Semantica Statica

Analisi
composizionale e contestuale

Analisi

- Principali applicazioni
- Tipi: espressioni.
- Sistema di tipi, analisi e inferenza.

Esercita vari tipi di controllo (contestuale) sulle strutture riconosciute sintatticamente legali

1) **unicita'**

dichiarazione di identificatore in un blocco (ambiente) deve essere unica

2) **occorrenze correlate**

in PASCAL il corpo di una funzione, f, deve contenere almeno un assegnamento all'identificatore f
(una locazione trasparente utilizzata per il value-return)

identificatori dichiarati devono essere usati
(warning)

identificatori usati devono essere dichiarati
(severe)

3) **struttura di controllo** (flusso)

in C il costrutto break deve occorrere allo interno di un blocco

in PASCAL un iteratore non deve essere controllato da un'espressione il cui valore non dipende dall'ambiente

una procedura con parametri by-value deve contenere almeno un effetto laterale

4) **type-checking**

ogni operatore deve essere applicato a strutture compatibili

TIPI

sono oggetti che:

- *sono assegnati* alle strutture del programma

- *servono a classificare* tali strutture per studiare la correttezza (semantica) della loro composizione

- *sono descritti* da opportune espressioni (sistema di tipi)

Applicazioni su una grammatica

- Alcune analisi di *occorrenze correlate*
- Come si imposta un'analisi: scelta degli attributi
- La grammatica ad attributi
- Un secondo esempio

Una grammatica LL(1) per
un linguaggio di programmazione ad espressioni intere, con dichiarazioni, sequenzializzazione, assegnamento, e iteratore nondeterminato.

[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**::= **;** Command **Cs** | [6] ε

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

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

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

ed espressioni con operatori a due livelli di priorita’

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

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

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

[15]**F'**::= **op-hi**ght **T F'** | [16] ε

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

Analisi a occorrenze correlate possibili:

1) tutti gli identificatori usati siano dichiarati

2) tutti gli identificatori dichiarati siano usati

3) tutte le variabili siano definite

4) le espressioni di controllo dell'iteratore
siano booleane [**attenzione coinvolge tipi**]

1) tutti gli identificatori usati siano dichiarati

Attributi utilizzati

3 attributi: u=insieme usati
d=insieme dichiarati
r=u≤d

Valori e funzioni ausiliarie usate

implementazione insiemi

liste: *cons*: ide X ide-list --> ide-list
emptylist: --> ide-list
append: ide-list X ide-list --> ide-list
include: 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]$\mathbf{O_1}$::= **, ide** $\mathbf{O_2}$	$\mathbf{O_1}$.d::= cons(**ide**.lexeme, $\mathbf{O_2}$.d)
[4]**O**::=ε	**O**.d::= emptylist
[5]$\mathbf{Cs_1}$::= **; C** $\mathbf{Cs_2}$	$\mathbf{Cs_1}$.u:= app(**C**.u, $\mathbf{Cs_2}$.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]$\mathbf{E'_1}$::= **op-l F** $\mathbf{E'_2}$	$\mathbf{E'_1}$.u:= app(**F**.u, $\mathbf{E'_2}$.u)
[13]**E**::= ε	**E**.u:= emptylist
[14]**F**::= **T F'**	**F**.u:= app(**T**.u, **F'**.u)
[15]$\mathbf{F'_1}$::= **op-hight T** $\mathbf{F'_2}$	$\mathbf{F'_1}$.u:= app(**T**.u, $\mathbf{F'_2}$.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

3) tutte le variabili siano definite

3 attributi: uin=insieme assegnate prima
uout=insieme assegnate
r=per ogni ide denotabile: ide$\in$ uin

uin=ereditato solo per comandi ed espressioni
uout=sintetizzato solo per comandi
r=sintetizzato solo per programma

implementazione insiemi

liste: *cons*: ide X ide-list --> ide-list
append: ide-list X ide-list --> ide-list
include: ide-list X ide-list --> boolean
emptylist: --> ide-list
isin: ide X ide-list --> boolean
isempty: ide-list --> boolean

[0]**P**::= **D Cs**	**P**.r:= **Cs**.r , **C**s.uin:=emptylist
[1]**P**::= **Cs**	**P**.r:= **Cs**.r , **C**s.uin:=emptylist
[2]**D**::= **var ide O** [3]**O**$_1$::= **, ide O**$_2$ [4]**O**::=ε	?
[5]**Cs**$_1$::= **; C Cs**$_2$	**Cs**$_1$.r:=(**C**.r&**Cs**$_2$.r), **C**.uin:=**Cs**$_1$.uin **Cs**$_1$.uout:=**Cs**$_2$.uout,**Cs**$_2$.uin:=**C**.uout
[6]**Cs**::= ε	**Cs**.r:= true,**C**s.uout:=**C**s.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