Static (Semantic) Analysis

(Third) Last Step of Compiler Front-End

Compositional and Contextual Property Analysis

Compositional and Contextual Property Analysis

- **Main Properties**:
  - Uniqueness
  - Well formed (control) Structures
  - Correlated Occurrences
- **Types:**
  - Type Systems
  - Type Checking
  - Type Inference

Uniqueness - Control Structure

Uniqueness

- no name collision (for instance, in a block, or definition..)

Well Formed Structures

- Checkings for correct use of construct composition:
  - in C, *break* may only, occur inside blocks;
  - in Java, no *hiding variable* is permitted in blocks
  - in Java, classes implementing interface must contain definitions for the interface methods
  - in Pascal, the *for-block* cannot modify *for-index*
  - Expressions used as *by-reference* parameters must have *l-values*

  ….

Correlated Occurrences

Correlated Occurrences

- Specific Checkings for the correct use of the constructs:
  - A *declared identifier* must occur in some use
  - In many languages, a used identifier must be declared with the right scope.
  - in Pascal, the *function body* must contain an assignment to the function name;
  - In C, non-void procedure bodies must contains *return exp*

Types

Are a Special Kind of Terms that:

- *are assigned* to the program structures
- *are fundamental for classifying* program structures with the aim of:
  - studying (semantic) correctness of the structure use
  - preventing run-time errors
  - allow code optimizations at compile/run-time
- *are expressed* by a suitable set of expressions:
  - called **Type Expressions** and
  - are obeying the laws of a specific system, called **Type System**

Type Expressions

1) **Basic (or Atomic) Types**

real, int, char, file, unit

2) *Type Constructors for* **Derived Types**

array: I x T -> array(I,T)
product: T1 x T2 -> T1 ✖ T2
record: ({i1}xT1)x..x({ik}xTk) -> record(i1:T1...ik:Tk)
enumerated: {v1,..,vk} -> (v1,..,vk)
pointer: T -> pointer(T)
function: TD x TC -> TD ➔ TC
procedure: TD -> TD ➔ unit

3) **Type Identifier** (for naming)

4) **Type Variables** (for polimorphic types)

Type System = Types + Rules

Rules

- Form the set of rules that associate types to the Lang. Structure
- They depend on the language
- May exhibit very different properties for the different languages
- But, always only one type can be associated to each program structure

$$\frac{\Gamma \rightarrowtail g:T1 \Leftarrow T2 \qquad \Gamma \rightarrowtail e:T1}{\Gamma \rightarrowtail g(e):T2}$$

Function invocation in a
type Checking System

$$\frac{\Gamma \rightarrowtail g:S \mid C_g \qquad \Gamma \rightarrowtail e:T \mid C_e \qquad C = C_g \cup C_e \cup \{S = T1 \Leftarrow T2 \,,\, T = T1\}}{\Gamma \rightarrowtail g(e):T2 \mid C}$$

Function invocation in a
type Inference System
using Unification as Constraint Sover for C

Applications

- Selection of the Grammar G
- Examples of *correlated occurrences* for L(G)
- How to do Analysis: Choise of the attributes
- The Attribute Grasmmar
- Another Example