

Specifica di un Linguaggio di Programmazione

- Successo di un LP → descrizione $\begin{cases} \text{concisa} \\ \text{comprensibile} \end{cases}$
- Pb **livello** di formalizzazione (notazione rigorosa per evitare ambiguità):
 - Formale → incomprensibile → pb di accettazione del LP (barriera, Es: ALGOL)
 - Informale → comprensibile → proliferazione di dialetti (ambiguità)
- Pb diversificazione dei destinatari (“audience”) → specifica polimorfa:
 - Revisori (potenziali utenti) → successo = f (chiarezza, eleganza, ...)
 - Implementatori del LP → chiaro significato dei costrutti (tutti!)
 - Utenti del LP → necessità del Reference Manual

Specifica di un Linguaggio di Programmazione (ii)

- Definizione di un L (anche naturale) comprende $\left\{ \begin{array}{l} \text{lessico} \\ \text{sintassi} \\ \text{semantica} \end{array} \right\}$ delle sue frasi

■ Data: $\overbrace{\text{DD/DD/DDDD}}^{\text{lessico}} \quad \overbrace{01/02/2009 \left\{ \begin{array}{l} \text{USA} \rightarrow 2 \text{ Gennaio } 2009 \\ \text{Italia} \rightarrow 1 \text{ Febbraio } 2009 \end{array} \right.}}^{\text{semantica}}$

■ Selezione in C $\left\{ \begin{array}{l} \overbrace{\text{if (expr) statement}}^{\text{sintassi}} \\ \underbrace{\text{"Se valore di } expr \neq 0 \text{ allora esegui } statement"}_{\text{semantica}} \end{array} \right.$

- Sintassi: non garantisce una frase semanticamente corretta → però: semplicità ...

Specifica di un Linguaggio di Programmazione (iii)

	<i>Complessità</i>	<i>Standardizzazione</i>	<i>Formalismo</i>
Lessico	piccola	si	Espressioni regolari
Sintassi	media	si	G non contestuali
Semantica	grossa	no	Sem. Operazionale Assiomatica Denotazionale

- Sebbene separate nella descrizione, intimamente collegate:
 - Sintassi: deve “suggerire” la semantica (`if`, `while`, + ...)
 - Semantica: guidata dalla sintassi

Lessico

- Terminologia {
 - stringa lessicale** = sequenza significativa di caratteri (parola)
 - simbolo** = astrazione di una classe di stringhe lessicali: *estensione* = { stringhe }
 - pattern** = regola per descrivere l'estensione dei simboli

Simbolo	Istanze	Pattern	} enumerazine
while	while	while	
begin	begin	begin	
relop	< <= > >= != = ==	{ <, <=, >, >=, !=, =, == }	
id	partenza tempo m24 X2	lettera seguita da lettere e/o cifre	
num	3 25 3.5 4.37E12	parte intera seguita opzionalmente da parte decimale e/o parte esponenziale	
strconst	"Hello world!"	sequenza di caratteri racchiusa tra "	

- Attributi lessicali**: quando $Card(estensione) > 1$ (per evitare perdita di informazione)
 - while** $\rightarrow Ext(\mathbf{while}) = \{ \text{while} \}$
 - relop** $\rightarrow Ext(\mathbf{relop}) = \{ <, <=, >, >=, !=, =, == \}$
 - id** $\rightarrow Ext(\mathbf{id}) = \{ \dots \}$

$\alpha = (\beta * 3) \quad \Rightarrow \quad \langle \mathbf{id}, \uparrow \alpha \rangle \langle \mathbf{assign}, \rangle \langle \mathbf{left}, \rangle \langle \mathbf{id}, \uparrow \beta \rangle \langle \mathbf{times}, \rangle \langle \mathbf{num}, 3 \rangle \langle \mathbf{right}, \rangle$

Specifica dei Simboli

- **Alfabeto** \equiv **insieme** finito di caratteri

$\{0, 1\}$ = alfabeto binario

$\{a, b, c, d\}$

ASCII = alfabeto per computer

Unicode

- **Stringa** su un alfabeto \equiv **sequenza** finita di caratteri dell'alfabeto

- $|x|$ \equiv lunghezza della stringa x

- ε \equiv *stringa nulla* $\rightarrow |\varepsilon| = 0$

- Operatore di **concatenazione**: $\begin{cases} xy & x = \text{alfa}, y = \text{beta} \rightarrow xy = \text{alfabeta} \\ \varepsilon & \text{elemento identità nella concatenazione: } x\varepsilon = \varepsilon x = x \end{cases}$



pensato come prodotto



possibilità di definire l'operatore **potenza** su stringhe

$$\begin{cases} x^0 = \varepsilon \\ x^1 = x \\ x^2 = xx \\ \dots \end{cases}$$

Specifica dei Simboli (ii)

- **Linguaggio** \equiv **insieme** di stringhe su un certo alfabeto

\emptyset (valenza formale)

$\{\varepsilon\}$

$\{\text{frasi sintatticamente corrette del Pascal}\}$

$\{\text{bytes}\}$

Operazione	Forma	Definizione
Unione	$L \cup M$	$\{x \mid x \in L \vee x \in M\}$
Concatenazione	$L M$	$\{xy \mid x \in L \wedge y \in M\}$
Potenza	L^n	$\begin{cases} \{\varepsilon\} & \text{se } n = 0 \\ L^{n-1} L & \text{se } n \geq 1 \end{cases}$ $L L L \dots L \text{ (n volte)}$
Chiusura di Kleene	L^*	$\bigcup_{i=0}^{\infty} L^i$ $L^0 \cup L^1 \cup L^2 \cup \dots$
Chiusura Positiva	L^+	$\bigcup_{i=1}^{\infty} L^i$ $L^1 \cup L^2 \cup L^3 \cup \dots$

Specifica dei Simboli (iii)

- Esempio di linguaggio: $L = \{ A, B, \dots, Z, a, b, \dots, z \}$
 $M = \{ 0, 1, \dots, 9 \}$ \Rightarrow isomorfismo tra $\begin{cases} \text{linguaggio} \\ \text{alfabeto} \end{cases}$

1. $L \cup M = \{ \text{lettere e cifre} \}$

2. $LM = \{ \text{stringhe composte da una lettera seguita da una cifra} \}$

3. $L^3 = \{ \text{stringhe di 3 lettere} \}$

4. $L^* = \{ \text{stringhe di lettere, inclusa } \varepsilon \}$

id 5. $L(L \cup M)^* = \{ \text{stringhe alfanumeriche che iniziano con una lettera} \}$

num 6. $M^+ = \{ \text{stringhe di almeno una cifra} \} = \{ \text{costanti intere senza segno} \}$

Espressioni Regolari

- Potente formalismo per specificare $\left\{ \begin{array}{l} \text{un pattern (id, num, ...)} \\ \text{l'estensione di un simbolo} \end{array} \right.$
identificatore = lettera (lettera | cifra)*
- Similitudine $\left\{ \begin{array}{l} \text{espressioni regolari} \\ \text{espressioni aritmetiche} \end{array} \right. \Rightarrow \text{mediante operatori applicati a sottoespressioni ...}$
fino alle espressioni atomiche
 $(34+25) * (12+3)$

```
graph TD; A[ ] --- B[caratteri alfabeto]; A --- C[numeri];
```
- Differenza $\left\{ \begin{array}{l} \text{espressioni aritmetiche} \rightarrow \text{risultato} = \text{numero} \\ \text{espressioni regolari} \rightarrow \text{"risultato"} = \{ \text{stringhe} \} = \text{linguaggio} = \text{estensione del simbolo} \end{array} \right.$

Definizione di Espressione Regolare

- Def di espressione regolare (su un alfabeto Σ) \rightarrow mediante regole induttive:
 - ε è una espressione regolare che denota il linguaggio $\{ \varepsilon \}$
 - Se $a \in \Sigma$, allora a è una espressione regolare che denota il linguaggio $\{ a \}$
 - Se x, y sono expr regolari che denotano rispettivamente $L(x), L(y)$, allora:
 - (x) è una expr regolare che denota $L(x)$ \implies possibilità di includere parentesi (non intrusive)
 - $(x) \mid (y)$ è una expr regolare che denota $L(x) \cup L(y)$
 - $(x)(y)$ è una expr regolare che denota $L(x)L(y)$
 - $(x)^*$ è una expr regolare che denota $(L(x))^*$

- Regola 2 \rightarrow diversi significati di a
 - carattere dell'alfabeto Σ
 - espressione regolare
 - stringa

- Eliminazione delle parentesi mediante regole di
 - precedenza
 - associatività

Associazione	Operatore
sinistra	\mid
sinistra	Concatenazione
sinistra	$*$

- Insieme regolare** \equiv linguaggio determinato da una espressione regolare

Esempi di Espressioni Regolari

1. $\Sigma = \{ a, b, c \}$. Determinare l'expreg corrispondente alle stringhe che contengono esattamente un b:

$(a \mid c)^* b (a \mid c)^*$

2. $\Sigma = \{ a, b, c \}$. Determinare l'expreg corrispondente alle stringhe che contengono al più un b:

$(a \mid c)^* (b \mid \epsilon) (a \mid c)^*$

Oss: expreg equivalente: $(a \mid c)^* \mid (a \mid c)^* b (a \mid c)^* \Rightarrow \exists$ diverse expreg per lo stesso L!

Formalmente: $L(x) = L(y) \Rightarrow x \approx y$

3. Limite della potenzialità espressiva (**insiemi non regolari**):

$\Sigma = \{ a, b \}$. $L = \{ a^n b a^n \mid n > 0 \} = \{ aba, aabaa, aaabaaa, \dots \}$ = astrazione di costrutti bilanciati



non specificabile con una espressione regolare \rightarrow non può contare!

$a^* b a^*$



non sufficientemente vincolante:
complete ma non sound

Espressioni Regolari (v)

- Proprietà algebriche delle espressioni regolari (analogamente alle espressioni aritmetiche)

<i>Proprietà</i>	<i>Descrizione</i>
$x \mid y = y \mid x$	\mid è commutativo
$x \mid (y \mid z) = (x \mid y) \mid z$	\mid è associativo
$(x y) z = x (y z)$	Concatenazione è associativa
$x (y \mid z) = xy \mid xz$ $(x \mid y) z = xz \mid yz$	Proprietà distributiva della concatenazione rispetto a \mid
$\varepsilon x = x \varepsilon = x$	ε = elemento identità per concatenazione
$x^* = (x \mid \varepsilon)^*$	Relazione tra ripetizione ed ε
$x^{**} = x^*$	Ripetizione è idempotente

Espressioni Regolari Estese

- Non cambia il potere espressivo (però: concisione, scrivibilità)

1. Una o più ripetizioni: $x^+ \equiv xx^*$ $\rightarrow (L(x))^+$

Esempio: Numeri naturali in forma binaria $(0|1)^+ \approx (0|1)(0|1)^*$

2. Qualsiasi carattere in Σ : $.$

Esempio: Tutte le stringhe che contengono almeno un b $.^*b.^*$

3. Un range di caratteri: $[\dots]$ $[abc] \equiv a | b | c$ $[a-z] \equiv a | b | \dots | z$

$[0-9] \equiv 0 | 1 | \dots | 9$ $[a-zA-Z] \equiv \{\text{lettere minuscole e maiuscole}\}$ $[A-Za-z] \neq [A-z]$ (anche in ASCII)

4. Qualsiasi carattere al di fuori di un certo **insieme**: \sim

$\sim a \equiv \Sigma - \{a\}$ $\sim(a | b | c) \equiv \Sigma - \{a, b, c\}$

5. Sottoespressioni opzionali: $?$ $(+ | -) ? [0-9]^+ \equiv \text{naturale con possibile segno}$

$(x)? \equiv (x | \epsilon)$

Definizioni Regolari

- Associazioni di nomi a espressioni regolari (semantica MACRO) → modularizzazione

nome₁ → x₁

nome₂ → x₂

...

nome_n → x_n

- Nomi distinti
- x_i = expreg sui simboli in $\Sigma \cup \{ \text{nome}_1, \text{nome}_2, \dots, \text{nome}_{i-1} \}$
- Non ricorsive! (altrimenti → aumento potere espressivo)

Esempi:

1. Identificatori Pascal:

lettera → [A-Za-z]

cifra → [0-9]

id → **lettera** (**lettera** | **cifra**)*

≈

[A-Za-z] ([A-Za-z] | [0-9])*

2. Numeri con esponente

7.25E-2 = 7.25 * 10⁻² = 0.0725

: 3 parti

{
intera
decimale
esponenziale

nat → [0-9]⁺

snat → (+|-) ? **nat**

num → **snat** ("." **nat**) ? (**E snat**) ?

Espressioni Regolari per Token nei LP

- Classificazione elementi lessicali:

- **Keywords** = { if, then, while, do, begin, end, procedure, function, case, repeat ... }
- **Simboli speciali** = { +, -, *, /, =, >, >=, !=, ++, --, &&, ||, ... }
- **Identificatori** = { stringhe alfanumeriche che iniziano con una lettera }
- **Costanti** = { 25, .34, 2.7E-3, "alfa", 'a', ... }
- **Pseudosimboli** (spaziatura, commenti)

- Keywords:

```
if → if
then → then
...
while → while
keyword → if | then | ... | while
```

- Simboli speciali:

```
plus → "+"
minus → "-"
...
equal → ==
assign → =
plusplus → "++"
minusminus → "--"
```

- Costanti:

```
charconst → '.'
strconst → \"(\\\"\\)*\" ←----- caso semplice, senza metacaratteri
...
```

Espressioni Regolari per Token nei LP (ii)

- **Commenti:** \exists diversi stili

{	Pascal:	<code>{ commento }</code>	<code>{ (~)* }</code>
	C:	<code>/* commento */</code>	
	Ada:	<code>-- commento</code>	<code>--(~\n)*</code>

Stile C: $ba \dots ab$ (astrazione) $ba(b^*(a^*(a|b)b^*)^*a^*)ab$

$(\sim(ab))^*$ ← no!

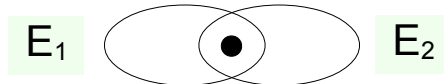
- **Ambiguità:** quando la stessa stringa di caratteri è compatibile con più espressioni regolari

Esempi:

- `while` { `id`
`while`
- `<>` { `noteq`
`less greater`

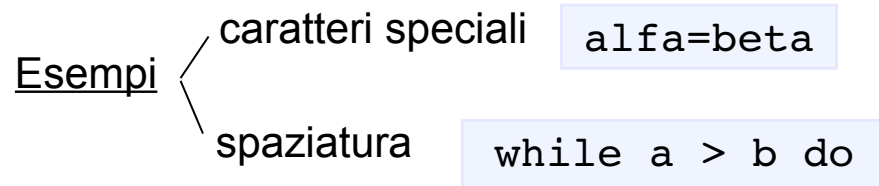
LP → regole di disambiguamento $\left\{ \begin{array}{l} \text{keywords = reserved words} \\ \text{principio della sottotringa più lunga (maximal munch)} \end{array} \right.$

in generale: insufficienti



Espressioni Regolari per Token nei LP (iii)

- Delimitatori: caratteri sui quali una stringa cessa di rappresentare un simbolo (“confini” dei token)



Def di pseudosimbolo: **whitespace** \rightarrow (**blank** | **tab** | **newline** | **comment**)⁺

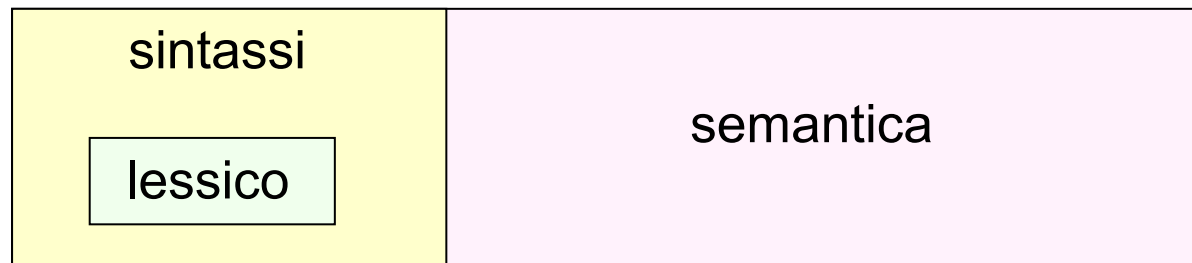
▲ quando riconosciuto, viene ignorato \rightarrow **Linguaggi liberi dal formato**

Sintassi

- $L = \{ \text{stringhe di caratteri su un certo alfabeto} \} \equiv \{ \text{frasi} \}$
- **Regole sintattiche**: specificano la struttura delle frasi di L

Oss: L $\left\{ \begin{array}{l} \text{naturale} \rightarrow \text{regole} \left\{ \begin{array}{l} \text{numerose} \\ \text{complesse} \end{array} \right. \\ \text{artificiale} \rightarrow \text{poche (semplici) regole} \end{array} \right.$

- Convenzione: sintassi \neq descrizione del **lessico** (unità sintattiche di più basso livello)

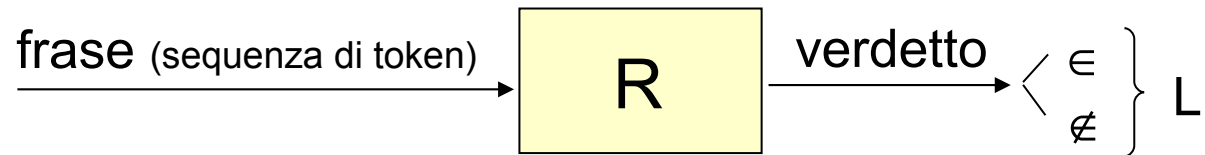


Sintassi (ii)

- Def formale di un L per $\left\langle \begin{array}{l} \text{riconoscimento} \\ \text{generazione} \end{array} \right\rangle$ non pratica!

1. Riconoscitori del linguaggio

Hp: L su alfabeto $\Sigma \rightarrow R \equiv$ strumento di riconoscimento:

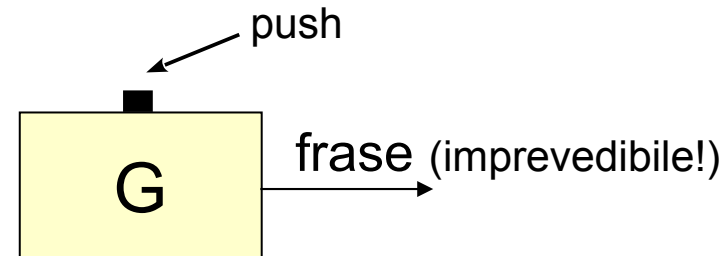


Pb: L infinito $\rightarrow R \left\langle \begin{array}{l} \text{non adatto per enumerare le frasi di } L \\ \text{usato per verificare la correttezza sintattica di un programma} \end{array} \right\rangle$

Sintassi (iii)

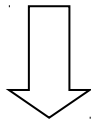
2. Generatori del linguaggio

- $G \equiv$ strumento per generare le frasi di L :



- Doppio pb $\left\langle \begin{array}{l} R: \text{strumento poco utile per definire } L \text{ poich\`e usato "per tentativi"} \\ G: \text{aleatoriet\`a della frase generata} \end{array} \right.$

- Stretta connessione tra $\left\langle \begin{array}{l} \text{generazione} \\ \text{riconoscimento} \end{array} \right.$ = scoperta della computer science

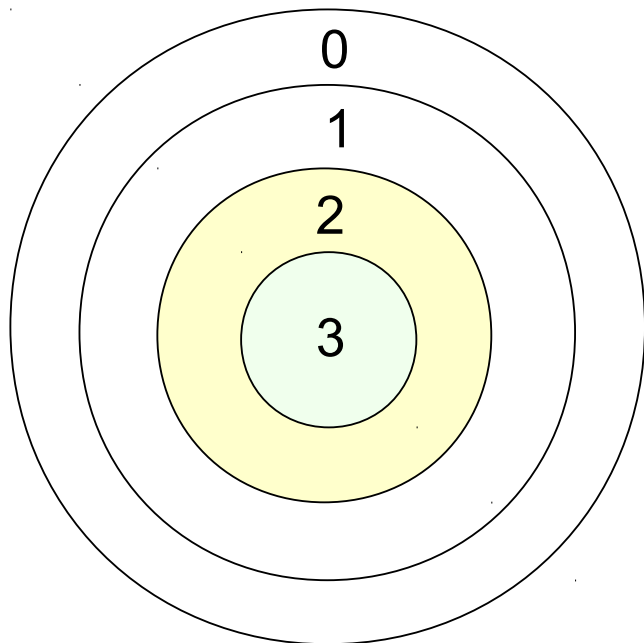


Meccanismo di R basato su quello di G

Metodi formali per la definizione della sintassi

- 1950s: $\left\langle \begin{array}{l} \text{John Backus} \\ \text{Noam Chomsky} \end{array} \right\rangle$ inventano la stessa notazione in contesti diversi

Chomsky (linguista): specifica 4 classi di strumenti generativi di grammatiche
→ definiscono 4 classi di linguaggi



Classe $\left\langle \begin{array}{l} 2 \equiv \text{G non-contestuali} \rightarrow \text{sintassi} \\ 3 \equiv \text{G regolari} \rightarrow \text{lessico} \end{array} \right.$

Metodi formali per la definizione della sintassi (ii)

Backus (1959): presenta ALGOL 58 → sintassi specificata in BNF

- BNF (Backus-Naur Form) : quasi identica allo strumento 2 di Chomsky
- Paradosso: BNF $\begin{cases} \text{non accettata prontamente dagli utenti dei LP} \\ \text{poi, standard per la descrizione della sintassi dei LP} \end{cases}$
- BNF = linguaggio per descrivere un LP → **metalinguaggio**

BNF

- Idea di fondo della notazione BNF: astrazioni per definire le strutture sintattiche

Assegnamento in C:

assign \rightarrow *var* = *expr*



LHS



RHS = { token, stringhe lessicali, astrazioni }

Parafrasi della regola (produzione):

“Un’istanza dell’astrazione *assign* è definita come un’istanza dell’astrazione *var*, seguita dalla stringa lessicale ‘=’, seguita da un’istanza dell’astrazione *expr*”.

netto = lordo - tara

- Necessario definire tutte le astrazioni (tipicamente: ricorsivamente)

BNF (ii)

- Nomenclatura: nella produzione $\left\{ \begin{array}{l} \text{astrazione} \equiv (\text{simbolo}) \text{ nonterminale} \\ \text{token / stringa lessicale} \equiv (\text{simbolo}) \text{ terminale} \end{array} \right.$
- Def: **Grammatica** $\equiv \{ \text{produzioni} \}$
- Formalmente: $G = (T, N, P, A)$ $\left\{ \begin{array}{l} \text{terminali} \\ \text{nonterminali} \\ \text{produzioni} \\ \text{assioma (astrazione di più alto livello)} \end{array} \right.$
- Astrazione: può essere definita in \neq modi \rightarrow **alternative** in un'unica regola (leggibilità)

BNF (iii)

1. Selezione in Pascal:

$if-stat \rightarrow if\ bool-expr\ then\ stat$
 $if-stat \rightarrow if\ bool-expr\ then\ stat\ else\ stat$



$if-stat \rightarrow if\ bool-expr\ then\ stat \mid$
 $if\ bool-expr\ then\ stat\ else\ stat$

2. Specifica sintetica di liste (finite ma illimitate)

- In matematica: 1, 2, ...
- In BNF \nexists '...' \Rightarrow ricorsione: $id-list \rightarrow id \mid id\ ,\ id-list$ (piede ricorsione, passo ricorsivo)
- BNF = strumento $\begin{cases} \text{semplice} \\ \text{potente} \end{cases}$

BNF (iv)

- BNF = strumento **generativo** per definire LP
- **Derivazione** \equiv “Frase generata mediante una serie di applicazioni delle regole (**riscrittura**), partendo dall’assioma”

$assign \rightarrow id := expr$

$id \rightarrow A \mid B \mid C$

$expr \rightarrow id + expr \mid$
 $id * expr \mid$
 $(expr) \mid$
 id



$A := B * (A + C)$



$assign \Rightarrow id := expr$

$\Rightarrow A := expr$

$\Rightarrow A := id * expr$

$\Rightarrow A := B * expr$

$\Rightarrow A := B * (expr)$

$\Rightarrow A := B * (id + expr)$

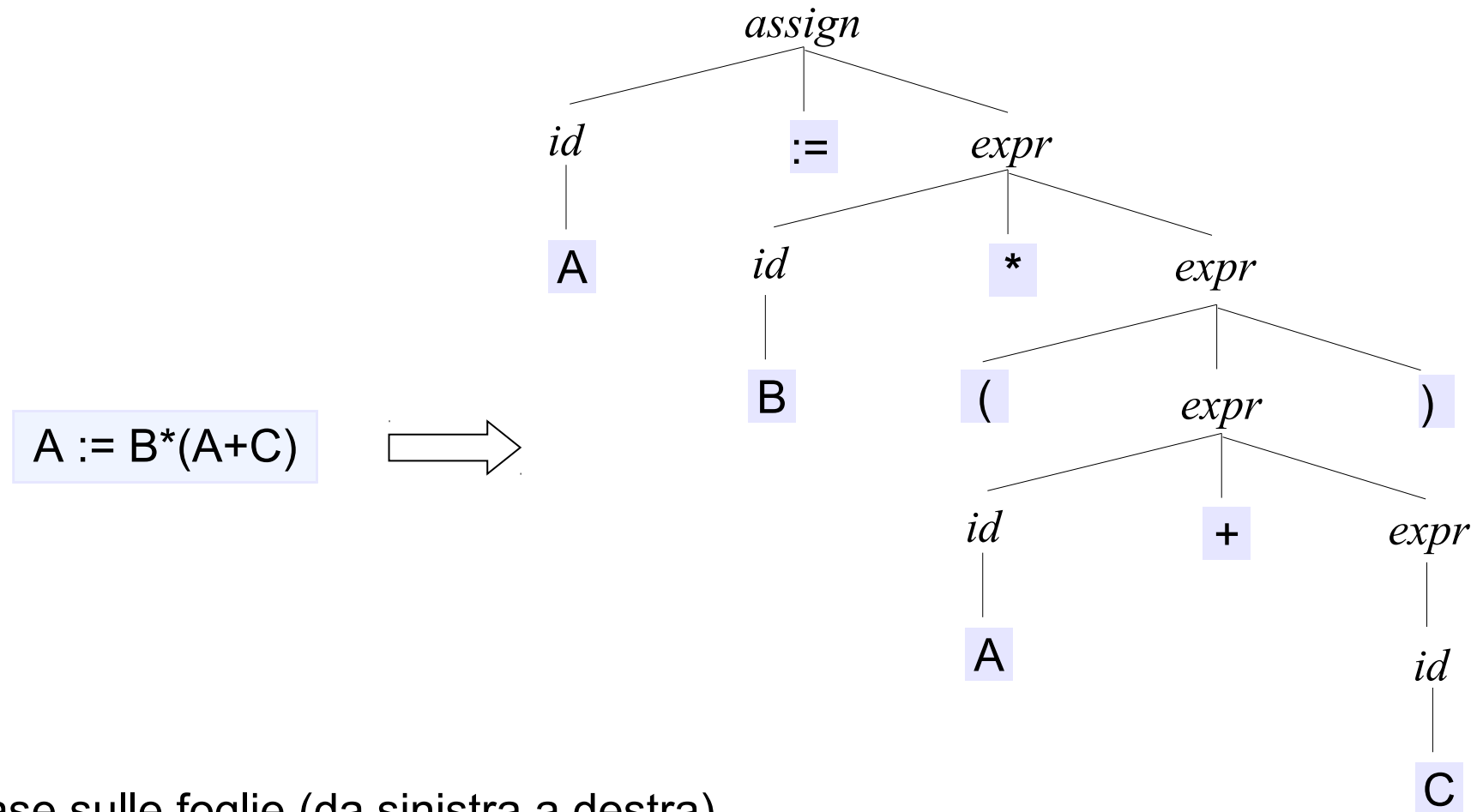
$\Rightarrow A := B * (A + expr)$

$\Rightarrow A := B * (A + id)$

$\Rightarrow A := B * (A + C)$

BNF (v)

- Rappresentazione della struttura gerarchica delle frasi mediante **alberi sintattici**



- Frase sulle foglie (da sinistra a destra)

BNF: Linguaggio per tabelle

R

A	B	C
3	alfa	true
5	beta	false

```
def R (A: integer, B: string, C: boolean)
def S (D: integer, E: string)
R := {(3, "alfa", true)(5, "beta", false)}
S := {(125, "sole")(236, "luna")}
```

S

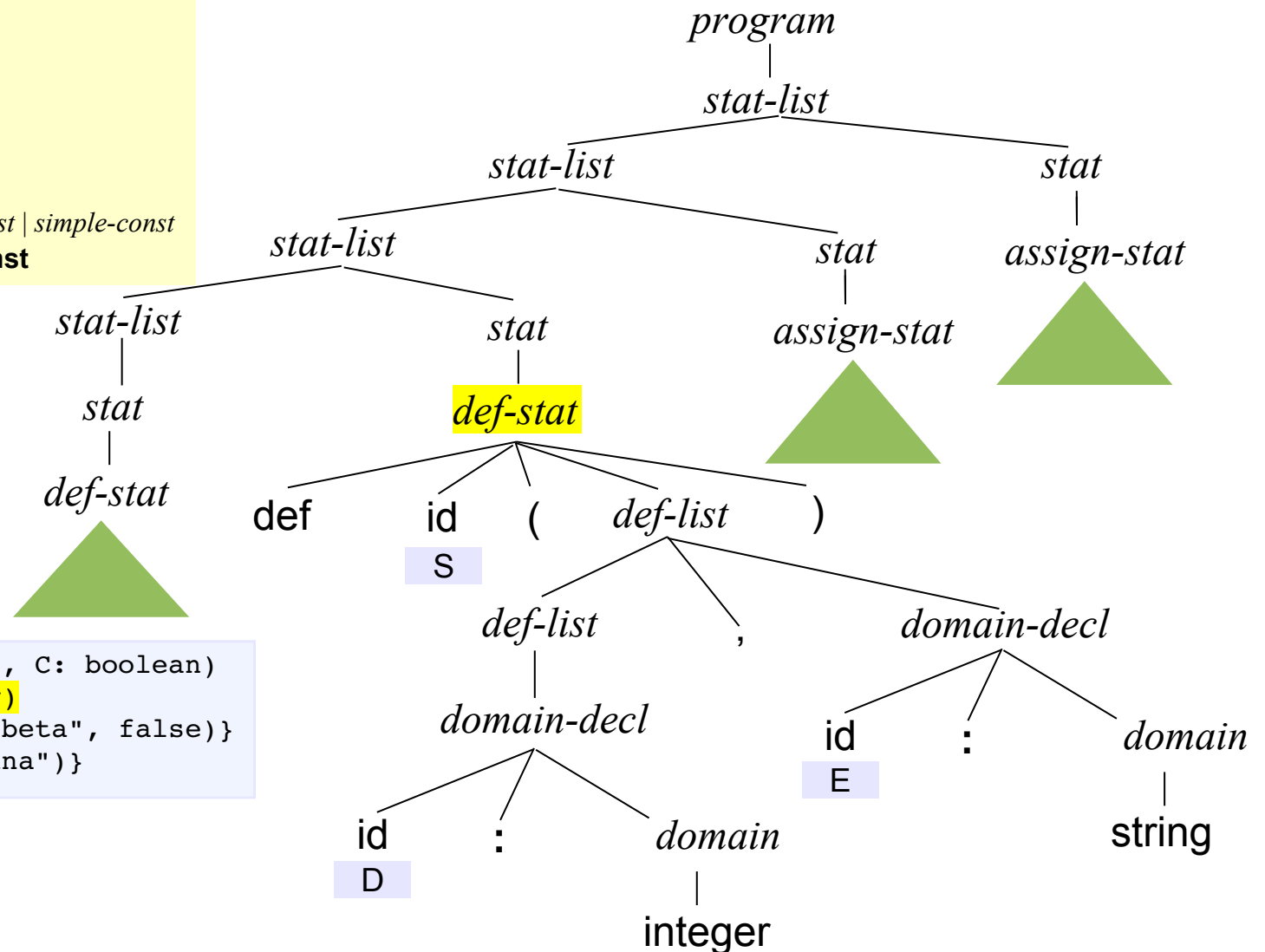
D	E
125	sole
236	luna

```
program → stat-list
stat-list → stat-list stat | stat
stat → def-stat | assign-stat
def-stat → def id ( def-list )
def-list → def-list, domain-decl | domain-decl
domain-decl → id: domain
domain → integer | string | boolean
assign-stat → id := { tuple-list }
tuple-list → tuple-list tuple-const | ε
tuple-const → ( simple-const-list )
simple-const-list → simple-const-list, simple-const | simple-const
simple-const → intconst | strconst | boolconst
```

BNF: Linguaggio per tabelle (ii)

```

program → stat-list
stat-list → stat-list stat | stat
stat → def-stat | assign-stat
def-stat → def id ( def-list )
def-list → def-list, domain-decl | domain-decl
domain-decl → id: domain
domain → integer | string | boolean
assign-stat → id := { tuple-list }
tuple-list → tuple-list tuple-const | ε
tuple-const → ( simple-const-list )
simple-const-list → simple-const-list, simple-const | simple-const
simple-const → intconst | strconst | boolconst
    
```



```

def R (A: integer, B: string, C: boolean)
def S (D: integer, E: string)
R := {(3, "alfa", true)(5, "beta", false)}
S := {(125, "sole")(236, "luna")}
    
```

EBNF

- Stesso potere espressivo → aumento di $\begin{matrix} \text{scrivibilità} \\ \text{leggibilità} \end{matrix}$
- Estensioni → nuovi metacaratteri: [] { } () +

1. Opzionalità

$if-stat \rightarrow \text{if } (expr) stat [\text{else } stat]$

2. Ripetizione

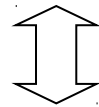
$id-list \rightarrow id \{ , id \}$

3. Disgiunzione

$for-stat \rightarrow \text{for } var := expr \text{ (to | downto) } expr \text{ do } stat$

EBNF (ii)

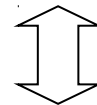
- BNF:

$$\begin{aligned} \text{expr} &\rightarrow \text{expr} + \text{term} \mid \text{expr} - \text{term} \mid \text{term} \\ \text{term} &\rightarrow \text{term} * \text{factor} \mid \text{term} / \text{factor} \mid \text{factor} \end{aligned}$$


- EBNF:

$$\begin{aligned} \text{expr} &\rightarrow \text{term} \{ (+ \mid -) \text{term} \} \\ \text{term} &\rightarrow \text{factor} \{ (* \mid /) \text{factor} \} \end{aligned}$$

Ripetizione non vuota:

$$\text{comp-stat} \rightarrow \text{begin } \text{stat} \{ \text{stat} \} \text{end}$$

$$\text{comp-stat} \rightarrow \text{begin } \{ \text{stat} \}^+ \text{end}$$

EBNF: Linguaggio per tabelle

R

A	B	C
3	alfa	true
5	beta	false

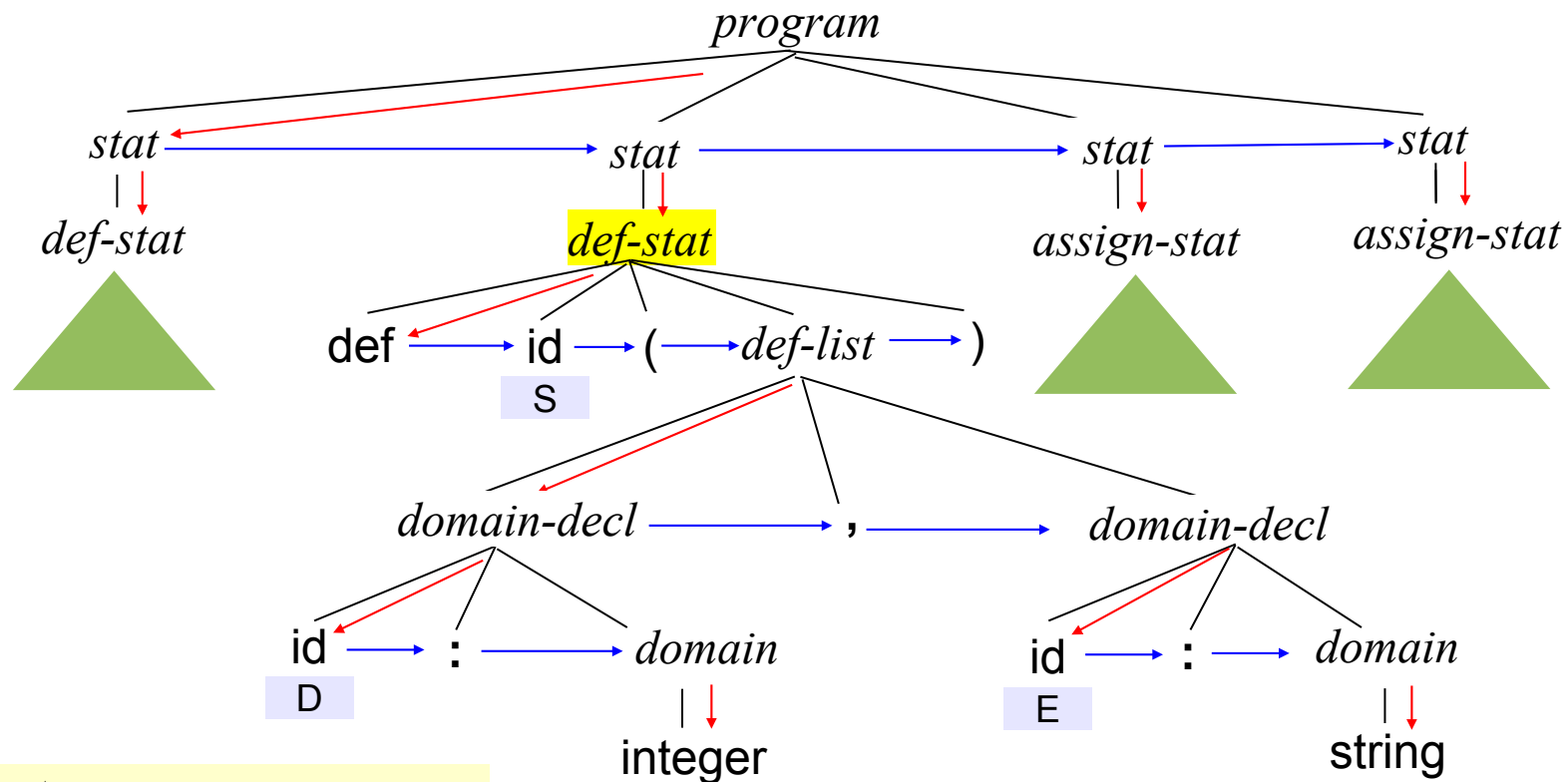
```
def R (A: integer, B: string, C: boolean)
def S (D: integer, E: string)
R := {(3, "alfa", true)(5, "beta", false)}
S := {(125, "sole")(236, "luna")}
```

S

D	E
125	sole
236	luna

```
program → {stat }+
stat → def-stat | assign-stat
def-stat → def id “(“ def-list “)”
def-list → domain-decl {, domain-decl}
domain-decl → id: domain
domain → integer | string | boolean
assign-stat → id := “{” {tuple-const} “}”
tuple-const → “(“ simple-const {, simple-const} “)”
simple-const → intconst | strconst | boolconst
```

EBNF: Linguaggio per tabelle (ii)



```

program → {stat }+
stat → def-stat | assign-stat
def-stat → def id "(" def-list ")"
def-list → domain-decl {, domain-decl}
domain-decl → id: domain
domain → integer | string | boolean
assign-stat → id := "{" {tuple-const} "}"
tuple-const → "(" simple-const {, simple-const} ")"
simple-const → intconst | strconst | boolconst
    
```

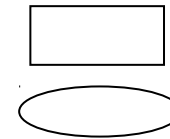
```

def R (A: integer, B: string, C: boolean)
def S (D: integer, E: string)
R := {(3, "alfa", true)(5, "beta", false)}
S := {(125, "sole")(236, "luna")}
    
```

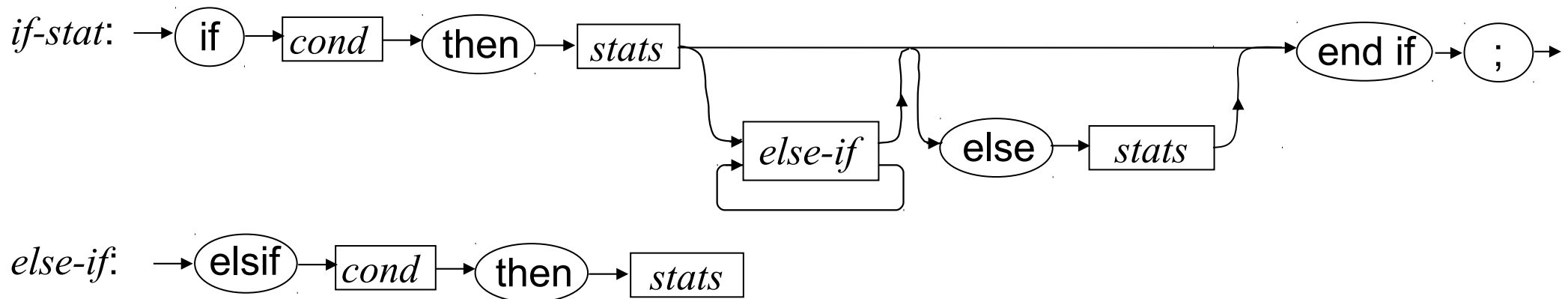

Diagrammi sintattici

- \exists un diagramma \forall unità sintattica (nonterminale)

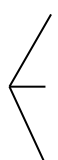

- Diversa rappresentazione grafica per $\left\{ \begin{array}{l} \text{nonterminali} \\ \text{terminali} \end{array} \right.$



- Selezione Ada: $if\text{-}stat \rightarrow \text{if } cond \text{ then } stats \{ \text{else-if} \} [\text{else } stats] \text{ end if } ;$
 $else\text{-}if \rightarrow \text{elsif } cond \text{ then } stats$



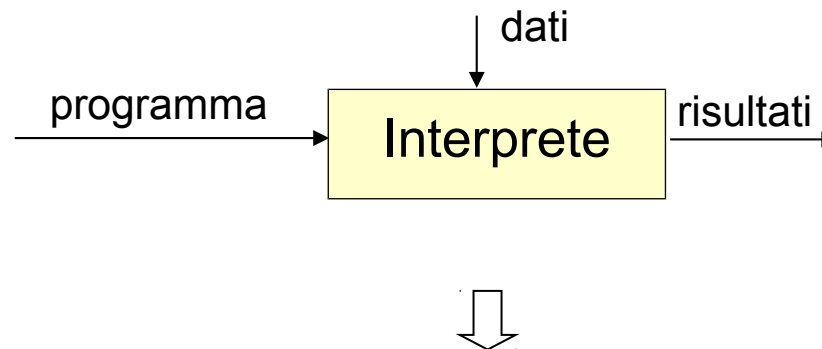
Semantica dinamica

- Per descrivere il significato dei programmi
- Vantaggi 
 - specifica non ambigua del LP (evitamento proliferazione dialetti)
 - possibilità di generazione automatica del compilatore
 - supporto alla prova formale di correttezza dei programmi
- Formalismi: semantica 
 - operazionale**
 - assiomatica
 - denotazionale**

Semantica operativa

(PL/I, 1972)

- Significato di un costrutto descritto dall'esecuzione su macchina $\left\{ \begin{array}{l} \text{reale (< livello)} \\ \text{virtuale (interprete)} \end{array} \right.$
- Se virtuale \rightarrow necessità di costruire 2 componenti $\left\{ \begin{array}{l} \text{traduttore } L \rightarrow L' \\ \text{macchina virtuale di } L' \end{array} \right.$
- Macchina virtuale (interprete) di L' :



Semantica di una istruzione = cambiamento di stato della macchina virtuale

Semantica operativa (ii)

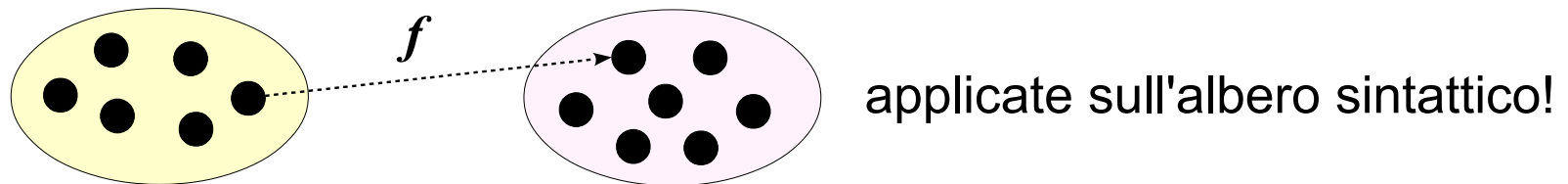
<i>Istruzione C</i>	<i>Semantica operativa</i>
<code>for(<i>expr</i>₁; <i>expr</i>₂; <i>expr</i>₃) <i>statements</i></code>	<code><i>expr</i>₁; loop: if <i>expr</i>₂ = 0 goto out <i>statements</i> <i>expr</i>₃; goto loop out:</code>

Oss: Semantica operazionale espressa algoritmicamente, non matematicamente



Semantica denotazionale

- Semantica denotazionale = formalismo rigoroso più usato per descrivere il significato dei programmi (basato sulla teoria delle funzioni **ricorsive**)
- Idea: \forall astrazione del LP: def $\left\langle \begin{array}{l} \text{dominio matematico} \\ f: \text{istanze della astrazione} \rightarrow \text{elementi del dominio} \end{array} \right.$

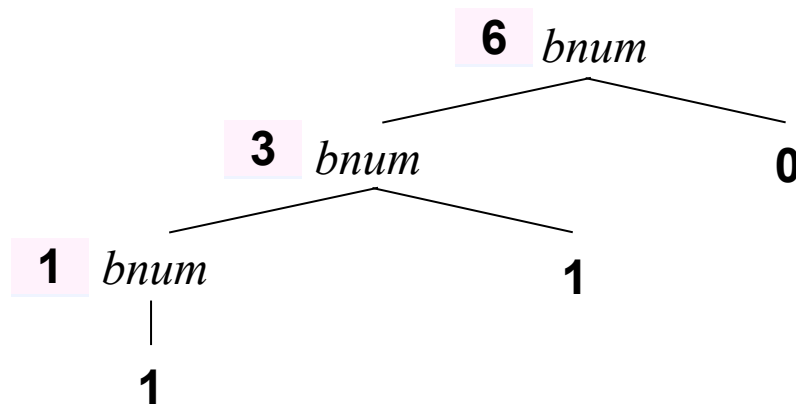


- Oggetti matematici *denotano* il significato delle corrispondenti entità linguistiche

Semantica denotazionale (ii)

Es: Numeri binari:

$bnum \rightarrow 0 \mid 1 \mid bnum\ 0 \mid bnum\ 1$



110 \rightarrow possibili diverse semantiche:

- numero binario
- and / or logico
- vettore di booleani
- ...

- $\mathcal{N} = \{ \text{naturali} \} \equiv$ dominio degli oggetti matematici associati

- $M_b \equiv$ funzione di mapping $\left\{ \begin{array}{l} M_b('0') = 0 \\ M_b('1') = 1 \\ M_b(bnum\ '0') = 2 * M_b(bnum) \\ M_b(bnum\ '1') = 2 * M_b(bnum) + 1 \end{array} \right.$

Semantica denotazionale (iii)

Es: Numeri decimali: $dnum \rightarrow 0 \mid 1 \mid \dots \mid 9 \mid dnum\ 0 \mid dnum\ 1 \mid \dots \mid dnum\ 9$

$$M_d('0') = 0, M_d('1') = 1, \dots, M_d('9') = 9$$

$$M_d(dnum\ '0') = 10 * M_d(dnum)$$

$$M_d(dnum\ '1') = 10 * M_d(dnum) + 1$$

...

$$M_d(dnum\ '9') = 10 * M_d(dnum) + 9$$

Semantica denotazionale (iv)

- **Stato di un programma** $\equiv \{(i_1, v_1), (i_2, v_2), \dots, (i_n, v_n)\}$ $\left\{ \begin{array}{l} i_k = \text{nome di una variabile} \\ v_k = \text{valore corrente di } i_k \\ \text{(eventualmente } \textit{undef}) \end{array} \right.$
- $\mu(i_k, s) \equiv$ funzione che computa v_k

Es: **Espressione** (senza effetti collaterali):

$expr \rightarrow dnum \mid var \mid bexpr$
 $bexpr \rightarrow expr_1 \ op \ expr_2$
 $op \rightarrow + \mid *$

Dominio associato = $\mathbb{Z} \cup \{ \textit{errore} \}$

Interi

$M_e(dnum, s) = M_d(dnum).$

$M_e(var, s) = \text{if } \mu(var, s) = \textit{undef} \text{ then}$
errore
 else $\mu(var, s).$

$M_e(bexpr, s) = M_{be}(bexpr, s).$

$M_{be}(bexpr, s) = \text{if } M_e(expr_1, s) = \textit{errore} \text{ or}$
 $M_e(expr_2, s) = \textit{errore} \text{ then}$
errore
 elsif $M_o(op) = '+' \text{ then}$
 $M_e(expr_1, s) + M_e(expr_2, s)$
 else
 $M_e(expr_1, s) * M_e(expr_2, s).$

$M_o(+) = ('+').$

$M_o(*) = ('*').$

Semantica denotazionale (v)

$expr \rightarrow dnum \mid var \mid bexpr$
 $bexpr \rightarrow expr_1 \ op \ expr_2$
 $op \rightarrow + \mid *$

$s = \{(x, 3), (y, 4)\}$

$x + 12 * y$

$M_e(dnum, s) = M_d(dnum).$

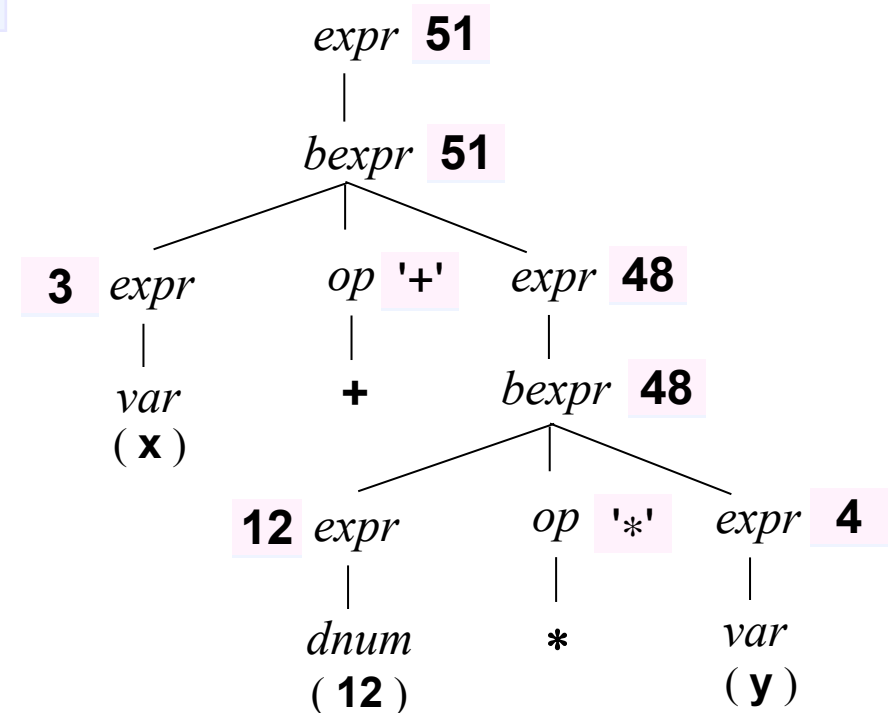
$M_e(var, s) = \text{if } \mu(var, s) = \text{undef} \text{ then}$
 errore
 else $\mu(var, s).$

$M_e(bexpr, s) = M_{be}(bexpr, s).$

$M_{be}(bexpr, s) = \text{if } M_e(expr_1, s) = \text{errore} \text{ or}$
 $M_e(expr_2, s) = \text{errore} \text{ then}$
 errore
 elseif $M_o(op) = '+' \text{ then}$
 $M_e(expr_1, s) + M_e(expr_2, s)$
 else
 $M_e(expr_1, s) * M_e(expr_2, s).$

$M_o(+)=('+').$

$M_o(*)=('*').$



Semantica denotazionale (vi)

Es: **Assegnamento**: $x := E$ $\left\{ \begin{array}{l} \text{valutazione di } E \\ \text{setting di } x \text{ con il valore di } E \end{array} \right.$

$$\begin{aligned} M_a(x := E, s) = & \\ & \text{if } M_e(E, s) = \mathbf{errore} \text{ then} \\ & \quad \mathbf{errore} \\ & \text{else} \\ & \quad s' = \{(i_1, v_1'), (i_2, v_2'), \dots, (i_n, v_n')\}, \forall k \in [1 .. n] \left[v_k' = \begin{cases} \mu(i_k, s) & \text{if } i_k \neq x \\ M_e(E, s) & \text{otherwise} \end{cases} \right] \end{aligned}$$

confronto fra nomi! \nearrow

Semantica denotazionale (vii)

Es: Ciclo a condizione iniziale: **while B do L**

Assumiamo $\exists \begin{cases} M_{\text{list}} : \langle \text{istruzioni} \rangle \rightarrow \text{stato} \\ M_{\text{bool}} : \text{expr booleana} \rightarrow \{ \text{true}, \text{false}, \text{errore} \} \end{cases}$

```
Mw(while B do L, s) =  
    if Mbool(B, s) = errore then  
        errore  
(piede della ricorsione) else if Mbool(B, s) = false then s  
    else if Mlist(L, s) = errore then errore  
(ricorsione) else Mw(while B do L,  $\underbrace{M_{\text{list}}(L, s)}_{\text{stato dopo l'esecuzione di L}}$ )
```

Oss:

- Conversione iterazione \rightarrow ricorsione
- Se \nexists terminazione: computazione di nulla!

Semantica denotazionale (viii)

```
Mw(while B do L, s) =  
  if Mbool(B, s) = errore then  
    errore  
  else if Mbool(B, s) = false then s  
  else if Mlist(L, s) = errore then errore  
  else Mw(while B do L, Mlist(L, s))
```

```
while i<3 do  
  a[i] := i+1;  
  i := i+1  
end;
```

	i	a[0]	a[1]	a[2]	M _{bool} (B, s)
Prima iterazione →	0	0	0	0	true
Seconda iterazione →	1	1	0	0	true
Terza iterazione →	2	1	2	0	true
	3	1	2	3	false

Semantica denotazionale (ix)

Valutazione:

- Uso di semantica denotazionale come ausilio alla progettazione del LP



Costrutti con semantica denotazionale < $\begin{matrix} \text{complessa} \\ \text{difficile} \end{matrix}$ \Rightarrow da riprogettare

- Possibilità (teorica) di generazione automatica del compilatore:
in pratica: \nexists metodi praticabili per generare compilatori utili

- Semantica denotazionale < $\begin{matrix} \text{poco utile agli utenti del LP (difficile)} \\ \text{eccellente per descrivere concisamente un LP} \end{matrix}$