

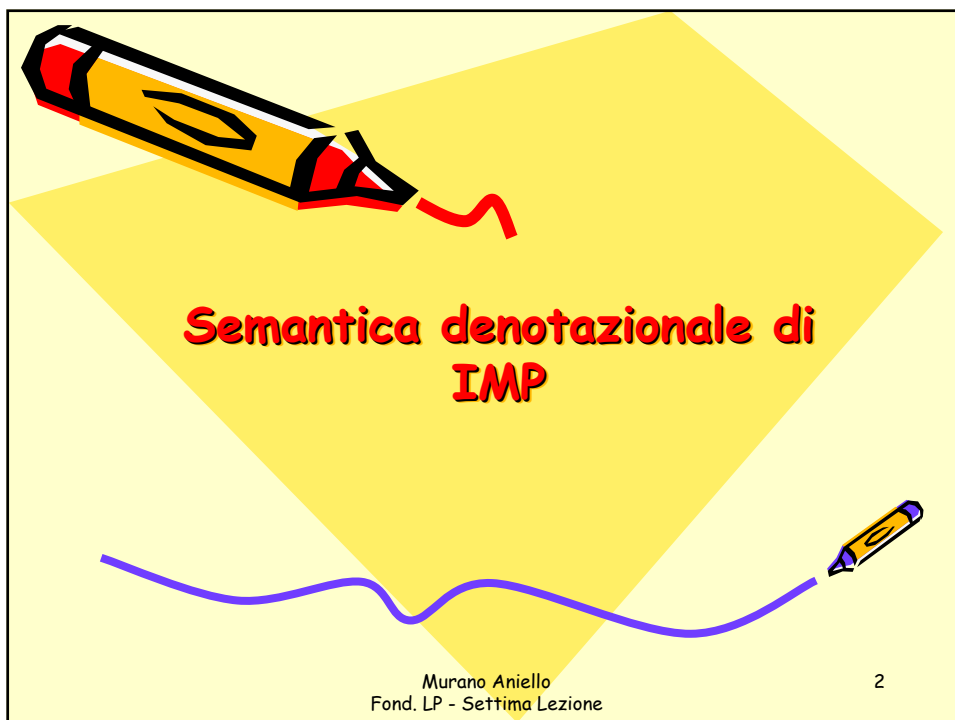


# Fondamenti dei linguaggi di programmazione

Aniello Murano  
Università degli Studi di Napoli  
"Federico II"

Murano Aniello  
Fond. LP - Settima Lezione

1



# Semantica denotazionale di IMP

Murano Aniello  
Fond. LP - Settima Lezione

2

## Introduzione alla VII lezione

- Nella prima parte del corso abbiamo introdotto il linguaggio imperativo IMP
- Per questo linguaggio, abbiamo introdotto una semantica operazionale e abbiamo mostrato che questa semantica si basa su regole di derivazione. In particolare:
  - È legata alla sintassi
  - Interpreta i programmi come relazioni di transizione.
- Sebbene **relativamente semplice** e molto efficiente per valutare certe proprietà di un linguaggio quali il **determinismo**, l'utilizzo di tali regole comporta alcune limitazioni:
  - Mostra soltanto come possono essere utilizzati gli elementi di un linguaggio, ma non il loro significato intrinseco.
  - La dipendenza dalla sintassi rende difficile il confronto di programmi scritti in linguaggi di programmazione differenti



Murano Aniello  
Fond. LP - Settima Lezione

3

## Esempio

- Supponiamo di voler valutare l'equivalenza della funzione "quadrato di un numero" espressa nelle seguenti notazioni

**Algol:**  
integer procedure square(x);  
integer x;  
begin square := x \* x end;

**StdC/C++/Java:**  
int square(int x)  
{ return (x \* x); }

**Teoria degli insiemi:**  
 $\{(x,y) \mid \forall x,y \in \mathcal{N} : y = x^2\}$

**Pascal:**  
function square (x:integer) :  
integer;  
begin square := x \* x end;

**ML97:**  
fun square x = x \* x;  
fun square (x:int) = x \* x;  
val square = fn x => x \* x;

**Algebra**  
 $f : \mathbb{N} \rightarrow \mathbb{N}; f(x) = x^2;$

- Utilizzando la sola semantica operazionale questa valutazione non è possibile



Murano Aniello  
Fond. LP - Settima Lezione

4

## Un po' di storia

- Negli anni sessanta, Christopher Strachey e Dana Scott riuscirono a superare le limitazioni della semantica operativa introducendo una semantica più astratta basata sull'utilizzo di **funzioni** semantiche (chiamate denotazioni) come modelli matematici di rappresentazione (del significato) dei programmi.
- L'idea di base utilizzata in questa semantica (denotazionale) può essere mostrata con un esempio:
- In IMP, due comandi  $c_0$  e  $c_1$  sono equivalenti ( $c_0 \sim c_1$ ) sse
$$\forall \sigma, \sigma' . \langle c_0, \sigma \rangle \rightarrow \sigma' \Leftrightarrow \langle c_1, \sigma \rangle \rightarrow \sigma'$$
- In modo equivalente, possiamo dire che  $c_0 \sim c_1$  sse
$$\{(\sigma, \sigma') \mid \langle c_0, \sigma \rangle \rightarrow \sigma'\} = \{(\sigma, \sigma') \mid \langle c_1, \sigma \rangle \rightarrow \sigma'\}$$
- Cioè sse  $c_0$  e  $c_1$  definiscono la stessa **funzione parziale** sugli stati. Dunque una valutazione di equivalenza tra linguaggi viene rimandata alla valutazione di equivalenza tra oggetti matematici



Murano Aniello  
Fond. LP - Settima Lezione

5

## Semantica denotazionale

- L'obiettivo della semantica denotazionale è quello di esprimere il significato di un programma in termini di funzioni semantiche (funzioni matematiche).
- In pratica, ad ogni frase del linguaggio viene associata una denotazione (significato) come funzione delle denotazioni delle sue sottofrasi.
- La semantica denotazionale cerca di catturare il significato interno di un programma piuttosto che la strategia di implementazione. Per questo motivo è più astratta di quella operativa ed è indipendente dalla macchina su cui si lavora.



Murano Aniello  
Fond. LP - Settima Lezione

6

## Semantica denotazionale di IMP

- In generale, per definire la *semantica* di un oggetto di un linguaggio definito da una certa *sintassi* utilizzeremo la notazione  $\llbracket \ ]$  nel seguente modo:

$\llbracket \langle \text{sintassi} \rangle \rrbracket =$  denotazione di  $\langle \text{sintassi} \rangle$

- dove le parentesi  $\llbracket \ ]$  sono chiamate parentesi semantiche e  $\llbracket \langle \text{sintassi} \rangle \rrbracket$  va letto come la "denotazione di  $\langle \text{sintassi} \rangle$ ".
- Sostanzialmente le parentesi  $\llbracket \ ]$  sono una funzione matematica.



Murano Aniello  
Fond. LP - Settima Lezione

7

## Semantica operativa di Aexp

- Ricordiamo la sintassi delle espressioni aritmetiche Aexp di IMP  
 $a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 * a_1$

- La semantica denotazionale per  $a \in \text{Aexp}$  è una funzione

$\mathcal{A}[\![a]\!] : \Sigma \rightarrow \mathcal{N}$  o equivalentemente  $\mathcal{A} : \text{Aexp} \rightarrow \Sigma \rightarrow \mathcal{N}$

- La denotazione di  $a$  (semantica denotazionale  $\mathcal{A}[\![a]\!]$  per  $a$ ) è una relazione tra stati e numeri ed è definita per induzione sulla struttura dell'espressione nel modo seguente:

- $\mathcal{A}[\![n]\!] = \{(\sigma, n) \mid \sigma \in \Sigma\}$        $\mathcal{A}[\![X]\!] = \{(\sigma, \sigma(X)) \mid \sigma \in \Sigma\}$
- $\mathcal{A}[\![a_0 + a_1]\!] = \{(\sigma, n_0 + n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\}$
- $\mathcal{A}[\![a_0 - a_1]\!] = \{(\sigma, n_0 - n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\}$
- $\mathcal{A}[\![a_0 * a_1]\!] = \{(\sigma, n_0 * n_1) \mid (\sigma, n_0) \in \mathcal{A}[\![a_0]\!] \ \& \ (\sigma, n_1) \in \mathcal{A}[\![a_1]\!]\}$

- Si noti che le funzioni " $\Sigma \rightarrow \mathcal{N}$ " sono rappresentate (in modo equivalente) in termini di insiemi di coppie "stato, numero" e che i simboli "+", "-", e "\*" nella parte sinistra delle uguaglianze sono simboli sintattici e nella parte destra sono operatori



Murano Aniello  
Fond. LP - Settima Lezione

8

## Esempi di valutazione di Aexp

- Per qualsiasi stato  $\sigma$ , la semantica denotazionale di "3+5" è

$$\mathcal{A}[[3 + 5]]\sigma = \mathcal{A}[[3]]\sigma + \mathcal{A}[[5]]\sigma = 3+5 = 8$$

- dove  $\mathcal{A}[[a]]\sigma$  semplifica  $\mathcal{A}[[a]](\sigma)$ .
- Sia  $\sigma$  lo stato in cui la locazione X vale "2". La semantica denotazionale di "X\*3" è

$$\mathcal{A}[[X*3]]\sigma = \mathcal{A}[[X]]\sigma * \mathcal{A}[[3]]\sigma = \sigma(X)*3 = 6$$

- E se avessimo avuto anche la divisione ( $a_0/a_1$ ) in Aexp, come può essere definita la semantica denotazionale per  $a_0/a_1$ ?
- Bisogna tener presente quando il denominatore è zero:
  - $\mathcal{A}[[a_0/a_1]]\sigma$  è indefinito se  $\mathcal{A}[[a_1]]\sigma = 0$ .
  - Altrimenti è il quoziente intero della divisione tra  $\mathcal{A}[[a_0]]\sigma$  e  $\mathcal{A}[[a_1]]\sigma$



Murano Aniello  
Fond. LP - Settima Lezione

9

## $\lambda$ -calculus

- Un modo alternativo di rappresentare funzioni è dato dalla notazione  $\lambda$
- In pratica, la notazione  $\lambda x.e$  denota la funzione che mappa  $x$  in  $e$ .
- Per esempio la funzione successione può esser scritta come

$$\lambda x.x+1$$

- Utilizzando questa notazione, è possibile riscrivere la semantica delle espressioni aritmetiche di IMP nel seguente modo
  - $\mathcal{A}[[n]] = \lambda \sigma.n$
  - $\mathcal{A}[[X]] = \lambda \sigma.\sigma(X)$
  - $\mathcal{A}[[a_0 + a_1]] = \{(\sigma, n) \mid \sigma \in \Sigma \wedge n = \mathcal{A}[[a_0]]\sigma + \mathcal{A}[[a_1]]\sigma\}$
  - $\mathcal{A}[[a_0 - a_1]] = \{(\sigma, n) \mid \sigma \in \Sigma \wedge n = \mathcal{A}[[a_0]]\sigma - \mathcal{A}[[a_1]]\sigma\}$
  - $\mathcal{A}[[a_0 * a_1]] = \{(\sigma, n) \mid \sigma \in \Sigma \wedge n = \mathcal{A}[[a_0]]\sigma * \mathcal{A}[[a_1]]\sigma\}$



Murano Aniello  
Fond. LP - Settima Lezione

10

## Intermezzo Background del $\lambda$ -calculus

- Developed in 1930's by Alonzo Church.
- Subsequently studied (and still studied) by many people in logic and computer science.
- Considered the "testbed" for procedural and functional languages.
  - Simple.
  - Powerful.
  - Easy to extend with features of interest.
- "Whatever the next 700 languages turn out to be, they will surely be variants of lambda calculus." (Landin '66)



Murano Aniello  
Fond. LP - Settima Lezione

11

## Intermezzo Alonzo Church

- PhD from Princeton University, 1927
  - Advisor was Oswald Veblen who also advised R. L. Moore of UT Austin
- Studied with David Hilbert, Paul Bernays & L.E.J. Brouwer in Germany
- Many of his PhD students are "founding fathers" of theoretical computer science
  - Stephen Kleene, 1934 (recursive function theory)
  - J. Barkley Rosser, 1934 (Church-Rosser theorem)
  - Alan Turing, 1938 (computational logic & computability)
  - Martin Davis, 1950 (logic & computability theory)
  - J. Hartley Rogers, 1952 (recursive function theory)
  - Michael Rabin, 1956 (probabilistic algorithms - Turing award)
  - Dana Scott, 1958 (prob. algorithms, domain theory - Turing award)
  - Raymond Smullyan, 1959 (logic, tableau proof, formal systems)



Murano Aniello  
Fond. LP - Settima Lezione

12

## Intermezzo

### The $\lambda$ -calculus in Computer Science

- A formal notation, theory, and model of computation
- Church's thesis:  $\lambda$ -calculus & Turing Machines describe the same set of objects, i.e., effectively computable functions
  - equivalence was proved by **Stephen Kleene**
- Foundation for the functional style of programming
  - Lisp, Scheme, ISWIM, ML, Miranda™, Haskell, ...
- Notation for Scott-Strachey denotational semantics.
- It can be used to abstractly represent numbers, booleans, predicates, functions, variables, block scopes, expressions, ordered pairs, lists, records & recursion, ecc.



Murano Aniello  
Fond. LP - Settima Lezione

13

### Denotazione di Bexp (1)

- Ricordiamo la sintassi delle espressioni booleane Bexp di IMP
$$b ::= \text{true} \mid \text{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \wedge b_1 \mid b_0 \vee b_1$$
- La semantica denotazionale per  $b \in \text{Bexp}$  è una funzione
$$\mathcal{B}[[b]] : \Sigma \rightarrow \{\text{true}, \text{false}\}$$
- La funzione semantica per le espressioni booleane è definita in termini delle operazioni logiche di congiunzione ( $\wedge$ ), disgiunzione ( $\vee$ ) e negazione ( $\neg$ )
- La denotazione di  $b \in \text{Bexp}$  (semantica denotazionale  $\mathcal{A}[[b]$  per  $b$ ) è una relazione tra stati e valori di verità, definita per induzione sulla struttura dell'espressione booleane nel modo seguente:



Murano Aniello  
Fond. LP - Settima Lezione

14

## Denotazione di Bexp (2)

- $\mathcal{B}[\text{true}] = \{(\sigma, \text{true}) \mid \sigma \in \Sigma\}$
- $\mathcal{B}[\text{false}] = \{(\sigma, \text{false}) \mid \sigma \in \Sigma\}$
- $\mathcal{B}[a_0 = a_1] = \{(\sigma, \text{true}) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[a_0]\sigma = \mathcal{A}[a_1]\sigma\} \cup \{(\sigma, \text{false}) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[a_0]\sigma \neq \mathcal{A}[a_1]\sigma\}$
- $\mathcal{B}[a_0 \leq a_1] = \{(\sigma, \text{true}) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[a_0]\sigma \leq \mathcal{A}[a_1]\sigma\} \cup \{(\sigma, \text{false}) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[a_0]\sigma \not\leq \mathcal{A}[a_1]\sigma\}$
- $\mathcal{B}[\neg b] = \{(\sigma, \neg t) \mid \sigma \in \Sigma \ \& \ (\sigma, t) \in \mathcal{B}[b]\}$
- $\mathcal{B}[b_0 \wedge b_1] = \{(\sigma, t_0 \wedge t_1) \mid \sigma \in \Sigma \ \& \ (\sigma, t_0) \in \mathcal{B}[b_0] \ \& \ (\sigma, t_1) \in \mathcal{B}[b_1]\}$
- $\mathcal{B}[b_0 \vee b_1] = \{(\sigma, t_0 \vee t_1) \mid \sigma \in \Sigma \ \& \ (\sigma, t_0) \in \mathcal{B}[b_0] \ \& \ (\sigma, t_1) \in \mathcal{B}[b_1]\}$
- Si noti che i simboli "true", "false", "¬", "∧" e "∨" nella parte sinistra delle uguaglianze sono simboli sintattici mentre nella parte destra sono operatori. Inoltre, le funzioni " $\Sigma \rightarrow \{\text{true}, \text{false}\}$ " sono state indicate come insiemi di coppie "stato, valore di verità"



Murano Aniello  
Fond. LP - Settima Lezione

15

## Esempi di valutazione di Bexp

- Per ogni stato  $\sigma$ , la semantica denotazionale di "true  $\vee$  false" è  
 $\mathcal{B}[\text{true} \vee \text{false}]\sigma = \mathcal{B}[\text{true}]\sigma \vee \mathcal{B}[\text{false}]\sigma = \text{true} \vee \text{false} = \text{true}$
- dove  $\mathcal{B}[b]\sigma$  semplifica  $\mathcal{B}[b](\sigma)$ .
- Sia  $\sigma$  lo stato in cui la locazione X vale "2" e Y vale "3". La semantica denotazionale di "X+3 = Y+2" è  
 $\mathcal{B}[X+3 = Y+2]\sigma = \text{true}$  perché  $\mathcal{A}[X+3]\sigma = 5 = \mathcal{A}[Y+2]\sigma$



Murano Aniello  
Fond. LP - Settima Lezione

16

## Denotazione di Com (1)

- Ricordiamo la sintassi dei comandi Com di IMP
$$c ::= \text{skip} \mid X:=a \mid c_0;c_1 \mid \text{if } b \text{ then } c_0 \text{ else } c_1 \mid \text{while } b \text{ do } c$$
- La semantica denotazionale per  $c \in \text{Com}$  è una funzione
$$\mathcal{C}[[c]] : \Sigma \rightarrow \Sigma$$
- La denotazione di  $c \in \text{Com}$  (semantica denotazionale  $\mathcal{C}[[c]]$  per  $c$ ) è una relazione tra stati, definita per induzione sulla struttura dei comandi.
- Si noti che la funzione di valutazione è parziale  $\mathcal{C}[[c]] : \Sigma \rightarrow \Sigma$  in quanto su alcuni comandi la funzione può non essere definita (per esempio sui loop infiniti)



## Denotazione di Com (2)

- $\mathcal{C}[[\text{skip}]] = \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$
- $\mathcal{C}[[X:=a]] = \{(\sigma, \sigma[n/X]) \mid \sigma \in \Sigma \ \& \ \mathcal{A}[[a_0]]\sigma = n\}$
- $\mathcal{C}[[c_0;c_1]] = \mathcal{C}[[c_1]] \circ \mathcal{C}[[c_0]]$  (si noti l'inversione in accordo alla regola di composizione)
- $\mathcal{C}[[\text{if } b \text{ then } c_0 \text{ else } c_1]] = \{(\sigma, \sigma') \mid \mathcal{B}[[b]]\sigma = \text{true} \ \& \ (\sigma, \sigma') \in \mathcal{C}[[c_0]]\} \cup \{(\sigma, \sigma') \mid \mathcal{B}[[b]]\sigma = \text{false} \ \& \ (\sigma, \sigma') \in \mathcal{C}[[c_1]]\}$
- Particolarmente difficile è invece la valutazione del comando while per il quale è necessario utilizzare i concetti matematici del punto fisso introdotti nella lezione precedente.



## Denotazione del while

- Dalla semantica operativa (II lezione) abbiamo osservato che esiste la seguente equivalenza:
- Sia  $w \equiv \text{while } b \text{ do } c$  allora
$$w \sim \text{if } b \text{ then } c; w \text{ else skip}$$
- Possiamo allora scrivere la semantica del comando while utilizzando le regole precedenti nel seguente modo:
- $\mathcal{C}[w] = \mathcal{C}[\text{if } b \text{ then } c; w \text{ else skip}]$ 
$$= \{(\sigma, \sigma') \mid \mathcal{B}[b]\sigma = \text{true} \ \& \ (\sigma, \sigma') \in \mathcal{C}[w] \circ \mathcal{C}[c]\} \cup \{(\sigma, \sigma) \mid \mathcal{B}[b]\sigma = \text{false}\}$$
- Come si vede, il termine  $w$  compare in entrambi i lati dell'uguaglianza e dunque  $\mathcal{C}[w]$  non ha una soluzione immediata.
- Risolvere questa funzione equivale a calcolare un punto fisso.



## Idea di valutazione con fixpoint

- Si consideri la concatenazione del comando  $X:=0$  e del comando  $w = \text{while } X \leq 10 \text{ do } X:=X+1$
- La denotazione di  $x:=0; w$  è  $\mathcal{C}[x:=0; w]\sigma = \mathcal{C}[w]\sigma \circ \mathcal{C}[x:=0]\sigma$ .
- Sappiamo che  $\mathcal{C}[x:=0]\sigma = \{(\sigma, \sigma[0/X])\}$
- Per  $\mathcal{C}[w]\sigma$  occorrono 12 valutazioni di  $X \leq 10$  e 11 esecuzioni di  $X:=X+1$ .
- Dopo la prima valutazione di  $X \leq 10$  (ed esecuzione di  $X:=X+1$ ), la denotazione di  $w$  è quella di  $w$  valutato solo 10 volte, in combinazione con  $\mathcal{C}[X:=X+1]\sigma = \{(\sigma, \sigma[\sigma(X)+1/X])\}$ . Questo combinato con  $\mathcal{C}[x:=0]\sigma = \{(\sigma, \sigma[0/X])\}$  restituisce la denotazione di  $\mathcal{C}[x:=0; w]\sigma$
- Iterando,  $\mathcal{C}[x:=0; w]\sigma$  è anche equivalente alla denotazione del while valutato 9 volte combinato alla funzione  $\{(\sigma, \sigma[\sigma(X)+1/X])\}$ , combinato a  $\{(\sigma, \sigma[\sigma(X)+1/X])\}$  e infine combinato a  $\{(\sigma, \sigma[0/X])\}$ .
- Questo termina quando non dobbiamo più valutare  $X \leq 10$ , o meglio, quando ulteriori valutazioni non cambiano il risultato.
- Dunque,  $\mathcal{C}[x:=0; w]\sigma = \{(\sigma, \sigma[11/X])\}$ . Generalizziamo questa idea.



## Valutazione di while con fixpoint(1)

- Nella lezione precedente abbiamo parlato di ordinamento parziale tra funzioni:
- date due funzioni parziali  $I, J : \Sigma \rightarrow \Sigma$ , con  $I \leq J$  indichiamo che  $J$  raffina  $I$  o che  $J$  estende  $I$
- Tornando alla funzione parziale di while, si potrebbe pensare che ad ogni iterazione di un while si può raffinare la conoscenza della sua valutazione
- Caso base: prima che  $b$  sia valutata, la conoscenza su  $\mathcal{C}[[w]]$  è
- $\mathcal{C}_0[[w]] := \perp : \Sigma \rightarrow \Sigma$ , che denota la funzione non definita in nessun stato. Quindi  $\mathcal{C}_0[[w]]$  corrisponde a nessuna informazione.



## Valutazione di while con fixpoint(2)

- Si supponga adesso di valutare  $b$  "false" in uno stato  $\sigma$ . Dunque la denotazione del while ritorna  $\{(\sigma, \sigma)\}$ ,
- Questo ci permette di raffinare la nostra conoscenza del while da  $\mathcal{C}_0[[w]]$  ad una nuova funzione parziale  $\mathcal{C}_1[[w]] : \Sigma \rightarrow \Sigma$ , che è una funzione identità sugli stati  $\sigma$  in cui  $b$  è valutata "false"
- Se invece  $b$  è valutato "true" in  $\sigma$  e il comando  $c$  viene valutato  $\{(\sigma, \sigma')\}$ , e  $b$  valutato "false" in  $\sigma'$ , cioè il while termina in una iterazione, allora possiamo raffinare  $\mathcal{C}_1[[w]]$  con  $\mathcal{C}_2[[w]] : \Sigma \rightarrow \Sigma$ , che oltre ad essere una funzione identità dove  $\mathcal{C}_1[[w]]$  è definita, associa a  $\mathcal{C}_2[[w]]\sigma$  la funzione  $\{(\sigma, \sigma')\}$ . Chiaramente  $\mathcal{C}_1[[w]] \leq \mathcal{C}_2[[w]]$ .
- Iterando il processo, per ogni  $k$  si può definire una funzione parziale  $\mathcal{C}_k[[w]] : \Sigma \rightarrow \Sigma$ , definita su tutti gli stati su cui il while termina in al più  $k$  valutazioni di  $b$  (cioè  $k - 1$  esecuzioni di  $c$ ).
- Dunque abbiamo una  $\omega$ -catena  $\mathcal{C}_0[[w]] \leq \mathcal{C}_1[[w]] \leq \dots \leq \mathcal{C}_k[[w]] \leq \dots$  e un ordine parziale completo con  $\perp = \mathcal{C}_0[[w]]$



## Valutazione di while con fixpoint(3)

- Sia  $f$  una funzione totale  $f: (\Sigma \rightarrow \Sigma) \rightarrow (\Sigma \rightarrow \Sigma)$  tale che :

$$f(\alpha) = \begin{cases} (\sigma, \sigma) & \text{if } \mathcal{B}[[b]]\sigma = \text{false} \\ (\sigma, \alpha(\mathcal{C}[[c]]\sigma)) & \text{if } \mathcal{B}[[b]]\sigma = \text{true} \end{cases}$$

- In pratica,  $f$  è definita dalle seguenti regole:

$$\begin{cases} \emptyset \rightarrow (\sigma, \sigma) & \text{if } \mathcal{B}[[b]]\sigma = \text{false} \\ (\sigma'', \sigma') \rightarrow (\sigma, \sigma') & \text{if } \mathcal{B}[[b]]\sigma = \text{true} \ \& \ \mathcal{C}[[c]]\sigma = \sigma'' \end{cases}$$

- Siano  $f^0(\perp) = \perp = \mathcal{C}_0[[w]]$  e  $f^1(\perp) = f(f^0(\perp)) = \mathcal{C}_1[[w]]$ , allora possiamo definire induttivamente
- $f^k(\perp) = f(f^{k-1}(\perp)) = \mathcal{C}_k[[w]]$  come il risultato di  $k$  composizioni di  $f$ .
- La funzione  $f$  è continua e per quanto detto nella lezione precedente, il fixpoint di  $f$  (che è anche il fixpoint di  $\mathcal{C}[[w]]$ ) è il "least upper bound" della catena.



## Osservazioni sulla definizione di $f$

- Osserviamo ancora come la definizione di  $f$  tramite le regole

$$\begin{cases} \emptyset \rightarrow (\sigma, \sigma) & \text{if } \mathcal{B}[[b]]\sigma = \text{false} \\ (\sigma'', \sigma') \rightarrow (\sigma, \sigma') & \text{if } \mathcal{B}[[b]]\sigma = \text{true} \ \& \ \mathcal{C}[[c]]\sigma = \sigma'' \end{cases}$$

permette di valutare ricorsivamente il comando iterativo  $w$  in  $\sigma$ .

- Consideriamo i seguenti scenari di denotazione di  $b$ :
- $b$  denotata false in  $\sigma$ , allora la denotazione di  $w$  è  $(\sigma, \sigma)$
- $b$  vale true in  $\sigma$ , e false in  $\sigma''$ . Siccome al secondo ciclo si utilizza la prima regola di  $f$ , risulta  $(\sigma'', \sigma') = (\mathcal{C}[[c]]\sigma, \mathcal{C}[[c]]\sigma)$ . Dunque la denotazione di  $w$  è  $(\sigma, \mathcal{C}[[c]]\sigma)$ , e corrisponde ad  $f$  applicata ad  $f$  al passo ricorsivo precedente cioè su:  $b$  valutata false in  $\mathcal{C}[[c]]\sigma''$ .
- $b$  vale  $k$  true partendo da  $\sigma$ , e false dopo  $k$  volte. Allora si applica  $k$  volte la seconda regola di  $f$  e poi una sola volta la prima. Dunque la valutazione di  $w$  in questo caso è data dalla funzione  $f$  applicata su  $f$  al passo ricorsivo precedente cioè su:  $b$  valutata  $k-1$  true partendo da  $\sigma$ , e poi false dopo  $k-1$  volte.



## Semantica Assiomatica

- Per motivi di tempo, in questo corso si è deciso di non trattare la semantica assiomatica
- Axiomatic Semantics is an approach based on mathematical logic to prove the correctness of computer programs
- An axiomatic semantics consists of:
  - A language for making assertions about programs.
  - Rules for establishing the assertions.
- Some typical kinds of assertions:
  - This program terminates.
  - The variables  $x$  and  $y$  have the same value throughout the execution of the program whenever  $z$  is 0.
  - All array accesses are within array bounds.
- Some typical languages of assertions:
  - First-order logic.
  - Other logics (e.g., temporal logic).



Murano Aniello  
Fond. LP - Settima Lezione

25

## Applications of axiomatic semantics

- Some application:
  - Guidance in design and coding.
  - Proving the correctness of algorithms and hardware description (or finding bugs).
  - Documentation of programs and interfaces.
- The project of defining and proving everything formally has not succeeded (at least not yet).
- Dijkstra said: " Program testing can be used to show the presence of bugs, but never to show their absence ".
- Hoare said: "Thus the practice of proving programs would seem to lead to solution of three of the most pressing problems in software and programming, namely, **reliability**, **documentation**, and **compatibility**. However, program proving, certainly at present, will be difficult even for programmers of high caliber; and may be applicable only to quite simple program designs".



Murano Aniello  
Fond. LP - Settima Lezione

26