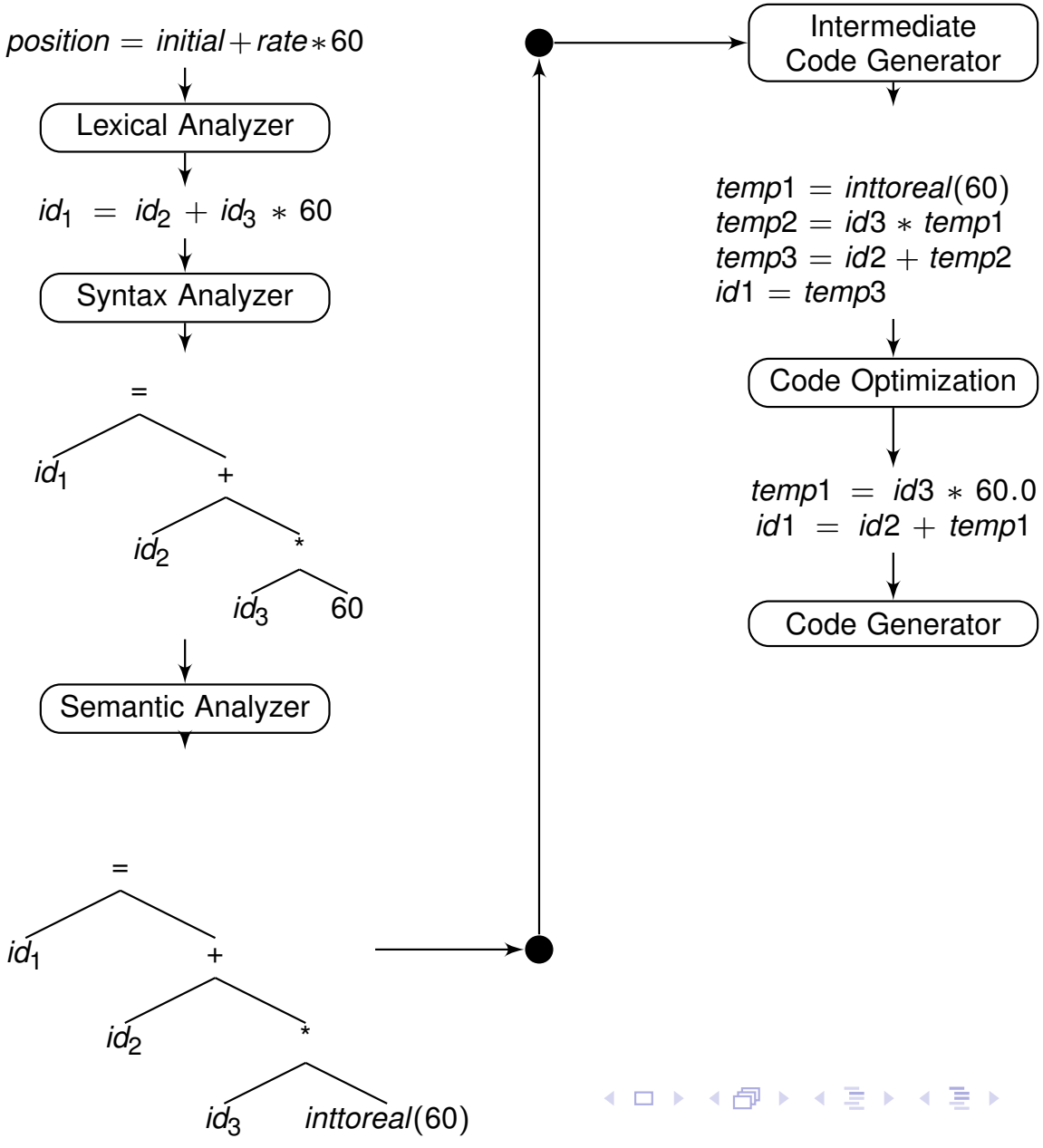
SPACE FOR
SYMBOL TABLE



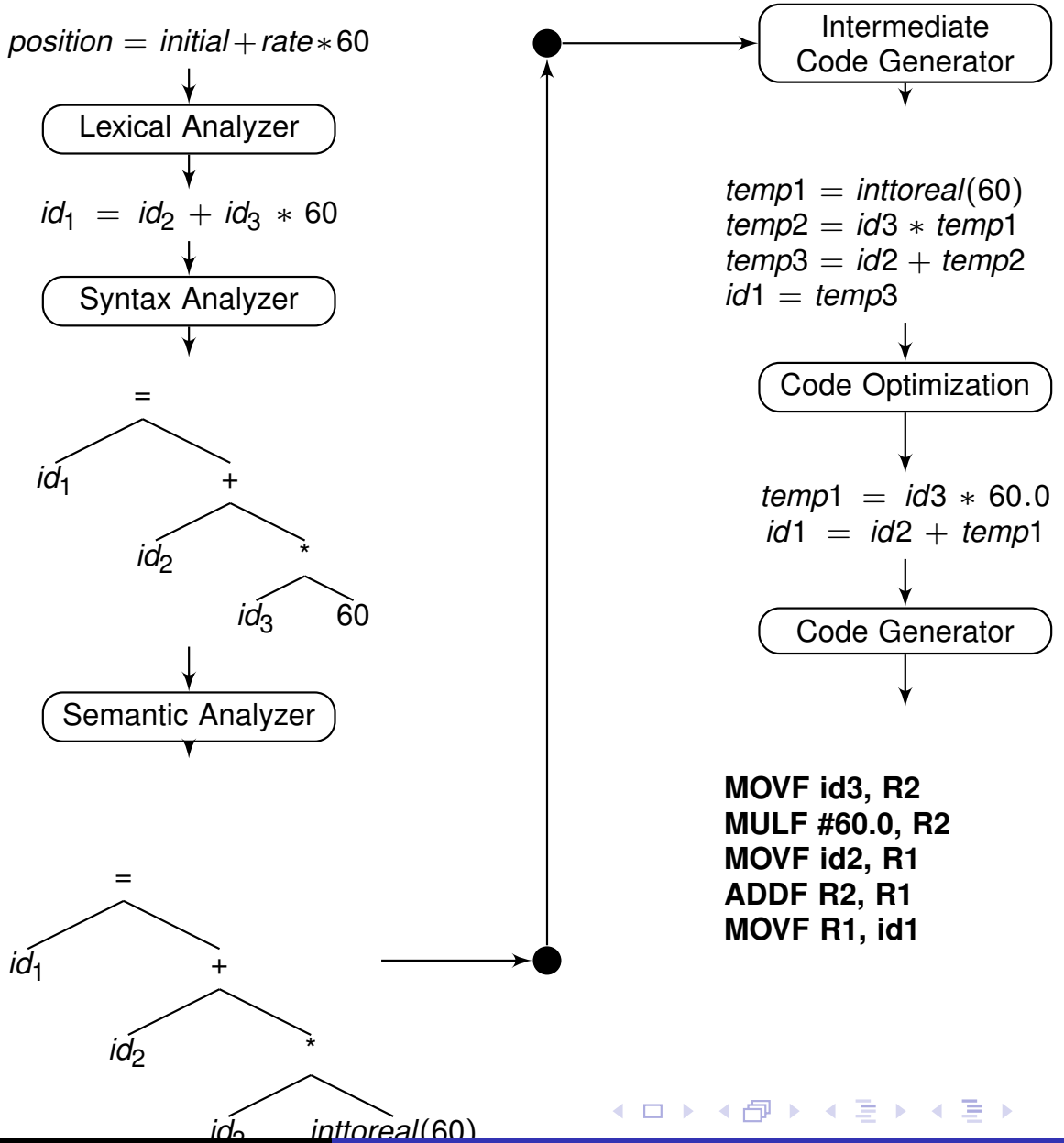
SPACE FOR
SYMBOL TABLE



SPACE FOR
SYMBOL TABLE



SPACE FOR
SYMBOL TABLE



Compiler Construction Tools:

Software development tools are available to implement one or more compiler phases-

- **Scanner Generators** that produce lexical analysers from a regular expression.
- **Parser Generators** that automatically produce syntax analyser.
- Syntax Directed translation Engine.
- *Code Generator* that produce a code generator from a collection of rules for translating each operation of the intermediate language into the machine language for a target machine.
- Compiler Construction tool kits that provide an integrated set of routines for constructing various phases of a compiler.

References

- Alfred V. Aho, Ravi Sethi, Jeffrey D Ullman, “Compilers Principles Techniques and Tools”, Pearson Education.

Thank You