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 compositions:
  - 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**

Applications

- Selection of the Grammar G
- Examples of *correlated occurrences* for L(G)
- Example of Type System and Type Checking

The Grammar
Declarations-Commands

The Imperative Language Simple

[0]**P**rogram= **D**eclaration **C**ommand**s** | [1]**C**ommand**s**

[2]**D**::= **ide O**Theridentifiers **;**

[3]**O**::= **ide O** | [4] ε

[5]**Cs**::= **C**ommand **; Cs** | [6] ε

[7]**C**::= **A**ssign | [8] **W**hile

[9]**A**::= **ide := E**xpression

[10]**W**::= **while E do C Cs endwhile**

The Grammar
Expressions

[11]**E**::= **F E'**

[12]**E'**::= **op-l**ower **F E'** | [13] ε

[14]**F**::= **T**erm **F'**

[15]**F'**::= **op-h**ight **T F'** | [16] ε

[17]**T**::= **num** | [18] **ide** | [19] **(E)**

Correlated Occurrences

1) All the used identifiers are correctly declared

2) All ther declared identifier are used

3) All the variables have an assigned values before the use

4) Iterator guard expressions have type boolean

All the used identifiers are correctly declared
Choice of the attributes

Attribute Plan: names and properties of the attributes

u: - set of the used identifiers
- **synthesized of Su={Cs,C,A.W,E,E',F,F',T}**
- **$\forall X \in Su$, X.u=I iff $X \Rightarrow^* \alpha = \alpha_1 ... \alpha_n \in \Sigma^*$ and if α_i=ide then α_i.lexeme∈I**

d: - set of the declared identifiers -
- **synthesized of Sd={D,O}**
- …

r: - set inclusion of the used into the declared ones
- **synthesized of {P}**
- **r=u≤d**

Values and auxiliary functions that are used in the actions

Set are handled by list with the following operations

cons: ide X ide-list --> ide-list
emptylist : --> ide-list
append: ide-list X ide-list --> ide-list
included: ide-list X ide-list --> boolean
isempty: ide-list --> boolean

[0]**P**::= **D Cs**	**P**.r:= include(**Cs**.u, **D**.d)
[1]**P**::= **Cs**	**P**.r:= isempty(**Cs**.u)
[2]**D**::= **var ide O**	**D**.d:= cons(**ide**.lexeme, **O**.d)
[3]**O1**::= **, ide O2**	**O1**.d::= cons(**ide**.lexeme, **O2**.d)
[4]**O**::=ε	**O**.d::= emptylist
[5]**Cs1**::= **; C Cs2**	**Cs1**.u:= app(**C**.u, **Cs2**.u)
[6]**Cs**::= ε	**Cs**.u:= emptylist
[7]**C**::= **A**	**C**.u:= **A**.u
[8]**C**::=**W**	**C**.u:=**W**.u
[9]**A**::= **ide := E**	**A**.u:= cons(**ide**.lexeme, **E**.u)
[10]**W**::=**while E do C Cs edw**	**W**.u:= app(**E**.u, app(**C**.u, **Cs**.u))
[11]**E**::= **F E'**	**E**.u:= app(**F**.u, **E'**.u)
[12]**E'1**::= **op-l F E'2**	**E'1**.u:= app(**F**.u, **E'2**.u)
[13]**E**::= ε	**E**.u:= emptylist
[14]**F**::= **T F'**	**F**.u:= app(**T**.u, **F'**.u)
[15]**F'1**::= **op-hight T F'2**	**F'1**.u:= app(**T**.u, **F'2**.u)
[16]**F'**::= ε	**F'**.u:= emptylist
[17]**T**::= **num**	**T**.u:= emptylist
[18]**T**::= **ide**	**T**.u:= cons(**ide**.lexeme, emptylist)
[19]**T**::= **(E)**	**T**.u:= **E**.u

E'::= ε

All the variables are correctly initialized before the use

uin: - set of the variables that have been assigned to, in the sequence that precedes the current statement
- **inherited of Su={Cs,C,A.W,E,E',F,F',T}**
- **....**

uout: - set of the variables assigned to, in the sequence ended by the current statement
- **synthesized of Sd={Cs,C,A.W,E,E',F,F',T}**
- **....**

r: - predicate rhat holds if the used ave been previously assigned to
- **synthesized of all the program structures, but declarations, {P,C,Cs,A,W,E,E',...}**
- **...**

Values and auxiliary functions that are used in the actions

Set are handled by list with the following operations

cons: ide X ide-list --> ide-list
emptylist : --> ide-list
append: ide-list X ide-list --> ide-list
included: ide-list X ide-list --> boolean
isempty: ide-list --> boolean

Try to do it yourself + then compare your solution with the one in the next slide.

[0]**P**::= **D Cs**	**P**.r:= **Cs**.r , **Cs**.uin:=emptylist
[1]**P**::= **Cs**	**P**.r:= **Cs**.r , **Cs**.uin:=emptylist
[2]**D**::= **var ide O** [3]**O1**::= **, ide O2** [4]**O**::=ε	?
[5]**Cs1**::= **; C Cs2**	**Cs1**.r:=(**C**.r&**Cs2**.r), **C**.uin:=**Cs1**.uin **Cs1**.uout:=**Cs2**.uout,**Cs2**.uin:=**C**.uout
[6]**Cs**::= ε	**Cs**.r:= true,**Cs**.uout:=**Cs**.uin
[7]**C**::= **A**	**C**.r:= **A**.r, **A**.uin:=**C**.uin, **C**.uout:=**A**.uout
[8]**C**::=**W**	**C**.r:= **W**.r, **W**.uin:=**C**.uin, **C**.uout:=**W.out**
[9]**A**::= **ide := E**	**A**.r:= **E**.r, **E**.uin:=**A**.uin, **A**.uout:=cons(**ide**.lexeme,**A**.uin)
[10]**W**::= **while E do C endw**	**W**.r:= (**E**.r & **C**.r), **E**.uin:=**W**.uin, **C**.uin:=**W**.uin,**W**.uout:=C.uout
[11]**E**::= **F E'**	**E**.r:= (**F**.r & **E'**.r), **F**.uin:=**E**.uin, **E'**.uin:=**E**.uin,
[12]**E'1**::= **op-l F E'2** [13]**E**::= ε [14]**F**::= **T F'** [15]**F'1**::= **op-h T F'2**	?
[16]**F'**::= ε	**F'**.r:= true
[17]**T**::= **num**	**T**.r:= true
[18]**T**::= **ide**	**T**.r:= isin(**ide**.lexeme,**T**.uin)
[19]**T**::= **(E)**	**T**.r:= **E**.r, **E**.uin:= **T**.uin

C.uout = emptylist

W.uout = W.uin

To Be Completed

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)