

SEGMENTICA DI P_0

DEF. 3.2

Un assegnamento proposizionale B è una funzione

$$B(\alpha) = 1$$

$$\bar{B}(\alpha) = \beta(\alpha) = 1$$

$$B : \begin{matrix} \text{VARIABILI} \\ \uparrow \\ \text{PROPOSIZIONI} \\ \downarrow \end{matrix} \rightarrow \{0, 1\}$$

p, q, r, \dots

DEF. 3.3

Un assegnamento proposizionale B è esteso per induzione ad una valutazione \bar{B} del linguaggio P_0 nel modo seguente:

- $\bar{B}(p) = B(p) \quad \forall p$
- $\bar{B}(\alpha \rightarrow \beta) = \begin{cases} 0 & \text{se e solo se } \bar{B}(\alpha) = 1 \text{ e } \bar{B}(\beta) = 0 \\ 1 & \text{altrimenti} \end{cases} \quad (\bar{B}(\beta) = 1)$
- $\bar{B}(\neg \alpha) = 1 - \bar{B}(\alpha)$

DEF. 3.4

Una fbf α di P_0 è detta essere:

TAUTOLOGIA $\forall \beta \Rightarrow \bar{B}(\alpha) = 1$

SODDISFACIBILE $\exists \beta \mid \bar{B}(\alpha) = 1$

CONTRODITTORIA $\exists \beta \mid \bar{B}(\beta) = 1 \quad \forall \beta \in \Gamma$

PROPOSIZIONE 3.13

Una fbf α di P_0 è una tautologia se e solo se $\neg \alpha$ è un'antidittoria

DIM
Se utilizziamo la definizione 3.3, notiamo che $\bar{B}(\neg \alpha) = 1 - \underbrace{\bar{B}(\alpha)}_{=1} = 0$

PROPOSIZIONE 3.14

Se α e β sono tautologie $\Rightarrow \beta$ è una tautologia

$\bar{B}(\alpha) = 1 \quad \bar{B}(\alpha \rightarrow \beta) = 1 \Rightarrow \bar{B}(\beta) = 1$ NON PUÒ ESSERE CHE $\bar{B}(\beta) = 0$
perché senon $\bar{B}(\alpha \rightarrow \beta) = 0$

LEMMA 3.1

DATA una fbf α di P_0 , $\bar{B}(\alpha)$ dipende solo del valore assegnato
di B alle variabili proposizionali presenti in α . (perché B è
una funzione!)

TEOREMA 3.4

Date una fbf α di P_0 , è decidibile se α è una tautologia
 \Rightarrow

DM.

Lo potrete verificare costruendo le tabelle di verità

DEF. 3.5 (CONSEGUENZA TAUTLOGICA)

Dato una fbf α di P_0 e un insieme Γ di fbf di P_0 , si dice che
 α è conseguenza tautologica di Γ se è solo se per ogni assegnamento

proposizionale B si ha che :

Se per ogni fbf $\beta \in \Gamma$ vale $\bar{B}(\beta) = 1$ allora $\bar{B}(\alpha) = 1$

$\forall \beta \in \Gamma \mid \bar{B}(\beta) = 1 \Rightarrow \bar{B}(\alpha) = 1$ e scriveremo

$\Gamma \models \alpha$

TEOREMA DI CORRETEZZA DI P_0 (teorema 3.5)

Siano Γ un insieme di fbf di P_0 e α una fbf di P_0 .

Se $\Gamma \vdash_{P_0} \alpha$ allora $\Gamma \models \alpha$

"Se Γ deduce α allora α è una conseguenza tautologica"

COLLARIO 3.3

P_0 è consistente

PROPOSIZIONE 3.15

Se Γ un insieme di Fbf di P_0 .

Se Γ è soddisfacibile allora Γ è consistente

Lemme 3.3 (SODDISFACIBILITÀ)

Γ soddisfacibile \Leftrightarrow Γ consistente

Dato Γ un insieme di fbf di P_0

Se Γ è consistente allora Γ è soddisfacibile

TEOREMA 3.6 (DI COMPLETEZZA)

Se Γ un insieme di fbf di P_0 e α una fbf di P_0
da $\Gamma \models \alpha$ segue $\Gamma \vdash_{P_0} \alpha$

VERSO OPPOSTO DEL TEOREMA DI CORRETTEZZA

QUINDI TEOREMA 3.5 + TEOREMA 3.6

=

Teorema di Correttezza e Completezza

COROLLAARIO 3.5

Dato un fbf α di P_0 , è decidibile il problema di dire se

α è un teorema di P_0 (PROBLEMA DI DECISIONE)

Questo è riducibile al problema di dire se α è una tautologia (che è a sua volta decidibile)

INTRODUZIONE DI MODELLI COMPUTAZIONALI E NOZIONI FONDAMENTALI DELLA PROGRAMMAZIONE FUNZIONALE.

COSA SIGNIFICA COMPUTARE?

TRASFORMARE UN'INFORMAZIONE DA UNA FORMA IMPLICITA DA
UNA FORMA ESPlicita

$$(3+4)*2 \Rightarrow 14$$

Un modello computazionale è una specifica formalizzazione metamathematica
del concetto di "computazione" e delle nozioni ad esso collegate
come per esempio informazioni, trasformazioni ec... .

I linguaggi di programmazione sono ispirati direttamente o indirettamente
ad uno o più modelli computazionali

LINGUAGGI DI PROGRAMMAZIONE IMPERATIVI (C, Pascal, Fortran ...)

Sono quelli basati su modelli come MT e le URM.

URM = Unlimited Register Machine è un modello computazionale basato su un numero illimitato di registri numerici, ognuno dei quali può contenere un numero naturale.

Questo tipo di modello utilizza istruzioni semplici (incremento, decremento...) per manipolare i registri e definire algoritmi

L'URM è equivalente alle macchine di Turing in termini di potenza computazionale e viene usato per lo studio delle complessità e delle computabilità,

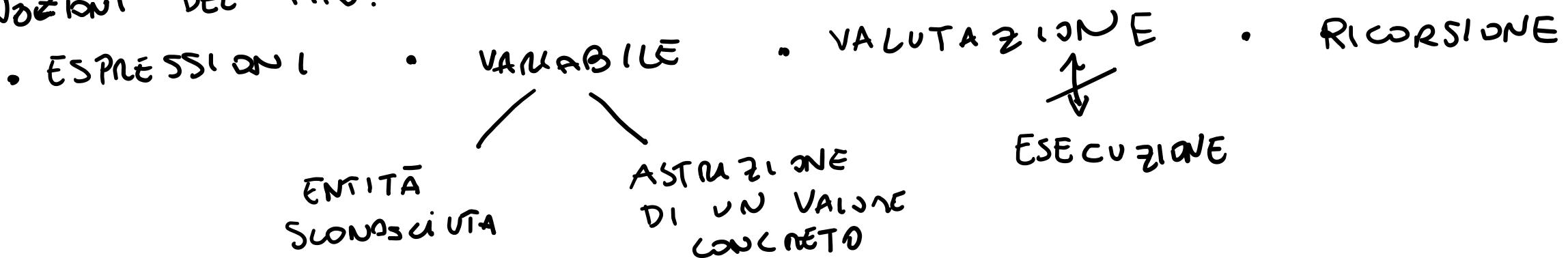
- assegnamento
- istruzione
- iterazione

LE NOZIONI FONDAMENTALI PI TALI LINGUAGGI SONO:

- MEMORIA - VARIABILE (qualcosa che si può leggere o modificare)

LINGUAGGI DI PROGRAMMAZIONE FUNZIONALI (come Haskell, Scheme, Erlang...)
TAI LINGUAGGI fanno riferimento al modello computazione
del λ -calcolo.

NOZIONI DEL TIPO:



LINGUAGGI DI PROGRAMMAZIONE LOGICI (come Prolog)

Questi si basano su alcune proprietà di sottosistemi della logica del primo ordine.

Il concetto di computazione coincide con il concetto di ricerca di una deduzione per me formula logica

LA PROGRAMMAZIONE FUNZIONALE

DEF. 1.GI

LA FUNZIONE CHE RESTITUISCE L'INVERSA DI UNA STRINGA $x \in \Sigma^*$ è definita come

$$\tilde{x} = \begin{cases} \epsilon & \text{se } x = \epsilon \\ \tilde{y}a & \text{se } x = ay \text{ con } a \in \Sigma \text{ e } y \in \Sigma^* \end{cases}$$

Questa definizione, oltre ad identificare univocamente una funzione sulle stringhe dell'alfabeto Σ , descrive implicitamente un metodo per calcolarla.

$$\tilde{cbd} = \tilde{bdc} = \tilde{dbc} = \epsilon dbc = dbc$$

$$\text{inv}[] = []$$

$$\text{inv}(a:y) = (\text{inv } y) ++ [a]$$

Quando chiediamo al linguaggio Haskell di calcolare l'inversa delle stringhe cbd

> `inv "cbd"`
> `"dbc"` (risposta)

sistemi di equazioni differenziali

$$f: \mathbb{R} \rightarrow \mathbb{R} \quad g: \mathbb{R} \rightarrow \mathbb{R}$$

$$f'' - 6g' = 6 \sin(x)$$

$$6g'' + 0.2f' = 6 \cos(x)$$

$$f(0) = 0, \quad f'(0) = 0, \quad g(0) = 1, \quad g'(0) = 1$$

ESEMPPIO PER CHE NON
TUTTE LE DEFINIZIONI DI
FUNZIONI DESCRIVANO
ANCHE UN PROCEDIMENTO
PER IL LORO CALCOLO

UNA VERSIONE EQUIVALENTE DI inv del linguaggio Haskell:

inv = \y → if (null y) then y
else (inv (tail y)) ++ [head y]

null : restituisce True se l'argomento è lista vuota, False altrimenti

tail : prende come argomento e restituisce una lista upvale, ma senza il primo elemento

head: prende una lista come argomento e restituisce il suo primo elemento

++: prende due liste " " " " " la loro concatenazione

if...then ... else espressioni condizionali

\y → questa è una funzione che, se prende in input un argomento y, restituisce...

inv = "Inv è il nome di..."

square $x = x^*x$

FORMA COMPATTA PER DEFINIRE IL QUADRATO DI x
In LINGUAGGIO HASKELL

square $\backslash x \rightarrow x^*x$

DEFINIZIONE IL FATTOORIALE:

$$\text{fact } 0 = 1$$

$$\text{fact } n = n * (\text{fact } (n-1))$$

FORMA COMPATTA

$\text{fact} = \backslash n \rightarrow \text{if } (n == 0) \text{ then } 1 \text{ else } n * (\text{fact } (n-1))$

ANALIZZIAMO COME DEFINIRE UNA FUNZIONE IN UN LINGUAGGIO
DI PROGRAMMAZIONE FUNZIONALE:

1) COSTRUIRE UN'ESPRESSIONE MATEMATICA:

variabili, costanti, funz. predefinite, e inserire nomi di funzioni
già pre-esistenti

=> APPLICAZIONE FUNZIONALE

2) COSTRUIRE UNA FUNZIONE ANONIMA

DEFINIRE TALE ESPRESSIONE CHIAMATA CORPO DELLA FUNZIONE.
Identifichiamo una variabile (che chiamiamo parametro formale delle funzioni)
e utilizziamo l'operatore funzionale

OPERATORI DI

ASSEGNAZIONE FUNZIONALE

\ →

3) ASSOCIARE le funzioni anonymous ad un nome utilizzando l'operatore =

COMPUTAZIONI NEI LINGUAGGI FUNZIONALI

> inv "cbd"

Indichiamo con $\rightsquigarrow A$ $\rightsquigarrow B$ $\rightsquigarrow C$ differenti possi di computazione

• 1^o PASSO COMPUTAZIONALE
 $\rightsquigarrow A$ sostituire UN NOME CON L'ESPRESSIONE ASSOCIAТА AD ESSO

• $\rightsquigarrow B$ SOSTITUIRE IL PARAMETRO ATTUALE AL POSTO DEL PARAMETRO FORMALE

• $\rightsquigarrow C$ CALCOLA DI UNA FUNZIONE PREDEFINITA + UNO ARGOMENTO
APPLICATO UN VALORE ESPLICITO

$$f(x) = x + 3$$

$$f(x) = x + 3 \quad //$$

$$f(1) = 1 + 3 = 4$$

" f_e "

parametro attuale delle funzione f

ogni passo computazionale \rightarrow TRASFORMARE UNA SOTTOESPRESSIONE

P.8

ESERCIZIO VEDERE ORE VENGONO CONNETTATI $\approx A$, $\approx B$, $\approx C$, possi
comparazioni si in un linguaggio di
Haskell per determinare
l'inverso di me stige-

LINGUAGGIO IMPERATIVO : SCHEMA

BOOLEAN even := TRUE; //variabile globale

:

PROCEDURE calculate (INTEGER value) : INTEGER;

BEGIN

even := NOT even; //variabile globale dell'ambiente

IF even

THEN RETURN (value + 1)

ELSE RETURN (value + 2)

END-ENDIF

: print (calculate(6));

: print (calculate(6));

il valore di "calculate"
dipende del numero di
volte che viene eseguito
il codice

SIDE EFFECT : che è proprio dei linguaggi imperativi, e non dei linguaggi funzionali

- LE DUE PRINT SONO IDENTICHE

$$7 = (6+1)$$

- POSSONO PRODURRE RISULTATI DIVERSI

$$8 = (6+2)$$

e questo

non dipende dell'argomento che do alle funzioni

Questo valore dipende delle variabili even che era una
variabile globale, la cui modifica rappresenta un SIDE
EFFECT!

Invece nella programmazione funzionale, poiché le funzioni definite sono delle
vere e proprie funzioni matematiche, il valore di me sottospressione
è sempre lo stesso, indipendentemente da dove si trova e da quante
volte viene valutata. Queste proprietà si chiamano referential transparency.