Compiler Construction CS510

Lecture Ten

Semantic Analysis

Errors

- Syntactic errors: violet grammar rules and caught by compliers.
- Static Semantic errors : e.g. identifiers are not declared caught by compliers.
- Runtime Errors e.g. division by zero.
- Semantic errors: meaning may not be different from programmer’s intension.
  - Crashes (stops running)
  - Runs forever
  - Produces an answer but not the desired one.

Semantic Analysis

- Parsing cannot catch some errors:

  e.g. :

  - Multiple declarations: a variable should be declared (in the same scope) at most once.
  - Undeclared variable: a variable should not be used without being declared.
  - Type mismatch: e.g., type of the left-hand side of an assignment should match the type of the right-hand side. y=y+3 error (string +number)
  - Wrong arguments: methods should be called with the right number and types of arguments.
  - Classes defined only once.
  - Methods in a class defined only once.

Attribute Grammars

- Regular expressions used for scanner phase.
- Context-free grammars used for parser phase.
- Attribute grammars method of describing semantic analysis.
- An attribute is any property of a programming language construct
  - The data type of a variable
  - The value of an expression
  - The location of a variable in memory
- Attributes are associated with the grammar symbols of the language. If X is a grammar symbol, and a is an attribute associated to X, then we write X.a for the value of a associated to X.

Attribute Grammars

- **<u>Example</u>** :

  num → nm digit |digit

  digit → 0|1|2|3|4|5|6|7|8|9

- *Grammar Rule: num→ digit*
- *Semantic Rule: num. val= digit.val*

- *Grammar Rule : →num digit*

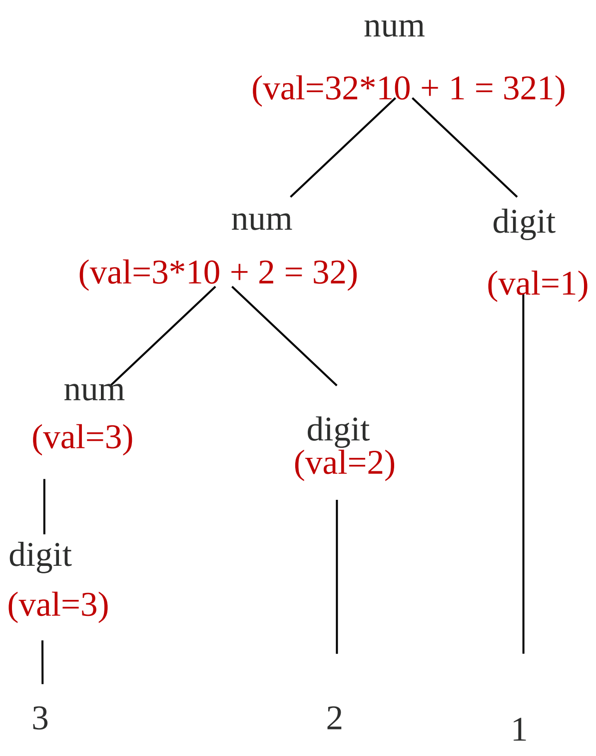
  The *number* on the right will have a different value from that of the *number* on the left

- *Semantic Rule:* $numb_1 \rightarrow num_2\ digit$

Attribute Grammars

Grammar Rule	Semantic Rules
$num_1 \rightarrow num_2\ digit$	$num1.val = num_2.val * 10 + digit.val$
$num \rightarrow digit$	$num.val = digit.val$
$digit \rightarrow$ 0	$digit.val = 0$
$digit \rightarrow$ 1	$digit.val = 1$
$digit \rightarrow$ 2	$digit.val = 2$
$digit \rightarrow$ 3	$digit.val = 3$
$digit \rightarrow$ 4	$digit.val = 4$
$digit \rightarrow$ 5	$digit.val = 5$
$digit \rightarrow$ 6	$digit.val = 6$
$digit \rightarrow$ 7	$digit.val = 7$
$digit \rightarrow$ 8	$digit.val = 8$
$digit \rightarrow$ 9	$digit.val = 9$

parse tree for the number **321**



Attribute Grammars

The computation of attributes is described using equations or semantic rule.

There are two types of attributes:

- Synthesized attributes

    Values computed from children

- Inherited attributes

    Values computed from parent and siblings

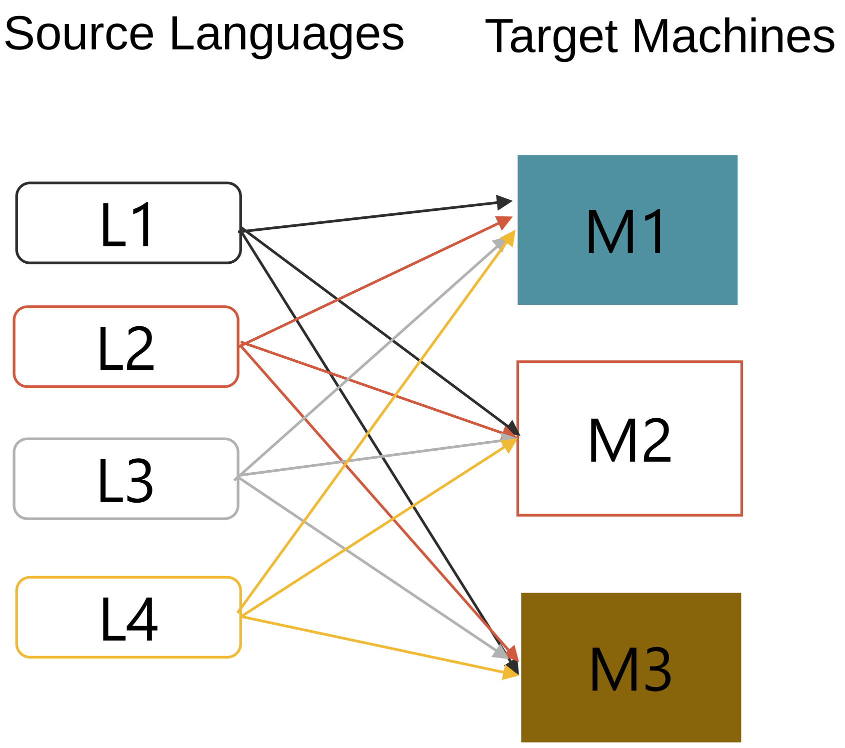
Runtime Environments

Code Generation

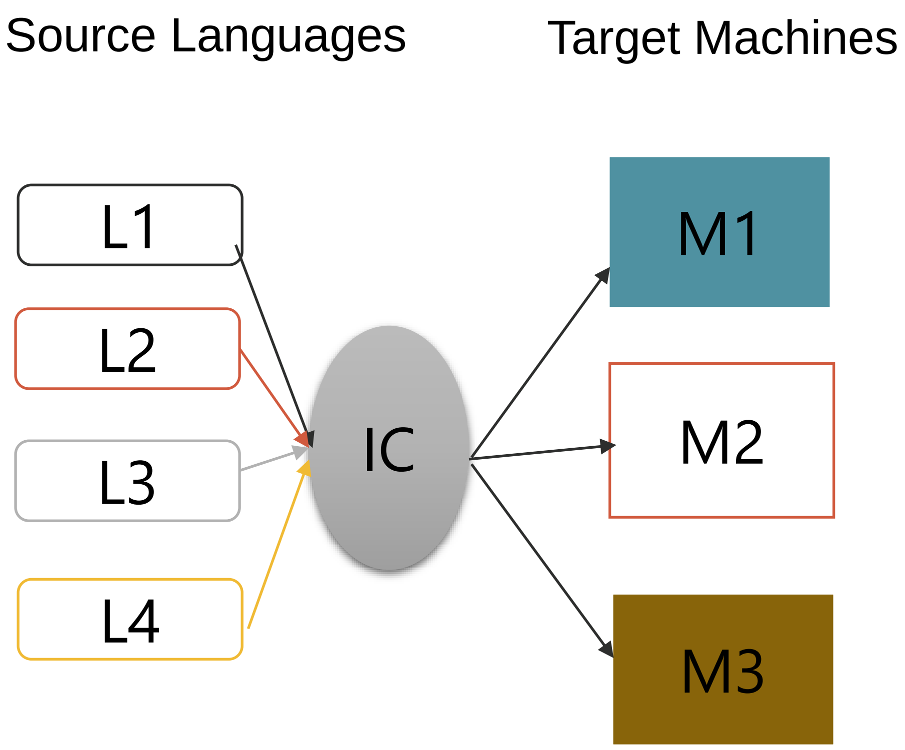
Code generation

- Code generation phase depends on :
  - Target architecture.
  - The structure of the runtime environment.
  - Operating system of the target machine.
- In this Lecture we will study generate intermediate code (universal form of assembly code that must be processed further by an assembler)
- Intermediate code is relatively target machine independent.
- Two popular forms of intermediate code: three-address code and P-code.

Why Intermediate Code?



L*M Code Generator



L+M Code Generator

Why Intermediate Code?

- Generating machine code directly from source code. With L languages and M target machines, L*M code generators is needed.
- converting source code to an intermediate code (machine-independent). With L languages and M target machines, L+M code generators is needed.

Three-Address Code

X=y op z

2*a+(b-3)

The three-address code for 2*a+(b-3)

tl = 2 *a

T2=b-3

t3 = t1 + t2

Example

a+b*c-d/(b*e)

- t1 = b*c
- t2 = a+t1
- t3 = b*e
- t4 = d/t3
- t5 = t2-t4

Three-Address Code Instructions

If Statement

If (E) S

t1=E

If false t1 goto L1

code for S

L1:

Exit

Three-Address Code Instructions

If Statement

If (E) S1 else S2

```
t1=E
If false t1 goto L1
        code for S1
        goto L2
L1:
        code for S2
L2:
        Exit
```

Three-Address Code Instructions

While Statement

while (E) do S

```
L1:
        t1=E
        If false t1 goto L2
                code for S
                goto L1
L2:
        Exit
```