Semantic Analysis

Mooly Sagiv
Tel Aviv University
sagiv@math.tau.ac.il
and
Reinhard Wilhelm
Universität des Saarlandes
wilhelm@cs.uni-sb.de

June 5, 1997

Subjects

- The task of semantic analysis
- Concrete and abstract syntax
- Syntax directed translation
- Examples
  - Context conditions
  - Scope and visibility rules
  - Type consistency
- The UClax front-end
- Symbol table organization
- Attribute grammars

“Standard” Compiler Structure

source(text)	
↓	
lexical analysis(7)	finite automata
↓	
tokenized-program	
↓	
syntax analysis(8)	pushdown automata
↓	
syntax-tree	
↓	
semantic-analysis(9)	attribute grammar evaluators
↓	
decorated syntax-tree	
↓	
optimizations(10)	abstract interpretation + transformations
↓	
intermediate rep.	
↓	
code-generation(11, 12)	tree automata + dynamic programming + $\cdots$
↓	
machine-program	

Program Rejection in the Analysis Phase

- analysis phase tasks reject programs violating constraints,

  scanner: words not in the regular language,
  parser: sequences of symbols (tokens) not in the context–free language.

- the rejection power of the mechanisms is limited,

  regular expressions cannot describe matching parentheses,
  context–free grammars cannot describe declarativity rules.

“Semantic” Constraints

- Requirements related to the “context” in which a construct occurs
- These **cannot** be formulated using context free grammars
- Examples:
  - Each variable used in a program must be declared in an enclosing scope
  - Variables must be uniquely declared (within a scope)
  - Types of operands and operators in expressions must match

Strings violating semantic constraints are rejected by the compiler.

Concrete and abstract syntax

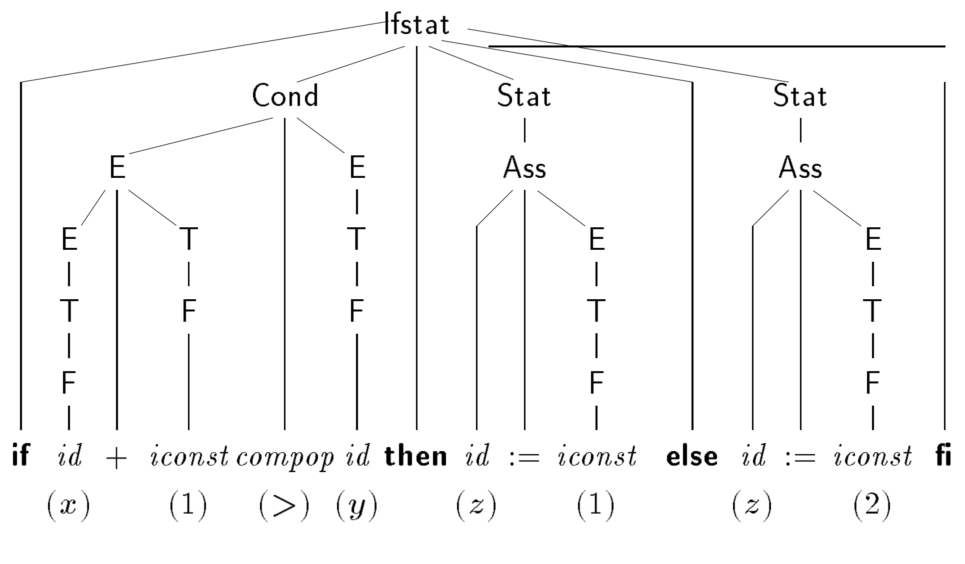
Concrete syntax:

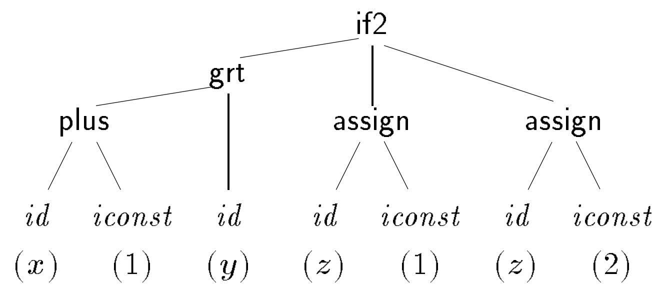
- Used in parsing
- Keywords and punctuation symbols are used to identify the structure of the input
- Hierarchies of nonterminals are used to express precedences of operators (E, T, F, P in expression grammar)
- Neither is relevant after parsing, i.e. once the tree exists

Abstract syntax:

- Used for program representation and translation
- Nodes in the abstract tree may be linked (for further usages)

Example


Ifstat
Cond
Stat
Stat
E
E
Ass
Ass
E T E E
T F T T T
F F F F
if id + iconst compop id then id := iconst else id := iconst fi
(x) (1) (>) (y) (z) (1) (z) (2)


if2
grt
plus
assign
assign
id iconst id id iconst id iconst
(x) (1) (y) (z) (1) (z) (2)

Terminology

identifier denotes objects[1], e.g., variables, constants, types, functions, procedurs, . . .

declaration introduces an identifier, binds it to an object,

defining occurrence of an identifier, an occurrence of an identifier in its declaration,

applied occurrences of an identifier, all others.

scope of a defining occurrence, the part of the program, in which an applied occurrence of an identifier may refer to this defining occurrence,

visibility of a defining occurrence of an identifier, directly visible in the scope or made visible by directives, name extensions, etc.

[1] Not in the sense of OO languages!

type of an object constraints, which operations can be applied to the object at execution.

identification of identifiers associates applied occurrences of identifiers with the defining occurrences they refer to.

overloading, more than one defining occurrrence visible at an applied occurrence, different parameter profiles required,

overload resolution identifies the one defining occurrence for an applied occurrence.

UClax Example

```
PROGRAM test ;
DECLARE x: INTEGER ;
BEGIN
x := 1 ;
END.
```

Decorated Abstract Syntax

```
Program(Id(test),
Block(NstLevel(0),DeclList(, Decl(NstLevel(0), LineNumber(2), VarDecl(Id(x), SimpleType(Integer)))
)
,
StatSeq(,
Stat(AssignStat(Var(SimpleVar(Id(x), NstLevel(0), VarDecl(Id(x), SimpleType(Integer))),SimpleType(Integer)), Expr(IntConstExpr(IntConst(1)),SimpleType(Integer)), SimpleType(Integer))
LineNumber(4))
)
)
)
```