Basic Components — Lexical

Steven Zeil

Aug. 28, 2003

Contents

3 Translation **1**
3.1 Phases of Translation 1
3.1.1 Compilers 1
3.1.2 Interpreters 1
3.2 Lexical 2

Basic Components of Programming Languages

1. History
2. Classification
3. Translation

---

3 Translation

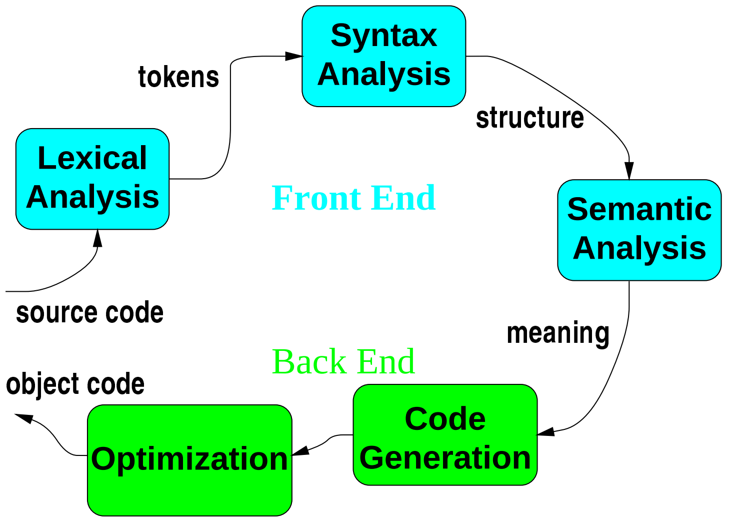
1. Phases of Translation
2. Lexical: What are the *words*?
3. Syntax: What is the *grammatical structure* combining the words into sentences?
4. Semantics: What do the sentences *mean*?

3.1 Phases of Translation

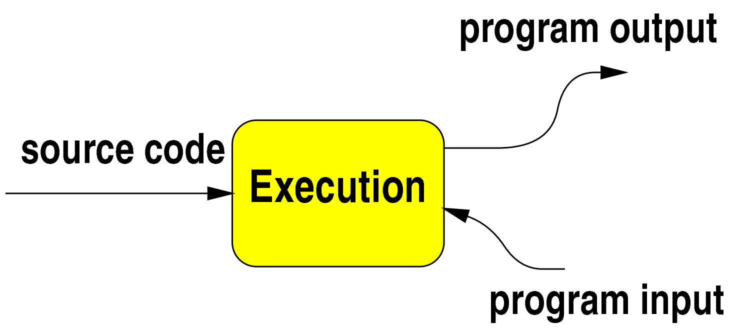
Translators are divided into

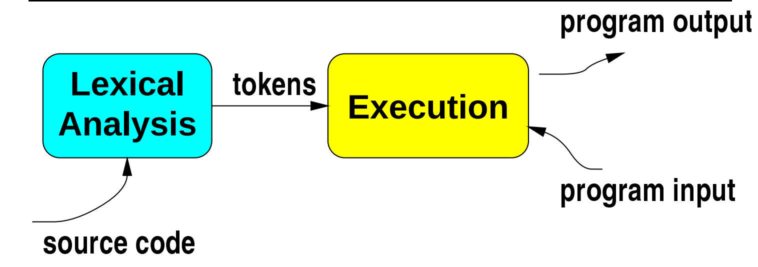
- **compilers**, that produce machine code as output
- **interpreters**, that directly execute the source program

3.1.1 Compilers

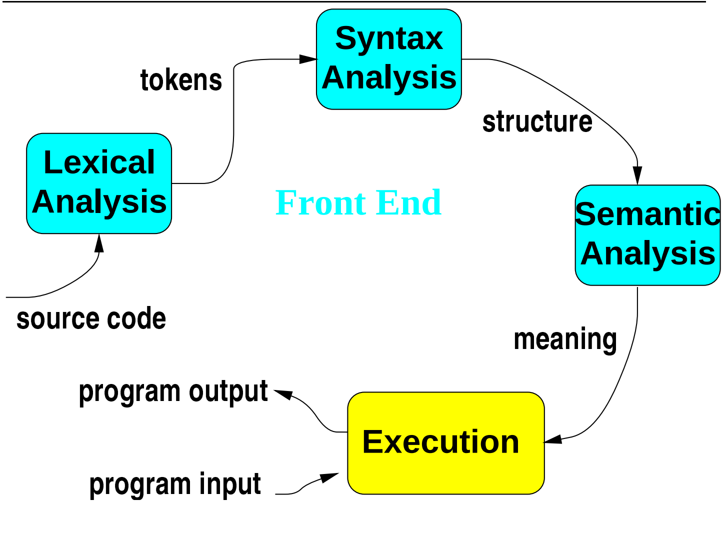


3.1.2 Interpreters



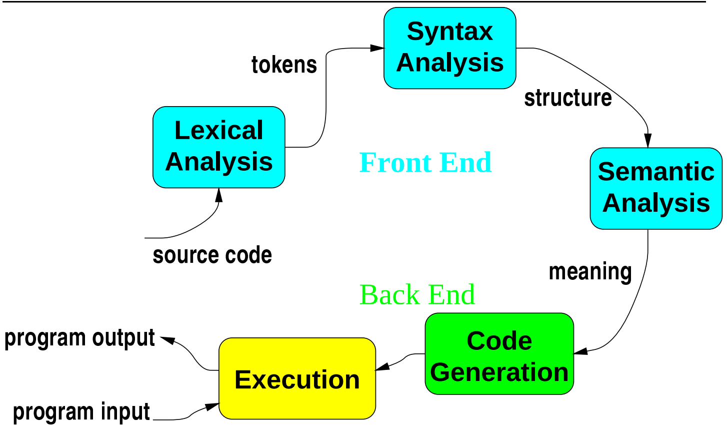


In practice, almost all intepreters share at least some front-end phases with compilers.



"Hybrid" interpreters

- compile for an imaginary **virtual machine**,
- then **emulate** execution of the virtual machine.



Hybrid Interpreter

3.2 Lexical

Characters occur in groups that have an "atomic" meaning in a language.

Such groups are called **tokens**. A language may have many different kinds of tokens.

The string of characters that corresponds to a given token is called a **lexeme**.

For example, in the code

```
if X > 1.5 THen
```

we have

tokens:	if	identifier	greater	number	then
lexemes:	`if`	`X`	`>`	`1.5`	`THen`

Tokens

The kinds of tokens vary from one language to another, but some common ones are

- constants, operators, identifiers, keywords
  - If a keyword is not allowed to be used as an identifier, it is called a **reserved word**.

In addition, some strings of characters don't contribute to any tokens:

- white space
- comments

Lexemes

The strings that can make up a given kind of token will also vary between languages. E.g.,

identifiers X, longName, long_Name, $name, name, Name, NA ME

reserved words if, IF, iF, end, fi, endif

constants 'abc', "abc", 0.275, .275, 0.275E3, 0.275G3

We describe the lexemes for a token kind either via grammars or via regular expressions.

Regular Expressions

In their simplest form, a **regular expression** R must be one of

- a single character
- ST, the concatenation of two other regular expressions
- $S|T$, the choice of two regular expressions

⋮

Regular Expressions (cont.)

⋮

- $S*$, 0 or more repetitions of a regular expression S
  - (known as the **Kleene closure**)
- (S), a regular expression within parentheses

Example: integers in most languages look sort of like this:

$$(+|-)(0|1|2|3|4|5|6|7|8|9)*$$

But this isn't quite right. Why not?

Common Extensions To Regular Expressions

- $R+$ denotes 1 or more repetitions of R
- $R?$ denotes 0 or 1 occurrence of R
- R_m^k denotes between m and k occurrences of R

Common Extensions (cont.)

- $[abc\ldots]$ is short for $(a|b|c|\ldots)$, where only single characters can appear between the [ ].
  - The notation $a-z$ is also allowed within [ ], to denote a range of consecutive characters.
- Quote characters like $'+'$ that would otherwise be confused.

With these, we can reduce our description of integer lexemes to

$$('+'|-)?[0-9]+$$

What do the following regular expressions describe?

- $0*1*$
- $(0*1*)*$
- $(0*1*)+$
- $(00|01|10|11)*$

What do the following regular expressions describe?

- $0*1*$
- $(0*1*)*$
- $[01]+$
- $(00|01|10|11)*$

What do the following regular expressions describe?

- $0*1*$
- $(0*1*)*$
- $[01]+$
- $(00|01|10|11)*$

What do the following regular expressions describe?

- $0*1*$
- $(0*1*)*$
- $[01]+$
- $(00|01|10|11)*$

What do the following regular expressions describe?

- $0*1*$
- $(0*1*)*$
- $[01]+$
- $(00|01|10|11)*$

Here are some descriptions of identifier lexemes in different languages.

- Pascal:
  $[a-zA-Z][a-zA-Z0-9]*$
- C: $[a-zA-Z\_][a-zA-Z0-9\_]*$
- FORTRAN: $[a-zA-Z][a-zA-Z0-9]_0^5$

Can you explain the differences?

What would be the lexemes for the reserved word "for" in C?
in Pascal?

Not all lexical conventions can be described via regular expressions.

- For example, older languages such as FORTRAN and COBOL had column dependencies.
- ALGOL (published form) used typesetting information: `if` was an identifier, but **`if`** is a reserved word.