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) **unicità**

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

In Java unicità all'interno di ogni definizione

2) **occorrenze correlate**

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

in C un costrutto *return exp* deve occorrere nel corpo di una non-void procedure

in Java un costrutto continue può occorrere solo all'interno di un blocco di un iteratore

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 (stato)

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

4) **type-checking**

ogni operatore deve essere applicato a strutture compatibili

ogni espressione e comando deve avere un unico tipo che ne classifica i possibili usi

TIPI

sono oggetti che:

- *sono assegnati* alle strutture del programma

- *servono a classificare* tali strutture per studiare la correttezza (semantica) del loro uso

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

UN SISTEMA DI TIPI

definisce:

- le *espressione di tipo*
  (forma degli oggetti)

- *regole per* assegnare tipi alle strutture

Espressioni di tipo (una semplice struttura)

1) *tipi basici*

real, int, char, file, unit

2) *costruttori* (tipi strutturati o derivati)

array: I x T -> array(I,T)

prodotto: T1 x T2 -> T1 ✖ T2

record:

({i1}xT1)x..x({ik}xTk) -> record(i1:T1...ik:Tk)

enumerato: {v1,..,vk} -> (v1,..,vk)

pointer: T -> pointer(T)

funzione: TD x TC -> TD ➔ TC

procedura: TD -> TD ➔ unit

3) *identificatori*

4) *variabili*

Regole per assegnare espressioni di tipo

dipendono dal linguaggio:

- struttura sintattica
- semantica

analisi di tipi

forward reasoning

So:

Concludo:

$$\frac{X{:}\text{int} \;\&\; 5{:}\text{int} \;\&\; +{:}\text{int} \times \text{int} \rightarrow \text{int}}{X + 5{:}\text{int}}$$

Inferenza di tipi

backward reasoning

So: $X + 5 : int$

Concludo: $+ : int \times int \rightarrow int$

Concludo: $X : int \;\&\; 5 : int$

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**::= **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**

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

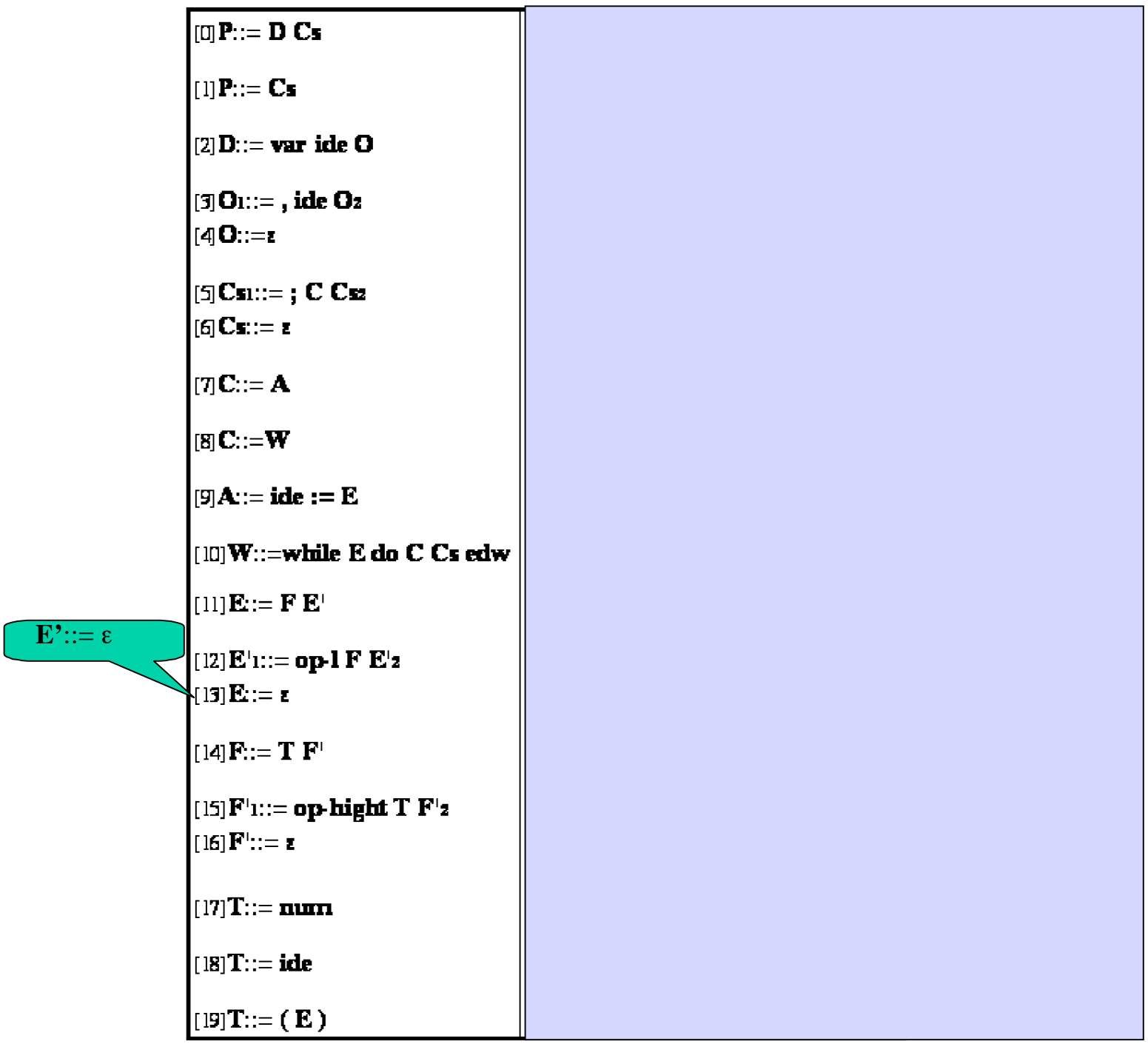
Attributi utilizzati

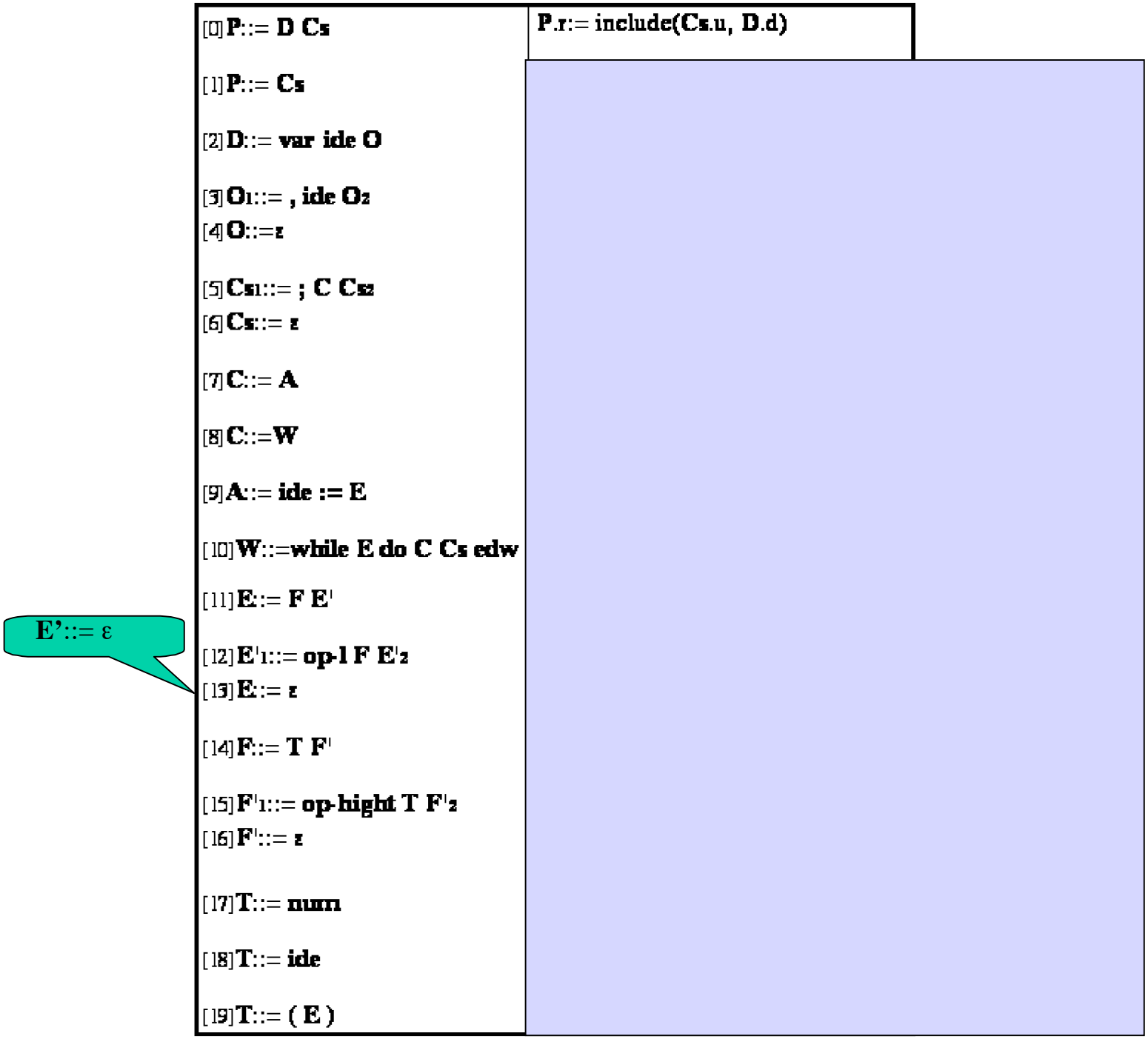
3 attributi: u=insieme usati
d=insieme dichiarati
r=u≤d

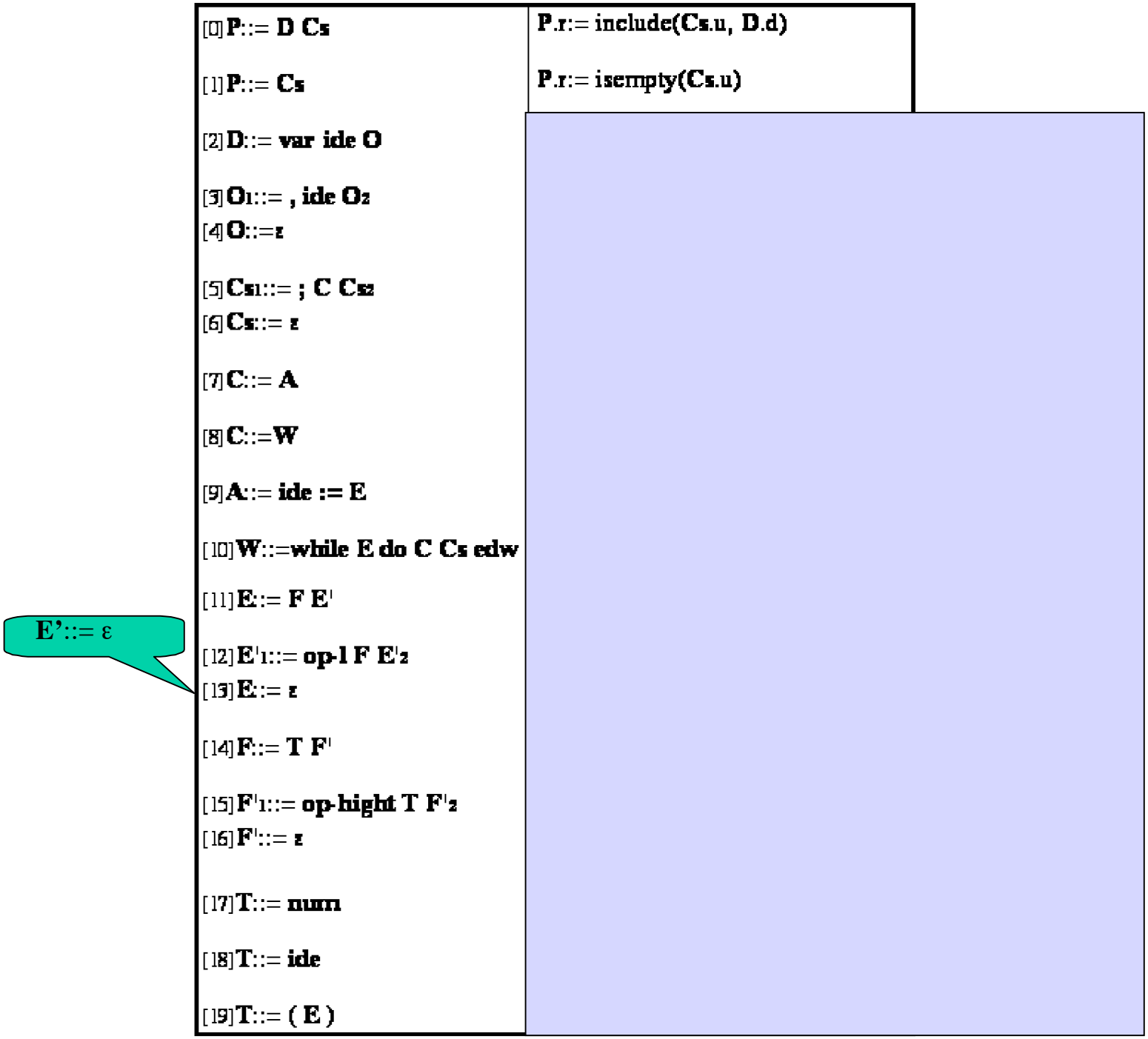
Valori e funzioni ausiliarie usate

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


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


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

[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] O_1 ::= **, ide** O_2	
[4] **O** ::= ε	
[5] Cs_1 ::= **; C** Cs_2	
[6] **Cs** ::= ε	
[7] **C** ::= **A**	
[8] **C** ::= **W**	
[9] **A** ::= **ide := E**	
[10] **W** ::= **while E do C Cs edw**	
[11] **E** ::= **F E'**	
[12] E'_1 ::= **op-l F** E'_2	
[13] **E'** ::= ε	
[14] **F** ::= **T F'**	
[15] F'_1 ::= **op-hight T** F'_2	
[16] **F'** ::= ε	
[17] **T** ::= **num**	
[18] **T** ::= **ide**	
[19] **T** ::= **(E)**	

E' ::= ε

[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** $\mathbf{Cs}_2$	
[6] **Cs** ::= ε	
[7] **C** ::= **A**	
[8] **C** ::= **W**	
[9] **A** ::= **ide := E**	
[10] **W** ::= **while E do C Cs edw**	
[11] **E** ::= **F E'**	
[12] $\mathbf{E'}_1$ ::= **op-l F** $\mathbf{E'}_2$	
[13] **E'** ::= ε	
[14] **F** ::= **T F'**	
[15] $\mathbf{F'}_1$ ::= **op-hight T** $\mathbf{F'}_2$	
[16] **F'** ::= ε	
[17] **T** ::= **num**	
[18] **T** ::= **ide**	
[19] **T** ::= **(E)**	

E' ::= ε

[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**	
[10] **W** ::= **while E do C Cs edw**	
[11] **E** ::= **F E'**	
[12] $\mathbf{E'}_1$ ::= **op-l F** $\mathbf{E'}_2$	
[13] **E'** ::= ε	
[14] **F** ::= **T F'**	
[15] $\mathbf{F'}_1$ ::= **op-hight T** $\mathbf{F'}_2$	
[16] **F'** ::= ε	
[17] **T** ::= **num**	
[18] **T** ::= **ide**	
[19] **T** ::= **(E)**	

E' ::= ε

[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**	
[11] **E** ::= **F E'**	
[12] $\mathbf{E'}_1$ ::= **op-l F** $\mathbf{E'}_2$	
[13] **E'** ::= ε	
[14] **F** ::= **T F'**	
[15] $\mathbf{F'}_1$ ::= **op-hight T** $\mathbf{F'}_2$	
[16] **F'** ::= ε	
[17] **T** ::= **num**	
[18] **T** ::= **ide**	
[19] **T** ::= **(E)**	

E' ::= ε

[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'**	
[12] $\mathbf{E'}_1$ ::= **op-l F** $\mathbf{E'}_2$	
[13] **E'** ::= ε	
[14] **F** ::= **T F'**	
[15] $\mathbf{F'}_1$ ::= **op-hight T** $\mathbf{F'}_2$	
[16] **F'** ::= ε	
[17] **T** ::= **num**	
[18] **T** ::= **ide**	
[19] **T** ::= **(E)**	

E' ::= ε

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

E' ::= ε

[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 v. assegnate nella s. che precede
uout= insieme v. assegnate nel progr. attraver.
r= per ogni ide memorizzabile: ide $\in$ uin

uin=ereditato solo per comandi ed espressioni
uout=sintetizzato solo per comandi (incl. Sequenze)
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**
[1]**P**::= **Cs**
[2]**D**::= **var ide O** [3]**O1**::= **, ide O2** [4]**O**::=ε
[5]**Cs1**::= **; C Cs2**
[6]**Cs**::= ε
[7]**C**::= **A**
[8]**C**::=**W**
[9]**A**::= **ide := E**
[10]**W**::= **while E do C endw**
[11]**E**::= **F E'**
[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
[17]**T**::= **num**
[18]**T**::= **ide**
[19]**T**::= **(E)**

[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**$_2$	
[6]**Cs**::= ε	
[7]**C**::= **A**	
[8]**C**::=**W**	
[9]**A**::= **ide := E**	
[10]**W**::= **while E do C endw**	
[11]**E**::= **F E'**	
[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	
[17]**T**::= **num**	
[18]**T**::= **ide**	
[19]**T**::= **(E)**	

[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**$_2$	
[6]**Cs**::= ε	
[7]**C**::= **A**	
[8]**C**::=**W**	
[9]**A**::= **ide := E**	
[10]**W**::= **while E do C endw**	
[11]**E**::= **F E'**	
[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	
[17]**T**::= **num**	
[18]**T**::= **ide**	
[19]**T**::= **(E)**	

[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	
[7]**C**::= **A**	
[8]**C**::=**W**	
[9]**A**::= **ide := E**	
[10]**W**::= **while E do C endw**	
[11]**E**::= **F E'**	
[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	
[17]**T**::= **num**	
[18]**T**::= **ide**	
[19]**T**::= **(E)**	

[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**	
[8]**C**::=**W**	
[9]**A**::= **ide := E**	
[10]**W**::= **while E do C endw**	
[11]**E**::= **F E'**	
[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	
[17]**T**::= **num**	
[18]**T**::= **ide**	
[19]**T**::= **(E)**	

[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**	
[10]**W**::= **while E do C endw**	
[11]**E**::= **F E'**	
[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	
[17]**T**::= **num**	
[18]**T**::= **ide**	
[19]**T**::= **(E)**	

[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**	
[11]**E**::= **F E'**	
[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	
[17]**T**::= **num**	
[18]**T**::= **ide**	
[19]**T**::= **(E)**	

[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'**	
[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	
[17]**T**::= **num**	
[18]**T**::= **ide**	
[19]**T**::= **(E)**	

[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

W.uout = W.uin

Type Checking (1)

- Estendiamo il linguaggio con tipi basici: una grammatica
- Pianifichiamo un'analisi dei tipi: symbol-table
- Definizione degli attributi:ereditiamo la lista degli identificatori
- Usiamo un'altra grammatica:
  - Ereditiamo il tipo: ma non è L-attributata
  - Usiamo solo sintetizzati

Una grammatica LL(1) per il Semplice linguaggio di programmazione al quale aggiungiamo i tipi *basici*

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

[2]**Ds**::= **Var Dt**yped**s**

[3]**Dts**::= **Dt Dts'**

[4]**Dts'** ::= **; Dt Dts'** | [5] ε

[6]**Dt**::= **ide O**theridentifiers

[7]**O**::= , **ide O** | [8] **:** t**Y**pe

[9]**Cs**::= **;** C**ommand Cs** | [10] ε

[11]**C**::= **A**ssign | [12] **W**hile

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

[14]**W**::= **while E do C Cs endw**hile

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

[16]**E'**::= **op-l**ower **F E'** | [17] ε

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

[19]**F'**::= **op-h**ight **T F'** | [20] ε

[21]**T**::= **num** | [22] **ide** | [23] **(E)**

[24]**Y**::= **boolean** | [25] **integer** | [26] **file** | ...

Associamo nella Sym-Table tipi alle variabili

attributi: entry=locazione in sym-table
t=espressione di tipo di **Y**
ty=lista variabilii

entry=sintetizzato (scanner) solo per **ide**
t=sintetizzato solo per **Y**
ty=ereditato solo per dichiarazioni

operiamo effetti laterali (SDD)

operazioni primitive su Sym-Table

Addtype: ide X type

[0] **P** ::= **Ds Cs**	
[1] **P** ::= **Cs**	
[2] **Ds** ::= **Var Dts**	
[3] **Dts** ::= **Dt Dts'**	
[4] **Dts'** ::= **; Dt Dts'**	
[5] **Dts'** ::= **ε**	
[6] **Dt** ::= **ide O**	**O**.ty := **cons**(**ide**.entry, **Dt**.ty)
[7] O_1 ::= , **ide** O_2	O_2.ty := **cons**(**ide**.entry, **Dt**.ty)

[8] **O** ::= **: Y**	temp := **O**.ty;
	repeat {addtype(car(temp), **Y**.t);
	temp := cdr(temp)}
	until isempty(temp)

[9] Cs_1 ::= **; C** Cs_2	
[10] **Cs** ::= **ε**	
[11] **C** ::= **A**	
[12] **C** ::= **W**	
[13] **A** ::= **ide := E**	
[14] **W** ::= **while E do C Cs edw**	
[15] **E** ::= **F E'**	
[16] **E'** ::= **op-1 F E'**	
[17] **E'** ::= **ε**	
[18] **F** ::= **T F'**	
[19] **F'** ::= **op-h T F'**	
[20] 1] **F'** ::= **ε**	
[21] **T** ::= **num**	
[22] **T** ::= **ide**	
[23] **T** ::= **(E)**	
[24] **Y** ::= **boolean**	**Y**.t := **boolean**
[25] **Y** ::= **integer**	**Y**.t := integer
[26] **Y** ::= **file**	**Y**.t := file

Usiamo un'altra grammatica

attributi: entry=locazione in sym-table
t=espressione di tipo di **Y**
ty=espressione di tipo

entry=sintetizzato (scanner) solo per **ide**
t=sintetizzato solo per **Y**
ty=ereditato solo per dichiarazioni

[0] **P**::= **Ds Cs**	
[1] **P**::= **Cs**	
[2] **Ds**::= **Var Dts**	
[3] **Dts**::=**Dt : Y Dts'**	**Dt.ty**:= **Y**.t
[4] **Dts'** ::= **; Dt : Y Dts'**	**Dt.ty**:=**Y**.t
[5] **Dts'** ::= **ε**	
[6] **Dt**:=**ide O**	**O.ty**:= **Dt.ty**, **addtype(ide.entry,Dt.ty)**
[7] **O1**::= , **ide O2**	**O2.ty**:=**O1.ty**, **addtype(ide.entry, O1.ty)**
[8] **O**::= **ε** [9] **Cs1**::= **; C Cs2** [10] **Cs**::= **ε** [11] **C**::= **A** [12] **C**::=**W** [13] **A**::= **ide := E** [14] **W**::=**while E do C Cs edw** [15] **E**:= **F E'** [16] **E'**::= **op-l F E'** [17] **E'**::= **ε** [18] **F**:= **T F'** [19] **F'**::= **op-h T F'** [201] **F'**::= **ε** [21] **T**::= **num** [22] **T**::= **ide** [23] **T**::= **(E)**	
[24] **Y**::= **boolean**	**Y**.t:= **boolean**
[25] **Y**::= **integer**	**Y**.t:= **integer**
[26] **Y**::= **file**	**Y**.t:= **file**

La precedente grammatica non L-attributata

perchè nella produzione [4]
Dt eredita dal fratello destro **Y**

Possiamo usare attributi in modo differente

attributi: entry=lista delle locazioni in sym-table
t=espressione di tipo di **Y**

entry=sintetizzato (scanner) solo per **Ide** e per **Dt**
t=sintetizzato solo per **Y**

[0] **P**::= **Ds Cs**	
[1] **P**::= **Cs**	
[2] **Ds**::= **Var Dts**	
[3] **Dts**::=**Dt : Y Dts'**	addtype-set(Dt.entry,Y.t)
[4] **Dts'** ::= **; Dt : Y Dts'**	addtype-set(Dt.entry,Y.t)
[5] **Dts'** ::= **ε**	
[6] **Dt**::=**ide O**	Dt.entry:= cons(ide.entry,O.entry)
[7] O_1::= , **ide** O_2	O_1.entry:=cons(ide.entry, O_2.entry)
[8] **O**::= **ε**	O.entry:=emptylist
[9] Cs_1::= **; C** Cs_2 [10] **Cs**::= **ε** [11] **C**::= **A** [12] **C**::=**W** [13] **A**::= **ide := E** [14] **W**::=**while E do C Cs edw** [15] **E**::= **F E'** [16] **E'**::= **op-l F E'** [17] **E'**::= **ε** [18] **F**::= **T F'** [19] **F'**::= **op-h T F'** [20] **F'**::= **ε** [21] **T**::= **num** [22] **T**::= **ide** [23] **T**::= **(E)**	
[24] **Y**::= **boolean**	Y.t:= boolean
[25] **Y**::= **integer**	Y.t:= integer
[26] **Y**::= **file**	Y.t:= file

Nel precedente piano:

usiamo solo sintetizzati

la grammatica ci consente di farlo con poca fatica

Anche la prima grammatica consentiva di usare solo sintetizzati. Un differente piano

attributi: entry=locazione in sym-table
t=espressione di tipo di **Y**

entry=sintetizzato (scanner) solo per **ide**
t=sintetizzato solo per **Y**,**O**

Una differente definizione degli attributi

Dt::=**ide O** addtype(**ide**.entry,**O**.t)

Completare con il calcolo del sintetizzato .t di O

Conclusioni (dall'esempio):

Scegliere la gramm. con attenzione

grammatiche differenti conducono a piani di attributi differenti e con attributi differenti per tipo (ered., sint.) e per numero

Scegliere il piano con attenzione

alcune grammatiche consentono più piani di attributi di differente difficoltà

per una grammatica LL (LR) si possono trovare anche piani non L-attributati

Type Checking (2)

- Associamo tipi alle strutture: regole di inferenza
- Estendiamo gli attributi della grammatica prec.
- Tipi derivati: regole di inferenza
- Coercion e overload: regole di inferenza
- Estendiamo gli attributi della grammatica prec.

Associamo espressioni di tipo alle strutture di un programma del linguaggio Semplice

regole di inferenza:

$$\frac{i:t \quad e:t}{i:=e:N} \qquad \frac{e:bool \quad c_s:N}{\text{while } e \text{ do } c_s:N} \qquad \frac{c:N \quad c_s:N}{c;c_s:N}$$

$$\frac{e_1:t_1 \quad e_2:t_2 \quad op: t_1 \times t_2 \rightarrow t}{op\ e_1\ e_2 : t}$$

attributi: type=tipo delle espressioni
in=tipo sottoespressione sinistra
r=tipo comandi

type=sintetizzato
in=ereditato sottoespressione sinistra
r=sintetizzato (true=N, false=error)

[0] **P::= Ds Cs**	**P.r:= Cs .r**
[1] **P::= Cs**	**P.r:= Cs .r**
[2] **Ds::= Var Dts**	
[3] **Dts::=Dt : Y Dts'**	**addtype-set(Dt.entry,Y.t)**
[4] **Dts' ::= ; Dt : Y Dts'**	**addtype-set(Dt.entry,Y.t)**
[5] **Dts' ::= ε**	
[6] **Dt:=ide O**	**Dt.entry:= cons(ide.entry,O.entry)**
[7] **O_1::= , ide O_2**	**O_1.entry:=cons(ide.entry, O_2.entry)**
[8] **O::= ε**	**O.entry:=emptylist**
[9] **Cs_1::= ; C Cs_2**	**Cs_1.r::= C.r & Cs_2.r**
[10] **Cs::= ε**	**Cs.r::= true**
[11] **C::= A**	**Cs.r::= A.r**
[12] **C::=W**	**Cs.r::= W.r**
[13] **A::= ide := E**	**A.r:= (ide.type = E.type), E.in:=N**
[14] **W::= while E do C Cs edw**	**W.r:= (E.type=boolean)& C.r & Cs.r E.in:=N**

[16] **E** ::= **F E'**	**E.type**:=**E'.type**, **F.in**:=**E.in**, **E'.in**:=**F.type**
[17] **E'**$_1$::= **op-1 F E'**$_2$	**F.in**:=**N** let **op-1.type** = $t_1 \times t_2 \rightarrow t$ in if (t_1=**E'**$_1$**.in**) & (t_2=**F.type**) then begin **E'**$_2$**.in**:=t; **E'**$_1$**.type**:=**E'**$_2$**.type end** else **E'**$_1$**.type**:= **error**
[18] **E'**::= **ε**	**E'.type**:=**E'.in**
	|
	|
[25] **Y**::= **boolean**	**Y**.t:= **bool**
[26] **integer**	**Y**.t:= **integer**
[27] **file**	**Y**.t:= **file**

Osservazioni:

l'attributo .in è ereditato dai soli simboli: **E**, **E'**, **F**, **F'**

solo le produzioni [17], di E', e [19] di F', richiedono tale attributo

allora, **E** ed **F** potrebbero non avere attributo .in e la grammatica essere semplificata

Cosa succede quando estendiamo il sistema dei tipi??

regole di inferenza:

$$\frac{e:\mathrm{array}(i,t) \quad e':i' \quad i \approx i'}{e[e']:t}$$

$$\frac{e:\mathrm{record}(i1:t1..ik:tk) \quad (i=ij,\ 1\le j\le k)}{e.i:tj}$$

Equivalenza $\approx$ tra tipi

strutturale:

$$\frac{t1 \approx t1' \quad tk \approx tk'}{y[t1,\ldots,tk] \approx y[t1',\ldots,tk']}$$

referenziale

$$\frac{t \equiv t'}{t \approx t'}$$

Coercion e overloading

regole di inferenza:

$$\frac{e:t \quad \text{into}: t \rightarrow t'}{\text{into } e: t'}$$

$$\frac{f:\{t_1 \rightarrow t_1', \ldots, t_k \rightarrow t_k'\} \quad (e:t_j \ , \ 1 \leq j \leq k)}{f(e):t_j'}$$

Aggiungiamo a Semplice funzioni overloaded:

1) produzioni attribuite per le dichiarazioni

2) produzioni attribuite per le espressioni

(attenzione alla nuova espressione di tipo {__})

nota: N = neutro rispetto ×, quindi N×t1×...×tn×N=t1×...×tn

[0] **P ::= Ds Dff Cs**	**P.r := Cs .r**
[1] **P ::= Cs**	**P.r := Cs .r**
[2] **Ds ::= Var Dts**	
[3] **Dts ::= Dt : Y Dts'**	**addtype-set(Dt.entry, Y.t)**
[4] **Dts'$_1$ ::= ; Dt : Y Dts$_2$'**	**addtype-set(Dt.entry, Y.t)**
[5] **Dts' ::= ε**	
[6] **Dt ::= ide O**	**Dt.entry := cons(ide.entry, O.entry)**
[7] **O$_1$ ::= , ide O$_2$**	**O$_1$.entry := cons(ide.entry, O$_2$.entry)**
[8] **O ::= ε**	**O.entry := emptylist**
[28] **Dff ::= Fun Df Dff'**	
[29] **Dff'$_1$ ::= ; Df Dff'$_2$**	
[30] **Dff' ::= ε**	
[31] **Df ::= funct ide FPP : Y ; P**	**overload(ide.entry, FPP.t → Y.t)**
[32] **FPP ::= (FP : Y FPP'**	**FPP.t := Y.t × FPP'.t**
[33] **FPP ::= ε**	**FPP.t := N**
[34] **FPP'$_1$ ::= , FP : Y FPP'$_2$**	**FPP'$_1$.t := Y.t × FPP'$_2$.t**
[35] **FPP' ::=)**	**FPP'.t := N**
[36] **T ::= apply ide AP**	**let S := ide.type** **in T.type := find(AP.type, S)**