Lezione 2 03/10/2018

lunedì 24 settembre 2018 20:54

MADONIA

ALFABETO → Insieme finito di simboli
Viene indicato con la lettera greca sigma Σ

[Es. $\Sigma = \{0,1\}$ / $\Sigma = \{a,b,c\}$ / $\Sigma = \{N,R\}$]

PAROLA / STRINGA → Sequenza finita di simboli di un alfabeto

DEFINIZIONE FORMALE DI STRINGA

Dato un alfabeto Σ, l'insieme delle stringhe Σ^* è definito:
- $\varepsilon \in \Sigma^*$
- $ax \in \Sigma^* \ \forall a \in \Sigma, x \in \Sigma^*$

Per convenzione esiste la parola ε (epsilon) che è detta "PAROLA VUOTA", cioè che non contiene nessun simbolo. N.B. UN ALFABETO NON CONTIENE MAI LA PAROLA VUOTA

CONCATENAZIONE TRA DUE STRINGHE → Affiancamento di due stringhe [Es. $x = 101$, $y = 00$, $x \circ y = 10100$ ("$x \circ y$" si dice "x concatenato y")]

N.B. NON GODE DELLA PROPRIETA' COMMUTATIVA

E' un'operazione binaria tra stringhe.

Dunque $x \circ \varepsilon = x$ e $\varepsilon \circ x = x$ → la PAROLA VUOTA è l'elemento neutro.

MONOIDE → E' una STRUTTURA ALGEBRICA che deve avere un'operazione interna e un elemento neutro (il monoide è sempre un insieme)

[Es. Nell'insieme dei numeri interi $\mathbb{Z}$, la somma è un'OPERAZIONE INTERNA (Poiché il risultato è parte dell'insieme) e ha un elemento neutro (0)] Dunque $\mathbb{Z}$ è un MONOIDE.

L'insieme di tutte le possibili stringhe sull'alfabeto Σ, viene indicato Σ^* (sigma star). N.B. Σ contiene solo simboli
Σ^* contiene solo parole composte da simboli di Σ

[Es. $\Sigma = \{0,1\}$, $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 111, 0001, \text{ecc.}\}$]

$(\Sigma^*, \circ, \varepsilon)$ è un **monoide** → Abbiamo un insieme "Σ^*"
$\circ$ → OP. CONCATENAZIONE
un'operazione "$\circ$"
un elem. neutro "ε"

Dato un alfabeto Σ, denotiamo con $(\Sigma^*, \circ, \varepsilon)$ il MONOIDE LIBERO (o SINTATTICO) su Σ.

L'operazione "$\circ$": $\Sigma^* \times \Sigma^* \to \Sigma^*$ (da Σ^* per Σ^* in Σ^*) è detta OPERAZIONE DI CONCATENAZIONE

$\Sigma^* \times \Sigma^*$ → E' UN PRODOTTO TRA DUE INSIEMI, QUINDI DA' COPPIE ORDINATE DI ELEMENTI $((x,y) \neq (y \neq x))$

[Es. $\Sigma\{0,1\}$. $\Sigma^* \times \Sigma^* = \{(0,1), (01,10), (10,01), \text{ecc}\}$]

In un prodotto cartesiano $A \times B$ è infinito se A, B o A e B sono infiniti (basta che uno solo dei due sia infinito).

Per definire l'operazione "$\circ$", prendo una coppia di stringhe data da $\Sigma^* \times \Sigma^*$ e la concateno. Il risultato sarà interno a Σ^*

Con la notazione x^h indichiamo la **POTENZA DI UNA STRINGA**, ovvero la concatenazione di x con sé stessa h volte, con $h \geq 0$.

[Es. $001^3 = \underbrace{001001001}_{h \text{ volte}}$]

! $x^0 = \varepsilon$ (Così come negli esponenziali $x^0 = 1$, poiché 1 è l'elemento neutro, nella concatenazione $x^0 = \varepsilon$, poiché ε è l'elemento neutro.)

La lunghezza di una stringa si indica con $|x|$.

$|x|$ = lunghezza di x

$|\varepsilon| = 0$

DEFINIZIONE FORMALE: $|xa| = |x| + 1$ con $\begin{cases} x \in \Sigma^* \rightarrow \text{E' una stringa} \\ a \in \Sigma \rightarrow \text{E' un simbolo} \end{cases}$ $(x \circ a)$

$|\varepsilon| = 0$ è il caso base

[Es. $\Sigma = \{0, 1\}$, $a = 1$, $b = 0$, $x = 00$, $xb = 001$, $|xb| = |\underbrace{001}_{x\,b}| = |\underbrace{00}_{x\,a}| + 1 = |0| + 1 + 1 = |\varepsilon| + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3$]

$|0| = |\varepsilon 0|$

LINGUAGGIO → Un qualunque sottoinsieme di Σ^*.

DEFINIZIONE: Dato un alfabeto Σ, si definisce "Linguaggio su Σ", un qualunque sottoinsieme di Σ^*.

[Es. $\Sigma = \{a, b\}$, $\{ab, ba, bb\} \subseteq \Sigma^*$ /oppure/ $\{a\} \subseteq \Sigma^*$ /oppure/ $\{\varepsilon, a, aa, aaa, \dots\} \subseteq \Sigma^*$]

Per rappresentare un insieme (in questo caso linguaggio) si può usare la notazione matematica [Es. $\{\varepsilon, a, aa, aaa, \dots\} = \{a^n \mid n \geq 0\}$].

LINGUAGGIO VUOTO → ϕ non contiene niente e si indica con lambda "Λ" ($\Lambda = \phi$).

N.B. $\Lambda \neq \{\varepsilon\}$ → Da distinguere LINGUAGGIO VUOTO e LINGUAGGIO CONTENENTE LA SOLA ε.

Avendo $\Sigma = \{a, b\}$, $L = \{aa, bb\}$ → $\overline{L} = \{\varepsilon, ab, aab, ba, \dots\}$ (Praticamente $\overline{L} = \Sigma^* \setminus L$)

→ COMPLEMENTARE DI L

→ \ negli insiemi è il simbolo di differenza.

N.B $\Sigma \subseteq \Sigma^*$ (E' un sottoinsieme formato solo dalle parole formate da un solo simbolo)

I LINGUAGGI SI POSSONO CONCATENARE

[Es. $L_1 = \{00, 01\}$

$L_2 = \{101, 11\}$

$L_1 \circ L_2 = \{00101, 0011, 01101, 0111\}$ → E' un insieme che contiene tutte le possibili concatenazioni delle parole dei due insiemi

N.B. NON SONO INSIEMI ORDINATI]

DEFINIZIONE: la concatenazione di due linguaggi $L_1, L_2 \subseteq \Sigma^*$ è il linguaggio $L_1 \circ L_2 = \{x \in \Sigma^* \mid \exists y_1 \in L_1, \exists y_2 \in L_2 \mid x = y_1 \circ y_2\}$

$\{\varepsilon\}$ è l'elemento neutro nella concatenazione tra due linguaggi.

<u>INFATTI</u>: $L \circ \{\varepsilon\} = L$ e $\{\varepsilon\} \circ L = L$

<u>N.B.</u> $L \circ \Lambda = \Lambda$ poiché non possiamo concatenare un elemento con nulla.

[Es. $L_1 = \{\text{ASTRO, FISIO}\}$ e $L_2 = \{\text{LOGIA, NOMIA}\}$

$L_1 \circ L_2 = \{\text{ASTROLOGIA, ASTRONOMIA, FISIOLOGIA, FISIONOMIA}\}$]

Esiste anche la **POTENZA DI LINGUAGGIO** (L^n), ovvero L concatenato n volte con sé stesso.

Dunque $L^n = \begin{cases} L \circ L^{n-1} & \text{se } n \geq 1 \\ \{\varepsilon\} & \text{se } n = 0 \end{cases} \rightarrow L^0 = \{\varepsilon\}$ <u>N.B.</u> Anche $\Lambda^0 = \{\varepsilon\}$

[Es. $L_1 = \{00, 11\}$

$L^2 = L \circ L = \{0000, 0011, 1100, 1111\}$

Es $L = \{00, 11\}$

$L^3 = L \circ L^2$

$L^2 = L \circ L = \{0000, 0011, 1100, 1111\}$

$L^3 = \{00, 11\} \circ \{0000, 0011, 1100, 1111\}$

$= \{000000, 000011, 001100, 001111, 110000, 110011, 111100, 111111\}$]

Lezione 3 08/10/2018

lunedì 24 settembre 2018 20:54

BARBANERA

I **SISTEMI FORMALI** sono la rappresentazione di un ragionamento.

RAGIONAMENTO / DEDUZIONE / PROVA → Sequenza di passi elementari che portano ad affermazioni/giudizi validi.

GIUDIZIO = **fbf** (formule ben formate).

Per descrivere un ragionamento, bisogna definire un alfabeto.

[Es. $S\{0, s, (,), \in, \mathbb{N}, x\}$] → η è il sistema formale

Un giudizio è formato da una stringa di caratteri dell'alfabeto.

DECIDIBILITA' → Un insieme è decidibile se preso un elemento dell'insieme, possiedo un algoritmo che mi permette di stabilire se l'elemento fa parte dell'insieme o no.

Un errore comune è quello di non considerare **SINTASSI** (insieme di caratteri) e **SEMANTICA** (insieme di significati ad essi attribuiti) come due cose distinte.

Un paradosso contro cui ci si potrebbe imbattere è l'affermazione "Questa frase è falsa". Se cerchiamo di "costringere" un significato in queste parole, scambiamo un insieme di caratteri per una frase di senso compiuto.

[Nel nostro esempio, abbiamo l'alfabeto $S\{0, s, (,), \in, \mathbb{N}\}$ → Sono una serie di simboli

Potremmo scrivere la stringa $((00 \in \mathbb{N}$ che apparentemente non significa niente

Anche la stringa $s(0) \in \mathbb{N}$ apparentemente può non significa niente.

Ma se noi diamo una particolare semantica $s(0) \in \mathbb{N}$ acquista il significato "il successore di zero appartiene all'insieme $\mathbb{N}$".

Introduciamo l'insieme W di fbf definito $W = \{x \in S^* \mid \exists y \in S \mid x = y \in \mathbb{N}\}$] → W è l'insieme delle formule ben formulate x appartenenti all'insieme S^* delle stringhe sull'alfabeto S, tale che esiste un carattere y dell'alfabeto S, tale che possiamo definire la fbf $x = y \in \mathbb{N}$ (y appartiene all'insieme $\mathbb{N}$)

N.B. $W \subseteq S^*$

L'insieme Ax di assiomi è un sottoinsieme dell'insieme W di fbf

$Ax \subseteq W$

ASSIOMA → E' un affermazione che non ha bisogno di giustificazioni o dimostrazioni precedenti. L'insieme degli assiomi è decidibile.

In un sistema formale, un assioma può essere inserito tutte le volte che si vuole e in qualsiasi momento

[Nel nostro esempio $Ax = \{0 \in \mathbb{N}\}$ ⟶ OVVERO ⟶ $\frac{}{0 \in \mathbb{N}}$]

Per dire che una stringa è un assioma si usa la simbologia $\frac{}{(\text{ASSIOMA})}$ che significa "se posso affermare quello che c'è sopra, potrò affermare quello che c'è sotto" e visto che sopra non c'è nulla, quello che c'è sotto non ha bisogno di affermazioni precedenti, dunque è un assioma.

N.B. Questa definizione di sistemi formali è stata data da HILBERT

Ora pariamo a stabilire quali sono i passi deduttivi del nostro ragionamento.

L'insieme R di passi è costituito da **REGOLE DI INFERENZA** denominate R_i (ci permettono di stabilire y partendo da x)

Allora $R = \{R_i\}_{i \in I}$ → Insieme di regole di inferenza grande quanto un insieme I (dunque numerato).

INFERENZA → Procedimento attraverso cui si passa da una cosa vera ad un'altra cosa vera.

R_i è una relazione tra premesse vere e conclusioni vere

R_i è definita in $W \times W$ (poiché collega fbf con fbf).

Possiamo avere tante premesse per una conclusione, ma in un sistema formale, una premessa può avere una sola conclusione.

Dunque definiamo R_i in $W \times W \times \ldots \times W \rightarrow W^{m_i}$ → Potenza di linguaggio (v. LEZIONE 2, MADONIA)

[R_i $\{(\epsilon\epsilon) \in \mathbb{N}, S(\epsilon\epsilon) \in \mathbb{N}\}$] → Se posso affermare che $(\epsilon\epsilon)$ appartiene a $\mathbb{N}$, potrò affermare che il successivo appartiene a $\mathbb{N}$.

W moltiplicato n volte, quanti sono gli elementi di I

[Nel mio esempio avrò l'assioma $(Ax)\ \frac{}{0 \in \mathbb{N}}$ e la regola $(R)\ \frac{x \in \mathbb{N}}{S(x) \in \mathbb{N}}$]

Se ho un sistema formale D, un RAGIONAMENTO (o PROVA o DEDUZIONE) una sequenza ordinata di fbf $(a_1, a_2, \ldots, a_k)$.

RAGIONAMENTO = $a_1, a_2, \ldots, a_k$ tale che per ogni indice i:

- a_i → ASSIOMA $a_i \in Ax$ OPPURE
- a_i → IPOTESI $a_i \in M$ OPPURE → M è l'insieme delle ipotesi
- a_i → FBF $a_i \in W$

[Nel nostro esempio, dati $Ax\ \frac{}{0 \in \mathbb{N}}$ e $R_i\ \frac{x \in \mathbb{N}}{S(x) \in \mathbb{N}}$

1. $0 \in \mathbb{N}$ (Ax) → Per l'assioma
2. $S(0) \in \mathbb{N}$ $(R_{(1)})$ → Regola a partire da (1)
3. $0 \in \mathbb{N}$ (Ax) → Assioma (Possiamo ripeterlo quante volte vogliamo)
4. $S(S(0)) \in \mathbb{N}$ $(R_{(2)})$ → Regola a partire da (2)

Questo è il nostro ragionamento.

$S(S(0)) \in \mathbb{N}$ è la conclusione, detta DERIVAZIONE.]

Una derivazione si indica con il simbolo $\vdash_M S(S(0)) \in \mathbb{N}$ → "$S(S(0)) \in \mathbb{N}$ derivazione nel sistema formale M".

SEMANTICA → Stabilire cosa significa se una fbf è vera o falsa

[Es. scrivo $((00 \in \mathbb{N}$ → Nell'esempio citato finora, questa fbf è falsa.
Ma se diamo la semantica che la fbf α è vera se il numero di caratteri usati è pari, allora $((00 \in \mathbb{N}$ risulta vera, mentre $0 \in \mathbb{N}$ risulterebbe falsa].

Se tutti i passi e le fbf del nostro ragionamento sono semanticamente vere si parla di **CORRETTEZZA** del ragionamento.

COMPLETEZZA → Se posso ottenere come conclusione di un ragionamento qualsiasi fbf del mio sistema.

[Es. $S(S(S(S(0))))\in \mathbb{N}$ è vera e posso ottenerla come conclusione del ragionamento:

1. $0 \in \mathbb{N}$ (Ax)
2. $S(0) \in \mathbb{N}$ (R(1))
3. $S(S(0)) \in \mathbb{N}$ (R(2))
4. $S(S(S(0))) \in \mathbb{N}$ (R(3))
5. $S(S(S(S(0)))) \in \mathbb{N}$ (R(4))

(5) è la conclusione del ragionamento.]

Se il nostro ragionamento non parte da un assioma, o da una regola deducibile, deve partire da un'ipotesi.
Si parla allora di **RAGIONAMENTO IPOTETICO**.

Dato un insieme H di ipotesi, se l'affermazione α si può ottenere da una deduzione a partire dall'insieme H, allora possiamo scrivere $H \vdash_{\eta} \alpha$ → "H deriva α in η"

CONSISTENZA → È un concetto puramente sintattico (ancor prima di dare una semantica, possiamo capire se il sistema ha un senso).
Un sistema è **INCONSISTENTE (O CONTRADDITTORIO)** se permette di derivare qualsiasi fbf (se $\vdash_{\eta} \alpha, \forall \alpha \in W \Rightarrow \eta$ è inconsistente)
Un sistema è **CONSISTENTE** se esiste almeno una fbf non derivabile (se $\exists\, \alpha \in W \mid \nvdash_{\eta} \alpha \Rightarrow \eta$ è consistente).

Sia Γ (gamma γ maiuscola) un insieme finito di fbf in η, l'insieme $Con(\Gamma)$ delle conseguenze a partire da Γ è definito:

$$Con(\Gamma) = \{\alpha \mid \Gamma \vdash_{\eta} \alpha\}$$

Chiamiamo **TEOREMA** una conseguenza che non deriva da ipotesi $(\alpha \in W \mid \vdash_{\eta} \alpha)$

Con $Con(\emptyset)$ indichiamo l'insieme dei teoremi
Un sistema dove $Con(\emptyset) = W$ (dove tutte le fbf sono teoremi), è un sistema inconsistente.
Se invece $Con(\Gamma) \subseteq W$ (se quindi tra le fbf non ci sono solo teoremi), abbiamo un sistema consistente

TEORIA → Γ si dice teoria se, utilizzando ipotesi di Γ, posso ottenere solo elementi di Γ → $Con(\Gamma) = \Gamma$

Viene detto "**INSIEME CHIUSO** rispetto alla relazione di derivazione".

È la **PROPRIETÀ DI PUNTO FISSO** dove $a \in W$, $f(a) = a$, dunque a è un punto fisso.

TEORIA PURA → L'insieme $Con(\emptyset)$ di Teoremi, è una teoria $(Con(Con(\emptyset)) = Con(\emptyset))$ detta TEORIA PURA.

Lezione 4 10/10/2018

lunedì 24 settembre 2018 20:54

MADONIA (Pg. 28)

STAR DI KLEENE di L → Il linguaggio L^* definito da:

$$L^* = \bigcup_{h=0}^{\infty} L^h$$

[Es. $L = \{00, 11\}$

$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots = \{\varepsilon\} \cup \{00, 11\} \cup \{0000, 0011, 1100, 1111\} \cup \ldots$]

[Es. $L_1 = \{BIS\}$ $L_2 = \{NONNO\}$

$L_1^* \circ L_2 = \{\underbrace{NONNO}_{L_2}, \underbrace{BIS}_{L_1^*}\underbrace{NONNO}_{L_2}, \underbrace{BISBIS}_{L_1^*}\underbrace{NONNO}_{L_2}, \underbrace{BISBISBIS}_{L_1^*}\underbrace{NONNO}_{L_2}, \ldots\}$]

Il linguaggio L^+ definito da:

STAR POSITIVA DI KLEENE → Il linguaggio L^+ è definito:

$$L^+ = \bigcup_{h=1}^{\infty} L^h$$

→ Praticamente parte da L^1 ε può appartenere a L^+ se L la contiene

Possiamo dedurre che $L^+ \subseteq L^*$:

$L^* = L^+ \cup L^0 = L^+ \cup \{\varepsilon\}$

In analogia con Σ e Σ^*, possiamo definire Σ^+

e poiché $\varepsilon \notin \Sigma$, potremo scrivere sempre $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$

LINGUAGGI (Pg. 59)

Esistono due metodi di rappresentare linguaggi infiniti con meccanismi finiti

- METODO GENERATIVO: Permette di generare tutte le stringhe del linguaggio
- METODO RICONOSCITIVO: Costituito da un dispositivo che prende in input una stringa e ci dice se questa $\in L$ o no.

I "RICONOSCITORI" vengono chiamati **AUTOMI**.

Formato da un dispositivo che si trova in un certo stato. → Se cerchiamo la stringa che contiene due "a", se ha già incontrato la prima "a" è in uno stato, se non l'ha incontrata è in un altro stato.

E' dotato di un nastro con una testina mobile (si muove di una sola cella a dx o sx o sta ferma) che controlla il contenuto della stringa in input. Sul nastro si scrive la stringa, e ogni cella contiene un simbolo. Il nastro può essere infinito o semiinfinito. Gli stati del riconoscitore sono finiti N.B. Il nastro è un array infinito.

[Es. x = aabbb

A seconda del contenuto della cella, il riconoscitore compie un passo successivo:
- Scrivere sulla cella
- Muovere la testina a dx o sx
- Cambiare stato

→ COMPUTAZIONE

Le regole che gestiscono i passi della computazione consistono nella FUNZIONE DI TRANSIZIONE

La computazione è una sequenza di CONFIGURAZIONI

CONFIGURAZIONE → E' la "fotografia" dell'automa in un qualsiasi momento (avremo uno stato q, un carattere in lettura, una posizione).

Lo stato di inizio (in posizione Q_0) si chiama CONFIGURAZIONE INIZIALE.

Si definisce "cella vuota", la cella (o le celle) che contengono l'elemento BLANK $\not b$

Negli automi a stati finiti, per terminare, bisogna raggiungere la CONFIGURAZIONE FINALE.

N.B. Le macchine di Turing sono gli automi più potenti.
Qualsiasi funzione per cui si possa costruire un algoritmo, può essere calcolate con una macchina di Turing. Questa è la TESI DI CHURCH-TURING e al momento non è stata confutata.

AUTOMI A STATI FINITI (ASF) (Pg. 71)

Sono riconoscitori in cui la testina non può scrivere e deve spostarsi a dx.

AUTOMA A STATI FINITI DETERMINISTICI → (ASFD) è una quintupla (5 elementi)

$A = (\Sigma, Q, \delta, q_0, F)$ dove:
- Σ = alfabeto di input (finito);

Negli stati finiti ogni configurazione permette al più una sola configurazione successiva

- Q = insieme degli stati (non vuoto e finito);
- $\delta : Q \times \Sigma \to Q$, funzione di transizione
- q_0 = stato iniziale;
- $F \subseteq Q$ = insieme degli stati finali.

"Se sono nello stato q_1 e leggo il carattere a allora devo passare allo stato q_2."

(Associa a elementi di Q elementi di Σ)

Perché se l'automa è nello stato q_m, in base al carattere in input letto, la funzione fa cambiare lo stato. $\delta(q,a)=p$ con $q \in Q$, $a \in \Sigma$ e $p \in Q$.

[Es. (Pg.72, Tabella 3.1)

$\Sigma = \{a, b\}$

$Q = \{q_0, q_1, q_2\}$

$F = \{q_1\}$

DEFINIAMO δ

δ	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

Es. $\delta(q_1, b) = q_2$

Poiché abbiamo tutte le caselle piene, dunque, se la funzione è definita per ogni coppia stato-simbolo, la chiamiamo **FUNZIONE TOTALE**

① a b b

q_0 $\delta(q_0, a) = q_0$ CONFIG: (q_0, abb)

② a b b

q_0 $\delta(q_0, b) = q_1$ CONFIG: (q_0, bb)

③ a b b

q_1 $\delta(q_1, b) = q_2$ CONFIG: (q_1, b)

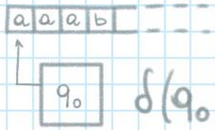
Scriviamo lo stato, e il nastro che ancora posso leggere. Poiché negli ASFD, posso/devo muovermi solo verso dx.

④ a b b

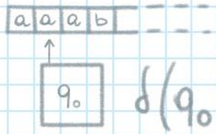
q_2 Poiché non abbiamo alcun elemento ci fermiamo

⑤ Controlliamo se lo stato q_2 fa parte dell'insieme degli stati finiti F. Non lo è, quindi non accettiamo la stringa "abb" nel nostro linguaggio.]

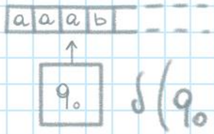
Esercizio: Controllare se la stringa "aaab" è accettabile nel linguaggio.

1. 

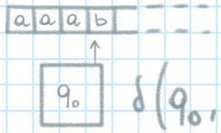
$\delta(q_0, a) = q_0$

2. 

$\delta(q_0, a) = q_0$

3. 

$\delta(q_0, a) = q_0$

4. 

$\delta(q_0, b) = q_1$

5. Controlliamo. $q_1 \in F$
Dunque possiamo accettare la stringa "aaab".

lunedì 24 settembre 2018 20:54

BARBANERA

Chiamiamo CL un sistema formale.

S (ALFABETO) $= \{K, S, (,), =\}$ → Finché non diamo una semantica, sono solo simboli.

W (INSIEME DI FBF) $= \{P = Q \mid P, Q \in \tau\}$ → Stringhe che hanno questa forma

$\tau \subseteq S^*$ è l'insieme di termini così definito:

$K \in \tau$, $S \in \tau$ e se $P, Q \in \tau \Rightarrow (PQ) \in \tau$

Ax: Per ogni $P, Q, R \in \tau$

- $((KP)Q) = P$ — Ax K
- $P = P$ — Ax rifl
- $(((SP)Q)R) = ((PR)(QR))$ — Ax S

} Sono schemi di assiomi poiché P, Q, R non appartengono a S

$R = \{R_1, R_2, R_3, R_4\}$

- $R_1 \ \dfrac{P = Q}{Q = P}$ → Regola simmetrica
- $R_2 \ \dfrac{P = Q \quad Q = R}{P = R}$ → Regola transitiva
- $R_3 \ \dfrac{R = R' \quad (PR) = (QR)}{(PR) = (QR')}$
- $R_4 \ \dfrac{R = R' \quad (RP) = (RQ)}{(RP) = (R'Q)}$

DEDUZIONE: dimostriamo che $\vdash_{CL} (((SK)K)K) = K$ → Ovvero che è un teorema di CL

1. $(((SK)K)K) = ((KK)(KK))$ — Axs con $P=K, Q=K, R=K$

$\quad\quad\;\; P\;\;Q\;\;R \qquad\quad P\;R \quad QR$

2. $((KK)(KK)) = K$ — AxK con $P=K, Q=KK$

$\quad\;\; P \quad Q \qquad P$

3. $(((SK)K)K) = K$ — $R_2\ (1,2) \rightarrow$ TRANSITIVA

Genericamente per ogni $M \in \tau$, $\vdash_{CL} (((SK)K)M) = M$

Introduciamo la definizione $I = ((SK)K) \rightarrow$ ipotesi

Dunque $\vdash_{CL} (IM) = M$

Dimostriamo che $\vdash_{CL} (((SI)I)M) = (MM)$ per ogni $M \in \tau$:

1. $(((SI)I)M) = ((IM)(IM))$ — Axs $\rightarrow P=I, Q=I, R=M \Rightarrow PR=IM, QR=IM$

2. $(IM) = M$ — esempio precedente

3. $((IM)(IM)) = ((IM)(IM))$ — Axrifl

4. $((IM)(IM)) = (M(IM))$ — $R_4\ (2,3)$

$$\begin{cases} ② \ R=R' \ [R=(IM),\ R'=M] \\ ③ \ RP=RQ \ [P=(IM),\ Q=(IM) \Rightarrow RP=((IM)(IM)),\ RQ=((IM)(IM))] \\ \Rightarrow RP=R'Q \ [RP=((IM)(IM)),\ R'Q=(M(IM))] \end{cases}$$

5. $((IM)(IM)) = (MM)$ — $R_3\ (2,4)$

$$\begin{cases} ② \ R=R' \ [R=(IM),\ R'=M] \\ ④ \ PR=QR \ [P=(IM),\ Q=M \Rightarrow PR=((IM)(IM)),\ QR=(M(IM))] \\ \Rightarrow PR=QR' \ [PR=((IM)(IM)),\ QR'=MM] \end{cases}$$

6. $(((SI)I)M) = (MM)$ — $R_2\ (1,5)$

$$\begin{cases} ① \ P=Q \ [P=(((SI)I)M),\ Q=((IM)(IM))] \\ ⑤ \ Q=R \ [R=MM] \\ \Rightarrow \ P=R \end{cases}$$

CALCOLO PROPOSIZIONALE

Sistema formale P_0

$S=\{(,),\rightarrow,\wedge,\vee,\neg,p,q,r,r',p',t,\dots \text{ecc}\dots\}$ → Chiamiamo questi simboli come variabili propositionali

W (definito per induzione):

- Qualsiasi variabile propositionale di S è una fbf ($\in W$)
- se $\alpha \in W \Rightarrow \neg(\alpha) \in W$
- se $\alpha,\beta \in W \Rightarrow (\alpha \vee \beta) \in W$
- se $\alpha,\beta \in W \Rightarrow (\alpha \wedge \beta) \in W$
- se $\alpha,\beta \in W \Rightarrow (\alpha \rightarrow \beta) \in W$

N.B. Una proposizione $a \wedge b$ è VERA se in un "mondo" (in un contesto) a e b sono vere.

A noi interessa trovare fbf vere in ogni mondo possibile.

Si parla di ovvietà, dunque **TAUTOLOGIE**

Se il nostro sistema non prevedesse $\wedge$ e $\vee$, potremmo scrivere le relazioni $\alpha \vee \beta$ e $\alpha \wedge \beta$ a partire da

- $\alpha \vee \beta$ ..... $(\neg\alpha) \rightarrow \beta$ se α vera $\neg\alpha$ falsa $\neg\alpha \rightarrow \beta$ è vera se o $\neg\alpha$ è vera o β è vera
- $\alpha \wedge \beta$ ..... $\neg(\neg\alpha \vee \neg\beta) \Rightarrow \neg(\neg\neg\alpha \rightarrow \neg\beta)$

Quindi il nostro alfabeto è:

$S=\{(,),\rightarrow,\neg,p,q,r,r',p',t,\dots \text{ecc}\dots\}$

W:

- $\alpha \rightarrow (\beta \rightarrow \alpha)$

REGOLA DERIVABILE → Regola di cui si può fare a meno:

Data R $\frac{\alpha_1, \alpha_2, \dots, \alpha_m}{\beta}$, se posso dimostrare β a partire da $\alpha_1, \alpha_2, \dots, \alpha_m$, senza usare R, allora R è una regola derivabile

[Es. Sistema $\mathcal{M}$

Ax $0 \in \mathbb{N}$

R $\frac{x \in \mathbb{N}}{S(x) \in \mathbb{N}}$

Consideriamo R_1 $\frac{x \in \mathbb{N}}{S(S(x)) \in \mathbb{N}}$

R_1 è derivabile perché posso ottenere $S(S(x)) \in \mathbb{N}$ a partire da $x \in \mathbb{N}$

DIMOSTRIAMO:

1. $x \in \mathbb{N}$
2. $S(x) \in \mathbb{N}$
3. $S(S(x)) \in \mathbb{N}$

dunque $x \in \mathbb{N} \vdash_{\mathcal{M}} S(S(x)) \in \mathbb{N}$]

Usiamo queste regole per semplicità (per evitare di fare sempre tutti questi passaggi).

ESERCIZIO 100: Dimostriamo che la fbf $(KSK)(KKK) = SK$ è un teorema del sistema formale CL

$KSK = S$	Ax K
$KKK = K$	Ax K
$(KSK)(KKK) = (KSK)(KKK)$	Ax rifl
$(KSK)(KKK) = S(KKK)$	(CONGR 2) (1,3)
$(KSK)(KKK) = SK$	(CONGR 1) (4,2)

ESERCIZIO 101: Aggiungiamo la regola (CONG) $\dfrac{M=N \quad P=Q}{MP=NQ}$
si dimostri $(KSK)(KKK) = SK$

$KSK = S$	Ax K
$KKK = K$	Ax K
$KSKKKK = SK$	CONGR

ESERCIZIO 102: Chiamiamo CL' il sistema uguale a CL ma con (CONG) in più
L'insieme dei teoremi di CL è uguale all'insieme di teoremi di CL'
DOBBIAMO DIMOSTRARE CHE: $\{\alpha \in W \mid \vdash_{CL} \alpha\} = \{\alpha' \in W \mid \vdash_{CL'} \alpha'\}$
Basta dimostrare che la regola (CONG) è derivabile, quindi non porta niente di nuovo.

CONG $\dfrac{M=N \quad P=Q}{MP=NQ}$

Dunque $M=N, P=Q \vdash_{CL} MP=NQ$

1. $M=N$ ipotesi
2. $P=Q$ ipotesi
3. $MP=MP$ Axrefl
4. $MP=MQ$ (CONGR 2)(2,3)
5. $MP=NQ$ (CONGR 1)(1,4)

ESERCIZIO 103: $\{K=SK\} \vdash_{CL} P=Q$ Riempi gli spazi vuoti

1. $K=SK$ (IPOTESI)
2. $KP=KP$ (Axrefl)
3. $KP=SKP$ (Cong 2)(1,2)
4. $KPQ=KPQ$ (Axrefl)
5. $KPQ=SKPQ$ (Cong 1)(3,4)
6. $KPQ=P$ (Ax K)
7. $SKPQ=KPQ$ (Symm)(5)
8. $SKPQ=P$ (Trans)(6,7)
9. $SKPQ=KQ(PQ)$ (Ax S)
10. $KQ(PQ)=Q$ (Ax K)
11. $SKPQ=Q$ (Trans)(9,10)
12. $P=SKPQ$ (Symm)(8)

13. $P = Q$ (Trans) (11, 12)

Esercizio 35.3: Sistema formale D

Ax $\dfrac{}{aaa}$

R $\dfrac{w}{waa}$ w è una qualsiasi sequenza di sole a

Consideriamo R' $\dfrac{b}{baa}$ è derivabile in D?

R' non è derivabile: nel nostro sistema dovrebbe esistere $\{b\} \vdash_D baa$, cioè senza usare R_1, dovremmo essere in grado di ottenere baa partendo dalla sola ipotesi b. Poiché nel sistema abbiamo solo regole che ci portano ad avere stringhe di a concatenate, non possiamo derivare baa.

Lezione 6 17/10/2018

lunedì 24 settembre 2018 20:54

MADONIA (Pg. 73 circa)

CONFIGURAZIONE: Dato un ASFD $(\Sigma, Q, \delta, q_0, F)$, una configurazione di $\mathcal{A}$ è una coppia (q, x) con $q \in Q$ e $x \in \Sigma^*$

Se ci fosse un carattere non appartente al linguaggio, la compilazione si blocca perché la funzione δ non può determinare uno stato successivo. LA STRINGA NON VIENE ACCETTATA.

Possiamo avere anche la parola VUOTA, cioè nel caso in cui terminiamo

DEFINIZIONE FORMALE DI CONFIGURAZIONE:

Una configurazione (q, x) con $q \in Q$ e $x \in \Sigma^*$ di $\mathcal{A}$, è detta:

- INIZIALE se $q = q_0$
- FINALE se $x = \varepsilon$
- ACCETTANTE se $q \in F$ e $x = \varepsilon$

DEFINIZIONE DI CONFIGURAZIONI SUCCESSIVE:

Dato un ASFD, $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ e due configurazioni (q, x) e (q_1, y),

avremo che $\underbrace{(q, x) \vdash_{\mathcal{A}} (q_1, y)}_{(q_1, y) \text{ segue } (q, x)}$ $\iff$ valgono le seguenti condizioni:

1) $\exists a \in \Sigma \mid x = ay \longrightarrow$ se $y = cc$, $x = acc$
2) $\delta(q, a) = q_1$

DEFINIZIONE: $c \vdash_{\mathcal{A}} c_1 \vdash_{\mathcal{A}} c_2 \vdash_{\mathcal{A}} \dots \vdash_{\mathcal{A}} c_m \Rightarrow c \overbrace{\vdash_{\mathcal{A}}^{*}}^{\text{con tot passi}} c_m$

Siano c e c' configurazioni, diciamo che da c passo a c' in più passi se esistono $c_0, c_1, \dots, c_m$ configurazioni tali che $c_0 = c$, $c_m = c'$ e $c_i \vdash_{\mathcal{A}} c_{i+1}$ con $i = 0, 1, 2, \dots, m-1$

Allora potremo scrivere $c \vdash_{\mathcal{A}}^{*} c'$

STRINGA X ACCETTATA DAL SISTEMA:

Una stringa $x \in \Sigma^*$ è accettata da un ASFD $\mathcal{A} \iff \overbrace{(q_0, x) \vdash_{\mathcal{A}}^{*} (q, \varepsilon)}^{\text{se dalla config. iniziale in più passi raggiungo la config. finale}}$ con $q \in F$

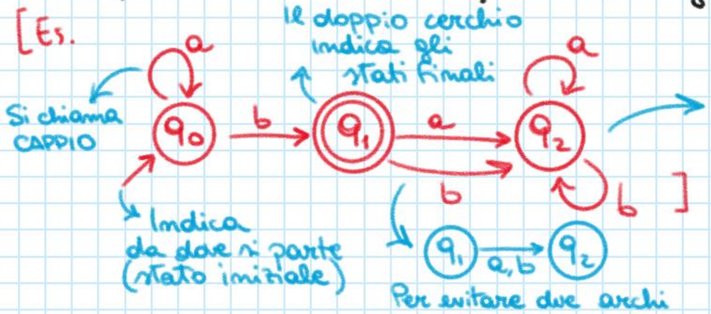
LINGUAGGIO L RICONOSCIUTO DAL SISTEMA:

Un linguaggio è riconosciuto in A se
$L(A) = \{x \in \Sigma^* \mid (q_0, x) \vdash_A^* (q, \varepsilon) \text{ con } q \in F\}$ → È formato da sole stringhe x accettate dal sistema A

N.B. Possono esistere più ASFD diversi che riconoscono lo stesso linguaggio.

Un automa si può rappresentare tramite un GRAFO DI TRANSIZIONE DI UN AUTOMA:
È composto da tanti nodi quanti sono gli stati e degli archi per i valori di δ.

[Es.



oppure per evitare due cappi.]

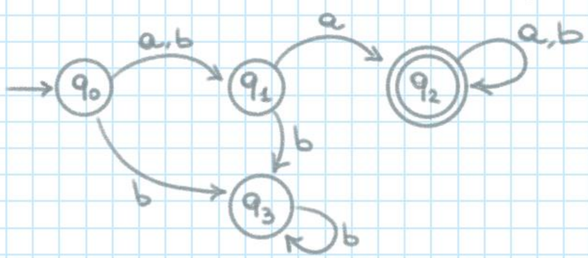
FUNZIONE DI TRANSIZIONE ESTESA:

Data la funzione di transizione $\delta: Q \times \Sigma \to Q$ (definita in $Q \times \Sigma$),
Chiamiamo FUNZIONE DI TRANSIZIONE ESTESA $\delta^*: Q \times \Sigma^* \to Q$. Si indica con $\delta^*(q, x)$.
Questa non legge simboli $(a \in \Sigma)$, ma direttamente stringhe $(x \in \Sigma^*)$.

[Es. Nel nostro esempio:
$\delta^*(q_0, aa) = q_0$ → ① $\delta(q_0, a) = q_0$, ② $\delta(q_0, a) = q_0$
$\delta^*(q_0, baa) = q_2$ → ① $\delta(q_0, b) = q_1$, ② $\delta(q_1, a) = q_2$, ③ $\delta(q_2, a) = q_2$]

Possiamo definire allora una stringa x accettabile se: $\delta^*(q_0, x) = q$, con $q \in F$

ESERCIZIO: Scrivere un ASFD su (a,b) che accetta tutte le stringhe che iniziano per aa



δ	a	b
q_0	q_1	q_3
q_1	q_2	q_3
q_2	q_2	q_2
q_3	q_3	q_3

Lezione 7 22/10/2018

lunedì 24 settembre 2018 20:54

BARBANERA

DERIVABILITA' E AMMISSIBILITA'

Una regola R è derivabile se nel sistema, partendo dalle stesse premesse di R, giungiamo alle conclusioni di R, ma senza utilizzarla.

$R\ \frac{\alpha_1, \alpha_2, \alpha_3, \dots \alpha_m}{\beta}$ è derivabile in D se $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_m \vdash_D \beta$

Una regola R è ammissibile se $\vdash_{D+R} \alpha$ e $\vdash_D \alpha$ (α è derivabile in D+R, ma lo è anche solamente in D)

Si dimostra che ogni regola derivabile è anche ammissibile.

N.B. Non vale il viceversa

DEFINIZIONE FORMALE di MONDO

Un mondo è una funzione B che assegna ad ogni variabile propositionale il valore di vero o falso (per convenzione falso = 0, vero = 1).

$B: \text{var prop} \rightarrow \{0, 1\}$

Queste funzioni vengono chiamate ASSEGNAMENTI PROPOSIZIONALI

DEFINIZIONE DI VERO / FALSO

Bisogna definire un metodo per associare ad ogni fbf un elemento di $\{0, 1\}$

Preso un mondo B, definiamo una funzione $\overline{B}$ che rappresenta il concetto di verità nel mondo B.

$\overline{B}: W \rightarrow \{0, 1\}$ (ad ogni fbf $\in W$ il valore di vero o falso)

Le fbf possono essere:

1) p Una qualsiasi variabile proposizionale
2) $\neg\alpha$ Negazione di una fbf
3) $\alpha \rightarrow \beta$ Una fbf ne implica un'altra

Dunque, preso un mondo B, definiamo le condizioni di verità $\overline{B}$ per queste fbf.

- $\overline{B}(p) = B(p)$ Il valore di verità della variabile p, dipende dal valore che il mondo B le assegna
- $\overline{B}(\neg\alpha) = \begin{cases} 0 & \text{se } \overline{B}(\alpha) = 1 \\ 1 & \text{se } \overline{B}(\alpha) = 0 \end{cases}$ → [Si può scrivere come $\overline{B}(\neg\alpha) = -\overline{B}(\alpha)$
  Se nel mondo B α è vero, $\neg\alpha$ assume il valore di verità opposto

- $\overline{B}(\alpha \to \beta) = \begin{cases} 0 & \text{se } \overline{B}(\alpha)=1 \text{ e } \overline{B}(\beta)=0 \\ 1 & \text{altrimenti } (\text{se } \overline{B}(\alpha)=1 \Rightarrow \overline{B}(\beta)=1) \end{cases}$

anche perché $\wedge$ non appartiene all'alfabeto S

Ci accorgiamo che fbf come $\alpha \wedge \beta$ sono superflue, perché sostituibili con fbf già presenti

$\overline{B}(\alpha \wedge \beta) = \begin{cases} 1 & \text{se } \overline{B}(\alpha)=1 \text{ e } \overline{B}(\beta)=1 \\ 0 & \text{altrimenti} \end{cases}$

Possiamo sostituire $\alpha \wedge \beta$ con una fbf accettabile nel nostro sistema, come $\neg(\alpha \to \neg\beta)$

$\overline{B}(\neg(\alpha \to \neg\beta)) = \begin{cases} 1 & \text{se } \overline{B}(\alpha \to \neg\beta)=0 \\ 0 & \text{altrimenti} \end{cases}$ → E quand'è che $\overline{B}(\alpha \to \neg\beta)=0$?

$\overline{B}(\alpha \to \neg\beta) = \begin{cases} 0 & \text{se } \overline{B}(\alpha)=1 \text{ e } \overline{B}(\neg\beta)=0 \\ 1 & \text{altrimenti} \end{cases}$ → $\overline{B}(\neg\beta)=0$ se $\overline{B}(\beta)=1$

Dunque, così come per $\overline{B}(\alpha \wedge \beta)$, $\overline{B}(\neg(\alpha \to \neg\beta))=1$ se $\overline{B}(\alpha)=1$ e $\overline{B}(\beta)=1$

Definiamo propositionalmente come può essere detta una fbf.

Una fbf del sistema P_0 può essere detta:

- TAUTOLOGIA se $\forall B$, $\overline{B}(\alpha)=1$ se α è vera in qualsiasi mondo
- SODDISFACIBILE se $\exists B \mid \overline{B}(\alpha)=1$ se esiste anche solo un mondo in cui α è vera
- CONTRADDITTORIA se $\nexists B \mid \overline{B}(\alpha)=1$ se non esiste un mondo in cui α è vera.

N.B. Se α è una tautologia, è anche soddisfacibile

N.B. Se α è contraddittoria, non è soddisfacibile

N.B. Se $\forall B$, $\overline{B}(\neg\alpha)=0$, α è una tautologia.

CONSEGUENZA TAUTOLOGICA ($\Gamma \models \alpha$) → α è una conseguenza tautologica di Γ se tutti e soli i mondi che rendono vere le formule di Γ, rendono vera anche α

CORRETTEZZA

Dato un insieme Γ di fbf, il sistema P_0 è corretto se $\Gamma \vdash_{P_0} \alpha \Rightarrow \Gamma \models \alpha$

Vedremo la prossima lezione perché.

α è una tautologia se $\models \alpha$ (ovvero $\emptyset \models \alpha$, poiché in tutti i mondi vale $\emptyset$)

Tutti gli assiomi sono tautologie.

DIMOSTRAZIONE: Data una fbf di P_0, $\overline{B}(\alpha)$ dipende solo dal valore assegnato da B alle variabili proposizionali che compongono α.

[Es. $(p \to p) \to q$ è vera, cioè $\overline{B}((p \to p) \to q) = 1$?
Dipende dai valori che B dà a p e q]

α di P_0 è decidibile se è una tautologia

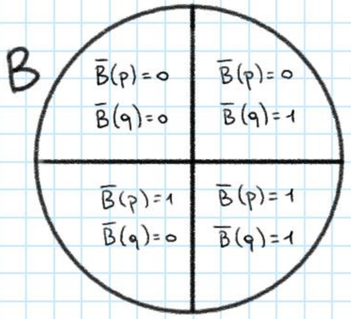
Quindi l'insieme delle tautologie di P_0 è decidibile.

Per potere stabilire se α è una tautologia dovremmo verificare la sua veridicità in ogni mondo possibile.

Non potendo valutare infiniti mondi, procediamo nel seguente modo:

Prendiamo l'fbf α (es. α: $(p \to p) \to q$). Sappiamo che il valore di verità dipende solo dal mondo B che ne assegna il valore.

Allora dividiamo l'insieme infinito di tutti i possibili mondi in 4 parti, in base ai possibili valori di p e q.



Questi sono i possibili valori assunti da p e q

Dopodiché si fa la tabella di verità

p	q	$(p \to p) \to q$
0	0	0
0	1	1
1	0	0
1	1	1

1) $p=0$ e $q=0$

falso → falso
$(p \to p)$, è vero che $F \to F$

vero → falso
$(p \to p) \to q$, è falso che $V \to F$

2) $p=0$ e $q=1$

falso → falso
$(p \to p)$, è vero che $F \to F$

vero → vero
$(p \to p) \to q$, è vero che $V \to V$

3) $p=1$ e $q=0$

vero → vero
$(p \to p)$, è vero che $V \to V$

vero → falso
$(p \to p) \to q$, è falso che $V \to F$

4) $p=1$ e $q=1$

vero → vero
$(p \to p)$, è vero che $V \to V$

vero → vero
$(p \to p) \to q$, è vero che $V \to V$

Allora α è SODDISFACIBILE perché $\exists B \mid \bar{B}(\alpha) = 1$,
ma non è una TAUTOLOGIA perché $\exists B \mid \bar{B}(\alpha) = 0$

N.B. Se ogni caso fosse stato 0, α sarebbe stata contraddittoria.

Lezione 8 29/10/2018

lunedì 24 settembre 2018 20:54

BARBANERA

TEOREMA DI CORRETTEZZA

Se $\Gamma \vdash \alpha \Rightarrow \Gamma \models \alpha$

Se α è derivabile da Γ, α è conseguenza Tautologica di Γ

Si dimostra per induzione sulla lungezza della derivazione

Derivazione: $\alpha_1, \alpha_2, \dots, \alpha_m$ con $\alpha_m \equiv \alpha \mid m \geq 1$

$$\frac{\varphi(1) \qquad [\forall m(\forall k<m.\ \varphi(k)) \rightarrow \varphi(m)]}{\forall m \geq 1.\ \varphi(m)}$$

Caso base: $m=1$, la derivazione presenta solo una fbf.

allora α può essere un assioma ($\alpha \in Ax$)

Dimostriamo allora che α è una tautologia.

Si dimostra con la tabella di verità per i 3 assiomi possibili ($AK, AS, A\neg$)

Se α è tautologia, $\models \alpha$, a maggior ragione sarà vera nei mondi in cui Γ è vera ($\Gamma \models \alpha$)

Caso induttivo:

- Se $\alpha \in \Gamma$, $\Gamma \models \alpha$
- Se $\alpha \in Ax$, $\models \alpha \Rightarrow \Gamma \models \alpha$ (se α è un assioma, sarà una tautologia, cioè è vera sempre, dunque anche nei mondi in cui Γ è vera)
- Se α è conseguenza di $MP(i,j)$, i deve avere la forma β e j deve avere la forma $\beta \rightarrow \alpha$.

DEDUZIONE

1	—	
⋮	—	
i	β	β deve derivare da passi precedenti ($\Gamma \models \beta$), allora $i<m$
⋮	—	
j	$\beta \rightarrow \alpha$	$\beta \rightarrow \alpha$ deve derivare da passi precedenti ($\Gamma \models \beta \rightarrow \alpha$), allora $j<m$
⋮	—	
m	α	Se $\Gamma \models \beta$, $\Gamma \models \beta \rightarrow \alpha$ e $\beta, \beta \rightarrow \alpha \models \alpha$, allora $\Gamma \models \alpha$

TEOREMA DI DEDUZIONE DI HERBRAND

$\Gamma \vdash_{P_0} \alpha \to \beta \iff \Gamma, \alpha \vdash_{P_0} \beta$

La dimostrazione si divide in 2 parti ($\Rightarrow$ e $\Leftarrow$).

- $\Rightarrow$ $\left(\Gamma \vdash_{P_0} \alpha \to \beta \Rightarrow \Gamma, \alpha \vdash_{P_0} \beta\right)$

1	—	
2	—	
.	—	
.	—	
.	—	
.	—	
K	$\alpha \to \beta$	
K+1	α	Ipotesi
K+2	β	MP(K, K+1)

da un sottoinsieme di Γ deriviamo $\alpha \to \beta$

- $\Leftarrow$ $\left(\Gamma, \alpha \vdash_{P_0} \beta \Rightarrow \Gamma \vdash_{P_0} \alpha \to \beta\right)$

Ragioniamo per induzione sulla lunghezza della deduzione di β in n passi.

- CASO BASE: $n = 1$

β può essere un assioma o un'ipotesi, oppure è l'ipotesi α stessa ($\beta = \alpha$).

Se β è assioma o ipotesi, deriviamo $\alpha \to \beta$ in questo modo:

1	β	Assioma / Ipotesi
2	$\beta \to (\alpha \to \beta)$	Ak
3	$\alpha \to \beta$	MP(1,2)

Se invece $\beta = \alpha$ potremo scrivere $\alpha \to \alpha$, poiché è un teorema in P_0.
Sapendo che $\beta = \alpha$ potremo allora scrivere tranquillamente $\alpha \to \beta$.

- IPOTESI INDUTTIVA: $n = m \mid 1 < m < n$ (Deduzioni più corte di $\Gamma, \alpha \vdash \beta$)

Chiamiamo i vari passi della deduzione che ha come conclusione β $\beta_1, \beta_2, \beta_3, \dots, \beta_m$ con $\beta_m = \beta$; supponiamo che l'ipotesi sia vera per tutte le deduzioni lunghe $m < n$. Potremo allora ottenere $\beta_1, \beta_2, \dots, \beta_{m-1}$ come conclusioni di deduzioni più corte di $\Gamma, \alpha \vdash \beta$.

- TESI:

Poiché abbiamo solo la regola MP $\frac{A \quad A \to B}{B}$, tutte le FbF β_x saranno della forma β_x o della formula $\beta_y \to \beta_z$. Poiché per ipotesi deriviamo β, avremo sicuramente due FbF precedenti β_i e β_j del tipo β_i e $\beta_j = \beta_i \to \beta$. Per ipotesi induttiva possiamo avere $\Gamma \vdash \alpha \to \beta_i$ e $\Gamma \vdash \alpha \to \beta_j$ (Poiché β_i e β_j sono conclusioni di prove di lunghezza $m < n$).

Dunque avremo $\alpha \to \beta_i$ e $\alpha \to (\beta_i \to \beta)$.

Grazie all'assioma AS potremo scrivere:

⋮		
k	$\alpha \to \beta_i$	MP(...,...)
k+1	$\alpha \to (\beta_i \to \beta)$	MP(...,...)
k+2	$(\alpha \to (\beta_i \to \beta)) \to ((\alpha \to \beta_i) \to (\alpha \to \beta))$	AS
k+3	$(\alpha \to \beta_i) \to (\alpha \to \beta)$	MP(k+1, k+2)
k+4	$\alpha \to \beta$	MP(k, k+3)

Dunque abbiamo dedotto $\alpha \to \beta$ senza usare α.

I mondi B sono statici: al passare del tempo i concetti di verità restano invariati.
Esistono delle LOGICHE TEMPORALI in cui il tempo implica delle modifiche.
Esistono anche LOGICHE COSTRUTTIVE, come la logica INTUIZIONISTA. Sono logiche in cui, se voglio dimostrare l'esistenza di un oggetto con delle caratteristiche, devo dimostrare di poterlo costruire. Si ragiona per assurdo. Dire "posso costruirlo", significa che posso produrre un algoritmo che mi faccia ottenere quell'oggetto.

TEOREMA

Un insieme Γ di fbf è contraddittorio se e solo se esiste α tale che $\Gamma \vdash_{P_0} \alpha$, $\Gamma \vdash_{P_0} \neg\alpha$ (equivalente a dire $Con(\Gamma) = W$)

Dunque Γ contraddittorio $\iff \Gamma \vdash_{P_0} \alpha, \Gamma \vdash_{P_0} \neg\alpha$

- $\Rightarrow$

Se Γ è contraddittorio, possiamo derivare qualsiasi fbf di W dunque anche α e $\neg\alpha$ $(\Gamma \vdash \alpha, \Gamma \vdash \neg\alpha)$

- $\Leftarrow$

* Si dimostra che $\alpha, \neg\alpha \vdash \beta$, per ogni β. Poiché $\Gamma \vdash \alpha$ e $\Gamma \vdash \neg\alpha$, per l'eliminazione dei lemmi, $\Gamma \vdash \beta$ per ogni β. Dunque Γ permette di derivare ogni cosa (è inconsistente).

COROLLARIO

Un sistema P_0 è inconsistente se e solo se $\exists \alpha \mid \vdash_{P_0} \alpha$ e $\vdash_{P_0} \neg\alpha$

- $\Rightarrow$

Se P_0 è contraddittorio, $Con(\emptyset) = W$, dunque tutte le fbf sono teoremi di P_0.

- $\Leftarrow$

* Si dimostra che $\alpha, \neg\alpha \vdash \beta$, per ogni β, e poiché $\vdash \alpha$ e $\vdash \neg\alpha$, allora $\vdash \beta$ per ogni β.

* DIMOSTRAZIONE $\vdash \alpha \to (\neg\alpha \to \beta) = \alpha, \neg\alpha \vdash \beta$

Usiamo come ipotesi α e $\neg\alpha$.

1	$\alpha \to (\neg\beta \to \alpha)$	Ak
2	α	ipotesi 1
3	$\neg\beta \to \alpha$	MP (1,2)
4	$\neg\alpha \to (\neg\beta \to \neg\alpha)$	Ak
5	$\neg\alpha$	ipotesi 2
6	$\neg\beta \to \neg\alpha$	MP(4,5)
7	$(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)$	A $\neg$
8	$(\neg\beta \to \alpha) \to \beta$	MP (6,7)
9	β	MP (3,8)

Lezione 9 31/10/2018

lunedì 24 settembre 2018 20:54

MADONIA (Pag 74, 77)

ESERCIZIO: ASFD che riconosce il linguaggio L di tutte le stringhe che contengono un numero pari di "a" o di "b".

Per Casa

DEFINIZIONE FORMALE DI FUNZIONE DI TRANSIZIONE ESTESA $\bar{\delta}$

→ N.B. Già vista nella lezione precedente, ma era stata chiamata δ^*.

E' la funzione $\bar{\delta} : Q \times \Sigma^* \to Q$ così definita.

- $\bar{\delta}(q, \varepsilon) = q \quad \forall q \in Q$
- $\bar{\delta}(q, xa) = \delta(\bar{\delta}(q, x), a)$ con $a \in \Sigma$, $x \in \Sigma^*$

A differenza di δ, che lavora con simboli, $\bar{\delta}$ lavora direttamente con stringhe.

[Es.

δ	a	b
q_0	q_1	q_3
q_1	q_2	q_3
q_2	q_2	q_2
q_3	q_3	q_3

$$\bar{\delta}(q_0, \overbrace{aa}^{\in \Sigma^*}\overbrace{b}^{\in \Sigma}) =$$

$$= \delta(\bar{\delta}(q_0, aa), b) =$$

$$= \delta(\delta(\bar{\delta}(q_0, a), a), b) =$$

$$= \delta(\delta(\delta(\bar{\delta}(q_0, \varepsilon), a), a), b =$$

$= \delta(\delta(\delta(q_0, a), a), b) =$

$= \delta(\delta(q_1, a), b) =$

$= \delta(q_2, b) = q_2$

Poiché $q_2 \in F$, aab viene accettata]

Quindi un linguaggio riconosciuto è $L = \{x \in \Sigma^* \mid \bar{\delta}(q_0, x) = q \in F\}$

Equivale a dire $L = \{x \in \Sigma^* \mid (q_0, x) \vdash^* (q, \varepsilon),\ q \in F\}$

Possono esistere diversi automi che riconoscono lo stesso linguaggio.

Ci sono anche linguaggi per cui non esistono ASFD che li riconoscono.

[Es. $a^n b^n$ non può essere riconosciuta]

Consideriamo l'insieme R dei linguaggi riconosciuti

$R = \{L \mid L \subseteq \Sigma^* \text{ e } \exists \text{ un ASFD} \mid L = L(A)\}$

Cioè R è l'insieme dei linguaggi per cui esiste un ASFD che li riconosce

R viene detto **CLASSE DEI LINGUAGGI REGOLARI**

AUTOMI A STATI FINITI NON DETERMINISTICI ASFND (Pag. 63, 77)

A differenza degli ASFD, da una configurazione, se ne possono raggiungere diverse successive. Ad una coppia stato, simbolo vengono associati più stati.

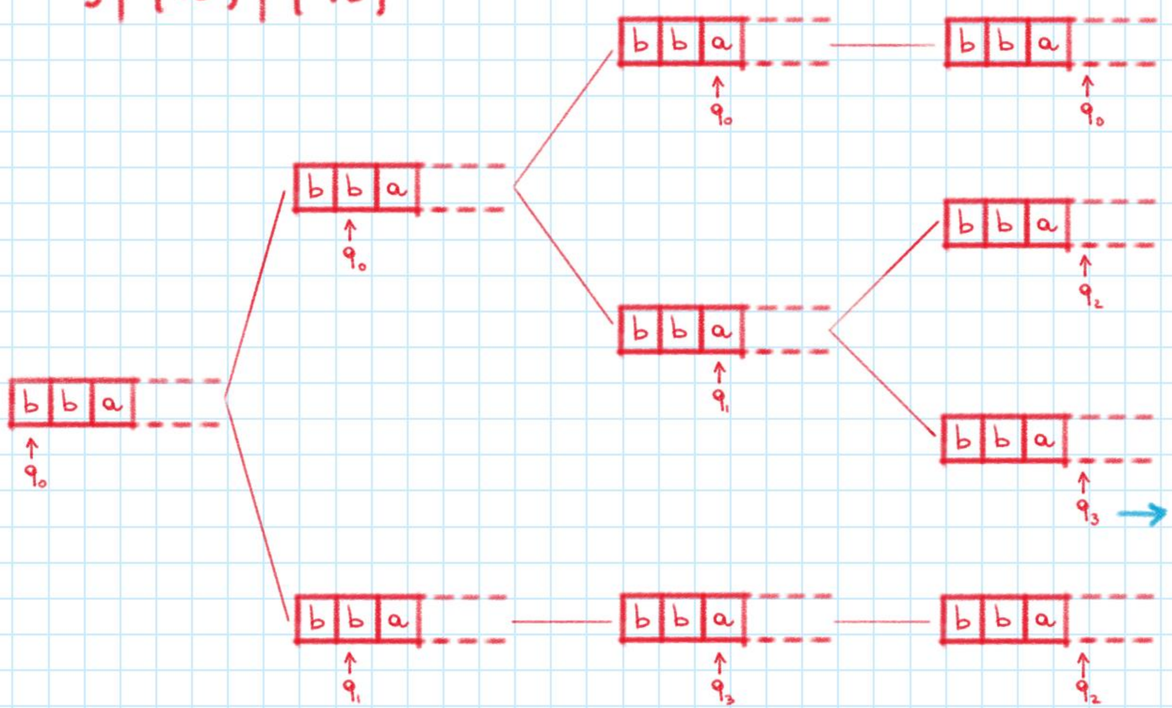
Se da una configurazione posso raggiungere più stati, è come se l'automa

si clonasse per raggiungere tutti gli stati.

Solitamente la funzione viene indicata con δ_N

[Es. ASFND $x = bba$ $F = \{q_3\}$

δ_N	a	b
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2, q_3\}$	$\{q_3\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\{q_2\}$	$\{q_2\}$



$\rightarrow q_3$ rientra negli stati finali ammessi

]

Lezione 10 05/11/2018

lunedì 24 settembre 2018 20:54

BARBANERA

DEDUZIONE NATURALE

Definiamo un nuovo tipo di sistema formale DN, in cui cambiano alcuni concetti:

Le deduzioni non sono sequenze di fbf, bensì alberi di fbf.

Cambiano anche le regole di inferenza.

La **RADICE** dell'albero è la conclusione della deduzione.

Le **FOGLIE** sono assiomi o ipotesi.

L'insieme dei simboli è $S = \{(,), \rightarrow, \neg, p, q, \wedge, \vee, \perp, \dots\}$

Il simbolo $\perp$ è detto **BOTTOM**, e non è un connettivo logico,

bensì una variabile proposizionale con il valore di "sempre falso" (ASSURDO).

Ricordiamo che:

- $\overline{B}(\alpha \wedge \beta) = \begin{cases} 1 & \text{se } \overline{B}(\alpha)=1 \text{ e } \overline{B}(\beta)=1 \\ \text{altrimenti} & \end{cases}$

- $\overline{B}(\alpha \vee \beta) = \begin{cases} 0 & \text{se } \overline{B}(\alpha)=0 \text{ e } \overline{B}(\beta)=0 \\ \text{altrimenti} & \end{cases}$

- $\overline{B}(\perp) = 0$

REGOLE DI INFERENZA

Le regole di inferenza sono molte e sono di due tipi:

- DI INTRODUZIONE (I): danno significato ai connettivi logici
- DI ELIMINAZIONE (E): dicono come usare i connettivi logici

- Regola $\wedge$I - Introduzione alla congiunzione

$$\frac{P \quad Q}{P \wedge Q}$$

- Regola $\wedge$E - Eliminazione della congiunzione

$$\frac{P \wedge Q}{P}$$

$$\frac{P \wedge Q}{Q}$$

- Regola $\vee$I - Introduzione alla disgiunzione

$$\frac{P}{P \vee Q}$$

$$\frac{Q}{P \vee Q}$$

- Regola $\rightarrow$E - Eliminazione della implicazione

$$\frac{P \quad P \rightarrow Q}{Q}$$

N.B. Non è altro che la regola MP (Modus Ponens) vista in P_0

- Regola $\to$ I - Introduzione alla implicazione

Utilizziamo il teorema di deduzione:

$\Gamma, \alpha \vdash \beta \iff \Gamma \vdash \alpha \to \beta$

Dunque se da un insieme di ipotesi Γ e dall'ipotesi α deriviamo β, allora potremo derivare da Γ $\alpha \to \beta$, ma senza usare α come ipotesi.

$$\frac{\begin{array}{c}[P]\\ \vdots \\ Q\end{array}}{P \to Q}$$

Vediamola per gradi:

1) Derivo Q da P e un insieme Γ di ipotesi

$$\begin{array}{c}P\\ \vdots \\ Q\end{array}$$

Con questo simbolo indico la derivazione da un insieme Γ $(\Gamma, P \vdash Q)$

2) Derivo $P \to Q$ da Γ senza usare P

$$\frac{\begin{array}{c}[P]\\ \vdots \\ Q\end{array}}{P \to Q}$$

$\to$ Con questo simbolo indico che ho "scaricato" P dalle mie ipotesi

$(\Gamma \vdash P \to Q)$

- Regola $\vee$E - Eliminazione della disgiunzione

$$\frac{P \vee Q \qquad \begin{array}{c}[P]\\ \vdots \\ R\end{array} \qquad \begin{array}{c}[Q]\\ \vdots \\ R\end{array}}{R}$$

Se posso dire $P \vee Q$, e deduco R sia da Γ e P, sia da Γ e Q, allora potrò affermare R scaricando P e Q

- Regola RAA - Reductio ad absurdum

$$\frac{\begin{array}{c}[\neg P]\\ \vdots \\ \bot\end{array}}{P}$$

Se da $\neg P$ arrivo all'assurdo, posso affermare P scaricando $\neg P$

- Regola $\bot$ - Regola del falso

$$\frac{\bot}{P}$$

Dal falso posso derivare qualsiasi cosa

Il sistema P_0 e il sistema DN sono equivalenti dal punto di vista della derivabilità:

se $\Gamma \vdash_{P_0} \alpha \Rightarrow \Gamma \vdash_{DN} \alpha$ e viceversa

Alla stessa maniera, tutti i teoremi di P_0 sono teoremi di DN e viceversa

In DN non ci sono assiomi.

Poiché se $\alpha \in Ax$ in P_0, allora $\vdash_{P_0} \alpha$, avremo $\vdash_{DN} \alpha$, dunque esisterà una deduzione che avrà α come conclusione

- AS

$\vdash_{DN} (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$

Per il teorema di deduzione di Herbrand:

$\alpha \to (\beta \to \gamma) \vdash_{DN} (\alpha \to \beta) \to (\alpha \to \gamma)$

$\alpha \to (\beta \to \gamma), \alpha \to \beta \vdash_{DN} \alpha \to \gamma$

$\alpha \to (\beta \to \gamma), \alpha \to \beta, \alpha \vdash_{DN} \gamma$ → Avremo queste ipotesi

$$\dfrac{\dfrac{\alpha \quad \alpha \to (\beta \to \gamma)}{\beta \to \gamma}(\to E) \qquad \dfrac{\alpha \quad \alpha \to \beta}{\beta}(\to E)}{\gamma}(\to E)$$

Ora dimostriamo l'inverso scaricando le ipotesi

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\alpha]_1 \quad [\alpha \to (\beta \to \gamma)]_3}{\beta \to \gamma} \qquad \dfrac{[\alpha]_1 \quad [\alpha \to \beta]_2}{\beta}}{\gamma}}{\alpha \to \gamma}(1)}{(\alpha \to \beta) \to (\alpha \to \gamma)}(2)}{\alpha \to (\beta \to \gamma) \to ((\alpha \to \beta) \to (\alpha \to \gamma))}(3)$$

Scarichiamo via via tutte le ipotesi della deduzione usando la regola di introduzione alla implicazione (Regola → I)

- A¬

$\vdash_{DN} (\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)$

$\neg\beta \to \neg\alpha \vdash_{DN} (\neg\beta \to \alpha) \to \beta$

$\neg\beta \to \neg\alpha,\ \neg\beta \to \alpha \vdash_{DN} \beta$

Usiamo anche $\neg\beta$ come ipotesi

$$
\dfrac{\dfrac{\dfrac{\dfrac{[\neg\beta]_1 [\neg\beta \to \neg\alpha]_3}{\neg\alpha} \quad \dfrac{[\neg\beta]_1 \quad [\neg\beta \to \alpha]_2}{\alpha}}{\bot}}{\beta}\,(1)}{(\neg\beta \to \alpha) \to \beta}\,(2)}{(\neg\beta \to \neg\alpha) \to ((\neg\beta \to \alpha) \to \beta)}\,(3)
$$

→ non posso affermare α e $\neg\alpha$

(1) → Regola RAA

- Ak

$\vdash_{DN} \alpha \to (\beta \to \alpha)$

$\alpha \vdash_{DN} \beta \to \alpha$

$\alpha, \beta \vdash_{DN} \alpha$

Vediamo α come una brevissima deduzione in cui α è sia ipotesi che conclusione, e non usiamo β.

$$\dfrac{\dfrac{[\beta]_1 \quad [\alpha]_2}{\beta \to \alpha}\,(1)}{\alpha \to (\beta \to \alpha)}\,(2)$$

α, β ipotesi

Dimostriamo $\alpha \to \alpha$ in DN

$\vdash_{DN} \alpha \to \alpha$

$\alpha \vdash_{DN} \alpha$

$$\dfrac{[\alpha]_1}{\alpha \to \alpha}\,(1)$$

Ipotesi e conclusione

Lezione 11 07/11/2018

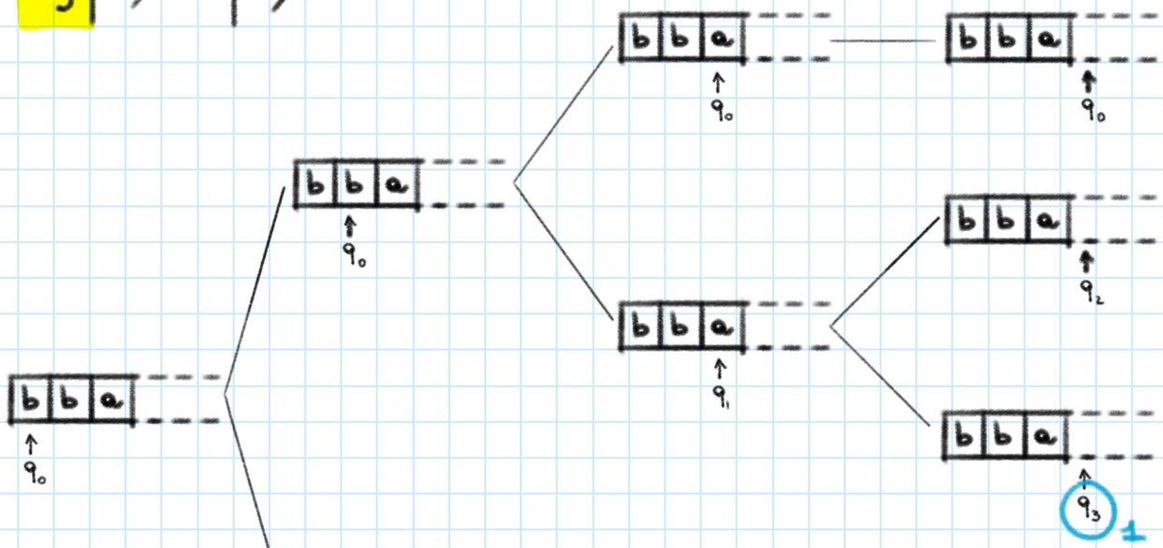
lunedì 24 settembre 2018 20:54

MADONIA (Pg. 78)

Prendiamo un esempio di ASFND N.B. E' un modello puramente teorico.

$\Sigma = \{a, b\}$ $F = \{q_3\}$ $x = bba$

δ_N	a	b
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2, q_3\}$	$\{q_3\}$
q_2	$\{q_3\}$	$\emptyset$
q_3	$\emptyset$	$\emptyset$



Quando $\delta(q_3, a) = \emptyset$ la computazione si blocca

N.B. q_3 è lo stato finale, ma nel caso 1 si termina con la configurazione di accettazione $(\delta(q_3, \varepsilon))$, mentre nel caso 2 non viene accettata perché non si raggiunge la configurazione di accettazione, infatti si ha $\delta(q_3, a)$.

Basta che esiste un ramo dell'albero che termina con una configurazione di accettazione per poter dire che la stringa x viene accettata dal ASFND.

DEFINIZIONE FORMALE DI ASFND

Un ASFND è una quintupla $\mathcal{A} = (\Sigma, Q, \delta_N, q_0, F)$ dove:

- Σ, Q, q_0, F sono definiti come per gli ASFD;
- $\delta_N : Q \times \Sigma \longrightarrow \underbrace{P(Q)}$ è una funzione di transizione.

è l'insieme delle parti di Q

Anche in questo caso esiste $\bar{\delta}_N$

DEFINIZIONE FORMALE DI FUNZIONE DI TRANSIZIONE ESTESA (ASFND)

Dato un ASFND, la funzione di transizione estesa è la funzione

$\bar{\delta}_N : Q \times \Sigma^* \longrightarrow P(Q)$ così definita:

- $\bar{\delta}_N(q, \varepsilon) = \{q\} \quad \forall q \in Q$ scriviamo $\{q\}$ perché si associa un insieme di stati

- $\bar{\delta}_N(q, xa) = \bigcup_{p \in \bar{\delta}_N(q,x)} \delta_N(p, a) \qquad \forall x \in \Sigma^*, a \in \Sigma$

[Es. $\bar{\delta}_N(q_0, \underbrace{bb}_{x \in \Sigma^*}\underbrace{a}_{a \in \Sigma}) = \bigcup_{p \in \bar{\delta}_N(q_0, bb)} \delta_N(p, a)$

$\bar{\delta}_N(q_0, bb) = \bigcup_{p \in \bar{\delta}_N(q_0, b)} \delta_N(p, b)$

$\bar{\delta}_N(q_0, b) = \bigcup_{p \in \bar{\delta}_N(q_0, \varepsilon)} \delta_N(p, b)$

$\bar{\delta}_N(q_0, \varepsilon) = \{q_0\}$

Giunti al caso base possiamo sostituire

$\bar{\delta}_N(q_0, b) = \bigcup_{p \in \bar{\delta}_N(q_0, \varepsilon)} \delta_N(q_0, b) = \{q_0, q_1\}$

$\bar{\delta}_N(q_0, bb) = \bigcup_{p \in \bar{\delta}_N(q_0, b)} \delta_N(p, b) = \delta_N(q_0, b) \cup \delta_N(q_1, b) = \{q_0, q_1\} \cup \{q_3\} = \{q_0, q_1, q_3\}$

$\bar{\delta}_N(q_0, bba) = \bigcup_{p \in \bar{\delta}_N(q_0, bb)} \delta_N(p, a) = \delta_N(q_0, a) \cup \delta_N(q_1, a) \cup \delta_N(q_3, a) = \{q_0\} \cup \{q_2, q_3\} \cup \emptyset = \{q_0, q_2, q_3\}$

DEFINIZIONE FORMALE DI LINGUAGGIO ACCETTATO (ASFND)

- In termini di funzione di transizione estesa:

$$L(A) = \{x \in \Sigma^* \mid \overline{\delta_N(q_0, x) \cap F \neq \emptyset}\}$$

Significa che tra gli stati di $\bar{\delta}_N(q_0, x)$ c'è almeno uno stato finale

- In termini di configurazioni:

Si parla di insiemi di configurazioni, perché abbiamo insiemi di stati successivi.

$$L(A) = \{x \in \Sigma^* \mid (\{q_0\}, x) \vdash^* (P, \varepsilon) \mid P \subseteq Q,\ P \cap F \neq \emptyset\}$$

Generalmente gli ASFND non sono più potenti degli ASFD, cioè le classi dei linguaggi riconosciuti sono uguali.

TEOREMA (Pg. 83, 3.1)

Dato un ASFD che riconosce un linguaggio L, esiste un ASFND che riconosce lo stesso linguaggio L.

VICEVERSA: Dato un ASFND che riconosce un linguaggio L, esiste un ASFD che riconosce lo stesso linguaggio L.

[Es. - ASFD, $\delta : Q \times \Sigma \to Q \mid \delta(q, a) = p$

- ASFND, $\delta_N : Q \times \Sigma \to P(Q) \mid \delta_N(q, a) = \{p\}$

Per ottenere un ASFND che riconosca lo stesso L, basta far corrispondere allo stato p dell'ASFD, l'insieme di stati che contiene il solo p.]

Gli ASFD sono particolari ASFND

Per passare da ASFND a ASFD, bisognerà utilizzare $P(Q)$ stati per sostituire tutti

i possibili insiemi di stati di ASFND.

AUTOMI MINIMALI

Sono automi che hanno il minore numero possibile di stati per riconoscere un certo L.

Si dimostra che per ogni linguaggio L esiste un unico AUTOMA MINIMALE.

Si dimostra che esistono linguaggi che non appartengono alla classe $\mathcal{R}$ di linguaggi regolari, cioè non esistono automi che li riconoscono

[Es. $x = a^n b^n$]

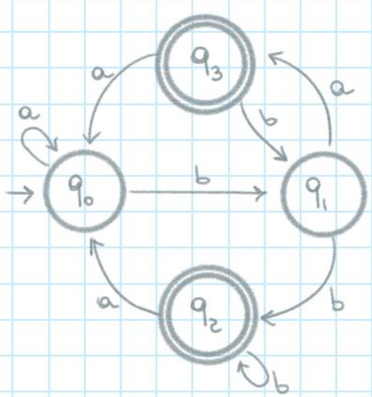
LINGUAGGIO ACCETTATO: se esiste almeno una stringa x accettata dall'automa

LINGUAGGIO RICONOSCIUTO: se per ogni stringa x la computazione termina e x viene accettata o rifiutata dall'autom

ESERCIZIO: Creare un ASFD e un ASFND che riconoscano tutte le stringhe con una b come penultimo carattere.

ASFND:

ASFD:



Lezione 12 12/11/2018

lunedì 24 settembre 2018 20:54

BARBANERA

MODELLI COMPUTAZIONALI [FB6]

COMPUTAZIONE → Insieme di passaggi che ci fanno passare dalla rappresentazione di un'informazione in forma implicita, alla rappresentazione in forma esplicita.

Dobbiamo formalizzare il concetto di computazione in forma matematica.
Esistono vari formalismi matematici, detti MODELLI COMPUTAZIONALI, che ci permettono di formalizzare il concetto di computazione.
Ad esempio il modello ad ASFD è detto "modello ad Automi".
Questo però non riesce a descrivere computazioni complesse.
Tra i vari modelli computazionali vedremo il λ-CALCOLO

Lezione 13 14/11/2018

lunedì 24 settembre 2018 20:54

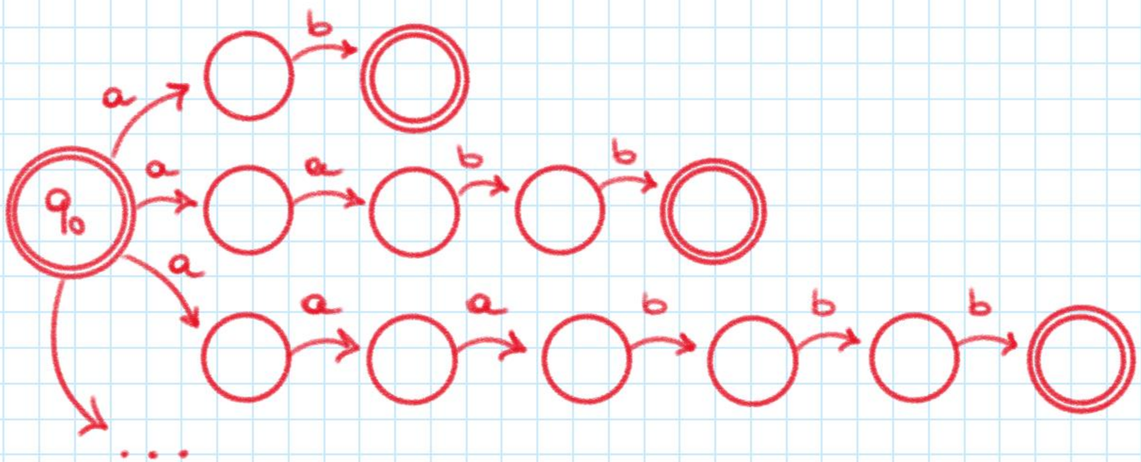
MADONIA

DEFINIZIONE CLASSE DI LINGUAGGI REGOLARI (Pg. 77)

$R = \{L \mid L \subseteq \Sigma^* \text{ esiste } A \mid L = L(A)\}$

Fissato K, il linguaggio $L = \{a^m b^m \mid 0 \leq m \leq k\}$ è regolare?

[Es. $K=5$ $L = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb\}$

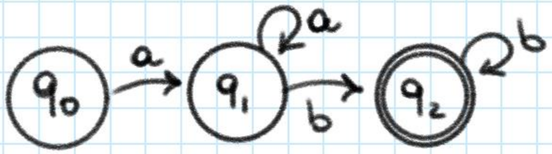


]

Se un linguaggio è finito è regolare (possiamo costruire una "strada" dell'automa per ogni stringa di L.

Se un linguaggio è infinito può essere regolare o no.

$L = \{a^n b^m \mid n, m > 0\}$ è regolare anche se infinito.



PUMPING LEMMA PER LINGUAGGI REGOLARI (Pg 92, 3.4)

N.B. Nell' Auriello è fatto male: consultare [MM]

E' una condizione NECESSARIA per i linguaggi regolari.

Se L è regolare $\Longrightarrow$ L soddisfa il PL.

Non vale per il viceversa (non è una condizione sufficiente)

E' importante perché basta che L non soddisfi PL per dire che L non è regolare.

Se L non soddisfa PL $\Rightarrow$ L non è regolare

TEOREMA PL

Se ho una stringa Z lunga del linguaggio L, posso suddividere Z in 3 pezzi e posso ripetere la parte centrale trovando ancora stringhe di L.

$Z = UVW$ con $V \neq \varepsilon$

$Z =$

$X =$

$X \in L$

ENUNCIATO: Per ogni linguaggio regolare L, esiste una costante m tale che se $z \in L$, $|z| \geq m \Rightarrow$ possiamo scrivere $z = uvw$ con $|uv| \leq m$ e $|v| \geq 1$ e ottenere che $uv^iw \in L$ con $i \geq 0$

m è il numero di stati che riconosce il linguaggio L

Se $|z| \geq m$, nel cammino $\delta(z, q_0) \vdash^* \delta(\varepsilon, q)$ avremo sicuramente dei ritorni in certi stati (perché abbiamo meno stati che simboli).

questo vale per qualsiasi numero di v.

DIMOSTRAZIONE:

Siccome L è regolare, esiste un ASFD $A = (\Sigma, Q, \delta, q_0, F)$ tale che $L = L(A)$ (il linguaggio L è riconosciuto da A).

Supponiamo che $|Q| = m$. Sia $z \in L$ con $|z| = k \geq m$

Siano $q_{i_0}, q_{i_1}, q_{i_2}, \dots, q_{i_k}$ con $i_0 = 0$ e $q_{i_k} \in F$, gli stati che attraverso leggendo i simboli di z.

Gli stati sono $k+1$, $k+1 > k \geq m$. Sicuramente almeno 2 stati coincidono.

Sia z_h il prefisso di z di h (z_h sarebbe u di uvw).

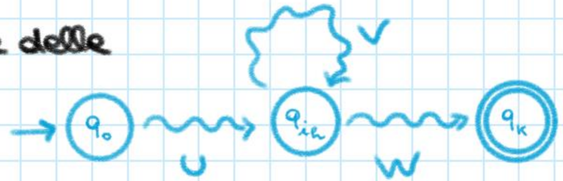
I primi h stati sono $q_{i_0}, q_{i_1}, q_{i_2}, \dots, q_{i_h}$.

Se $q_{i_1} = \bar{\delta}(q_0, z_0)$

$q_{i_2} = \bar{\delta}(q_0, z_1)$

$q_{i_h} = \bar{\delta}(q_0, z_h)$

Allo stato q_{i_h}, poiché siamo giunti a v, alcune delle configurazioni successive portano a q_{i_h}.



Dunque, tra i primi m stati, alcuni saranno ripetuti. Allora esistono almeno 2 indici r ed s, con $0 \leq r < s \leq m$ tali che

$q_{i_r} = q_{i_s}$, cioè $\bar{\delta}(q_0, z_r) = q_{i_r} = q_{i_s} = \bar{\delta}(q_0, z_s)$

Allora poniamo $u = z_r$, $uv = z_s$, $uvw = z$.

$|uv| = |z_s| = s \leq m$

CASO BASE: $i=0$

$$\bar{\delta}(q_0, u) = \bar{\delta}(q_0, z_k) = \bar{\delta}(q_0, z_1)$$

$uv^0w = uw \in L$

$z_1 = uv$

allora $\bar{\delta}(q_0, uw) = \bar{\delta}(\bar{\delta}(q_0, u), w) = \bar{\delta}(\bar{\delta}(q_0, z_1), w) =$

$= \bar{\delta}(\bar{\delta}(q_0, uv), w) = \bar{\delta}(q_0, uvw) = \underbrace{\bar{\delta}(q_0, z) \in F}_{\text{Perché } L \text{ è riconosciuto}}$

PASSO INDUTTIVO: Supposto vero per $i = m-1$, verifichiamo per $i = m$

DA FARE A CASA

Lezione 14 19/11/2018

lunedì 24 settembre 2018 20:54

BARBANERA

I linguaggi di programmazione funzionale (come Haskell), non sono basati su modelli computazionali come le macchine di Turing.

Non si basano sui concetti di stato, variabile, assegnamento, ecc.

I concetti sono: espressione, variabile matematica, valutazione, ricorsione, ecc.

La programmazione funzionale consiste in una serie di definizioni di funzioni.

Nella dispensa [FBb] il modello di Haskell è ben descritto.

Lezione 15 21/11/2018

lunedì 24 settembre 2018 20:54

MADONIA

Per la prima parte del PUMPING LEMMA v. lezione 13.

PASSO INDUTTIVO : $i > 0$

Supponiamo la condizione vera per $(i-1)$,

e la dimostro per i.

Supponiamo allora che $uv^{i-1}w \in L$

Dimostriamo che $uv^iw \in L$.

In termini di funzione di transizione $\bar{\delta}(q_0, uv^iw) \in F$.

$$\bar{\delta}(q_0, uv^iw) = \bar{\delta}(q_0, uvv^{i-1}w) = \bar{\delta}(\bar{\delta}(q_0, uv), v^{i-1}w)$$

Poiché $uv = z_s$ e $u = z_r$, e sappiamo che $\bar{\delta}(q_0, z_s) = \bar{\delta}(q_0, z_r)$

dunque $\bar{\delta}(q_0, uv) = \bar{\delta}(q_0, u)$

Allora possiamo scrivere:

$$\bar{\delta}(\bar{\delta}(q_0, uv), v^{i-1}w) = \bar{\delta}(\bar{\delta}(q_0, u), v^{i-1}w) = \bar{\delta}(q_0, uv^{i-1}w)$$

Per ipotesi $\bar{\delta}(q_0, uv^iw) \in F$

Allora $uv^iw \in L$.

ESERCIZIO: Dimostriamo che $L = \{a^k b^k \mid k \geq 0\}$

Vediamo se soddisfa il PUMPING LEMMA

Per il teorema $\exists m \mid$ se $z \in L$ e $|z| \geq m \Rightarrow z = uvw$ e avere $uv^iw \in L$.

Dimostriamo che $\forall m, \exists z \in L \mid z = uvw$ con $uv \leq m$, $v \neq \varepsilon$ allora $\exists i \mid uv^iw \notin L$

(In qualsiasi modo suddividiamo z in uvw, non possiamo ottenere $uv^iw \in L$)

Fissato m scelgo $z = a^m b^m$. Questo $z \in L$.

$|z| = 2m$ quindi $|z| \geq m$.

Allora z è così composta

Poiché $|uv| \leq m$, uv sarà formato da sole "a" ($uv \in a^*$).

Allora v sarà composto solo da "a" ($v \in a^*$).

Poiché $v \neq \varepsilon$ ($v \in a^+$), comunque ripetiamo v, avremo un numero totale di a maggiore di quello di b.

Possiamo concludere $uv^iw \notin L$

ESERCIZIO 41: Dimostrare che il linguaggio $L = \{a^h b^k \mid h, k \geq 0 \text{ con } h > k\}$ non è regolare.

$z = a^{m+1} b^m \in L \qquad |z| = 2m+1$

$$\underbrace{a\,a \cdots\cdots a}_{m}\,\underbrace{a\,b\,b \cdots\cdots b}_{m}$$

$|uv| \leq m$, dunque $uv \in a^+ \Rightarrow v \in a^+$.

Quindi v è composta da almeno una a.

Per $i=0$ avremo $uv^0w = uw$.

Praticamente togliamo una a.

Allora $h \leq k$, quindi $uv^iw \notin L$.

PROPRIETA' DI CHIUSURA DEI LINGUAGGI REGOLARI (Pg 94-102)

- 1° TEOREMA: Dati L_1 e L_2 regolari, $L_1 \cup L_2$ è regolare.
  "La classe di linguaggi regolare è chiusa rispetto all'unione".
- 2° TEOREMA: "La classe R è chiusa rispetto alla concatenazione".
- 3° TEOREMA: Il complemento $\overline{L}$ di un linguaggio regolare L.
- 4° TEOREMA: "La classe R è chiusa rispetto all'intersezione".

ESPRESSIONI REGOLARI (Pg. 29)

Sono modi finiti per rappresentare linguaggi anche infiniti (come gli ASF).

La classe R per gli ASF coincide con quella delle espressioni regolari.

Partendo da un alfabeto Σ, definiamo le espressioni regolari su Σ.

DEFINIZIONE

Dato un alfabeto Σ, si definisce espressione regolare su Σ una stringa $r \in (\Sigma \cup \{+, *, (,), \cdot, \emptyset\})^+$ tale che valga una delle seguenti condizioni:

- $r = \emptyset$ // $\emptyset$ è un'espressione regolare
- $r \in \Sigma$ // r può essere un simbolo di Σ
- $r = (s+t)$ oppure $r = (s \cdot t)$ oppure $r = s^*$, dove s e t sono espressioni regolari su Σ.

[Es. $\Sigma = (a, b)$

Possiamo avere come espressioni regolari:

$\emptyset, a, b, (a+b), a^*, ((a+b) \cdot a^*), \ldots$]

Ad ogni espressione regolare può essere associato un linguaggio.

[Es. $a \rightsquigarrow \{a\}$

$(a+b) \rightsquigarrow \{a\} \cup \{b\} = \{a, b\}$

$a^* \rightsquigarrow \{a\}^*$

$((a+b) \cdot a^*) \rightsquigarrow \{a, b\} \cdot \{a\}^*$]

DEFINIZIONE FORMALE DI LINGUAGGIO ASSOCIATO

Linguaggio associato a un'espressione regolare ($\mathcal{L}(r)$)

1. $\mathcal{L}(\emptyset) = \Lambda$
2. $\mathcal{L}(a) = \{a\}$ con $a \in \Sigma$
3. $\mathcal{L}(s+t) = \mathcal{L}(s) \cup \mathcal{L}(t)$
4. $\mathcal{L}(s \cdot t) = \mathcal{L}(s) \cdot \mathcal{L}(t)$
5. $\mathcal{L}(s^*) = (\mathcal{L}(s))^*$

Lezione 16 26/11/2018

lunedì 24 settembre 2018 20:54

BARBANERA

λ- CALCOLO

E' la definizione dei linguaggi di programmazione funzionale.

Nella definizione di una funzione abbiamo bisogno di:

- COSTANTI
- FUNZIONI DI BASE
- ESPRESSIONI
- VARIABILE MATEMATICA
- APPLICAZIONE FUNZIONALE
- ASTRAZIONE FUNZIONALE
- DARE NOMI AD ESPRESSIONI

$$\left[\text{fact} = \overbrace{\backslash n \rightarrow \text{if } (\underbrace{n == 0}_{\text{FUNZIONE DI BASE}}) \text{ then } \underbrace{1}_{\text{COSTANTE}} \text{ else } \underbrace{n}_{\text{VARIABILE MATEMATICA}} * \text{fact } (n-1)}^{\text{ESPRESSIONE}}\right]$$

In λ-CALCOLO sono indispensabili solo i concetti di:

- VARIABILE MATEMATICA
- APPLICAZIONE FUNZIONALE
- ASTRAZIONE FUNZIONALE.

Tutti gli altri concetti sono derivabili a partire da questi.

Si definisce un Λ-TERMINE come: (con Λ ::= indichiamo la definizione di un Λ-TERMINE)

- $\Lambda ::= X$ una variabile
- $\Lambda ::= (\Lambda \cdot \Lambda)$ concatenazione di due Λ-TERMINI
- $\Lambda ::= \lambda X . \Lambda$ astrazione di un Λ-TERMINE
- Nient'altro è un Λ-TERMINE

N.B. X è un insieme numerabile di variabili matematiche.

N.B. "." è il simbolo di ASTRAZIONE FUNZIONALE

Bisogna formalizzare il processo di trasformazione, computazione.

Il passo computazionale $\rightsquigarrow$ A (sostituire un nome con l'espressione associata) e il passo $\rightsquigarrow$ C (calcolo di una funzione predefinita a cui abbiamo applicato un valore esplicito),

Resta quindi il passo computazionale $\rightsquigarrow_\beta$ (sostituire il parametro attuale al parametro formale).

β-RIDUZIONE

Formalmente la "trasformazione" da parametro formale ad attuale è:

$(\lambda x.M)\,N \longrightarrow_\beta M[N/x]$ → M è un Λ-TERMINE

Applico λx.M al parametro attuale N.

Dunque la β riduzione ci restituirà M, ma sostituendo al parametro formale x il parametro attuale N

Bisogna definire il concetto di sostituzione ($[N/x]$)

RELAZIONE DI α-CONVERSIONE ($=_\alpha$)

Due Termini sono considerati uguali se sono identici a meno del nome delle variabili usate.

[Es. $(\lambda x.xz)$ chiamiamolo P

$(\lambda y.yz)$ chiamiamolo Q

$P =_\alpha Q$]

Dunque $(\lambda x.xz) =_\alpha (\lambda y.yz)[y/x]$

Bisogna formalizzare il concetto di variabile usata come argomento.

BOUND VARIABLE (BV)

- $BV(x) = \emptyset$ → x è una variabile
- $BV(P \cdot Q) = BV(P) \cup BV(Q)$ → P e Q sono Λ-TERMINI
- $BV(\lambda x.M) = BV(M) \cup \{x\}$ → M è un Λ-TERMINE

Sono i "segnaposti" per le variabili lette in INPUT.

FREE VARIABLE (FV)

- $FV(x) = x$ → x è una variabile
- $FV(P \cdot Q) = FV(P) \cup FV(Q)$ → P e Q sono Λ-TERMINI
- $FV(\lambda x.M) = FV(M) \setminus \{x\}$ → M è un Λ-TERMINE

La il concetto di sostituzione è applicabile solo alle variabili libere (FV).

DEFINIZIONE:

- $x[L/x] = L$
- $y[L/x] = y$ se $y \neq x$
- $(PQ)[L/x] = P[L/x] \cdot Q[L/x]$
- $(\lambda x.P)[L/x] = (\lambda x.P)$ Perché le x sono legate
- $(\lambda y.P)[L/x] = \lambda y.(P[L/x])$

Il problema è il seguente:

(ESEMPIO): $(\lambda y.zy)[(yy)/z] = \lambda y.(yy)y$

Non otteniamo più un algoritmo del tipo

$\lambda t.(yy)t$, ma un algoritmo del tipo $\lambda t.ttt$

Bisogna fare attenzione che si può fare

$(\lambda y.P)[L/x] = \lambda y.(P[L/x])$ solo se

$FV(L) \cap BV(L) = \emptyset$

Generalmente $P \xrightarrow[\beta]{} Q$

se P è in forma β-REDEX $\rightarrow (\lambda x.M)N$

e Q è in forma CONTRACTUM $\rightarrow M[N/x]$

[Es. $(\lambda x.\underbrace{xx}_{M})(\underbrace{\lambda x.xx}_{N}) \xrightarrow[\beta]{} (\lambda x.xx)(\lambda x.xx)$

E' infinito. viene chiamato Ω]

Anche il λ-CALCOLO ha una STRATEGIA

[Es. $(\lambda x.y)\,\Omega$

se la strategia facesse valutare prima Ω, avremmo un ciclo infinito.

Se prima valutiamo $\lambda x.y$, qualsiasi sia il parametro attuale, il risultato è y]

Lezione 17 28/11/2018

lunedì 24 settembre 2018 20:54

MADONIA

ESERCIZIO: Scrivere un'espressione regolare a cui si associano tutte le stringhe sull'alfabeto $\Sigma = \{a, b\}$, che hanno "b" come penultimo carattere.

$$(a+b)^* b (a+b)$$

ESERCIZIO: Scrivere un'espressione regolare a cui si associano tutte le stringhe sull'alfabeto $\Sigma = \{a, b\}$, che contengono un numero pari di "a".

$$(ab^*a + b)^*$$

oppure

$$(b^*ab^*ab^*)^* + b^*$$

ESERCIZIO: Scrivere un'espressione regolare a cui si associano tutte le stringhe sull'alfabeto $\Sigma = \{a, b\}$, che cominciano per "bb" o finiscono per "ab".

$$bb(a+b)^* + (a+b)^* ab$$

ESERCIZIO: Scrivere un'espressione regolare a cui si associano tutte le stringhe sull'alfabeto $\Sigma = \{a, b\}$, che contiene solo ε.

$\emptyset^* \rightarrow$ $\emptyset$ è un'espressione regolare

$\emptyset^0 = \varepsilon \qquad \emptyset^* = \emptyset$

oppure

$\Lambda^* \rightarrow$

$L(\emptyset) = \Lambda$

$\Lambda^* = \Lambda^0 \Lambda^1 \Lambda^2$

$\Lambda^0 = \{\varepsilon\}$

N.B. **TEOREMA**

Si può dimostrare che per tutti i linguaggi denotati da un'espressione regolare, esiste un ASF che riconosce lo stesso linguaggio.

Vale anche il viceversa.

Sono 2 teoremi costruttivi

GRAMMATICHE DI CHOMSKY (Pg. 35)

Sono dei metodi GENERATIVI (non RICONOSCITIVI).

Lo scopo di Chomsky era quello di riconoscere se una frase (in inglese) fosse sintatticamente corretta.

Si parte da una stringa e riscrivendo via via parte di essa seguendo delle REGOLE DI PRODUZIONE

[Es. $S \rightarrow aB$

$abcSbb \rightsquigarrow abcaBbb$]

Una grammatica ha un alfabeto di SIMBOLI TERMINALI e un alfabeto di SIMBOLI NON TERMINALI

DEFINIZIONE DI GRAMMATICA

Una grammatica G è una quadrupla

$G = \{V_T, V_N, P, S\}$ dove:

- V_T è l'ALFABETO TERMINALE, finito e non vuoto, di simboli terminali
- V_N è l'ALFABETO NON TERMINALE, finito e non vuoto, di simboli non terminali (o VARIABILI)
- P è l'insieme delle REGOLE DI PRODUZIONE, cioè coppie (α, β) con

$$\underbrace{\alpha \in (V_T \cup V_N)^* V_N (V_T \cup V_N)^*}_{\text{stringa contenente almeno un simbolo non terminale}} \quad e \quad \underbrace{\beta \in (V_T \cup V_N)^*}_{\text{stringa di simboli terminali e non terminali}}$$

- $S \in V_N$ detto SIMBOLO INIZIALE DELLA GRAMMATICA

[Es. $V_T = \{a, b, c\}$ $V_N = \{S, B, C\}$

$$\left.\begin{array}{l} S \to aS \\ S \to B \\ B \to bB \\ B \to bC \\ C \to cC \\ C \to c \end{array}\right] \text{Insieme di REGOLE DI PRODUZIONE}$$

]

Tutta questa è una grammatica.

N.B. Per convenzione, lettere maiuscole sono simboli non terminali, lettere minuscole sono simboli terminali.

Seguendo l'esempio partiamo da S, simbolo iniziale e applichiamo le regole in un qualsiasi modo finché otteniamo solamente simboli terminali

1 S

2 aS $(S \to aS)$

3 aB $(S \to B)$

4 abB $(B \to bB)$

5 $abbC$ $(B \to bC)$

6 $abbc$ $(C \to c)$

DEFINIZIONE DI DERIVAZIONE DIRETTA (Def 2.3)

Data una grammatica G, siano $\Phi, \Psi \in (V_T \cup V_N)^*$ dico che Φ deriva direttamente Ψ e scrivo $\Phi \underset{G}{\Rightarrow} \Psi$ se:

- $\Phi = \gamma \alpha \delta$
- $\Psi = \gamma \beta \delta$
- $\alpha \to \beta$ è in P

Lezione 18 11/12/2018

lunedì 24 settembre 2018 20:54

BARBANERA

Esiste una strategia di riduzione detta NORMAL ORDER che riduce il β-REDEX più a sinistra e più esterno.

TEOREMA DI STANDARDIZZAZIONE

Se un termine possiede una forma normale, attraverso la riduzione NORMAL ORDER si giunge a quella forma.

TEOREMA DI CONFLUENZA

N.B. con il simbolo $\twoheadrightarrow$ si indica un numero di passi di β-riduzione ≥ 0.

Se si parte da un termine M, e si ha $M \twoheadrightarrow M_1$, mentre attraverso un altro procedimento si ha che $M \twoheadrightarrow M_2$, allora esiste sempre

un termine L tale che $M_1 \twoheadrightarrow L$ e $M_2 \twoheadrightarrow L$

In forma di diagramma:

$$\begin{tikzcd}
 & M \arrow[dl, two heads] \arrow[dr, two heads] & \\
M_1 \arrow[dr, two heads] & & M_2 \arrow[dl, two heads] \\
 & L &
\end{tikzcd}$$

N.B. M, M_1, M_2, L potrebbero non essere in forma normale

COROLLARIO DI UNICITA' DELLA FORMA NORMALE

Se si ha:

$$\begin{tikzcd}
 & M \arrow[dl, two heads] \arrow[dr, two heads] & \\
N_1 & & N_2
\end{tikzcd}$$

con N_1 e N_2 in forma normale,

sappiamo dal teorema precedente che:

$$\begin{tikzcd} N_1 \arrow[dr, two heads] & & N_2 \arrow[dl, two heads] \\ & L & \end{tikzcd}$$

In questo caso L deve essere in forma normale:

$N_1 \twoheadrightarrow L$ avviene in 0 passi $(L = N_1)$

$N_2 \twoheadrightarrow L$ avviene in 0 passi $(L = N_2)$

Dunque $N_1 = N_2$

β-CONVERSIONE

Se $M =_\beta M'$, allora M e M' hanno lo stesso contenuto informativo.

M e M' sono diverse implementazioni dello stesso algoritmo.

Due termini sono β-CONVERTIBILI se uno può essere trasformato nell'altro attraverso zero o più β-espansioni o β-riduzioni.

Ovviamente se $M =_\beta M'$ allora

$$\begin{tikzcd} M \arrow[dr] & & M' \arrow[dl] \\ & L & \end{tikzcd}$$

Quanto è generale il modello del λ-calcolo come formalizzazione del concetto di computazione?

Il modello ha la massima generalità: qualsiasi maniera per calcolare una funzione si possa avere in mente, questa può essere definita con questo modello.

TESI DI CHURCH-TURING

Tutto ciò che è calcolabile in modo effettivo, si può calcolare mediante il λ-calcolo (e le macchine di Turing).

Anche somma e prodotto possono essere calcolate tramite λ-calcolo

add $= \lambda mnfx.\, nf(mfx)$

mult $= \lambda mnf.\, n(mf)$

Anche i numeri vengono rappresentati attraverso il **NUMERALE DI CHURCH**

$\overline{0} = \lambda fx.\, x$

$\overline{1} = \lambda fx.\, fx$

$\overline{2} = \lambda fx.\, f(fx)$

$\overline{3} = \lambda fx.\, f(f(fx))$

$\vdots$

$\overline{m} = \lambda fx.\, f^m x$

Esercizio: Fare 2×2 con il λ-calcolo

$mult = \lambda mnf.\, m(nf)$

$\overline{2} = \lambda fx.\, f(fx)$

$$\overbrace{(\lambda mnf.\, m(nf))}^{mult}\overbrace{(\lambda fx.\, f(fx))}^{\overline{2}}$$

$$\lambda mf.\, (\lambda fx.\, f(fx))(mf)$$

$$\lambda mf.\, (\lambda x.\, (mf)(mfx))$$

$$(\lambda mfx.\, (mf)(mfx))\overbrace{(\lambda fx.\, f(fx))}^{\overline{2}}$$

$$\lambda fx.\, (\lambda fx.\, f(fx))f\,(\lambda fx.\, f(fx))fx$$

$$\lambda fx.\, (\lambda x.\, f(fx))(\lambda x.\, f(fx))x$$

$$\lambda fx.\, (\lambda x.\, f(fx))(f(fx))$$

$$\lambda fx.\, f(f(f(fx)))$$

Questo λ-Termine corrisponde a $\overline{4}$

La teoria della β-Conversione è consistente in quanto è possibile trovare due elementi non β-convertibili. Ad esempio x e y.

$x \neq_\beta y$

infatti se $x =_\beta y$ esisterebbe un R tale che $x \twoheadrightarrow R$ e $y \twoheadrightarrow R$. Poiché la riduzione avverrebbe in zero passi, si avrebbe $R \equiv x$ e $R \equiv y$.

Dimostrare qualcosa e scrivere un programma sono due cose equivalenti.

Prendiamo la deduzione in DN di AS

$\vdash_{DN} (\sigma \to (\tau \to \gamma)) \to ((\sigma \to \tau) \to (\sigma \to \gamma))$

$$\dfrac{\dfrac{[\sigma \to (\tau \to \gamma)]^3 \quad [\sigma]^1}{\tau \to \gamma} \qquad \dfrac{[\sigma \to \tau]^2 \quad [\sigma]^1}{\tau}}{}$$

Dimostrare qualcosa e scrivere un programma sono due cose equivalenti.

Prendiamo la deduzione in DN di AS

$\vdash_{DN} (\sigma \to (\tau \to \gamma)) \to ((\sigma \to \tau) \to (\sigma \to \gamma))$

$$
\dfrac{\dfrac{\dfrac{\dfrac{[\sigma \to (\tau \to \gamma)]^3 \quad [\sigma]^1}{\tau \to \gamma} \qquad \dfrac{[\sigma \to \tau]^2 \quad [\sigma]^1}{\tau}}{\gamma}}{\sigma \to \gamma}\,(1)}{(\sigma \to \tau) \to (\sigma \to \gamma)}\,(2)}{(\sigma \to (\tau \to \gamma)) \to ((\sigma \to \tau) \to (\sigma \to \gamma))}\,(3)
$$

Poniamo:

- $x: \sigma \to (\tau \to \gamma)$
- $y: \sigma$
- $z: \sigma \to \tau$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x:\sigma\to(\tau\to\gamma)]^3 \quad [y:\sigma]^1}{xy:\tau\to\gamma} \qquad \dfrac{[z:\sigma\to\tau]^2 \quad [y:\sigma]^1}{zy:\tau}}{(xy)(zy):\gamma}}{\lambda y.(xy)(zy):\sigma\to\gamma}\,(1)}{\lambda z.\lambda y.(xy)(zy):(\sigma\to\tau)\to(\sigma\to\gamma)}\,(2)}{\lambda x.\lambda z.\lambda y.(xy)(zy):(\sigma\to(\tau\to\gamma))\to((\sigma\to\tau)\to(\sigma\to\gamma))}\,(3)
$$

$\lambda x.\lambda z.\lambda y.(xy)(zy)$ è il λ-termine corrispondente alla deduzione

ESERCIZIO: Fare 2×3 con il λ-calcolo

$mult = \lambda mmF.\ m(mF)$

$\overline{2} = \lambda fx.\ f(fx)$

$\overline{3} = \lambda fx.\ f(f(fx))$

$$\overbrace{\lambda mmf.\ m(mf)}^{mult}\ \overbrace{(\lambda fx.\ f(fx))}^{\overline{2}}\ \overbrace{(\lambda fx.\ f(f(fx)))}^{\overline{3}}$$

$\lambda f.\ (\lambda fx.\ f(fx))((\lambda fx.\ f(f(fx)))f)$

$\lambda f.\ (\lambda fx.\ f(fx))(\lambda x.\ f(f(fx)))$

$\lambda f.\ (\lambda x.\ (\lambda x.\ f(f(fx)))((\lambda x.\ f(f(fx)))\ x)$

$\lambda f.\ \lambda x.\ (\lambda x.\ f(f(fx)))(f(f(fx)))$

$\lambda fx.\ f(f(f(f(f(fx)))))$

Questo λ-termine è $\overline{6}$

Lezione 19 12/12/2018

lunedì 24 settembre 2018 20:54

MADONIA

Una produzione del tipo $\alpha \to \varepsilon$ $\varepsilon \in (V_N \cup V_T)^*$

si dice **PSEUDOPRODUZIONE**

Corrisponde ad eliminare α dalla stringa.

Attraverso più derivazioni dirette, si ottiene una DERIVAZIONE.

DEFINIZIONE DI DERIVAZIONE (Def 2.4)

Si ha che Φ deriva Ψ in G se $\exists\, \Phi_1, \Phi_2, \dots, \Phi_m \in (V_N \cup V_T)^*$

tali che: $\Phi = \Phi_1$, $\Psi = \Phi_m$ e $\Phi_i \Rightarrow \Phi_{i+1}$ $\forall i = 1, 2, \dots, m-1$.

In questo caso scriviamo $\Phi \overset{*}{\underset{G}{\Rightarrow}} \Psi$

ovvero $\Phi = \Phi_1 \underset{G}{\Rightarrow} \Phi_2 \underset{G}{\Rightarrow} \dots \underset{G}{\Rightarrow} \Phi_m = \Psi$

La derivazione diretta può essere vista

come relazione tra 2 elementi in $(V_N \cup V_T)^*$

Φ e Ψ sono in relazione tra loro se $\Phi \underset{G}{\Rightarrow} \Psi$

La derivazione diretta non gode della proprietà transitiva, né della riflessiva. → Perché bisogna fare almeno un passo

Invece la derivazione gode sia della transitiva che della riflessiva. Si dice **CHIUSURA RIFLESSIVA E TRANSITIVA DELLA DERIVAZIONE DIRETTA**.

DEFINIZIONE DI LINGUAGGIO GENERATO DA UNA GRAMMATICA (Def 2.6)

Il linguaggio generato da una grammatica $G = (V_N, V_T, P, S)$ è

$$L(G) = \{x \in V_T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} x\}$$

Insieme di tutte le stringhe di caratteri terminali derivate da S

$$\left.\begin{array}{l} S \to aSBC \\ CB \to BC \\ SB \to bF \\ FC \to cG \\ FB \to bF \\ GC \to cG \\ G \to \varepsilon \end{array}\right\} \quad L = \{a^m b^m c^m \mid m \geq 1\}$$

Esistono grammatiche che generano linguaggio vuoto

Es. $S \to Ab$
$A \to Sa$

Vediamo che non avremo mai stringhe di soli caratteri terminali, dunque nessuna stringa apparterrà al linguaggio generato dalla grammatica.

Due grammatiche che generano lo stesso linguaggio si dicono GRAMMATICHE EQUIVALENTI

DEFINIZIONE GRAMMATICHE EQUIVALENTI

Due grammatiche G_1 e G_2 sono equivalenti se $L(G_1) = L(G_2)$

ESERCIZIO: Date le seguenti grammatiche sono equivalenti

G_1 : $S \to aS \mid b$ "|" significa "oppure"

G_2 : $S \to b \mid Ab$

$A \to a \mid Aa$

Vediamo che G_1 e G_2 generano $L = \{a^n b \mid n \geq 0\}$

Prendiamo G_1

Usiamo n volte $S \to aS$ $\left(S \overset{n}{\underset{G_1}{\Rightarrow}} a^n S\right)$

Poi usiamo $S \to b$ $\left(a^n S \underset{G_1}{\Rightarrow} a^n b\right)$

Prendiamo G_2

Usiamo $S \to Ab$ $\left(S \underset{G_2}{\Rightarrow} Ab\right)$

Usiamo $n-1$ volte $A \to Aa$ $\left(Ab \overset{n-1}{\underset{G_2}{\Rightarrow}} a^{n-1}b\right)$

Usiamo $A \to a$ $\left(a^{n-1}b \underset{G_2}{\Rightarrow} a^n b\right)$

Abbiamo dimostrato che $L(G_1) \subseteq L(G_2)$

GRAMMATICHE DI TIPO 0

$\alpha \to \beta$ con $\alpha \in V^* V_N V^*$ e $\beta \in V^*$ con $V = (V_N \cup V_T)$

Dette anche UNRESTRICTED (senza restrizioni)

N.B. I linguaggi di tipo 0 vengono detti RICORSIVAMENTE ENUMERABILI

GRAMMATICHE DI TIPO 1

$\alpha \to \beta$ $\alpha \in V^* V_N V^*$ e $\beta \in V^+$ con $|\alpha| \leq |\beta|$

Ovviamente $\beta \neq \varepsilon$ perché si avrebbe $|\beta| = |\varepsilon| = 0$

e $\forall \alpha$, $|\alpha| > |\beta|$ e non può essere.

Si dicono **CONTEXT SENSITIVE** (perché se si ha una G di tipo 1, se ne può costruire una equivalente in cui tutte le regole di produzione sono del tipo $\alpha A \beta \to \alpha a \beta$)

GRAMMATICHE DI TIPO 2

$A \to \beta$ con $A \in V_N$ e $\beta \in V^+$

Sono dette **CONTEXT-FREE**

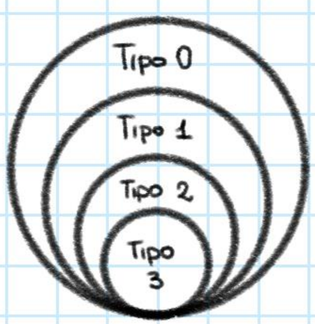
GRAMMATICHE DI TIPO 3

$A \to \beta$ con $A \in V_N$ e $\beta \in (V_N V_T) \cup V_T$

Dette **GRAMMATICHE REGOLARI**

Generano linguaggi regolari

β del tipo $\begin{cases} bB \\ Bb \\ b \end{cases}$



Si dimostra che $a^n b^n$ è di tipo 2, ma non 3.

Si dimostra che $a^n b^n c^n$ è di tipo 1, ma non 2.

Dunque l'inclusione è propria (del tipo $\subset$ e non $\subseteq$)

ALBERI DI DERIVAZIONE O PARSETREE (Pg. 159)

$A \to \beta$ con $A \in V_N$ e $\beta \in V^+$

$S \to aSbA \mid ab$

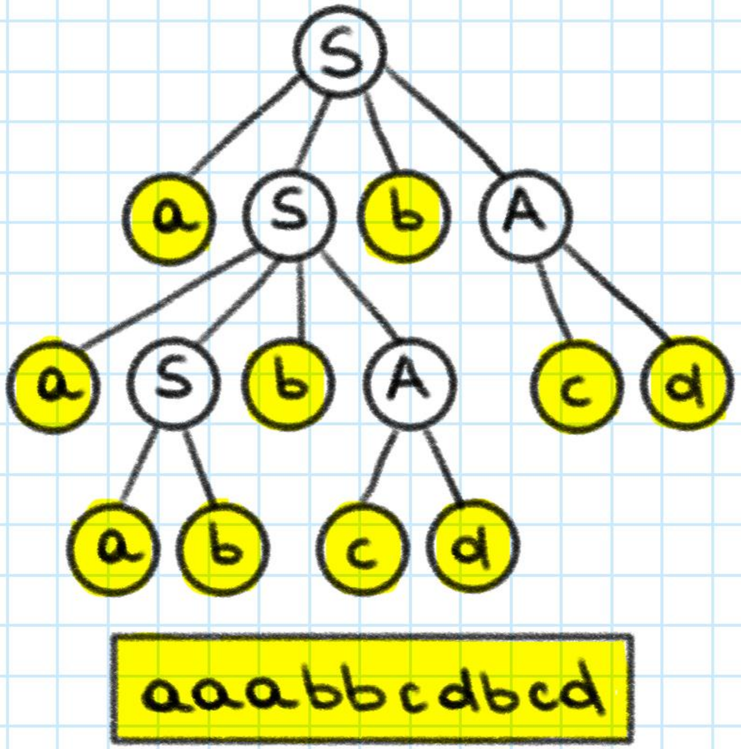
$A \to cAd \mid cd$

$S \Rightarrow aSbA \Rightarrow aaSbAbA \Rightarrow$

$\Rightarrow aaabbAbA \Rightarrow aaabbcdbA \Rightarrow$

$\Rightarrow aaabbcdbcd$

A questa derivazione può corrisponde un albero di derivazione :



Due derivazioni sono diverse se hanno un albero di derivazione diverso.

DERIVAZIONE SINISTRA

Si sostituisce il carattere non terminale più a sinistra. Ogni albero ne ha solo una

DEFINIZIONE DI GRAMMATICA AMBIGUA

Una grammatica si dice ambigua se esiste una stringa appartenente al linguaggio generato con 2 derivazioni sinistre (cioè hanno 2 alberi di derivazione).

Es. $V_N = \{E\}$ $V_T = \{a, +, *\}$

$$E \to E * E \mid E + E \mid a$$

$$\underline{E} \Rightarrow \underline{E} * E \Rightarrow a * \underline{E} \Rightarrow a * \underline{E} + E \Rightarrow a * a + \underline{E} \Rightarrow a * a + a$$

oppure

$$\underline{E} \Rightarrow \underline{E} + E \Rightarrow \underline{E} * E + E \Rightarrow a * \underline{E} + E \Rightarrow a * a + \underline{E} \Rightarrow a * a + a$$

Esistono degli alberi che si possono disambiguare.
Esistono alberi inerentemente ambigui (qualunque linguaggio generato è ambiguo).

Lezione 20. 17/12/2018

domenica 16 dicembre 2018 20:23

BARBANERA

Un programma è la descrizione di un processo computazionale.
Programmi corretti sono sequenze di caratteri che hanno una determinata semantica.
Occorre dare una semantica formale dei linguaggi di programmazione
Esistono diversi approcci (denotazionale, assiomatico, operazionale).
La SEMANTICA OPERAZIONALE STRUTTURALE (S.O.S.)
è la descrizione di un programma attraverso transizioni e regole di inferenza definite sul linguaggio del programma
Quella che vedremo è detta "BIG-STEP"

LINGUAGGIO WHILE

I giudizi hanno la forma $\vec{a}\ P\Downarrow\ \vec{b}$ dove:

- $\vec{a}$ è un vettore di valori
- $P\Downarrow$ è un programma
- $\vec{b}$ è un vettore di valori

Diremo che il programma è corretto (la formula è valida) se $\vec{a}$ rappresenta i valori delle variabili date in input al programma $P\Downarrow$ e $\vec{b}$ rappresenta il valore delle variabili dopo che sono state date al programma (output).

1° ASSIOMA DEL SISTEMA FORMALE

$$\frac{}{\vec{a},\ \text{begin end}\Downarrow,\ \vec{a}}$$

E' la descrizione di un programma vuoto.

Se diamo $\vec{a}$, otteniamo $\vec{a}$ ($P\Downarrow$ è vuoto)

REGOLA DI INFERENZA

$$\frac{m \geq 1 \quad \vec{a},\ \text{begin } \delta_1 \text{ end}\Downarrow,\ \vec{a'} \qquad \vec{a'},\ \text{begin } \delta_2;\dots;\delta_m \text{ end}\Downarrow \vec{b}}{\vec{a},\ \text{begin } \delta_1;\dots;\delta_m \text{ end}\Downarrow,\vec{b}}$$

Praticamente dice che: dire che attraverso la computazione dei passi $\delta_1;\dots;\delta_m$ passo da $\vec{a}$ a $\vec{b}$, vuol dire che prima si esegue la prima istruzione δ_1 e giungiamo ai valori intermedi $\vec{a'}$, e così via finché abbiamo eseguito tutti i δ.

2° ASSIOMA

$(a_1, a_i, a_{i+1}, \dots, a_k)$, begin $X_i := 0$ end $\Downarrow$, $(a_1, 0, a_{i+1}, \dots, a_k)$

Assegnamento del valore 0 ad a_i

3° ASSIOMA

$(a_1, a_i, a_{i+1}, \dots, a_k)$, begin $X_i := succ(X_j)$ end $\Downarrow$, $(a_1, a_{j+1}, a_{i+1}, \dots, a_k)$

Sostituiamo a_i con a_{j+1}

4° ASSIOMA

$(a_1, a_i, a_{i+1}, \dots, a_k)$, begin $X_i := pred(X_j)$ end $\Downarrow$, $(a_1, a_{j-1}, a_{i+1}, \dots, a_k)$

Sostituiamo a_i con a_{j-1}

5° ASSIOMA

$$\frac{a_i = a_j}{\vec{a}, \text{ begin while } X_i \neq X_j \text{ do } \vec{\delta} \text{ end} \Downarrow, \vec{a}}$$

E' un assioma solo se in $\vec{a}$ $a_i = a_j$, dunque il programma termina senza elaborare $\vec{\delta}$ (non fa niente).

REGOLA DI INFERENZA

$$\frac{a_i \neq a_j \quad \vec{a}, \text{begin } \delta \text{ end} \Downarrow, \vec{b} \quad \vec{b}, \text{begin while } X_i \neq X_j \text{ do } \vec{\delta} \text{ end} \Downarrow, \vec{c}}{\vec{a}, \text{ begin while } X_i \neq X_j \text{ do } \vec{\delta} \text{ end} \Downarrow, \vec{c}}$$

ESERCIZIO: Aggiungiamo l'istruzione:

<istruzione> ::= for X_1 do <istruzione>

Informalmente se l'istruzione del for decrementa X_1, si ripete l'istruzione X_1 volte.

Se $X_1 = 0$, non bisogna fare niente:

$$\frac{a_1 = 0}{\vec{a}, \text{begin for } X_1 \text{ do } \delta \text{ end} \Downarrow, \vec{a}}$$

(se X_1 è 0, il programma non fa nulla)

$$\frac{(a_1, \dots, a_m), \text{begin for } X_1 \text{ do } \delta \text{ end} \Downarrow, (a_1-1, a_2', a_3', \dots, a_m') \qquad (a_1-1, a_2', \dots, a_m'), \text{begin for } X_1 \text{ do } \delta \text{ end} \Downarrow, \vec{b}}{(a_1, \dots, a_m), \text{begin for } X_1 \text{ do } \delta \Downarrow, \vec{b}} \; \overset{\text{premessa}}{(a_1 \neq 0)}$$

Questo è il caso in cui δ decrementa a_1

Ora vediamo il caso in cui δ non decrementa a_1, allora il programma non deve fare nulla

$$\frac{(a_1, \dots, a_m), \text{begin for } X_1 \text{ do } \delta \text{ end} \Downarrow, (a_1-1, a_2', a_3', \dots, a_m')}{(a_1, \dots, a_m), \text{begin for } X_1 \text{ do } \delta \Downarrow, \vec{a}} \; \overbrace{(a_1 \neq 0)\underbrace{(a_1' \neq a-1)}_{\text{se } \delta \text{ non decrementa}}}^{\text{premesse}}$$

Lezione 21. 19/12/2018

domenica 16 dicembre 2018 20:23

MADONIA

LINGUAGGIO DI TIPO i

Un linguaggio L è detto di tipo i (con $i = 0, 1, 2, 3$),
se esiste una grammatica G di tipo i che lo genera,
cioè tale che $L(G) = L$

LINGUAGGIO STRETTAMENTE DI TIPO n

Un linguaggio L viene detto "strettamente di tipo n"
se esiste una grammatica G di tipo n, e non
esiste alcuna grammatica G' di tipo n' con $n' > n$ che lo generi.

Esiste un analogo del Pumping-Lemma per dimostrare
che un linguaggio non è CONTEXT-FREE
Così si può dimostrare ad esempio che $L = \{a^n b^n c^n\}$ è
strettamente di tipo 1 (dunque non è CONTEXT-FREE)

ESERCIZIO: Grammatica regolare sull'alfabeto $\{a, b\}$, che riconosca i linguaggi che contengono un numero dispari di "a".

$S \to aA \mid bS \mid a$

$A \to b \mid aS \mid bA$

[$S \Rightarrow aA \Rightarrow aaS \Rightarrow aabS \Rightarrow aabaA \Rightarrow aabab$]

Se la grammatica è di tipo CONTEXT-FREE, si può trasformare nella FORMA NORMALE DI BACKUS (BNF) per rendere più comprensibile quale sia il linguaggio generato dalla grammatica.

Ci sono alcune regole:

1. Caratteri non terminali prendono la forma <nome>
   [Es. <identificatore>, <espressione>]
2. La freccia di produzione $\to$, viene sostituita da ::=
3. Si usano le parentesi graffe { } per simboleggiare

la possibile ripetizione di un carattere.

[Es. $A \to x \mid xy \mid xyy \mid xyyy \mid xyyyy \mid$

diventa:

$A ::= x\{y\}^4$]

4. Le parentesi quadre [ ] per i caratteri che possono presentarsi o no

[Es. $A \to y \mid xy$

diventa

$A ::= [x]y$]

5. Le tonde ( ) per concatenazione

[Es. $A \to xu \mid xv \mid xw$

diventa

$A ::= x(u|v|w)$]

Esempio di grammatica che genera tutti gli identificatori del linguaggio Pascal di lunghezza n che iniziano per una lettera:

<identificatore> ::= <alfabetico> $\{$<alfanumerico>$\}^{n-1}$

<alfanumerico> ::= <alfabetico> | <cifra>

<alfabetico> ::= A | B | ... | Z | a | b | ... | z

<cifra> ::= 0 | 1 | ... | 9

MACCHINE DI TURING (Pg. 170)

Sono macchine che hanno un nastro infinito, ha una testina che può muoversi a sx e dx, e può scrivere sul nastro.

A seconda dello stato in cui si trova può:

- muoversi a sx o dx
- stare ferma
- scrivere sul nastro.

E' la generalizzazione più ampia del concetto teorico di algoritmo.

TESI DI CHURCH-TURING → Qualsiasi funzione calcolabile può essere calcolata con una TM

Le celle vuote del nastro contengono il BLANCK (b)

DEFINIZIONE FORMALE DI MTD

N.B. Consideriamo il modello deterministico

Una Macchina di Turing deterministica MTD è una sestupla $M = \{\Gamma, b, Q, q_0, F, \delta\}$ dove:

- Γ è l'alfabeto dei simboli di nastro (finito e non vuoto)
- $b \notin \Gamma$ è il simbolo che denota la cella vuota (blanck)
- Q è l'insieme degli stati (finito e non vuoto)
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati finali
- δ è la funzione di transizione così definita:

Destra
Sinistra
Immobile

$$\delta : \underbrace{Q \setminus F}_{} \times \Gamma \cup \{b\} \rightarrow Q \times \underbrace{\Gamma \cup \{b\}}_{} \times \underbrace{\{D, S, I\}}_{}$$

Direzione

La MTD può anche scrivere sul nastro

N.B. La computazione si ferma quando si giunge ad un carattere terminale, non quando si legge b

Si definisce $\overline{\Gamma} = \Gamma \cup \{\sqcup\}$

Si definisce $\Sigma \subseteq \Gamma$ alfabeto di input

(anche se il linguaggio riconosciuto è sui simboli di Σ, sul nastro possiamo avere anche altri simboli di Γ)

Lezione 22. 07/01/2019

domenica 16 dicembre 2018 20:23

BARBANERA

CALCOLO DEI PREDICATI

Puntiamo la nostra attenzione su mondi in cui esistono relazioni tra gli elementi.
Così le proprietà di un elemento sono relazioni unarie; le proprietà comuni a due elementi sono relazioni binarie; e così via.
Un giudizio di base, in logica dei predicati è un giudizio che afferma l'esistenza di una o più relazioni tra certi individui.
La prima cosa che si fa è definire una **SEGNATURA** (insieme di simboli) $\Sigma = \{\{ s.f. \}, \{ s.r. \}\}$

simboli di funzione (per denotare individui)

simboli di relazione (per denotare relazioni)

N.B. L'**ARITA'** di una funzione è il numero di argomenti.
Infatti f^0 è una costante poiché non ha argomenti

Come connettivi logici usiamo anche $\neg$ e $\rightarrow$.
Si possono utilizzare delle variabili generiche per generalizzare delle affermazioni.

Il connettivo di **QUANTIFICAZIONE UNIVERSALE $\forall$** serve a generalizzare.

Esempio visto a lezione:

$\forall x. B(c(c(x)))$

Per qualsiasi individuo $(\forall x)$, il cugino del cugino di quell'individuo $(c(c(x)))$ è bello (B).

Il connettivo di **QUANTIFICAZIONE ESISTENZIALE $\exists$** permette di fare affermazioni sull'esistenza di un generico elemento.
Poiché si può definire in termini di $\forall$, spesso non viene considerato.

Dire $\exists(\varphi(x)$ è equivalente a dire $\neg\forall. \neg\varphi(x)$

N.B. Bisogna specificare se la logica che stiamo usando comprende l'uguaglianza.
In quel caso potremo usare il simbolo $=$.
Bisogna dare un insieme di assiomi e regole di inferenza

Vi sono gli assiomi del calcolo proposizionale (A_K, A_S, $A_\neg$).

Esistono anche degli assiomi specifici.

Come regola abbiamo sempre il MODUS PONENS.

Il concetto di VALIDITA' di una fbf dipende dalla sua struttura logica

Un'interpretazione sulla segnatura Σ ($\mathcal{A}_\Sigma$) è una particolare visione dei simboli di Σ.

Un'interpretazione $\mathcal{A}_\Sigma$ è un "mondo possibile" in cui, ad ogni simbolo di funzione associamo un individuo e ad ogni simbolo di relazione associamo una precisa relazione.

Qual è la nozione di VERITA'?

Definiamo una funzione di interpretazione $[\![\ ]\!]$ che dipende dall'interpretazione $\mathcal{A}$.

Presa la fbf φ, restituisce 0 o 1.

$[\![\varphi]\!]^{\mathcal{A}_\Sigma} = 0 / 1$

Per definire questa funzione partiamo dalle fbf più semplici.

- $[\![\underbrace{P(t_1, t_2, \dots, t_n)}_{\substack{P \text{ è una relazione} \\ t_i \text{ sono gli individui}}}]\!]^{\mathcal{A}_\Sigma} = 1$ se gli elementi t_i sono accomunati dalla relazione P.

- $[\![\alpha \to \beta]\!]^{\mathcal{A}_\Sigma} = \begin{cases} 0 & \text{se } [\![\alpha]\!]^{\mathcal{A}_\Sigma} = 1 \text{ e } [\![\beta]\!]^{\mathcal{A}_\Sigma} = 0 \\ 1 & \text{altrimenti} \end{cases}$

- $[\![\neg\alpha]\!]^{\mathcal{A}_\Sigma} = \begin{cases} 0 & \text{se } [\![\alpha]\!]^{\mathcal{A}_\Sigma} = 1 \\ 1 & \text{se } [\![\alpha]\!]^{\mathcal{A}_\Sigma} = 0 \end{cases}$

N.B. Se il valore di verita' della funzione $P(x)$ dipende da un generico individuo x, dobbiamo definire un **AMBIENTE ρ** cioe' una funzione definita $\rho : \text{Var ind} \to |\mathcal{A}_\Sigma|$ cioe' che ad una variabile individuale associa una precisa interpretazione.

N.B. la notazione $\rho_x^a(y)$ ci dice che $\rho_x^a(y) = \begin{cases} \rho(y) & \text{se } y \neq x \\ a & \text{se } y = x \end{cases}$

- $[\![\forall x. \underbrace{\varphi(x)}_{\varphi(x) \text{ è una relazione generica}}]\!]_\rho^{\mathcal{A}_\Sigma} = \begin{cases} 1 & \text{se per ogni } a \in |\mathcal{A}_\Sigma|.\ [\![\varphi(x)]\!]_{\rho_x^a}^{\mathcal{A}_\Sigma} = 1 \\ 0 & \text{altrimenti} \end{cases}$

Se volessimo aggiungere $\exists x. \varphi(x)$

$$[\![\exists x. \varphi(x)]\!]_\rho^{\mathcal{A}_\Sigma} = \begin{cases} 1 & \text{se esiste } a \in |\mathcal{A}_\Sigma|.\ [\![\varphi(x)]\!]_{\rho_x^a}^{\mathcal{A}_\Sigma} = 1 \\ 0 & \text{altrimenti} \end{cases}$$

Una fbf α è valida se per qualsiasi $\mathcal{A}_\Sigma$ e qualsiasi ρ, $[\![\alpha]\!]_\rho^{\mathcal{A}_\Sigma} = 1$

Il sistema del calcolo dei predicati e corretto e completo rispetto al concetto di validita'.

VEDERE LA DEFINIZIONE 4.9, 4.12

Rispetto alla relazione di conseguenza logica $\models$,
il sistema è corretto e completo $(\Gamma \models \alpha \iff \Gamma \vdash \alpha)$

Abbiamo visto la definizione di logica dei predicati
nella VERSIONE ALLA HILBERT.

Ora vediamo la VERSIONE IN DEDUZIONE NATURALE

Non abbiamo assiomi, ma regole di inferenza.

Aggiungiamo alcune regole per $\forall$ e $\exists$:

- ELIMINAZIONE DELLA QUANTIFICAZIONE UNIVERSALE $(\forall E)$

$$\frac{\forall x A}{A[t/x]}$$

Se per qualsiasi x, la formula A è vera, allora posso sostituire in A qualsiasi elemento t al posto del generico x.

- INTRODUZIONE DELLA QUANTIFICAZIONE UNIVERSALE $(\forall I)$

$$\frac{A}{\forall x A}$$

La condizione è che non dobbiamo avere libere di x in assunzioni non scaricate nella sottoderivazione che termina con A.

- INTRODUZIONE DELLA QUANTIFICAZIONE ESISTENZIALE $(\exists I)$

$$\frac{A[t/x]}{\exists x A}$$

Se possiamo dire che A vale per almeno una t, allora potremo dire che esiste un valore x per cui A vale.

- **ELIMINAZIONE DELLA QUANTIFICAZIONE ESISTENZIALE** ($\exists E$)

$$\dfrac{\exists x A \qquad \begin{array}{c}[A]\\ \vdots \\ B\end{array}}{B}$$

Con la condizione che x non sia libera in B o in alcuna assunzione non scaricata della sottoderivazione che termina per B.

Lezione 23. 09/01/2019

domenica 16 dicembre 2018 20:23

MADONIA

(Pg. 169)

Nelle macchine di Turing la funzione δ non è una funzione totale: esistono coppie stato - simbolo per cui non è definita una configurazione successiva.

Le TM possono essere usate per riconoscere linguaggi su un alfabeto $\Sigma \subseteq \Gamma$.

Dunque $L \subseteq \Sigma^*$ dove $\Sigma \subseteq \Gamma$.

Si possono anche usare per calcolare funzioni.

In questo caso vengono chiamate TRASDUTTORI.

Dato un insieme L, la FUNZIONE CARATTERISICA DI L la funzione $f_L(x) = \begin{cases} 1 & \text{se } x \in L \\ 0 & \text{se } x \notin L \end{cases}$

Dunque le TM che riconoscono un linguaggio possono essere viste come dei trasduttori che calcolano f_L.

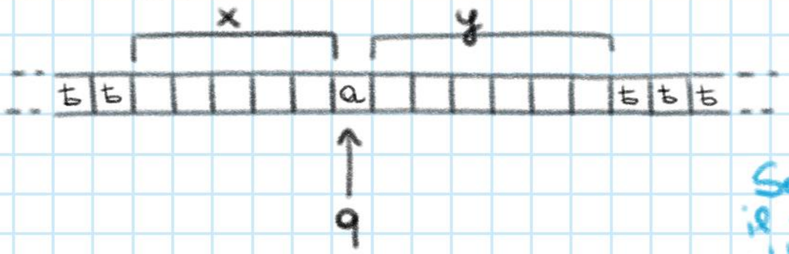
Vediamo un esempio di δ.

$\Gamma = \{0, 1, *, \$\}$

$Q = \{q_0, q_1, q_2, \dots\}$

δ	0	1	*	\$	$\sqcup$
q_0	$q_1, *, d$	$q_2, \$, d$	...	...	$q_f, \sqcup, i$
q_1	...	...	...	...	...
...	...	...	...	...	...

Durante la computazione, per rappresentare la configurazione, dobbiamo considerare la porzione di nastro in cui sono presenti caratteri non $\sqcup$, e la posizione della testina.



Se la testina sta leggendo il primo carattere, a sx abbiamo $\sqcup$, dunque $x = \varepsilon$

DEFINIZIONE DI CONFIGURAZIONE

Si definisce configurazione di una TM con alfabeto Γ e insieme di stati Q, una stringa xqy con $x \in \Gamma \cdot \overline{\Gamma}^* \cup \{\varepsilon\}$, $q \in Q$ e $y \in \overline{\Gamma}^* \cdot \Gamma \cup \{\text{b}\}$

Es.

b	a	b	b	b	b	b	b	b

$\uparrow$ q

CONFIG: abbbbbqb

N.B. Si possono chiamare:

$L_T = \Gamma \cdot \overline{\Gamma}^* \cup \{\varepsilon\}$

$R_T = \overline{\Gamma}^* \cdot \Gamma \cup \{\text{b}\}$

L per LEFT

R per RIGHT

Una configurazione xqy si dice:

- INIZIALE se $x = \varepsilon$, $q = q_0$ e $y \in \underbrace{\Gamma^+ \cup \{\text{b}\}}$

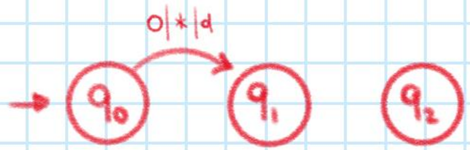
  Si pretende che la stringa y non contenga caratteri b al suo interno

  N.B. Se il nastro è tutto vuoto, la configurazione è q_0b.

- FINALE se $q \in F$ con $x \in L_T$ e $y \in R_T$

Esiste anche per le TM una rappresentazione tramite grafo.

$$\delta(q_0, 0) = (q_1, *, d)$$



DEFINIZIONI DI CONFIG. SUCCESS. (Pg. 174)

Facoltative!

DEFINIZIONE DI STRINGA ACCETTATA (Pg. 176)

Data una TM deterministica con alfabeto Γ e stato iniziale q_0, e dato $\Sigma \subseteq \Gamma$ alfabeto di input,

una stringa $x \in \Sigma^*$ è accettata
da M se $\exists$ una computazione
che parte dalla configurazione iniziale $q_0 x$
e termina in una configurazione finale.

La stringa invece non viene accettata se
giunti ad una coppia stato-simbolo
non è definita una configurazione successiva

N.B. La computazione potrebbe anche andare
all'infinito.

Esercizio: Definire una TM che riconosca $L = \langle aa(a+b)^* b \rangle$

δ	a	b	$\sqcup$
q_0	q_1, a, D	/	/
q_1	q_2, a, D	/	/
q_2	q_2, a, D	q_3, b, D	/
q_3	q_2, a, D	q_3, b, D	$q_4, \sqcup, D$
q_4	/	/	/

$F = \{q_4\}$

Esercizio: $L = \{a^h b^k \mid h > k\}$

δ	a	b	$\sqcup$
q_0	$q_1, \sqcup, D$	/	/
q_1	q_1, a, D	q_2, b, D	q_F
q_2	/	q_2, b, D	$q_3, \sqcup, S$
q_3	/	$q_4, \sqcup, S$	/
q_4	q_4, a, S	q_4, b, S	$q_0, \sqcup, D$

$F = \{q_F\}$

A Pag 180, paragrafo 5.3 - Calcolabilità secondo Turing

Lezione 24. 16/01/2019

domenica 16 dicembre 2018 20:23

BARBANERA

MACCHINE ASTRATTE (o VIRTUALI)

Introduciamo il concetto di Macchina Astratta Imperativa: è un insieme di algoritmi e strutture dati in grado di memorizzare ed eseguire dei programmi

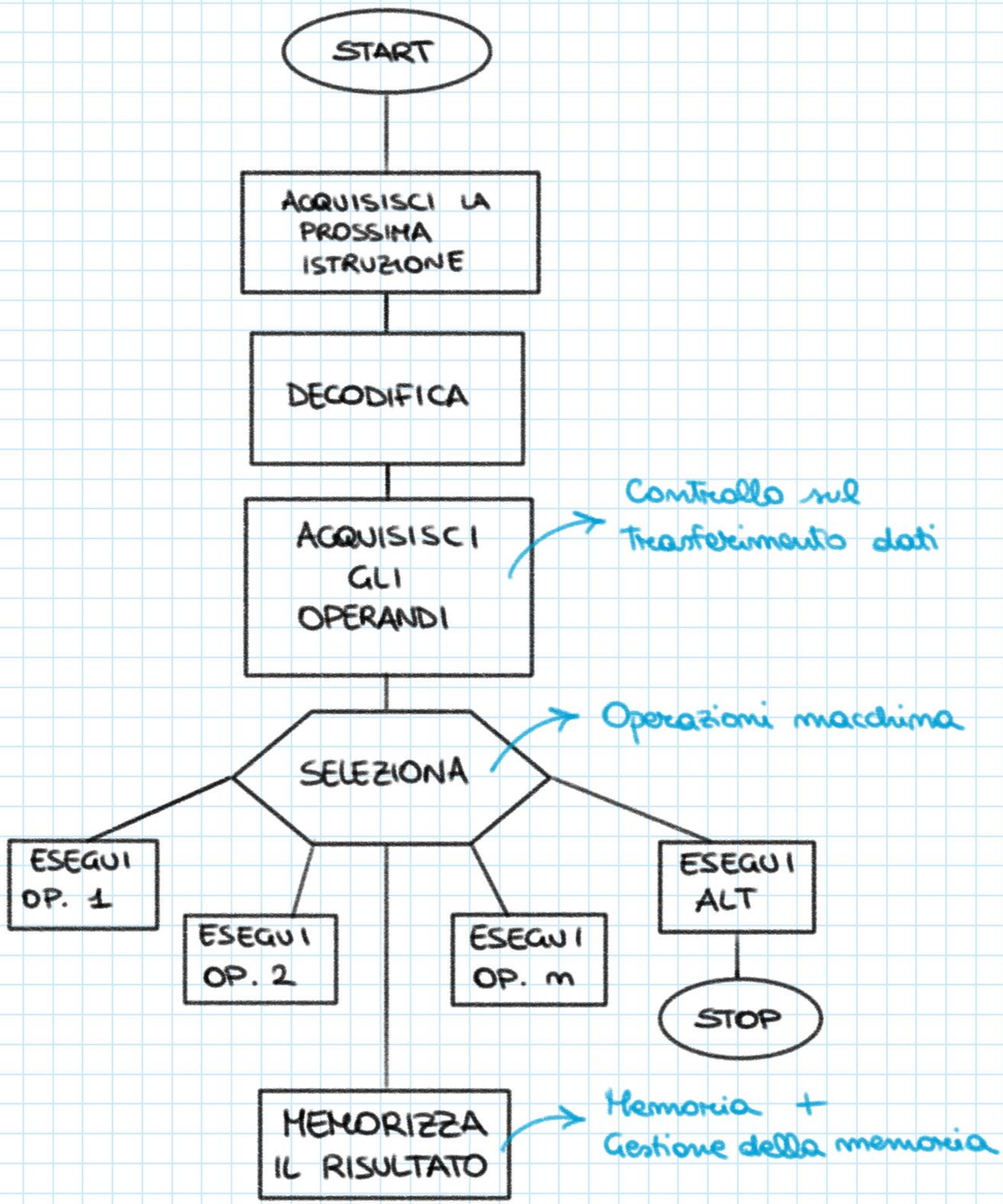
Per semplicità raggruppiamo algoritmi e strutture dati in diverse componenti che interagiscono tra loro.

Le principali componenti sono:

- Memoria: si occupa di memorizzare il programma e i dati.
  Tiene conto dell'ordine delle istruzioni del programma

- <u>Operazioni Macchina</u>: Insieme degli algoritmi e delle operazioni.
- <u>Controllo sequenza</u>: permette di eseguire i programmi nella corretta sequenza. Attraverso degli indici tiene conto dell'ordine delle istruzioni.
- <u>Controllo sul Trasferimento Dati</u>: permette di reperire i dati in memoria e utilizzarli nelle operazioni.
- <u>Gestione della Memoria</u>: opera sulla memoria.
- <u>Interprete</u>: è il cuore della macchina astratta che coordina tutte le componenti.

ALGORITMO DELL' INTERPRETE



Viene chiamato **CICLO DI FETCH-EXECUTE** (estrai ed esegui)

E' un ciclo potenzialmente infinito che si blocca se si esegue l'operazione di ALT.

Data una macchina astratta si possono "costruire" i linguaggi di programmazione.

Ad un linguaggio si può associare una macchina astratta

REALIZZAZIONE DI UNA MACCHINA ASTRATTA

Ci sono 3 tipi:

1) **Hardware**: Prendere gli algoritmi e costruire dei circuiti che rappresentano le strutture dati.

   Il problema sono i costi elevati.

   [PicoJava è una realizzazione hardware di Java]

   Tra gli hardware ci sono i FIRMWARE, dove tutte le componenti sono nello stesso chip.

   Dunque si possono rappresentare linguaggi semplici.

2) **Interpretativa**
3) **Compilativa**

   } Si possono utilizzare se si dispone già di una macchina astratta utilizzabile.

Se ad esempio voglio creare una macchina MA2 associata ad un linguaggio L_2, devo disporre di una MA1 esistente che opera su un linguaggio L_1.

REALIZZAZIONE INTERPRETATIVA

Presa una MA2 (creata al livello astratto), che opera su un L_2. Abbiamo bisogno di una MA1 già esistente che prende tutte le componenti di MA2 (algoritmi e strutture dati) e li ricrea in L_1.

REALIZZAZIONE COMPILATIVA

Avremo un programma C (compilatore) scritto in L_1. Il risultato è che noi diamo un programma P_{L_2} al compilatore e questo restituisce un programma P_{L_1} che fa la stessa cosa.

$$P_{L_2} \longrightarrow C \longrightarrow P_{L_1}$$

Per questioni di costo, una macchina creata in Hardware opera su un linguaggio semplice.

Lezione 25. 21/01/2019

domenica 16 dicembre 2018 20:23

MADONIA

CODICI (Testo CB)

Si tratta della rappresentazione delle informazioni.

Quando si vuole passare l'informazione da un sistema ad un altro bisogna eseguire una TRADUZIONE in modo corretto.

Supponiamo di avere:

- un insieme X (finito o infinito) di informazioni da trattare.
- un alfabeto A (finito) del sistema che riceve l'informazione.

Bisogna associare ad ogni elemento di X una stringa su A.

La FUNZIONE DI CODIFICA è definita $cod: X \rightarrow A^*$

E' una funzione INIETTIVA perché per $x_1 \neq x_2$, $cod(x_1) \neq cod(x_2)$.

Serve anche una FUNZIONE DI DECODIFICA che fa proprio il lavoro inverso:

$decod: A^* \rightarrow \overbrace{X \cup \{errore\}}^{*}$, tale che $decod(cod(y)) = y$

*

Non necessariamente tutti gli elementi di A^* hanno un'immagine in X (A^* è infinito, X può essere finito).

N.B. decod non è iniettiva:

possiamo avere più elementi y_i di A^* per cui $decod(y_i) =$ errore.

Per simboleggiare la fine di una stringa, se usiamo codici a lunghezza variabile, avremo bisogno di un ulteriore simbolo di "fine-stringa". Se invece abbiamo un codice a lunghezza fissa, sappiamo che in ogni blocco di memoria avremo una stringa.

RAPPESENTAZIONE POSIZIONALE (CODICI N-ARI POSIZIONALI)

- <u>base 10</u>:

$(57)_{10} = 5 \cdot 10^1 + 7 \cdot 10^0$

$(1432)_{10} = 1 \cdot 10^3 + 4 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0$

- <u>base 2</u>:

Si procede allo stesso modo

$(110)_2 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (5)_{10}$

Vediamo come si scrive il 57 in base 2

$57 = 28 \cdot 2 + 1$

$28 = 14 \cdot 2 + 0$

$14 = 7 \cdot 2 + 0$

$7 = 3 \cdot 2 + 1$

$3 = 1 \cdot 2 + 1$

$1 = 0 \cdot 2 + 1$

$(57)_{10} = (111001)_2$

Si può passare da una base ad un altra,
senza passare dalla base 10 solo in alcuni casi.
Vediamo il caso di basi che sono potenze di 2:

- base 2 $\rightarrow$ base 16

$(00101101)_2 = (45)_{10}$

Per la base 16 si usano i simboli $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \overset{10}{\overset{\shortparallel}{A}}, \overset{11}{\overset{\shortparallel}{B}}, \overset{12}{\overset{\shortparallel}{C}}, \overset{13}{\overset{\shortparallel}{D}}, \overset{14}{\overset{\shortparallel}{E}}, \overset{15}{\overset{\shortparallel}{F}})$

$(00101101)_2 = (2D)_{16}$

$(2D)_{16} = 2 \cdot 16^1 + 13 \cdot 16^0 = 32 + 13 = 45$

Per convertire basta prendere
blocchi di 4 cifre binarie $(16 = 2^4)$

$\underbrace{0010}_{2}\underbrace{1101}_{13}$

Per fare l'inverso basta scrivere ogni
cifra del numero in base 16 con
4 cifre in base 2.

supponiamo di avere una sequenza di 8 cifre in base 2.

$C_7 C_6 C_5 C_4 C_3 C_2 C_1 C_0$ base 2

$e_1 \; e_0$ base 16

Dove:

$e_1 = (C_7 C_6 C_5 C_4)_2$

$e_0 = (C_3 C_2 C_1 C_0)_2$

Infatti

$$x = C_7 \cdot 2^7 + C_6 \cdot 2^6 + C_5 \cdot 2^5 + C_4 \cdot 2^4 + C_3 \cdot 2^3 + C_2 \cdot 2^2 + C_1 \cdot 2^1 + C_0 \cdot 2^0$$

$$x = \underbrace{\left(C_7 \cdot 2^3 + C_6 \cdot 2^2 + C_5 \cdot 2^1 + C_4 \cdot 2^0\right)}_{e_1} \underbrace{(2^4)^1}_{16^1} + \underbrace{\left(C_3 \cdot 2^3 + C_2 \cdot 2^2 + C_1 \cdot 2^1 + C_0 \cdot 2^0\right)}_{e_0} \underbrace{(2^4)^0}_{16^0}$$

Per dire che un numero è scritto in esadecimale si precede il numero con "0x".

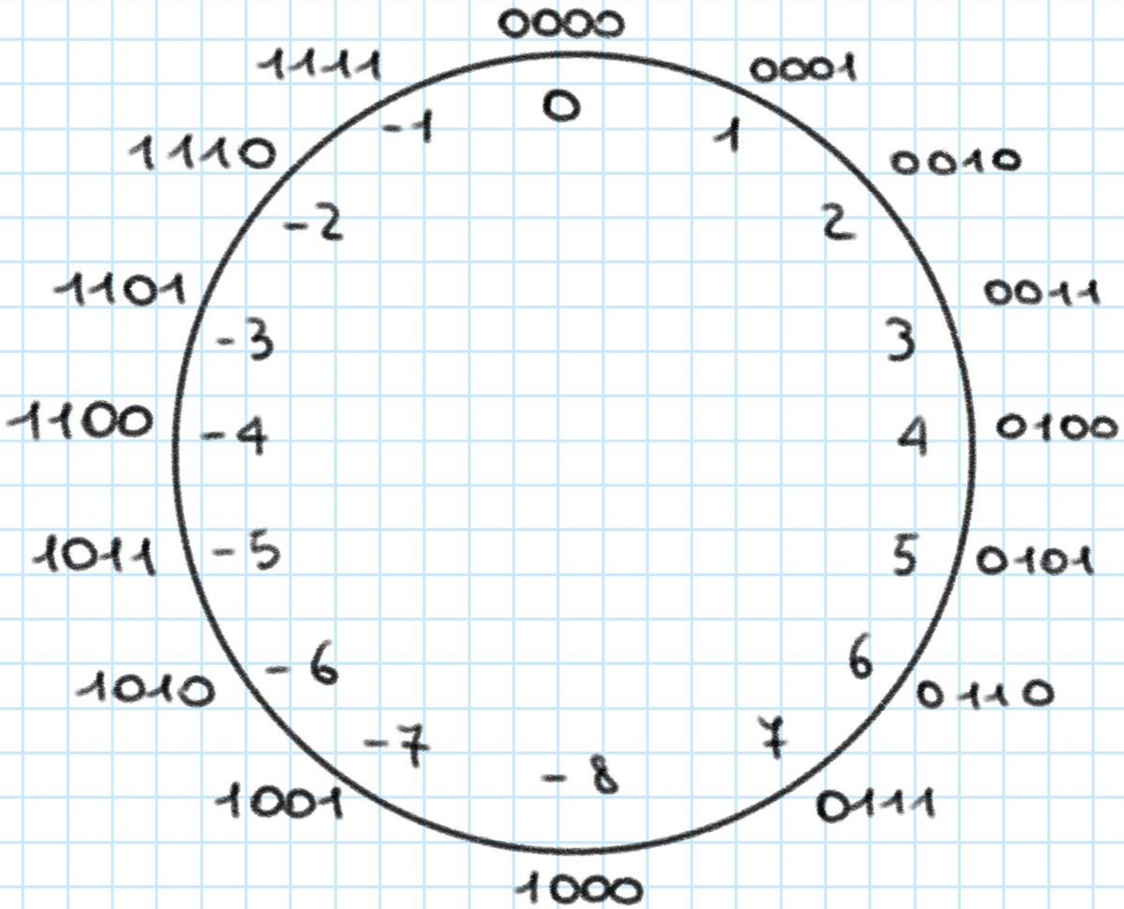
Tra i codici a lunghezza variabile ci sono i **CODICI A ESPANSIONE**:
Si divide l'insieme X (di elementi da rappresentare)
in sottoinsiemi, e si decide per ogni sottoinsieme,
una lunghezza fissa per la rappresentazione.

RAPPRESENTAZIONE IN COMPLEMENTO A 2

Ha una rappresentazione unica dello zero:
in molte rappresentazioni che prevedono
anche i numeri negativi, lo zero esiste come
zero positivo e zero negativo.

$p = 4$ (n° di bit usati)
$B = 2$ (base)

Con 4 bit si rappresentano:
- lo zero
- 7 numeri positivi ($1 \rightarrow 7$)
- 8 numeri negativi ($-1 \rightarrow -8$)



Tutti i negativi cominciano per 1

Tutti i positivi cominciano per 0

Se si vuole ottenere un numero

negativo n basta rappresentare in binario $2^4 + n$.

[Es. -5 (1011)

$2^4 - 5 = 16 - 5 = 11$

$(11)_{10} = (1011)_2$]

Si può anche prendere lo stesso numero ma positivo e sommare 1 al suo complemento.

[Es. -5 (1011)

Prendiamo il 5

$(5) = (0101)_2$

1010 è il complemento

Aggiungiamo 1 e otteniamo 1011.]

Lezione 26. 25/01/2019

domenica 16 dicembre 2018 20:23

MADONIA

Se chiamiamo:

p numero di bit,

B base,

possiamo definire:

$cod^{p}_{z_{CB}}(r)$ = codifica intera in complemento a B del numero r in p posizioni.

intero

Nella lezione 25 abbiamo visto la codifica di 7 numeri positivi e 8 negativi in complemento a 2, in 4 posizioni.

Es. $cod^{4}_{z_{C2}}(2) = 0010$

Vediamo la regola

$$cod^{p}_{z_{CB}}(r) = \begin{cases} r \geq 0 \rightarrow cod^{p}_{NB}(r) \; * \\ r < 0 \rightarrow cod^{p}_{NB}(B^{p} - |r|) \; ** \end{cases}$$

* E' la codifica naturale del numero

$$cod^4_{ZC2}(2) = cod^4_{N2}(2) = 0010$$

** E' la codifica naturale di $B^p - |x|$

$$cod^4_{ZC2}(-4) = cod^4_{N2}(2^4 - |-4|) =$$

$$= cod^4_{N2}(12) = 1100$$

In generale se siamo in base B e usiamo p posizioni, possiamo rappresentare numeri che vanno da $-B^{p-1}$ a $B^{p-1}-1$

$[-B^{p-1}, B^{p-1}-1]$

Con questi tipi di codifica, la somma è una semplice somma tra numeri:

base 2, $p=16$

105 - 128

$cod^{16}_{zC2}(105) = 0000000001101001$

$cod^{16}_{zC2}(-128) = 1111111110000000$

$cod^{16}_{zC_2}(105) \oplus cod^{16}_{zC_2}(-128) =$

$$
\begin{array}{rl}
= 0000000001101001 & + \\
1111111110000000 & = \\
\hline
1111111111101001 &
\end{array}
$$

$cod^{16}_{zC_2}(-23) = 1111111111101001$

DIMOSTRAZIONE SOMMA

Ovviamente vale solo se la somma tra i due numeri è rappresentabile in quel sistema.

$$cod^P_{ZCB}(X+Y) = cod^P_{ZCB}(X) \ (+) \ cod^P_{ZCB}(Y)$$

$$cod^P_{ZCB}(X-Y) = cod^P_{ZCB}(X) \ (+) \ cod^P_{ZCB}(-Y)$$

Vediamo solo questo caso particolare

con $X, Y > 0$ e $Y > X$

Vedi sopra

$$= cod^P_{ZCB}(X) \ (+) \ cod^P_{ZCB}(-Y) = cod^P_{NB}(X) \ (+) \ cod^P_{NB}(B^P - Y) =$$

$$= cod^P_{NB}(X + B^P - Y) = cod^P_{NB}(B^P - (Y - X)) =$$

$$= cod^P_{ZCB}(X - Y)$$

... LEFT BEHIND

CARDINALITA' DEI LINGUAGGI (1.3.4 [AAG])

Se abbiamo un alfabeto Σ, quante sono le possibili stringhe su Σ?

(Qual è la cardinalità di Σ^*?)

L'insieme Σ^* è numerabile:

possiamo instaurare una corrispondenza biunivoca con l'insieme $\mathbb{N}$.

Se prendiamo l'alfabeto $\{a,b\}$ possiamo stabilire un ORDINAMENTO LESSICOGRAFICO delle stringhe:

ε,	a,	b,	aa,	ab,	ba,	bb,	aaa, ...
1	2	3	4	5	6	7	8

dove la lunghezza è sempre crescente.

Se un insieme è numerabile, l'insieme delle parti non è numerabile. Ecco che quindi un linguaggio su un Σ^* numerabile non è numerabile.

Esistono programmi PASCAL che riconoscono linguaggi. Possiamo vedere i programmi come stringhe di Σ^*. Poiché l'insieme dei linguaggi è più che numerabile, esisteranno linguaggi non riconosciuti (non decidibili).

Lezione 27. 28/01/2019

domenica 16 dicembre 2018 20:23

BARBANERA

ESERCIZIO: 12. logica dei predicati

$\{\{c^0, k^0, f^1, g^2\}, \{P^1, Q^1\}\}$

- $\forall x. Q(x,x)$

VERA

$|A_\varepsilon| \to \mathbb{N}$

$Q \to$ "uguaglianza"

FALSA

$|A_\varepsilon| \to \mathbb{N}$

$Q \to$ "essere maggiore"

- $\forall x. \forall y. Q(x,y)$

VERA

$|A_\varepsilon| \to$ "una famiglia"

$Q \to$ "essere parenti"

FALSA

$|A_\varepsilon| \to$ "il mondo"

$Q \to$ "essere parenti"

ESERCIZIO: $\vdash (\forall x.\,(A \to B)) \to ((\forall x.A) \to (\forall y.B))$

$\forall x.(A \to B),\ \forall x.A \vdash \forall y.B$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{[\forall x.(A \to B)]^2 \qquad [\forall x.A]^1}{A \to B \qquad A}
}{B}
}{\forall y.B}
}{(\forall x.A) \to (\forall y.B)}\,(1)
}{(\forall x.(A \to B)) \to ((\forall x.A) \to (\forall y.B))}\,(2)
$$