CS251: Introduction to Language Processing

Intermediate Code Generation

Vishwesh Jatala

Department of CSE

Indian Institute of Technology Bhilai

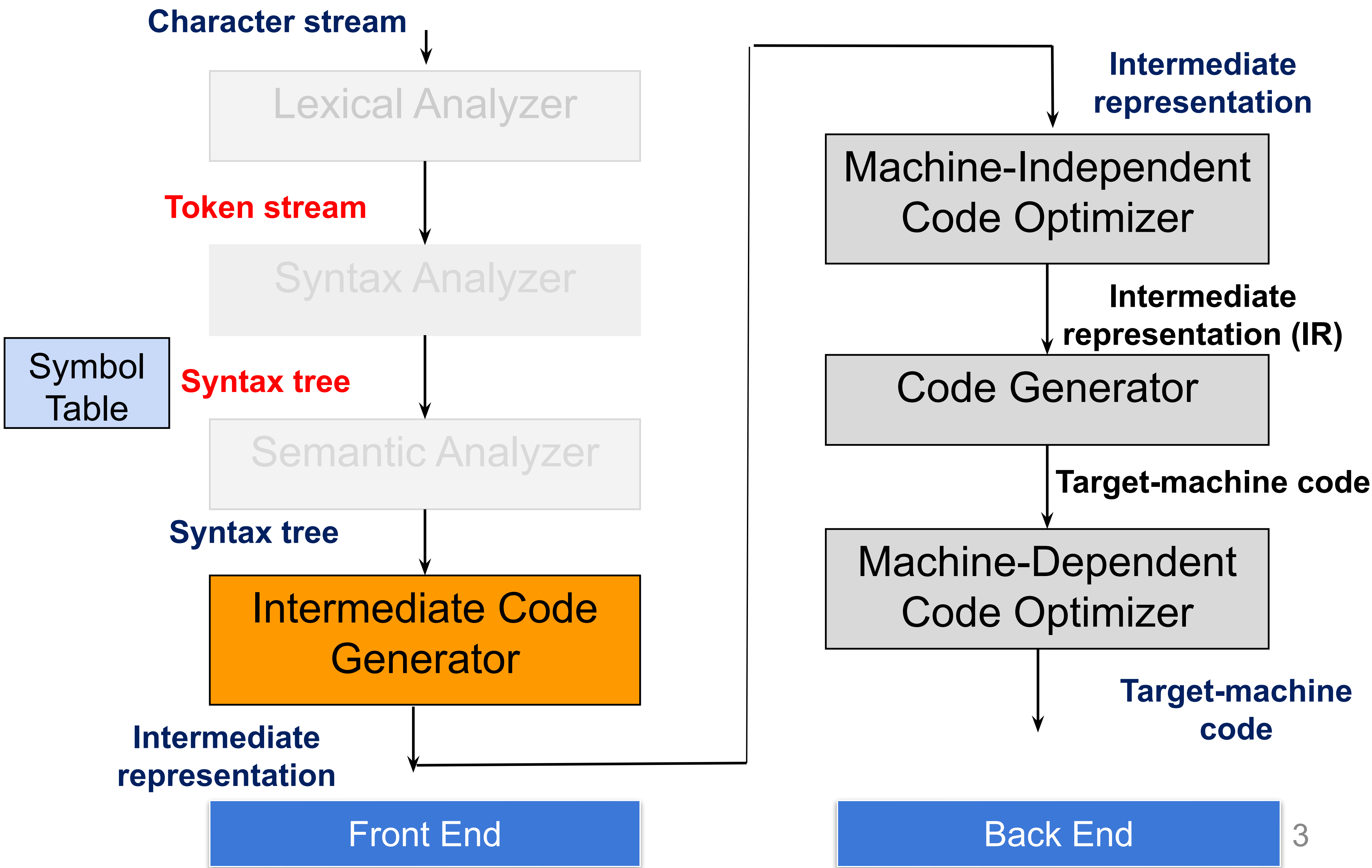
vishwesh@iitbhilai.ac.in

2023-24-M

Acknowledgement

- References for today's slides
    - *Lecture notes of Prof. Amey Karkare (IIT Kanpur) and Late Prof. Sanjeev K Aggarwal (IIT Kanpur)*
    - *Course textbook*

Compiler Design



Recap

- Intermediate code generation
  - Expressions
    - Arithmetic
    - Boolean
    - Arrays
  - Statements
    - Assignment
    - Control flow

      if, if-else, while

Outline

- Intermediate code generation
  - Declarations
    - Type checking

Type system

- What is a type in the programming language?

Type system

- A type is a set of values and operations on those values
- A language's type system specifies which operations are valid for a type
- The aim of type checking is to ensure that operations are used on the variable/expressions of the correct types

Type system

- Languages of three categories w.r.t type:
  - "untyped"
    - No type checking needs to be done
    - Assembly languages
  - Statically typed
    - All type checking is done at compile time
    - Algol class of languages
    - Also, called strongly typed
  - Dynamically typed
    - Type checking is done at run time
    - Mostly functional languages like Lisp, Scheme etc.

Type systems

- Static typing
  - Catches most common programming errors at compile time
  - Avoids runtime overhead
- Most code is written using static types languages
- In fact, developers for large/critical system insist that code be strongly type checked at compile time even if language is not strongly typed.

Type expression

- Type of a language construct is denoted by a type expression
  - It is either a basic type OR
  - it is formed by applying operators called *type constructor* to other type expressions

Type expression

- **Basic types:**
  - integer, char, float, boolean
- **Constructed type:**
  - array, record, pointers, functions
- **Enumerated type:** (violet, indigo, red)

Type Constructors

- Array: if T is a type expression then array(I, T) is a type expression denoting the type of an array with elements of type T and index set I

  int A[10];

  A can have type expression array(0 .. 9, integer)

- Product: if T1 and T2 are type expressions then their Cartesian product T1 * T2 is a type expression
  - Pair/tuple

Type constructors

- Records: it applies to a tuple formed from field names and field types. Consider the declaration

```
type student = record
        id:   integer;
        name:  array [1 .. 15] of char
       end;

var s: student;
```

Type constructors

- Pointer: if T is a type expression then pointer(T) is a type expression denoting type pointer to an object of type T

Specifications of a type checker

- Consider a language which consists of a sequence of declarations followed by a single expression

$P \rightarrow D \; ; E$

$D \rightarrow D \; ; D \mid id : T$

$T \rightarrow char \mid integer \mid T[num] \mid T*$

$E \rightarrow literal \mid num \mid E\%E \mid E\,[E] \mid *E$

Specifications of a type checker …

- A program generated by this grammar is

  key : integer;
  key %1999

- Assume following:
  - basic types are char, int, etc
  - all arrays start at 0
  - char[256] has type expression array(0 .. 255, char)

Rules for Symbol Table entry

$D \rightarrow id : T$	addtype(id.entry, T.type)
$T \rightarrow char$	T.type = char
$T \rightarrow integer$	T.type = int
$T \rightarrow T_1 *$	T.type = pointer(T_1.type)
$T \rightarrow T_1 [num]$	T.type = array(0..num-1, T_1.type)

Type checking for expressions

$E \longrightarrow$ literal	E.type = char
$E \longrightarrow$ num	E.type = integer
$E \longrightarrow$ id	E.type = lookup(id.entry)
$E \longrightarrow E_1 \% E_2$	E.type = if E_1.type == integer and E_2.type==integer then integer else type_error

Type conversion

- Consider expression like x + i where x is of type real and i is of type integer
- Internal representations of integers and reals are different in a computer
  - – different machine instructions are used for operations on integers and reals
- The compiler has to convert both the operands to the same type
- Language definition specifies what conversions are necessary.

Type conversion

- Type checker is used to insert conversion operations:

    x + i

    x + inttoreal(i)

- Type conversion is called implicit/coercion if done by compiler.
- It is limited to the situations where no information is lost
- Conversions are explicit if programmer has to write something to cause conversion

Type checking for expressions

$E \longrightarrow num$ E.type = int

$E \longrightarrow num.num$ E.type = real

$E \longrightarrow id$ E.type = lookup(id.entry)

$E \longrightarrow E_1 \text{ op } E_2$ E.type =

if E_1.type == int && E_2.type == int
then int
elif E_1.type == int && E_2.type == real then real
elif E_1.type == real && E_2.type == int
then real
elif E_1.type == real && E_2.type==real
then real

Next Lecture

- Intermediate code generation
  - Functions
  - Runtime environment