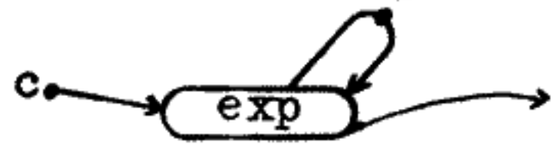
X

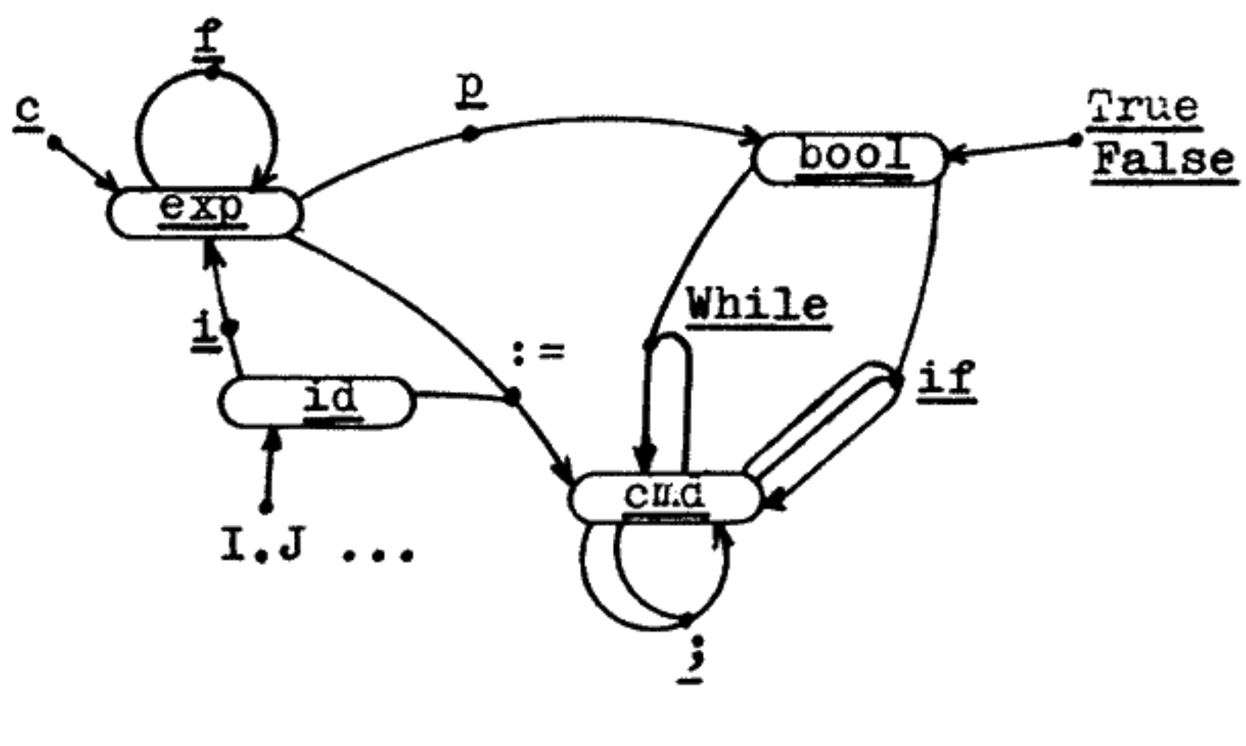
SEMANTICA DI LINGUAGGI

64.- Semantica di un semplice linguaggio imperativo

Consideriamo un tipo di dati $\underline{\underline{A}} = (A, f^A, c^A)$ con la seguente segnatura



Possiamo costruire su $\underline{\underline{A}}$ il linguaggio $\underline{\underline{A}}$-While avente sintassi astratta espresa dalla seguente segnatura di evidente lettura



(i rappresenta l'immersione di identificatori in espressioni che per brevità nel seguito non indicheremo)

Indichiamo con **EXP , ID , CMD** gli insiemi di termini di tale segnatura di sorte exp, id, cmd rispettivamente e definiamo il seguente insieme

$$\mathbf{STA} = \{\rho \mid \rho : \mathbf{ID} \longrightarrow \mathbf{A}\}$$

consideriamo le seguenti funzioni:

— aggiornamento $-[-/-] : \mathbf{STA} \times \mathbf{A} \times \mathbf{ID} \longrightarrow \mathbf{STA}$

ove $(\rho[a/I])I = a$

e $(\rho[a/I])J = \rho J \quad \forall\, J \neq I$

— condizionale $-\supset-,-$: $\{True, False\} \times STA \times STA \rightarrow STA$

ove $True \supset \rho, \rho' = \rho$

$False \supset \rho, \rho' = \rho'$

È del tutto naturale postulare delle funzioni di interpretazione $\mathcal{E}$, $\mathcal{B}$, $\mathcal{C}$ tali che:

$\mathcal{E} : EXP \times STA \rightarrow A \cup \{undefined\}$

$\mathcal{B} : BOOL \times STA \rightarrow \{True, False, undefined\}$

$\mathcal{C} : CMD \times STA \rightarrow STA \cup \{undefined\}$

per cui valgano $\forall \rho \in STA$ le seguenti condizioni ove I, J, E, H, C, C_1, C_2 sono elementi generici di ID , EXP, BOOL, CMD rispettivamente ($\mathcal{C}(\underline{while}\ \underline{True}\ \underline{do}\ C)_\rho$ =undefined)

$\mathcal{E}(\underline{c})\rho = c^A$

$\mathcal{E}(I)\rho = \rho I$

$\mathcal{E}(\underline{f}E)\rho = f^A(\mathcal{E}(E)\rho)$

$\mathcal{B}(\underline{True})\rho = True$

$\mathcal{B}(\underline{False})\rho = False$

$\mathcal{B}(\underline{p}E)_\rho = p^A(\mathcal{E}(E)\rho)$

$\mathcal{C}(C_1;C_2)\rho = \mathcal{C}(C_2)\ (\mathcal{C}(C_1)\rho)$

$\mathcal{C}(I:= E)\rho = \rho\,[\mathcal{E}(E)\rho/I]$

$\mathcal{C}(\underline{if}\ H\ \underline{then}\ C_1\ \underline{else}\ C_2)\rho = \mathcal{B}(H)\rho \supset \mathcal{C}(C_1)\rho, \mathcal{C}(C_2)\rho$

$\mathcal{C}(\underline{While}\ H\ \underline{do}\ C)\rho = \mathcal{B}(H)\rho \supset \mathcal{C}(\underline{While}\ H\ \underline{do}\ C)(\mathcal{C}(C)\rho), \rho$

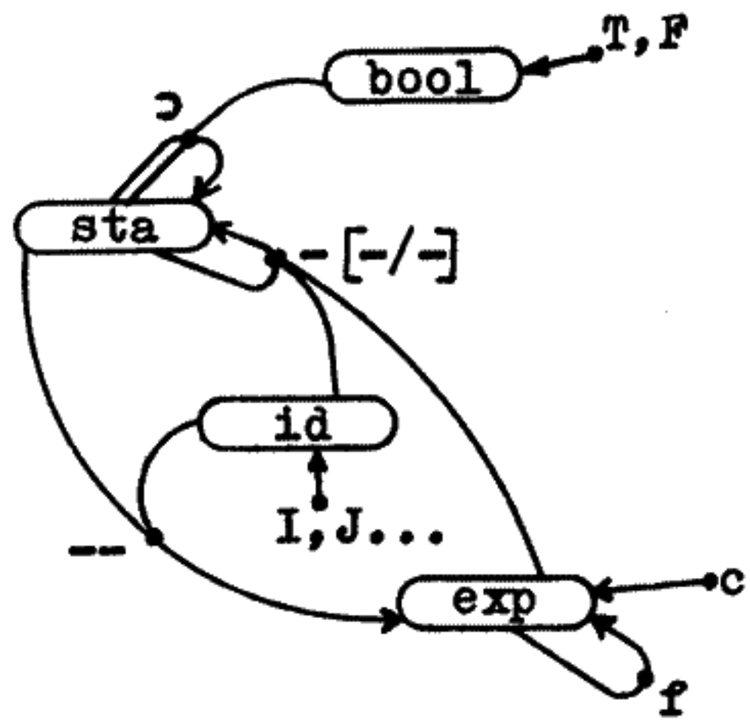
Anche se il significato di tali equazioni è del tutto intuitivo esse non possono essere considerate una definizione di $\mathcal{E}$ e $\mathcal{C}$ e questo per il fatto che

tali condizioni sono ricorsive in senso stretto.

Basta considerare equazioni del tipo $fx = fx$ oppure $fx = fx + 1$ ($x \in N$, $f: N \longrightarrow N$) per capire come una equazione ricorsiva può avere infinite o nessuna soluzione e in quanto tale non è assumibile come condizione definitoria. Un modo per superare tale difficoltà è quella di leggere tali condizioni in opportune strutture entro cui è possibile dare loro un carattere definitorio. È questa la soluzione adottata in **semantica denotazionale** tramite la **teoria del punto fisso** (cfr.[10]).

Qui presenteremo una soluzione del problema completamente algebrica.

Consideriamo la seguente segnatura Σ

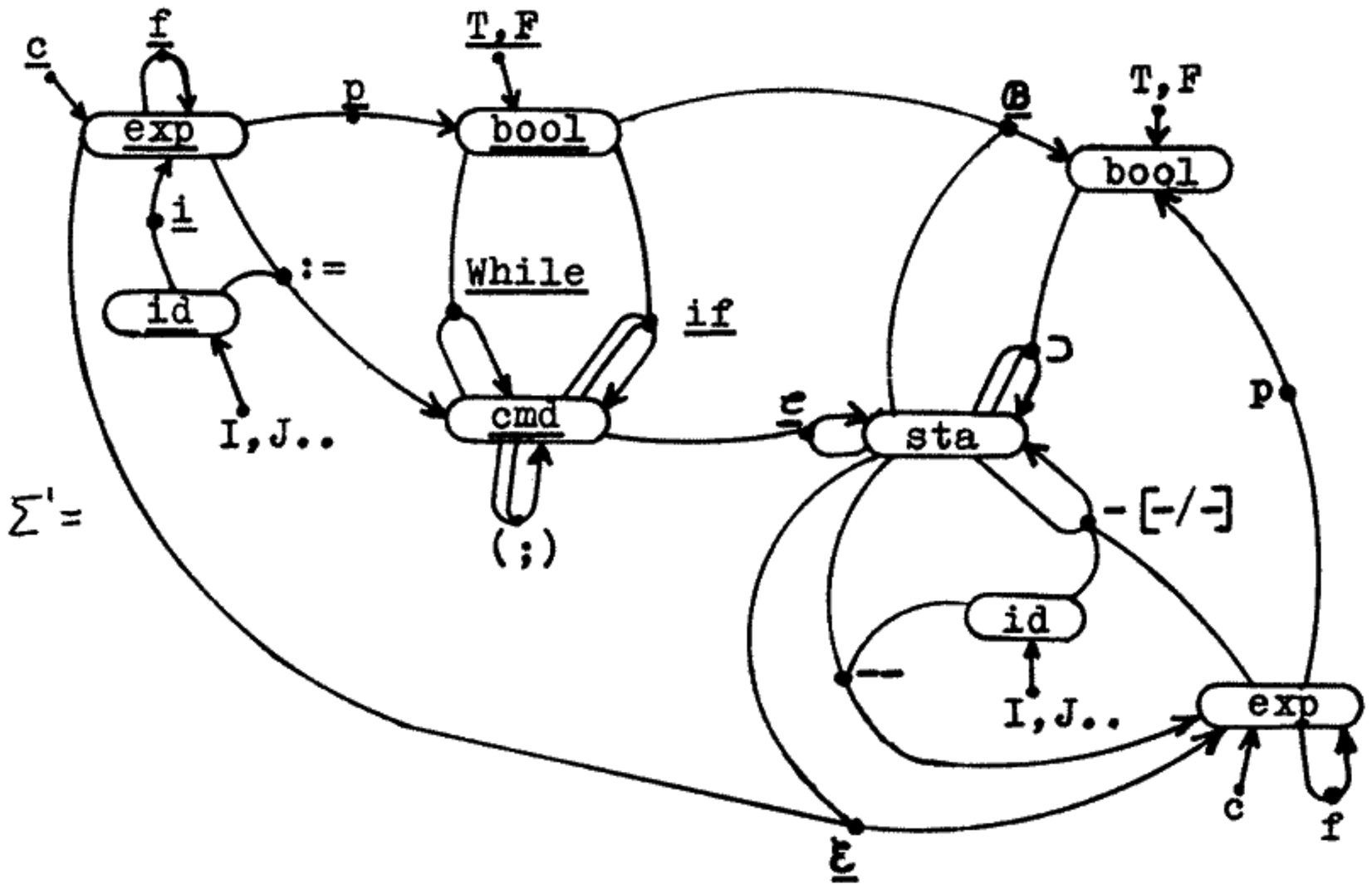


Sia $\underline{B}$ la Σ-algebra ove

$$B_{id} = ID \qquad B_{exp} = A \qquad B_{sta} = STA$$

e ove gli operatori $-[-/-]$, $-\supset-,-$, $--$, individuano le operazioni di aggiornamento e condizionale come sopra definite e quella di applicazione funzionale.

Sia quindi Σ' la segnatura ottenuta unendo a Σ la segnatura della sintassi prima definita :



ove $\underline{\mathfrak{E}}$, $\underline{\mathfrak{B}}$, $\underline{\mathfrak{C}}$ sono operatori associati alle funzioni di interpretazione $\mathfrak{E}$, $\mathfrak{B}$, $\mathfrak{C}$ sopra considerate.

Consideriamo le equazioni semantiche di prima e riscriviamole come schemi di assiomi **Ax** nella segnatura Σ', ovvero sottolineando i simboli $\mathfrak{E}$, $\mathfrak{B}$, $\mathfrak{C}$ e sostituendo c^A ed f^A con c ed f rispettivamente. Se si verifica che tali equazioni così riscritte risultano B-consistenti, allora per il teorema fondamentale sulle teorie basate (punto ii)) la classe delle algebre B-sovrabasate che verifica **Ax** ha un'algebra iniziale $\underline{I}$.

Evidentemente $\mathfrak{E}^I$, $\mathfrak{B}^I$, $\mathfrak{C}^I$ dove:

$$\mathfrak{E}^I : I_{\underline{exp}} \times I_{sta} \longrightarrow I_{exp}$$

$$\mathfrak{B}^I : I_{\underline{bool}} \times I_{sta} \longrightarrow I_{bool}$$

$$\mathfrak{C}^I : I_{\underline{cmd}} \times I_{sta} \longrightarrow I_{sta}$$

verificano le equazioni semantiche date all'inizio, ma

i domini e codomini di $\underline{\underline{\mathfrak{E}}}^I$, $\underline{\underline{\mathfrak{B}}}^I$, $\underline{\underline{\mathfrak{C}}}^I$ includeranno quelli specificati per le funzioni di interpretazione poichè $\underline{\underline{I}}$ è sovrabasata su $\underline{\underline{B}}$.

In effetti poichè in **Ax** non vi sono assiomi in cui si eguagliano termini di sorte **exp** o **cmd** avremo che $I_{exp} = EXP$ e $I_{cmd} = CMD$; però $I_{sta} \supset STA$ e l'inclusione (a meno di biunivocità, ovvero $I_{sta} \supset STA'$ e STA', STA biunivoci) è stretta in quanto, per esempio $\forall \rho$ non esiste un $\rho' \in STA$ tale che

$$\rho' = \underline{\underline{\mathfrak{C}}}^I(\underline{\text{While}}\ I = I\ \text{do}\ I := J)\rho$$

e quindi anche $I_{exp} \supset A$ (a meno di biunivocità).

Possiamo però definire facilmente le funzioni se= mantiche postulate all'inizio ponendo

$$\mathfrak{E}(E)\rho = \underline{\underline{\mathfrak{E}}}^I(E)\rho \in A \supset \underline{\underline{\mathfrak{E}}}^I(E)\rho \text{ , undefined}$$

$$\mathfrak{C}(c)\rho = \underline{\underline{\mathfrak{C}}}^I(c)\rho \in STA \supset \underline{\underline{\mathfrak{C}}}^I(c)\rho \text{ , undefined}$$

Lo schema adottato per l'$\underline{\underline{A}}$-While può generaliz= zarsi a qualsiasi linguaggio:

1) Si definisce la sintassi astratta;
2) Si individua una base $\underline{\underline{B}}$ e si scrivono le equazioni semantiche desiderate per delle funzioni di inter= pretazione;
3) Si riscrivono le equazioni semantiche come teoria algebrica la cui segnatura include quelle della sin= tassi astratta e quella della base e degli operatori interpretativi relativi alle funzioni interpretative che connettono sintassi con base;
4) Si dimostra la B-consistenza della teoria algebrica ottenuta, da cui discende l'esistenza di un'algebra $\underline{\underline{I}}$

in cui valgono le equazioni di partenza per funzio=
ni più definite di quelle semantiche desiderate
(che in genere sono parziali);

5) Si restringono le funzioni relative agli operatori interpretativi di $\underline{\underline{I}}$ sui domini della base $\underline{\underline{B}}$ con= siderandole non definite se forniscono come risul= tati elementi non appartenenti alla base .

Il problema fondamentale per la applicabilità di tale metodo è la dimostrazione di B-consistenza della teoria algebrica che specifica la semantica del lin= guaggio. Esistono in proposito dei metodi che stabili= scono tale consistenza utilizzando tecniche induttive o risultati sui sistemi di riscrittura (cfr.[26] [27]) e addirittura per la maggior parte dei linguaggi pro= programmativi tali metodi sono di applicazione semplicissima nel senso che si riducono a semplici controlli sintattici sulla specifica algebrica (cfr.[27])

Notiamo che il metodo presentato sviluppa una se= mantica che potremmo dire integrativa piuttosto che strettamente interpretativa, ovvero le funzioni seman= tiche non sono un morfismo tra algebre simili, ma connettono un' algebra sintattica a cui dare signifi= cato con un'algebra semantica (base) di oggetti assunti come già conosciuti.

Inoltre il calcolo equazionale permette in modo operativo di calcolare la semantica di un costrutto poichè se questo è un valore della base allora il cal= colo lo ottiene in un numero finito di passi (altrimen= ti il calcolo diventa infinito) (cfr.[27]) .

Consideriamo un comando in $\underline{\underline{N}}$-While ove

$$\underline{\underline{N}} = (N,\ +^N, <^N,\ 0^N,\ 1^N\ )$$

e calcoliamone la semantica utilizzando la sua specifica

algebrica su uno stato ρ_0 tale che $\rho_0 J = 0 \quad \forall J \in ID$
(abbrevieremo 0^N e 1^N con $0,1$)

$\mathcal{E}$(While I < 1+1 do I:= I+1)ρ_0 =

$\mathcal{B}(I < 1+1)\rho_0 = \mathcal{E}(I)\rho_0 <^N \mathcal{E}(1+1)\rho_0 =$

$= \rho_0 I <^N \mathcal{E}(1)\rho_0 +^N \mathcal{E}(1)\rho_0 =$

$= 0 <^N 1 +^N 1 = 0 <^N 2 =$ True

$= \mathcal{E}$(While I < 1+1 do I:= I+1)($\mathcal{E}$(I:= I+1)ρ_0) =

$\mathcal{E}(I := I+1)\rho_0 = \rho_0[\mathcal{E}(I+1)\rho_0 / I] =$

$= \rho_0[\mathcal{E}(I)\rho_0 +^N \mathcal{E}(1)\rho_0 / I] =$

$= \rho_0[\rho_0 I +^N 1 / I] = \rho_0[0 +^N 1 / I] =$

$= \rho_0[1 / I] = \rho_1$

$= \mathcal{E}$(While I < 1+1 do I:= I+1)ρ_1 =

$\mathcal{B}(I < I+1)\rho_1 = \mathcal{E}(I)\rho_1 <^N \mathcal{E}(1+1)\rho_1 =$

$= \rho_1 I <^N \mathcal{E}(1)\rho_1 +^N \mathcal{E}(1)\rho_1 =$

$= 1 <^N 1 +^N 1 = 1 <^N 2 =$ True

$= \mathcal{E}$(While I < 1+i do I:= I+1) ($\mathcal{E}$(I:= I+1)ρ_1) =

$\mathcal{E}(I := I+1)\rho_1 = \rho_1[\mathcal{E}(I+1)\rho_1 / I] =$

$= \rho_1[\mathcal{E}(I)\rho_1 +^N \mathcal{E}(1)\rho_1 / I] =$

$= \rho_1[\rho_1 I +^N 1 / I] = \rho_1[1 +^N 1 / I] =$

$= \rho_1[2 / I] = \rho_2$

$= \mathcal{E}$(While I < 1+1 do I:= I+1)ρ_2 =

$$\left| \; B(I < 1+1)\rho_2 = \rho_2 I <^N 2 = 2 <^N 2 = \text{False} \right.$$

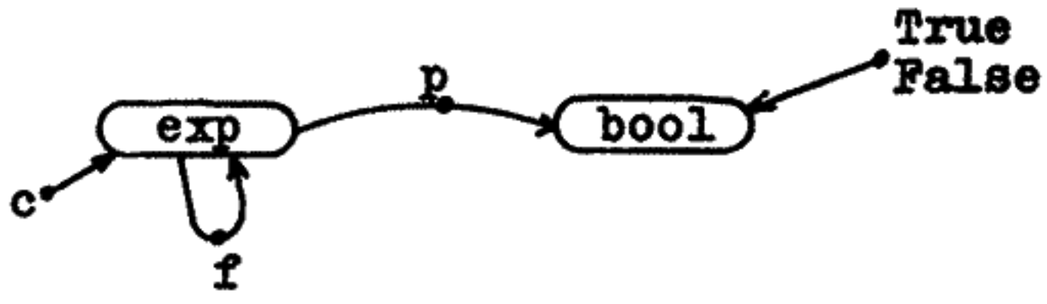
$$= \rho_2$$

65.- Semantica algebrica di un semplice linguaggio funzionale .

Consideriamo ancora un algebra $\mathbb{A}$

$$\mathbb{A} = (A, \{True, False\}, f^A, c^A, p^A)$$

di segnatura



Definiamo a partire da $\mathbb{A}$ il linguaggio $\mathbb{A}$-McCarthy

La sintassi astratta di tale linguaggio in BNF è la seguente

<Program> ::= eval (<Declaration>, <Expression>)

<Expression> ::= <Identifier> | f <Expression> | _

call <Function name> (<Expression>)

if <Boolean> then <Expression>
else <Expression>

<Declaration> ::= define <Function name> (<Identifier>) =
= <Expression> |

<Declaration> ; <Declaration>

<Boolean> ::= p <Expression> | True | False

<Identifier> ::= $I \mid I_1 \mid I_2 \ldots$

<Function name> ::= $F \mid F_1 \mid F_2 \ldots$

Indichiamo con **PRG**, **EXP**, DEC, BOOL, ID, FUN gli insiemi di termini della segnatura associata alla gramma= tica di sopra le cui sorte indicheremo con **prg**, **exp** **dec**, **bool**, **id**, **fun** rispettivamente.

Definiamo quindi un algebra base $\underline{B}$ di domini

$$B_{val} = A \cup \{error\}$$

$$B_{bool} = \{True, False, error\}$$

$$B_{exp} = EXP$$

$$B_{id} = ID$$

$$B_{fun} = FUN$$

$$B_{def} = \{\delta \mid \delta : FUN \rightarrow EXP \times ID \cup \{unbound\}\}$$

$$B_{sta} = \{\varrho \mid \varrho : ID \rightarrow A \cup \{unbound\}\}$$

Su tali domini assumiamo le funzioni definite dal= le seguenti condizioni:

— Applicazioni funzionali

$$\varrho \in B_{sta}, I \in B_{id} \Rightarrow \varrho I \in B_{exp} \text{ (applic.di } \varrho \text{ a } I)$$

$$\delta \in B_{def}, F \in B_{fun}, \delta F=(E,I) \Rightarrow \begin{cases} \delta F \downarrow EXP = E \\ \delta F \downarrow ID = I \end{cases}$$

— Condizionali

$$a,b,a_1,a_2 \in B_{val}, a_1 \neq a_2 \Rightarrow \begin{cases} True \supset a,b = a \\ False \supset a,b = b \\ a_1=a_2 \supset a,b = b \\ a_1=a_1 \supset a,b = a \end{cases}$$

— Aggiornamenti

$$(\varrho[a/I])J = \begin{cases} a & \text{se } I = J \\ \varrho J & \text{se } I \neq J \end{cases}$$

$$(\delta[(E,I)\ /\ F])G = \begin{cases} (E,I) & \text{se } F = G \\ \delta F & \text{se } F \neq G \end{cases}$$

— Test su espressioni ($E \in B_{exp}$)

$$\text{free}(E) = \begin{cases} \text{True} & \text{se } E \text{ contiene identificatori} \\ \text{False} & \text{altrimenti} \end{cases}$$

$$\text{free}(I,E) = \begin{cases} \text{True} & \text{se } E \text{ contiene identificatori diversi da } I \\ \text{False} & \text{altrimenti} \end{cases}$$

Nel seguito indicheremo gli operatori relativi a tali funzioni con la stessa simbologia adottata per le funzioni .

Possiamo quindi esprimere la semantica dell'$\underline{A}$-McCarthy tramite una segnatura composta a partire dalla sintassi e dalla segnatura di $\underline{B}$ aggiungendo i seguenti operatori (di interpretazione)

$\underline{\mathfrak{E}}$: exp def sta → val

$\underline{\mathfrak{B}}$: bool def sta → bool

$\underline{\mathfrak{D}}$: dec def → def

$\underline{\mathfrak{P}}$: prg → val

e dando le seguenti equazioni $\forall\ \varphi \in B_{sta}$, $\delta \in B_{def}$

con I elemento generico di sorta id e id· (B_{id}= ID)

E " " " " exp e exp (B_{exp}=EXP)

H " " " " bool

D " " " " dec

$\underline{\mathfrak{E}}\ (\underline{c})\delta\varphi = c$

$\underline{\mathfrak{E}}\ (I)\delta\varphi = \varphi I$

$\mathcal{E}(\underline{f}E)\delta\rho = f(\mathcal{E}(E)\delta\rho)$

$\mathcal{E}(\underline{\text{if}}\ H\ \underline{\text{then}}\ E_1\ \underline{\text{else}}\ E_2)\delta\rho = \mathcal{B}(H)\delta\rho = \text{error} \supset \text{error},$

$\mathcal{B}(H)\delta\rho = \text{True} \supset \mathcal{E}(E_1)\delta\rho, \mathcal{E}(E_2)\delta\rho$

$\mathcal{E}(\underline{\text{call}}\ F(E))\delta\rho = \mathcal{E}(E)\delta\rho = \text{error} \supset \text{error}, (\delta F = \text{unbound} \supset$

$\supset \text{error}, \mathcal{E}(\delta F{\downarrow}EXP)\delta\rho[\mathcal{E}(E)\delta\rho / \delta F{\downarrow}ID])$

$\mathcal{B}(\underline{\text{True}}) = \text{True}$

$\mathcal{B}(\underline{\text{False}}) = \text{False}$

$\mathcal{B}(\underline{p}E)\delta\rho = p(\mathcal{E}(E)\delta\rho)$

$\mathcal{D}(\underline{\text{define}}\ F(I)=E)\delta = \delta F = \text{unbound} \supset (\text{free}(I,E) \supset \text{error},$

$\delta[(E,I)/F]),\ \text{error}$

$\mathcal{P}(\underline{\text{eval}}\ E\ \underline{\text{in}}\ D) = \mathcal{D}(D)\delta_o = \text{error} \supset \text{error},\ \text{free}(E) \supset \text{error},$

$\mathcal{E}(E)(\mathcal{D}(D)\delta_o)\rho_o$

$\mathcal{D}(D_1;\ D_2)\delta = \mathcal{D}(D_2)(\mathcal{D}(D_1)\delta)$

$\rho_o I = \text{unbound}$

$\delta_o I = \text{unbound}$

E' del tutto naturale estendere la semantica sopra data nel caso in cui si ammettono chiamate di funzione con più di un argomento.

66- Descrizione algebrica del LISP

Un linguaggio di tipo LISP è costituito da un nucleo la cui sintassi astratta e la cui semantica sono riconducibili a quelle del Liste-McCarthy ove Liste è il tipo di dati di liste su stringhe alfanumeriche definibile formalmente come si è visto precedentemente.

Esprimendo la sintassi astratta in BNF avremo :

<Program> ::= eval (<Declaration> , <Expression>)

Ψ: <Expression> ::= <Identifier> | <List> | <Atom> |
car(<Expression>) | cdr(<Expression>) |
cons(<Expression> , <Expression>) |
<Function name> (<Argument>) |
if <Boolean> then <Expression>
else <Expression>

Π: <Argument> ::= <Expression> | <Argument> , <Expression> |

<Atom> ::= <Alfanum> <Atom> | <Alfanum>

<Alfanum> ::= A | B | ... | Z | Ø | 1 | ... | 9

Λ: <List> ::= l (l è lista del tipo di dati "liste su atomi")

Δ: <Declaration> ::= define <Function name> (<Identifier>)=
= <Expression> |
<Declaration> ; <Declaration>

χ: <Identifier> ::= I <Atom>

Φ : <Function name> ::= F <Atom>

Γ: <Boolean> ::= eq(<Atom>, <Atom>) | isatom(<Expression>) |
True | False

(Se Ψ è un'espressione scorretta del tipo car(nil) $\xi(\Psi)_{\delta\varsigma}$ =error)

Ciò che però è peculiare del LISP è il fatto di avere una sintassi concreta espressa entro il tipo di dato liste; ovvero vi è una funzione τ di realizzazione sintattica che traduce i domini EXP, BOOL, ... in liste.

Avviene quindi che una certa espressione del Liste-McCarthy che individua una funzione su liste è in LISP rappresentata tramite una lista. Cosi in modo del tutto naturale si possono rappresentare in LISP programmi che manipolano programmi, ovvero algoritmi a vari livelli di applicazione funzionale.

Tale aspetto rende il LISP estremamente indicato in vari contesti programmativi e sopratutto in intelligenza artificiale.

Definiamo come esempio una realizzazione sintattica τ :

$\tau(\chi) = \chi$

$\tau(\beta) = \beta$

$\tau(\Psi,\Gamma)) = (\tau(\Psi), \tau(\Gamma))$

$\tau(\underline{\text{True}}) = \text{TRUE}$

$\tau(\underline{\text{False}}) = \text{FALSE}$

$\tau(\text{nil}) = \text{NIL}$

$\tau(\Lambda) = (\text{QUOTE}\ \Lambda) \qquad \forall\ \Lambda \neq \text{nil}$

$\tau(\ \underline{\text{eq}}(\Psi_1, \Psi_2)\ ) = (\text{EQ}\ \ \tau(\Psi_1)\ \tau(\Psi_2)\ )$

$\tau(\underline{\text{isatom}}(\Psi)\ ) = (\text{ISATOM}\ \ \tau(\Psi)\ )$

τ(cons(Ψ_1,Ψ_2))= (CONS $\tau(\Psi_1)$ $\tau(\Psi_2)$)

τ(car(Ψ))= (CAR $\tau(\Psi)$)

τ(cdr(Ψ))= (CDR $\tau(\Psi)$)

τ(if Γ then Ψ_1 else Ψ_2)= (COND $\tau(\Gamma)$ $\tau(\Psi_1)$ $\tau(\Psi_2)$)

τ(define($\Xi(\chi)= \Psi$))= (DEFINE $\Xi(\chi)$ $\tau(\Psi)$)

$\tau(\Delta_1; \Delta_2) = (\tau(\Delta_1)\ \tau(\Delta_2))$

$\tau(F(\Pi)) = (F\ \tau(\Pi))$

τ(eval(Δ, Ψ)) = (EVAL $\tau(\Delta)$ $\tau(\Psi)$)

Osserviamo che l'atomo QUOTE serve a distinguere le liste che sono nomi di costrutti da quelle che sono liste vere e proprie, ovvero elementi del tipo di dato. L'uso di QUOTE è analogo alle virgolette con cui in italiano distin= guiamo Roma come nome di un ente extralinguistico e "Roma" come nome della parola.

Esempio di traduzione dal Liste-McCarthy al LISP.

P= eval(define FF(II)= if eq(II,nil) then FINE

else cons(CIAO,FF(cdr(II))) , FF((X X X)))

τ(P) = (EVAL (DEFINE FF(II) (COND (EQ II NIL) (QUOTE

FINE) (CONS (QUOTE CIAO) (FF (CDR II)))))

FF((QUOTE X X X)))

Esercizio : descrivere τ come morfismo