

# Come descrivere un Linguaggio di Programmazione

Sommario: 4-5 marzo, 2020

- Sintassi, Semantica, Pragmatica e Implementazione
- Grammatiche e Sintassi
- Grammatiche Context Free: Derivazione, Alberi, Ambiguità
- Sintassi: Proprietà contestuali
- Compilatore: Le fasi di analisi sintattica, semantica e di generazione del codice
- Esercizi Parte1: Front-End
- Semantica: Formalismi
- Semantica operativa strutturata: Stato, Transizione, Computazione.
- Esercizi Parte2: Semantica

- Un programma si presenta come una "lunga" sequenza di caratteri che rappresentano simboli e strutture del linguaggio con cui il programma è espresso.

```
#include<stdio.h> #include<stdlib.h> voidXSwap(intA[], intB[]){A[0] = ...
```

- **Lessico.** La definizione di **come** e **quali** sequenze di caratteri formino simboli (inclusi i separatori di simboli) è affidata al Lessico.
- **Sintassi.** La definizione di **come** e **quali** sequenze di simboli formino costrutti e la struttura del programma è affidata alla Sintassi.

- **Grammatiche.** Lessico e Sintassi possono essere completamente definite mediante grammatiche:
  - Lineari, per il lessico,  $\mathcal{G}^{\mathcal{R}}$ .
  - Libere da contesto (Context Free), per la sintassi,  $\mathcal{G}^{\mathcal{F}}$ .
  - $\mathcal{G}^{\mathcal{R}} \subset \mathcal{G}^{\mathcal{F}}$ .
- Sia  $A$  insieme finito di simboli (caratteri):
  - $A^*$  insieme di tutte le sequenze finite di simboli su  $A$ .
  - $A^*$  è chiuso rispetto all'operatore binario, associativo,  $\dots$ , chiamato *giustapposizione*<sup>1</sup>: esempio  $\text{Pi.sa} = \text{Pisa}$ .
  - $\epsilon \in A^*$  seq. vuota:  $\epsilon.q = q = q.\epsilon, \forall q \in A^*$

## Definition (Linguaggio (formale) su $A$ )

Un linguaggio (formale) su  $A$  è un sottoinsieme di  $A^*$

<sup>1</sup> la sua notazione è omessa quando non vi è ambiguità

## Definition (Grammatica Libera)

Una grammatica libera è una quadrupla  $(NT, T, R, S)$  tale che:

- **NT** insieme finito dei non terminali.
- **T** insieme finito dei terminali.
- **S** simbolo iniziale,  $S \in NT$ .
- **R** insieme finito delle produzioni,

$$R \equiv \{V_i \rightarrow w_i \mid V_i \in NT, w_i \in (T \cup NT)^*, i \in [1, k]\}$$

Esempio

$$G = (\{E\}, \{I, +, *, (, )\}, E, R)$$
$$R = \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow I, E \rightarrow (E)\}$$

**Notazione.**  $V \rightarrow \alpha_1 | \dots | \alpha_k$  invece di  $V_1 \rightarrow \alpha_1, \dots, V_k \rightarrow \alpha_k$  quando,  $V = V_1 = \dots = V_k$ .

Esempio

$$R = \{E \rightarrow E + E \mid E * E \mid I \mid (E)\}$$

Esempio

$$G = (\{E\}, \{I, +, *, (, )\}, E, R)$$

$$R = \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow I, E \rightarrow (E)\}$$

Qual'è il significato di  $G$ ? <sup>2</sup>

## Definition (La relazione $\Rightarrow$ )

Ogni grammatica libera  $G \equiv (NT, T, R, S)$  definisce una relazione binaria  $\Rightarrow_G$  (o, semplicemente  $\Rightarrow$ ) su  $(T \cup NT)^*$  tale che:

$$\forall \alpha.V.\rho, \alpha.\gamma.\rho \in (T \cup NT)^*, \alpha V \rho \Rightarrow \alpha \gamma \rho \text{ sse } V \rightarrow \gamma \in R$$

## Definition (Linguaggio generato)

Sia  $G \equiv (NT, T, R, S)$  libera. Il linguaggio definito da  $G$  :

$$\mathcal{L}(G) = \{\alpha \in T^* \mid S \Rightarrow^* \alpha\}$$

dove,  $\Rightarrow^*$  è la chiusura transitiva (e riflessiva) di  $\Rightarrow_G$

---

<sup>2</sup> "sse" sta per "se e solo se"

## Definition (Derivazione)

Sia  $G \equiv (NT, T, R, S)$  libera. Siano  $v, w \in (T \cup NT)^*$ . Sia  $v \Rightarrow^* w$ . Una derivazione, in  $G$ , da  $v$  a  $w$ , è una sequenza  $v \Rightarrow w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w$

## Esempio

La derivazione ci fornisce una prova di appartenenza. Si dimostri che  $I * I + I \in \mathcal{L}(G)$ .

$$G = (\{E\}, \{I, +, *\}, E, R)$$

$$R = \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow I, E \rightarrow (E)\}$$

Procediamo applicando una  $\Rightarrow$  alla volta a partire da  $E$

$$E \Rightarrow_2 E * E^3$$

$$\Rightarrow_3 I * E$$

$$\Rightarrow_1 I * E + E$$

$$\Rightarrow_3 I * I + E$$

$$\Rightarrow_3 I * I + I$$

Aggiungendo induzione possiamo dimostrare che:

$$\{I.\alpha_1.\dots.\alpha_n \mid \alpha_i \in \{+I, *I\}, n \geq 0\} \subset \mathcal{L}(G).$$

---

<sup>3</sup>Un pedice può indicare la produzione usata. In questo caso, usiamo la posizione a partire da 1 per la produzione più a sinistra

## Esempio

$$G = (\{E\}, \{I, +, *, (, )\}, E, R)$$

$$R = \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow I, E \rightarrow (E)\}$$

- La  $\Rightarrow$  su sequenze di simboli (stringhe) grammaticali ci dice proprio tutto?

$$E \Rightarrow_2 E * E \Rightarrow_3 I * E \Rightarrow_1 I * E + E \Rightarrow_3 I * I + E \Rightarrow_3 I * I + I$$

$$E \Rightarrow_2 E * E \Rightarrow_1 E * E + E \Rightarrow_3 I * E + E \Rightarrow_3 I * I + E \Rightarrow_3 I * I + I$$

$$E \Rightarrow_1 E + E \Rightarrow_2 E * E + E \Rightarrow_3 I * E + E \Rightarrow_3 I * I + E \Rightarrow_3 I * I + I$$

- In realtà NO.
  - 2 delle 3 derivazioni sopra ci dicono la stessa cosa, 1 dice una cosa differente.
  - Quali? In cosa, sono diverse?
  - Useremo la struttura degli alberi per rispondere.
- La sequenza di simboli grammaticali non è abbastanza espressiva

## Definition

- $[] \in \text{Tree}^A$
- $[a - (t_1, \dots, t_n)] \in \text{Tree}^A$ , per  $a \in A, t_i \in \text{Tree}^A, n \geq 0$

Notazione.  $[a - ()]$  può essere scritto  $[a]$ .

- Gli alberi possono essere utilizzati come relazioni sulla struttura di un termine
  - Ogni arco è una coppia della relazione *sottotermine*
- Hanno una rappresentazione grafica.
  - mostriamo quella di  $[E - ([E],[+],[E])]$



## Definition (La relazione $\Rightarrow^{\text{Tree}}$ )

Ogni grammatica libera  $G \equiv (NT, T, R, S)$  definisce una relazione binaria  $\Rightarrow_G^{\text{Tree}}$  (o, semplicemente  $\Rightarrow^{\text{Tree}}$ , ovvero  $\Rightarrow$ ) su  $\text{Tree}^{\text{TUNT}}$  tale che:

- $[V] \Rightarrow [V - ([u_1], \dots, [u_n])]$   
sse  $V \rightarrow u_1 \dots u_n \in R$  and  $u_i \in (T \cup NT)$
- $[V - (t_1, \dots, t_j, \dots, t_n)] \Rightarrow [V - (t_1, \dots, r_j, \dots, t_n)]$   
sse  $1 \leq j \leq n \wedge (\forall i) t_i \in \text{Tree}^{\text{TUNT}} \wedge t_j \Rightarrow r_j$

## Definition (Linguaggio generato su $\text{Tree}^{\text{TUNT}}$ - Parse Tree set)

Sia  $G \equiv (NT, T, R, S)$  libera. L'insieme dei parse tree di  $G$  :

$$\mathcal{PT}(G) = \{\alpha \in \text{Tree}^{\text{TUNT}} \mid [S] \Rightarrow^* \alpha\}$$

dove,  $\Rightarrow^*$  è la chiusura transitiva (e riflessiva) di  $\Rightarrow$

## Definition (Derivazione su Tree<sup>TUNT</sup>)

Sia  $G \equiv (NT, T, R, S)$  libera. Siano  $u, q \in \text{Tree}^{\text{TUNT}}$ . Sia  $u \Rightarrow^* q$ .  
Una derivazione, in  $G$ , da  $u$  a  $q$ , è una sequenza  
 $u \Rightarrow q_0 \Rightarrow q_1 \Rightarrow \dots \Rightarrow q$

### Esempio

$$G = (\{E\}, \{I, +, *, (, )\}, E, R)$$
$$R = \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow I, E \rightarrow (E)\}$$

### • La $\Rightarrow$ su stringhe ci dice proprio tutto?

$$E \Rightarrow_2 E * E \Rightarrow_3 I * E \Rightarrow_1 I * E + E \Rightarrow_3 I * I + E \Rightarrow_3 I * I + I$$
$$E \Rightarrow_2 E * E \Rightarrow_1 E * E + E \Rightarrow_3 I * E + E \Rightarrow_3 I * I + E \Rightarrow_3 I * I + I$$
$$E \Rightarrow_1 E + E \Rightarrow_2 E * E + E \Rightarrow_3 I * E + E \Rightarrow_3 I * I + E \Rightarrow_3 I * I + I$$

### • La $\Rightarrow$ su alberi?

$$[E] \Rightarrow_2 [E - ([E], [*], [E])] \Rightarrow_3 [E - ([E - ([I])], [*], [E])]$$
$$\Rightarrow^* [E - ([E - ([I])], [*], [E - ([E - ([I])], [+], [E - ([I])]])]$$

$$[E] \Rightarrow^* \dots$$

$$[E] \Rightarrow_1 [E - ([E], [+], [E])] \Rightarrow_2 [E - ([E - ([E], [*], [E])], [+], [E])]$$
$$\Rightarrow^* [E - ([E - ([E - ([I])], [*], [E - ([I])]), [+], [E - ([I])])]$$

# La relazione $\Rightarrow$ su alberi: Una vista grafica

Esempio

- La  $\Rightarrow$  su alberi?

$$\begin{aligned} [E] &\Rightarrow_2 [E - ([E], [*], [E])] \Rightarrow_3 [E - ([E - ([I])], [*], [E])] \\ &\Rightarrow^* [E - ([E - ([I])], [*], [E - ([E - ([I])], [+], [E - ([I])]])] \end{aligned}$$

$[E] \Rightarrow^* \dots$

$$\begin{aligned} [E] &\Rightarrow_1 [E - ([E], [+], [E])] \Rightarrow_2 [E - ([E - ([E], [*], [E])], [+], [E])] \\ &\Rightarrow^* [E - ([E - ([E - ([I])], [*], [E - ([I])])], [+], [E - ([I])])] \end{aligned}$$

## Definition (Frontiera di un albero)

La frontiera di un albero  $q \in \text{Tree}^A$  è una stringa di  $A^*$ , ovvero  $\text{fron}(q) \in A^*$  per ogni  $q$ , così definita:

- $\text{fron}([\ ])=\epsilon$
- $\text{fron}([a])=a$
- $\text{fron}([a - (t_1, \dots, t_n)]) = \text{fron}(t_1) \dots \text{fron}(t_n), \quad n > 0$

## Esempio

$$\begin{aligned} \text{fron}([E - ([E - ([I]), [*], [E - ([E - ([I]), [+], [E - ([I])])])])]) &= \\ &= \text{fron}([E - ([I])]) \cdot \text{fron}([*]) \cdot \text{fron}([E - ([E - ([I]), [+], [E - ([I])])]) \\ &= \text{fron}([I]) \cdot \text{fron}([*]) \cdot \text{fron}([E - ([I])]) \cdot \text{fron}([+]) \cdot \text{fron}([E - ([I])]) \\ &= \text{fron}([I]) \cdot \text{fron}([*]) \cdot \text{fron}([I]) \cdot \text{fron}([+]) \cdot \text{fron}([I]) \\ &= I \cdot * \cdot I \cdot + \cdot I = I * I + I \end{aligned}$$

## Definition (Linguaggio generato utilizzando Parse Tree)

Sia  $G \equiv (NT, T, R, S)$  libera. Il linguaggio definito da  $G$  :

$$\mathcal{L}(G) = \{\text{fron}(q) \in T^* \mid [S] \Rightarrow^* q\}$$

dove,  $\Rightarrow^*$  è la chiusura transitiva (e riflessiva) di  $\Rightarrow_G^{\text{Tree}^{\text{TUNT}}}$

## Definition (Grammatica Ambigua)

$G \equiv (NT, T, R, S)$  è ambigua se  $\exists q_1 \neq q_2 \in \text{Tree}^{\text{TUNT}}$ :

- $[S] \Rightarrow^* q_1, [S] \Rightarrow^* q_2$
- $\text{fron}([q_1]) = \text{fron}([q_2]) \in T^*$

- La **sintassi di un LP** deve essere espressa mediante una grammatica libera **non ambigua**
- Ogni **termine** di LP deve avere **un'unica struttura sintattica** che ne mostra i sottotermini componenti

## Esempio Grammatica Ambigua

$$G = (\{E\}, \{I, +, *\}, E, R)$$

$$R = \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow I, E \rightarrow (E)\}$$

## Esempio Grammatica non Ambigua

$$G' = (\{E, F, T\}, \{I, +, *\}, E, R')$$

$$R' = \{E \rightarrow E + F \mid F$$

$$F \rightarrow F * T \mid T$$

$$T \rightarrow I \mid (E)\}$$

$$\mathcal{L}(G) = \mathcal{L}(G')$$

- Lessico e Sintassi si integrano nella definizione di un Linguaggio

**Esempio** sottoLinguaggio delle Espressioni Aritmetiche su Naturali

Sintassi: Espressione

$$G = (\{E, F, T\}, \{N, +, -, *\}, E, R_G)$$

$$R_G = \{ E \rightarrow E + F \mid F$$

$$F \rightarrow F * T \mid T$$

$$T \rightarrow -T \mid N \mid (E) \}$$

Lessico: Naturale (Identificatore)

$$H = (\{L, N, D\}, \{-, +, *, 0, \dots, 9\}, L, R_H)$$

$$R_H = \{ L \rightarrow N \mid * \mid - \mid +$$

$$N \rightarrow D N \mid D$$

$$D \rightarrow 0 \mid 1 \mid \dots \mid 9 \}$$

Riconosciamo la stringa,  $13*-5+007 \in \mathcal{L}(\langle G, H \rangle)$ .

# Compiler/Interprete: Front-End e Back-End

Front-End  
(Compilatore  
Interprete)

SYMBOL TABLE	
1	position    . . .
2	initial    . . .
3	rate      . . .
4	

Back-End  
Compilatore

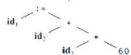
Aho, A.V. et al., Compilers: Principles,  
Techniques, & Tools, 2 ed., Addison-Wesley,  
2007, pag. 7, Fig. 1.7

position := initial + rate \* 60

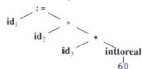
lexical analyzer

$id_1 := id_2 + id_3 * 60$

syntax analyzer



semantic analyzer



intermediate code generator

```
temp1 := intoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

code optimizer

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

code generator

```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```



- **Programmi (sintatticamente) Legali** sono i programmi sintatticamente corretti a cui la semantica del linguaggio può associare la funzione calcolabile descritta dal programma.
- La sintassi espressa da una grammatica non ambigua non è sufficiente a identificare tutti e solo i programmi legali.
- Ad esempio:  $x = 7$  è un assegnamento
  - sintatticamente corretto in tutti i programmi C<sup>4</sup>
  - ma non è legale se  $x$  non è stato *dichiarato*<sup>5</sup>

---

<sup>4</sup>e di tutti i L.P. dove  $x$ ,  $_=_$ ,  $7$  siano la sintassi per una variabile, per l'operatore di assegnamento, per un'espressione, rispettivamente.

<sup>5</sup>opportunitamente, nel programma

- **Proprietà contestuali** (delle regole di composizione) non possono essere espresse da una grammatica libera (da contesto, i.e. Context-Free):
  - identificatori usati devono essere dichiarati prima ...;
  - numero di parametri attuali e formali devono coincidere
  - compatibilità nell'assegnamento, tra il tipo di una variabile e il tipo dell'espressione assegnata
  - ...
- Queste proprietà contestuali **fanno parte della definizione del linguaggio**, ma devono essere espresse con strumenti adatti.
- **Analisi Statica** Sono procedimenti di analisi e formalismi specifici (ad es., Sistema dei Tipi) per controllare che il programma soddisfi tutte le proprietà contestuali del linguaggio.

- **Programmi (sintatticamente) Legali.** Compilatori e Interpreti trattano tutti questi aspetti nel front-end.
- Il **Front-End** di C./I.<sup>6</sup> provvede alla:
  - **Analisi Lessicale** in accordo al lessico
  - **Analisi Sintattica** in accordo alla sintassi
  - **Analisi Statica** in accordo alle proprietà contestuali
  - **Costruzione Abstract Tree**  $AT(p)$  che fornisce una rappresentazione interna del programma sorgente  $p$ .
- $AT(p)$  è successivamente utilizzato dal **Back-End** di C./I. per completare la realizzazione dell'esecutore.
- Vediamo tutto questo nella tipica struttura (in fasi) di un Compilatore.

---

<sup>6</sup>Compilatori e Interpreti

# Compiler/Interprete: Front-End e Back-End

Front-End  
(Compilatore  
Interprete)

SYMBOL TABLE	
1	position    . . .
2	initial    . . .
3	rate      . . .
4	

Back-End  
Compilatore

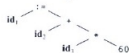
Aho, A.V. et al., Compilers: Principles,  
Techniques, & Tools, 2 ed., Addison-Wesley,  
2007, pag. 7, Fig. 1.7

position := initial + rate \* 60

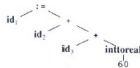
lexical analyzer

$id_1 := id_2 + id_3 * 60$

syntax analyzer



semantic analyzer



intermediate code generator

```
temp1 := intoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

code optimizer

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

code generator

```
MOVf id3, R2
MULF #60.0, R2
MOVf id2, R1
ADDF R2, R1
MOVf R1, id1
```

# Sintassi Concreta e Sintassi Astratta /1

## Definition (Sintassi Concreta)

Insieme delle sequenze di simboli, dato un lessico, che formano i costrutti ed i programmi esprimibili in un Linguaggio di Programmazione.

- Interfaccia per il programmatore
- Molto più facile scrivere un programma in S. Concreta che in S. Astratta.
- Ma occorre conoscere Sintassi del linguaggio utilizzato.

```
#include <stdio.h>
#include <stdlib.h>

int main (void){
    int x = 10, y = 5, z = 7;
    if (x>y) if (y-z>0) z=x;
    else z = z+y;
    printf("x = %d; y = %d; z = %d;\n", x, y, z);
    return(1);
}
```

- Conoscete la Sintassi del Linguaggio C?
- Allora spiegate la struttura dei condizionali usati nel programma, ovvero quali valori sono stampati dalla sua esecuzione.

## Definition (Sintassi Astratta)

Insieme degli alberi etichettati che rappresentano i costrutti ed i programmi esprimibili in un Linguaggio di Programmazione. La rappresentazione deve mostrare in modo univoco, la relazione tra il termine e i sotto-termini che lo compongono.

- Interfaccia per il Back-End
  - Faticoso scrivere un programma in S. Astratta: Ma MAI AMBIGUA
  - Contiene tutte e solo le proprietà del programma (niente fronzoli mnemonici sui simboli, o parentesi per associare/grouping termini, o convenzioni o sottintesi)
- 
- Durante lo studio di un LP la definizione della S. Concreta è data come l'ultima attività
  - Per gran parte dello sviluppo di un LP, i programmi del nuovo linguaggio sono scritti direttamente in S. Astratta.
  - Preparatevi a farlo in Laboratorio.

# Sintassi Astratta: Equazioni su Alberi

## Definition ("Grammatica" per Sintassi Astratta)

La S. Astratta è espressa con formalismi che utilizzano notazioni che ricordano una grammatica. Usano definizioni che ricordano le produzioni ed hanno forma:

$$A ::= [\text{csr}] A_1 \dots A_n$$

dove:

- $A, A_1, \dots, A_n$  sono tipi di alberi: Ogni tipo associato univocamente ad una categoria della sintassi concreta
- $[\text{csr}]$  è un costruttore di alberi con segnatura:  $A_1 \dots A_n \rightarrow A$
- I costruttori costruiscono alberi su  $\text{Tree}^T$ , con T sottoinsieme dei Terminali della Sintassi Concreta: L'etichetta di ogni nodo dipende dal costruttore.

**Esempio** sottoLinguaggio delle Espressioni Aritmetiche (omettiamo il lessico per N)

**Sintassi Astratta:** Una grammatica su  $\text{Tree}^T$  per le Espressioni

$$G^A = (\{E\}, \{N, +, -, *\}, E, R_{G^A})$$

$$R_{G^A} = \{E ::= [+ ] E E, E ::= [* ] E E, E = [- ] E, E = [N ]\}$$

Somma di fattori:  $[+ ]([N ], [* ]([N ], [N ]))$

Prodotti di addendi:  $[* ]([+ ]([N ], [N ]), [N ])$

Sintassi concreta:  $N + N * N$

# Esercizi L4

- Una grammatica  $G$  calcola un linguaggio. Si dica:
  - Quali sono gli oggetti di tale linguaggio;
  - Di quale insieme è sotto-insieme tale linguaggio
  - Quale relazione (fornire definizione) permette di definire tale linguaggio?
  - Si fornisca la definizione di tale linguaggio
- Sia  $G \equiv \{\{T\}, \{A, E\}, T, \{T \rightarrow A, T \rightarrow A, T \rightarrow A E T\}\}$ . Si applichino a  $G$  le formulazioni date nei punti (a)-(e) di esercizio L4.1.
- Si formalizzi la relazione tra  $\Rightarrow_G$  e  $\Rightarrow_G^{\text{Tree}}$
  - Quando usare l'una e quando l'altra?
- Si completi la definizione di una grammatica le cui produzioni sono sotto.
  - Si mostri che la grammatica ottenuta è ambigua.
$$E \rightarrow T \mid T + E \mid T - E$$
$$T \rightarrow A \mid A * E$$
- Si dia una grammatica non ambigua che generi tutte e sole le sequenze di parentesi angolate bilanciate
- Si chiama regolare una grammatica ... (v. Esercizi4Lezione4Parte1).  
Si mostri che il linguaggio  $L \equiv \{a^n b^m \mid n, m \geq 1\}$  è regolare.
- Si dia una grammatica non ambigua per il linguaggio  $L \equiv \{a^n b^m \mid n \leq m\}$ .
- A cosa serve la sintassi concreta di un  $L$ . di Programmazione?
  - Cosa la distingue dalla sintassi astratta.

Altri esercizi in:

\* Esercizi4Lezione4Parte1

\* Cap. 2 [GM]



- Associare significato ai termini del linguaggio.
- Vari formalismi con differenti accezioni di significato e differenti usi.
- Due da ricordare:
  - **Semantica Denotazionale**
    - *Significato*: Funzione Calcolata
    - *Usi*: Molteplici incluso definizione di C. e di I.
    - *Laboratorio*: Interpreti di  $\mathcal{L}$ , derivabili dalla S. Den. di  $\mathcal{L}$ , riscritta in un Linguaggio di P. Funzionale, OCaml
    - *Caratteristica*: Orientata allo studio di proprietà di  $\mathcal{L}$ , Astratta dall'implementazione di  $\mathcal{L}$ .
  - **Semantica Operazionale**

- Associare significato ai termini del linguaggio.
- Vari formalismi con differenti accezioni di significato e differenti usi.
- Due da ricordare:
  - **Semantica Denotazionale**
  - **Semantica Operazionale**
    - *Significato*: Computazione (su Macchina)
    - *Usi*: Definizione di I., anche didattici e per studio di programmi e programmazione in  $\mathcal{L}$
    - *Caratteristica*: Basata sulla nozione di Stato, transizione di stato, computazione
- Useremo una Semantica Operazionale *Strutturata* in cui la Macchina è una Macchina Astratta.

- La definizione di Stato, Transizione di stato e Computazione di una S.O.S, dipende dal Linguaggio.
- Vediamole per il Linguaggio sotto.
- Assumiamo che sintassi (concreta) e lessico siano già stati trattati, analizzati, fornendoci la sintassi astratta dei termini del linguaggio: "(AExp-AExp)" va letto come un albero con radice "-" e due alberi "AExp" come figli (i simboli "(" , ")" rimarcano questa lettura).

$$Num ::= 1 \mid 2 \mid 3 \mid \dots$$
$$Var ::= X_1 \mid X_2 \mid X_3 \mid \dots$$
$$AExp ::= Num \mid Var \mid (AExp + AExp) \mid (AExp - AExp)$$
$$BExp ::= \mathbf{tt} \mid \mathbf{ff} \mid (AExp == AExp) \mid \neg BExp \mid (BExp \wedge BExp)$$
$$Com ::= \mathbf{skip} \mid Var := AExp \mid Com; Com \mid$$
$$\mathbf{if} BExp \mathbf{then} Com \mathbf{else} Com \mid \mathbf{while} BExp \mathbf{do} Com$$

$$Num ::= 1 \mid 2 \mid 3 \mid \dots$$
$$Var ::= X_1 \mid X_2 \mid X_3 \mid \dots$$
$$AExp ::= Num \mid Var \mid (AExp + AExp) \mid (AExp - AExp)$$
$$BExp ::= \mathbf{tt} \mid \mathbf{ff} \mid (AExp == AExp) \mid \neg BExp \mid (BExp \wedge BExp)$$
$$Com ::= \mathbf{skip} \mid Var := AExp \mid Com; Com \mid$$
$$\mathbf{if} BExp \mathbf{then} Com \mathbf{else} Com \mid \mathbf{while} BExp \mathbf{do} Com$$

- La definizione di Stato.

- Identificatori di un unico tipo: Variabili (valori modificabili)
- Memoria simbolica per trattare le variabili (valori modificabili)
- Non abbiamo I/O, file systems,.... Lo stato è la sola memoria
- Lo stato è rappresentato da sequenza finita di coppie  $(X_i, n_i)$ , indicante ...
- Lo stato è denotato con le variabili  $\sigma, \tau$  (anche con pedici)

$$Num ::= 1 \mid 2 \mid 3 \mid \dots$$
$$Var ::= X_1 \mid X_2 \mid X_3 \mid \dots$$
$$AExp ::= Num \mid Var \mid (AExp + AExp) \mid (AExp - AExp)$$
$$BExp ::= \mathbf{tt} \mid \mathbf{ff} \mid (AExp == AExp) \mid \neg BExp \mid (BExp \wedge BExp)$$
$$Com ::= \mathbf{skip} \mid Var := AExp \mid Com; Com \mid$$
$$\mathbf{if} BExp \mathbf{then} Com \mathbf{else} Com \mid \mathbf{while} BExp \mathbf{do} Com$$

- La definizione di Transizione.
  - Esecuzione di un costrutto  $c$  nello stato  $\sigma$  della Macchina Astratta
  - La esprimiamo con:
    - $\langle c, \sigma \rangle \rightarrow \tau$ , indicante ...
    - $\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle$ , indicante ...
    - $\langle c_1, \sigma_1 \rangle \rightarrow \langle c'_1, \sigma'_1 \rangle, \dots, \langle c_k, \sigma_k \rangle \rightarrow \langle c'_k, \sigma'_k \rangle / \langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle$ ,  
k-premesse/1-conclusione, indicante ...
  - La semantica di  $\mathcal{L}$  fornisce le transizioni che sono specifiche per  $\mathcal{L}$ , ovvero quelle  $\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle$  che sono provate **vere**.

$$Num ::= 1 \mid 2 \mid 3 \mid \dots$$
$$Var ::= X_1 \mid X_2 \mid X_3 \mid \dots$$
$$AExp ::= Num \mid Var \mid (AExp + AExp) \mid (AExp - AExp)$$
$$BExp ::= \mathbf{tt} \mid \mathbf{ff} \mid (AExp == AExp) \mid \neg BExp \mid (BExp \wedge BExp)$$
$$Com ::= \mathbf{skip} \mid Var := AExp \mid Com; Com \mid$$
$$\mathbf{if} BExp \mathbf{then} Com \mathbf{else} Com \mid \mathbf{while} BExp \mathbf{do} Com$$

- La definizione di Computazione di  $p \in \mathcal{L}$ .
  - Sequenza degli stati attraversati effettivamente dalle transizioni usate nell'esecuzione del programma  $p$ .

$$Num ::= 1 \mid 2 \mid 3 \mid \dots$$
$$Var ::= X_1 \mid X_2 \mid X_3 \mid \dots$$
$$AExp ::= Num \mid Var \mid (AExp + AExp) \mid (AExp - AExp)$$
$$BExp ::= \mathbf{tt} \mid \mathbf{ff} \mid (AExp == AExp) \mid \neg BExp \mid (BExp \wedge BExp)$$
$$Com ::= \mathbf{skip} \mid Var := AExp \mid Com; Com \mid$$
$$\mathbf{if} BExp \mathbf{then} Com \mathbf{else} Com \mid \mathbf{while} BExp \mathbf{do} Com$$

- Notazione.

$(X_1, n_1), \dots, (X_k, n_k)$  stato con  $k$  variabili legate

$\sigma, \tau$  (anche con pedici) sono stati della macchina

$\sigma(X_i) = n_i$  quando  $\sigma = (X_1, n_1), \dots, (X_i, n_i), \dots, (X_k, n_k)$ ,

$\sigma[X_i \leftarrow m_i] = (X_1, n_1), \dots, (X_i, m_i), \dots, (X_k, n_k)$

quando  $\sigma = (X_1, n_1), \dots, (X_i, n_i), \dots, (X_k, n_k)$

$n, n_i \in Num$  valori numerici

$a, a_i \in AExp$  espressioni aritmetiche

$b, b_i \in BExp$  espressioni booleane

$\mathbf{tt}, \mathbf{ff}$  i valori true e false

$c, c_i \in Com$  comandi del linguaggio

$$Num ::= 1 \mid 2 \mid 3 \mid \dots$$
$$Var ::= X_1 \mid X_2 \mid X_3 \mid \dots$$
$$AExp ::= Num \mid Var \mid (AExp + AExp) \mid (AExp - AExp)$$

## Semantica delle Espressioni AExp.

$$\langle X, \sigma \rangle \rightarrow \langle \sigma(X), \sigma \rangle$$
$$\frac{\langle (n+m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}{\text{where } p = n+m} \quad \frac{\langle (n-m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}{\text{where } p = n-m \text{ e } n \geq m}$$
$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a' + a_2), \sigma \rangle} \quad \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a_1 + a''), \sigma \rangle}$$
$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a' - a_2), \sigma \rangle} \quad \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a_1 - a''), \sigma \rangle}$$

- L'ordine di valutazione degli argomenti delle operazioni è inessenziale
- Se lo volessimo da sinistra a destra: Come modificarla?



$$AExp = Var \mid Num \mid (AExp + AExp) \mid (AExp - AExp)$$

$\text{var: } \frac{X \in \text{Var}}{\langle X, \sigma \rangle \rightarrow \langle \sigma(X), \sigma \rangle}$	
$\text{primPlus: } \frac{n, m \in \text{Num} \quad p = n \oplus m}{\langle (n + m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$	$\text{primMinus: } \frac{n, m \in \text{Num} \quad p = n \ominus m}{\langle (n - m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$
$\text{plusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a' + a_2), \sigma \rangle}$	$\text{plusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a_1 + a''), \sigma \rangle}$
$\text{minusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a' - a_2), \sigma \rangle}$	$\text{minusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a_1 - a''), \sigma \rangle}$

Applichiamo le regole all'espressione  $a \equiv ((x - y) + 7)$  nello stato  $\sigma \equiv (x, 12), (y, 3)$

Step 1: PlusL

$$R(\text{PlusL}) \frac{\langle ((x - y) + 7), \sigma \rangle \rightarrow \langle (a' + 7), \sigma \rangle}{\langle (x - y), \sigma \rangle \rightarrow \langle a', \sigma \rangle}$$

Step 1: PlusL

Step 2: MinusL

$$R(\text{plusL}) \frac{\langle ((x - y) + 7), \sigma \rangle \rightarrow \langle (a' + 7), \sigma \rangle}{R(\text{minusL}) \frac{\langle (x - y), \sigma \rangle \rightarrow \langle (x' - y), \sigma \rangle}{\langle x, \sigma \rangle \rightarrow \langle x', \sigma \rangle}}$$

- continua

$$AExp = Var \mid Num \mid (AExp + AExp) \mid (AExp - AExp)$$

$\text{var: } \frac{X \in \text{Var}}{\langle X, \sigma \rangle \rightarrow \langle \sigma(X), \sigma \rangle}$	
$\text{primPlus: } \frac{n, m \in \text{Num} \quad p = n \oplus m}{\langle (n + m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$	$\text{primMinus: } \frac{n, m \in \text{Num} \quad p = n \ominus m}{\langle (n - m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$
$\text{plusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a' + a_2), \sigma \rangle}$	$\text{plusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a_1 + a''), \sigma \rangle}$
$\text{minusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a' - a_2), \sigma \rangle}$	$\text{minusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a_1 - a''), \sigma \rangle}$

Step 1: PlusL

Step 2: MinusL

Step 3: Var

$$\begin{array}{c}
 \text{R(plusL)} \frac{\langle (x - y) + 7, \sigma \rangle \rightarrow \langle (a' + 7), \sigma \rangle}{\langle (x - y), \sigma \rangle \rightarrow \langle (x' - y), \sigma \rangle} \\
 \text{R(minusL)} \frac{\langle (x - y), \sigma \rangle \rightarrow \langle (x' - y), \sigma \rangle}{\langle x, \sigma \rangle \rightarrow \langle x', \sigma \rangle} \\
 \text{R(Var)} \frac{\langle x, \sigma \rangle \rightarrow \langle x', \sigma \rangle}{x \in \text{Var}, x' = \sigma(x) = 12}
 \end{array}$$

Questo conduce alla seguente transizione:

$$\langle (x - y) + 7, \sigma \rangle \xrightarrow{(\text{plusL}, \text{minusL}, \text{Var})} \langle ((12 - y) + 7), \sigma \rangle$$

$$AExp = Var \mid Num \mid (AExp + AExp) \mid (AExp - AExp)$$

$\text{var: } \frac{X \in \text{Var}}{\langle X, \sigma \rangle \rightarrow \langle \sigma(X), \sigma \rangle}$	
$\text{primPlus: } \frac{n, m \in \text{Num} \quad p = n \oplus m}{\langle (n + m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$	$\text{primMinus: } \frac{n, m \in \text{Num} \quad p = n \ominus m}{\langle (n - m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$
$\text{plusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a' + a_2), \sigma \rangle}$	$\text{plusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a_1 + a''), \sigma \rangle}$
$\text{minusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a' - a_2), \sigma \rangle}$	$\text{minusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a_1 - a''), \sigma \rangle}$

Step 1: PlusL

Step 2: MinusR

Step 3: Var

$$\begin{array}{c}
 \text{R(plusL)} \frac{\langle ((12 - y) + 7), \sigma \rangle \rightarrow \langle (a' + 7), \sigma \rangle}{\langle (12 - y), \sigma \rangle \rightarrow \langle (12 - y'), \sigma \rangle} \\
 \text{R(minusR)} \frac{\langle (12 - y), \sigma \rangle \rightarrow \langle (12 - y'), \sigma \rangle}{\langle y, \sigma \rangle \rightarrow \langle y', \sigma \rangle} \\
 \text{R(Var)} \frac{\langle y, \sigma \rangle \rightarrow \langle y', \sigma \rangle}{y \in \text{Var}, y' = \sigma(y) = 3}
 \end{array}$$

Questo conduce alla seguente sequenza di transizioni:

$$\begin{array}{l}
 \langle ((x - y) + 7), \sigma \rangle \\
 (\text{plusL}, \text{minusL}, \text{Var}) \rightarrow \langle ((12 - y) + 7), \sigma \rangle \\
 (\text{minusR}, \text{Var}) \rightarrow \langle ((12 - 3) + 7), \sigma \rangle
 \end{array}$$

$$AExp = Var \mid Num \mid (AExp + AExp) \mid (AExp - AExp)$$

$\text{var: } \frac{X \in \text{Var}}{\langle X, \sigma \rangle \rightarrow \langle \sigma(X), \sigma \rangle}$	
$\text{primPlus: } \frac{n, m \in \text{Num} \quad p = n \oplus m}{\langle (n + m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$	$\text{primMinus: } \frac{n, m \in \text{Num} \quad p = n \ominus m}{\langle (n - m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$
$\text{plusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a' + a_2), \sigma \rangle}$	$\text{plusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a_1 + a''), \sigma \rangle}$
$\text{minusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a' - a_2), \sigma \rangle}$	$\text{minusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a_1 - a''), \sigma \rangle}$

Step 1: PlusL

Step 2: primMinus

$$R(\text{plusL}) \frac{\langle ((12 - 3) + 7), \sigma \rangle \rightarrow \langle (a' + 7), \sigma \rangle}{R(\text{primMinus}) \frac{\langle (12 - 3), \sigma \rangle \rightarrow \langle p, \sigma \rangle}{12, 3 \in \text{Num}, p = 12 \ominus 3} \quad a' \equiv p}$$

Questo conduce alla seguente sequenza di transizioni:

$$\langle (x - y) + 7, \sigma \rangle$$

$$(\text{plusL}, \text{minusL}, \text{Var}) \rightarrow \langle ((12 - y) + 7), \sigma \rangle$$

$$(\text{minusR}, \text{Var}) \rightarrow \langle ((12 - 3) + 7), \sigma \rangle$$

$$(\text{primMinus}) \rightarrow \langle (9 + 7), \sigma \rangle$$

$$AExp = Var \mid Num \mid (AExp + AExp) \mid (AExp - AExp)$$

$\text{var: } \frac{X \in \text{Var}}{\langle X, \sigma \rangle \rightarrow \langle \sigma(X), \sigma \rangle}$	
$\text{primPlus: } \frac{n, m \in \text{Num} \quad p = n \oplus m}{\langle (n + m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$	$\text{primMinus: } \frac{n, m \in \text{Num} \quad p = n \ominus m}{\langle (n - m), \sigma \rangle \rightarrow \langle p, \sigma \rangle}$
$\text{plusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a' + a_2), \sigma \rangle}$	$\text{plusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 + a_2), \sigma \rangle \rightarrow \langle (a_1 + a''), \sigma \rangle}$
$\text{minusL: } \frac{\langle a_1, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a' - a_2), \sigma \rangle}$	$\text{minusR: } \frac{\langle a_2, \sigma \rangle \rightarrow \langle a'', \sigma \rangle}{\langle (a_1 - a_2), \sigma \rangle \rightarrow \langle (a_1 - a''), \sigma \rangle}$

Step 1: primPlus

$$R(\text{primPlus}) \frac{\langle (9 + 7), \sigma \rangle \rightarrow \langle p, \sigma \rangle}{9, 7 \in \text{Num}, p = 9 \oplus 7}$$

Questo conduce alla seguente sequenza di transizioni:

$$\begin{aligned} & \langle ((x - y) + 7), \sigma \rangle \\ & (\text{plusL, minusL, Var}) \rightarrow \langle ((12 - y) + 7), \sigma \rangle \\ & (\text{minusR, Var}) \rightarrow \langle (((12 - 3) + 7), \sigma) \rangle \\ & (\text{primMinus}) \rightarrow \langle (9 + 7), \sigma \rangle \\ & (\text{primPlus}) \rightarrow \langle 16, \sigma \rangle \end{aligned}$$

$$\text{Com} ::= \text{skip} \mid \text{Var} := A\text{Exp} \mid \text{Com}; \text{Com} \mid$$
$$\text{if } B\text{Exp} \text{ then } \text{Com} \text{ else } \text{Com} \mid \text{while } B\text{Exp} \text{ do } \text{Com}$$

## Semantica dei Comandi Com.

$$\langle \text{skip}, \sigma \rangle \rightarrow \sigma \quad (c1)$$

$$\langle X := n, \sigma \rangle \rightarrow \sigma[X \leftarrow n] \quad (c2) \quad \frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle X := a, \sigma \rangle \rightarrow \langle X := a', \sigma \rangle} \quad (c3)$$

$$\frac{\langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle c_1; c_2, \sigma \rangle \rightarrow \langle c_2, \sigma' \rangle} \quad (c4) \quad \frac{\langle c_1, \sigma \rangle \rightarrow \langle c'_1, \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \rightarrow \langle c'_1; c_2, \sigma' \rangle} \quad (c5)$$

$$\langle \text{if tt then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle \quad (c6)$$

$$\langle \text{if ff then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_2, \sigma \rangle \quad (c7)$$

$$\frac{\langle b, \sigma \rangle \rightarrow \langle b', \sigma \rangle}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle \text{if } b' \text{ then } c_1 \text{ else } c_2, \sigma \rangle} \quad (c8)$$

$$\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \langle \text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else skip}, \sigma \rangle \quad (c9)$$

# SOS: Semantica dei Comandi

$$\langle \text{skip}, \sigma \rangle \rightarrow \sigma \quad (c1)$$

$$\langle X := n, \sigma \rangle \rightarrow \sigma[X \leftarrow n] \quad (c2) \quad \frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle X := a, \sigma \rangle \rightarrow \langle X := a', \sigma \rangle} \quad (c3)$$

$$\frac{\langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle c_1; c_2, \sigma \rangle \rightarrow \langle c_2, \sigma' \rangle} \quad (c4) \quad \frac{\langle c_1, \sigma \rangle \rightarrow \langle c'_1, \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \rightarrow \langle c'_1; c_2, \sigma' \rangle} \quad (c5)$$

$$\langle \text{if tt then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle \quad (c6)$$

$$\langle \text{if ff then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_2, \sigma \rangle \quad (c7)$$

$$\frac{\langle b, \sigma \rangle \rightarrow \langle b', \sigma \rangle}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle \text{if } b' \text{ then } c_1 \text{ else } c_2, \sigma \rangle} \quad (c8)$$

$$\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \langle \text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else skip}, \sigma \rangle \quad (c9)$$

Sia  $c$  la sequenza di comandi:  $X := 1; \text{while } \neg(X == 0) \text{ do } X := X - 1$

$$\begin{aligned} & \langle c, \sigma \rangle \\ & \rightarrow \langle c', \sigma[X \leftarrow 1] \rangle \\ & \rightarrow \langle \text{if } \neg(X == 0) \text{ then } X := (X - 1); c' \text{ else skip}, \sigma[X \leftarrow 1] \rangle \\ & \rightarrow \langle \text{if } \neg(1 == 0) \text{ then } X := (X - 1); c' \text{ else skip}, \sigma[X \leftarrow 1] \rangle \\ & \rightarrow \langle \text{if } \text{ff then } X := (X - 1); c' \text{ else skip}, \sigma[X \leftarrow 1] \rangle \\ & \rightarrow \langle \text{if tt then } X := (X - 1); c' \text{ else skip}, \sigma[X \leftarrow 1] \rangle \\ & \rightarrow \langle X := (X - 1); c', \sigma[X \leftarrow 1] \rangle \\ & \rightarrow \langle X := (1 - 1); c', \sigma[X \leftarrow 1] \rangle \\ & \rightarrow \langle X := 0; c', \sigma[X \leftarrow 1] \rangle \\ & \rightarrow \langle c', \sigma[X \leftarrow 0] \rangle \\ & \rightarrow \langle \text{if } \neg(X == 0) \text{ then } X := (X - 1); c' \text{ else skip}, \sigma[X \leftarrow 0] \rangle \\ & \rightarrow \langle \text{if } \neg(0 == 0) \text{ then } X := (X - 1); c' \text{ else skip}, \sigma[X \leftarrow 0] \rangle \\ & \rightarrow \langle \text{if } \text{tt then } X := (X - 1); c' \text{ else skip}, \sigma[X \leftarrow 0] \rangle \\ & \rightarrow \langle \text{if ff then } X := (X - 1); c' \text{ else skip}, \sigma[X \leftarrow 0] \rangle \\ & \rightarrow \langle \text{skip}, \sigma[X \leftarrow 0] \rangle \\ & \rightarrow \sigma[X \leftarrow 0] \end{aligned}$$

- 1 Significato, Uso, Caratteristica di semantica Denotazionale e semantica Operazionale dei L. di Programmazione
- 2 Quali sono le 3 strutture che devono essere definite per fornire la SOS di un L. di Programmazione?
- 3 Si modifichino le regole di inferenza delle espressioni aritmetiche, date a lezione, in modo da prescrivere una valutazione degli argomenti da sinistra a destra per somma e sottrazione.
- 4 Si indichino le regole di inferenza e si mostrino le transizioni ottenute applicando all'espressione  $(y-(7+2))$  la semantica SOS, data a lezione.
- 5 Si annoti, a sinistra di ogni freccia nella figura sotto (v. testo completo), le regole di inferenza (dei comandi c1-c9) utilizzate nella transizione, come abbiamo visto a lezione nel caso delle espressioni.
- 6 Siano c e d i seguenti comandi:  
c: `X:=1;`  
d: `while(X==1)do skip`  
Si dia la sequenza di transizioni ottenute per c;d a partire dallo stato  $\sigma = (X, 0)$ .

Altri esercizi in:

\* Esercizi4Lezione4Parte2

\* Cap. 2 [GM]