Linguaggi

Definiamo un linguaggio come un insieme di stringhe ammissibili su un certo alfabeto A

- Alfabeto: Insieme finito di simboli
- Stringhe: sequenze di lunghezza arbitraria di simboli

Esempio: nei linguaggi di programmazione le stringhe ammissibili sono i programmi

Abbiamo poi:

- A alfabeto
- A^* tutte le sequenze su A di qualsiasi lunghezza
- $L \subseteq A^*$ è il linguaggio L su A

Quindi per descrivere un linguaggio abbiamo bisogno di un modo per determinare un sottoinsieme di tutte le stringhe su un certo alfabeto, ovvero quelle ammissibili, con due techiche:

1. Generazione: con una grammatica si generano tutte le stringhe del linguaggiio
2. Accettazione/Riconoscimento: con un automa si accettano/riconoscono tutte e sole le stringhe del linguaggio

Questa è parte della teoria dei linguaggi formali

Alfabeto

Un Alfabeto è un insieme finito di simboli, esempi:

- $\{a, b, c, \ldots, y, z\}$
- $\{A, B, C, \ldots, Y, Z\}$
- $\{\alpha, \beta, \gamma, \ldots, \psi, \omega\}$
- $\{\mathrm{A}, \mathrm{B}, \Gamma, \ldots, \Psi, \Omega\}$
- $\{1, 2, 3, \ldots, 9\}$
- $\{1, 2, 3, \ldots, 9, A, B, \ldots, F\}$
- $\{0, 1\}$
- $\{0, 1, 2, \ldots, 9\} \cup \{+, \times, -, \div\}$

Stringhe su un Alfabeto A

Una stringa su A è una sequenza di lunghezza arbitraria di simboli di A. L'insieme delle stringhe su A è A^*. Si chiamano Parole in caso di un linguaggio naturale

Definizione induttiva di A^*:

[Caso Base] $\varepsilon \in A^*$ stringa vuota

[Passo Induttivo] $\omega \in A^* \Rightarrow a\omega \in A^*,\ \forall a \in A$

Esempi

- Su alfabeto latino minuscolo: *anna, bob, dsdshd, ε, iiiiii*
- *Su alfabeto cifre decimali: 1945, 27, 10, 50, 0020*
- *Su alfabeto cifre esadecimali: F2, A8, CAFE, 3A8*
- *Su alfabeto dell'aritmetica:* $1 + 4 \times 5 - 6 \div 9$

Si noti l'analogica con la definizione di liste su A

[Caso Base] $[\,] \in L_A \rightarrow$ Liste su A

[Passo Induttivo] $(a \in A \wedge lst \in L_A) \Rightarrow a : lst \in L_A$

Si può vedere come

[Caso Base] $[\,] \in L_A \rightarrow$ Liste su A

[Passo Induttivo] $(a \in A \wedge \omega \in L_A) \Rightarrow a : \omega \in L_A$

Si dimostra che $L_A \cong A^*$

> Le liste come le stringhe sono sequenze si elementi dell'insieme A di lunghezza arbitraria, se per la definizione induttiva di lista, la lista di elementi $a \in A$ di una lunghezza arbitraria n in una lista $lst \in L_A$, è una lista $[a_0, a_1, \ldots, a_n]$. Allo stesso modo per la definizione induttiva di stringa su un alfabeto A, la stringa di elementi $a \in A$ di una lunghezza arbitraria n in una stringa ω è una stringa $a_0 a_1 \ldots a_n$. L_A e A^* contengono quindi gli stessi elementi e la biiezione $L_A \cong A^*$ può essere vista simile ad una relazione $id_A : A \leftrightarrow A$ sull'insieme A. Di base la stringa vuota ε è in relazione con la lista vuota $[\,]$ per la definizione del caso base della dimostrazione induttiva di stringhe e lista.

Nel caso di stringhe solitamente A è un alfabeto, nel caso di liste invece è arbitrario

Linguaggio su A

Dato un alfabeto A, un linguaggio L su A è un insieme di stringe su A, quindi $L \subseteq A^*$

L'insieme di tutti i linguaggi su A è $\mathcal{P}(A^*)$, quindi $L \in \mathcal{P}(A^*)$

Esempi:

- Il vocabolario italiano è un linguaggio sull'alfabeto latino
- La lingua italiana è un linguaggio sull'alfabeto che contiene caratteri latini maiuscoli, minuscoli, spazi e punteggiatura
- Le espressioni aritmetiche sono un linguaggio sull'alfabeto $\{0, 1, 2, \ldots, 9\} \cup \{+, -, \times, \div\}$
- Un linguaggio di programmazione è un linguaggio sull'alfabeto $ASCII$ o $UNICODE$

Automi

Gli automi sono vari tipi di macchine a stati fondamentali per l'informatica.

Esempio gli Automi di Moore, di Mealy e le Macchine di Turing

Vedremo gli Automi a stati finiti (deterministici e non-deterministici) per il riconoscimento di linguaggi.

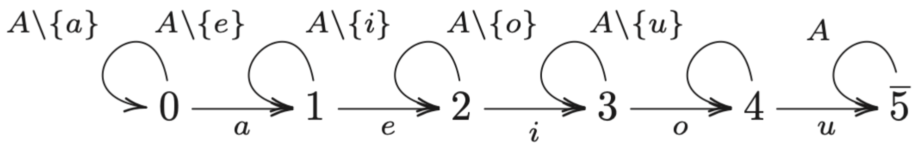
L'idea di fondo è:

- Si legge una stringa in input a partire da uno stato (iniziale)
- Ogni carattere causa una transizione di stato
- La stringa è riconosciuta dallo stato iniziale se alla fine lo stato raggiunto è di accettazione

Un esempio di automa

Sia L l'insieme delle parole inglesi che contengono le vocali $\{a, e, i, o, u\}$ in questo ordine

Un automa che riconosce questo linguaggio dovrà ricordare le vocali già lette e cambiare stato quando ne legge un'altra



Ad esempio: $\text{sacrilegious} \in L, \ \text{hello} \notin L$

Definizione formale di Automa

Dato un alfabeto A, un automa a stati finito su A è una tripla:

$$\mathcal{A} = (S, T, F)$$

- S è un insieme finito, l'insieme degli stati
- $T \subseteq (A \times S) \times S$ è la relazione di transizione
- $F \subseteq S$ è l'insieme di stati finali (o di accettazione)

Nell'automa precedente per esempio la transizione $3 \to 4$ è una relazione di transizione $(o, 3), 4)$, all'occorrenza di o passa dallo stato 3 allo stato 4

La parola face $\notin L$ perché non finisce in uno stato finale, stato finale che in questo caso è $F = \{5\}$

Riprendendo quindi l'automa visto in precedenza possiamo vedere l'automa rappresentato come

$$\mathcal{A} = (S, T, F) \quad S = \{0, 1, 2, 3, 4, 5\} \quad F = \{5\},\ F \subseteq S$$

$$T = \begin{array}{ll} \{((a,0),1), ((e,1),2), ((i,2),3), ((o,3),4), ((u,4),5)\} & \cup \\ \{((i,0),0) \mid i \in A \backslash \{a\}\} & \cup \\ \{((i,1),1) \mid i \in A \backslash \{e\}\} & \cup \\ \{((i,2),2) \mid i \in A \backslash \{i\}\} & \cup \\ \{((i,3),3) \mid i \in A \backslash \{o\}\} & \cup \\ \{((i,4),4) \mid i \in A \backslash \{u\}\} & \cup \\ \{((i,5),5) \mid i \in A\} & \end{array}$$

Raggiungibilità

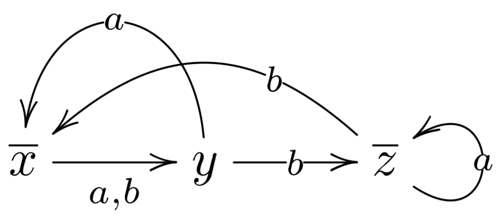
Sia $\mathcal{A} = (S, T, F)$ un automa sull'alfabeto A

Per ogni $a \in A$ definiamo

$$T_a = \{(x, y) \mid ((x, a), y) \in T\} \in Rel(S, S)$$

Esempio

$$A = \{A, B\} \quad \mathcal{A} = (S, T, F) \quad S = \{x, y, z\} \quad F = \{x, y\}$$
$$T = \{((a,x),y), ((b,x),y), ((a,y),x), ((b,y),z), ((a,z),z), ((b,z),x)\}$$



- $T_a = \{(x, y), (y, x), (z, z)\}$
- $T_b = \{(y, z), (z, x), (x, y)\}$

La relazione $T_a; T_b$ ci da la raggiungibilità di ab

Un altro esempio

Sia $\mathcal{A} = (S, T, F)$ automa sull'alfabeto $A = \{a, b\}$ con $S = \{x, y, z\}$ $F = \{x, z\},\ F \subseteq S$

Per $\omega \in A^*$ definiamo induttivamente $T_\omega \in Rel(S, S)$

[Caso Base] $T_\varepsilon = id_S$

[Passo Induttivo] $T_{a\omega} = T_a; T_\omega$

Se $(x,y) \in T_\omega$ allora y è raggiungibile da x con la stringa ω

Esempio:

$$\begin{aligned} &= T_a;T_{bb} \\ &= T_a;(T_b;T_b) \\ T_{abb} &= T_a;(T_b;(T_b;T_\varepsilon)) \\ &= T_a;(T_b;(T_b;id_S)) \\ &= T_a;T_b;T_b \end{aligned}$$

Linguaggio accettato da uno stato

Sia $\mathcal{A} = (S,T,F)$ automa sull'alfabeto $A = \{a,b\}$ con $S = \{x,y,z\}$ $F = \{x,z\},\ F \subseteq S$ e $x \in S$

Il linguaggio accettato da x è

$$\ll x \gg = \{\omega \in A^* \mid \exists y \in F\ .\ (x,y) \in T_\omega\}$$

Se $\omega \in \ll x \gg$ diciamo che ω è accettata dallo stato x

Esempio:

- $b \in \ll x \gg$? No, perché non raggiunge uno stato finale
- $bbb \in \ll x \gg$? Si, perché raggiunge lo stato finale $x \to y \to z \to \overline{x}$
- $bb \in \ll x \gg$?, Si, perché $x \to y \to \overline{z}$
- $a \in \ll z \gg$?, Si, perché $z \to \overline{z}$
- $\varepsilon \in \ll x \gg$?, Si, perché $\overline{x}$

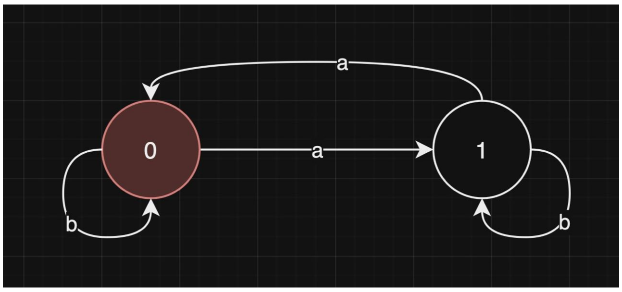
Esercizi:

Progettare un automa con uno stato che accetti tutte e sole le parole nell'alfabeto latino minuscolo che comincino con "al"

$$\begin{gathered} A = \{a,l\} \quad \mathcal{A} = (S,T,F) \quad S = \{0,1,2\} \quad F = \{2\} \\ T = \{((a,0),1),((l,1),2),((A,2),2)\} \end{gathered}$$

Sia $A = \{a,b\}$, Progettare un automa su A con uno stato che accetti tutte e sole le stringhe che contengono un numero pari di a

$$\begin{gathered} A = \{a,b\} \quad \mathcal{A} = (S,T,F) \quad S = \{0,1,2\} \quad F = \{2\} \\ T = \{((a,0),1),((a,1),0),((b,0),0),((b,1),1)\} \\ L = \{\omega \in A^* \mid \text{tutte le stringhe che contengono un numero pari di } a\} \end{gathered}$$

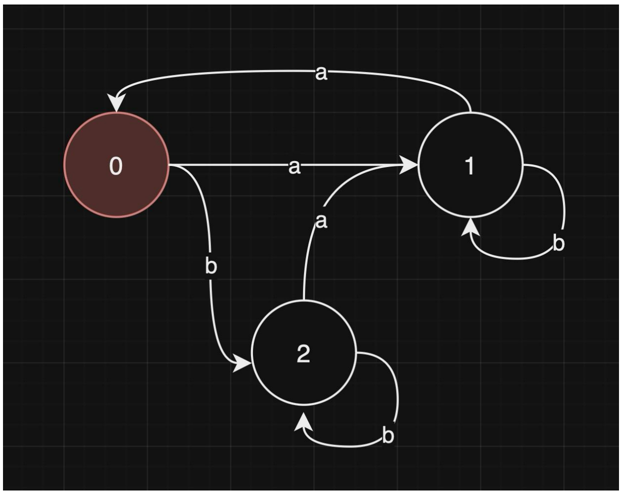


Questa soluzione accetta la stringa ω priva di a come stringa valida

- $bbbb \in L$
- $abba \in L$

Invece il seguente automa non la considera come valida

$$A = \{a, b\} \quad \mathcal{A} = (S, T, F) \quad S = \{0, 1, 2\} \quad F = \{2\}$$
$$T = \{((a, 0), 1), ((a, 1), 0), ((b, 1), 1), ((b, 0), 2), ((b, 2), 2), ((a, 2), 1)\}$$
$$L = \{\omega \in A^* \mid \text{tutte le stringhe che contengono un numero pari di } a\}$$



Questa soluzione non accetta la stringa ω priva di a come stringa valida, ci devono quindi essere almeno due a

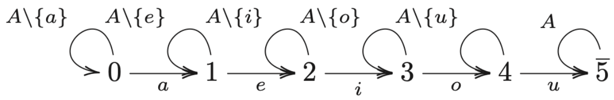
- $abba \in L$
- $bbb \notin L$
- $ababa \notin L$

Automi deterministici e non-deterministici

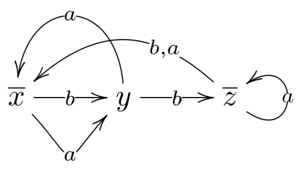
Un automa $\mathcal{A} = (S, T, F)$ è deterministico se e solo se la relazione di transizione è una funzione (quindi totale e univalente), ovvero per ogni coppia (a, S) esiste esattamente una transizione da prendere

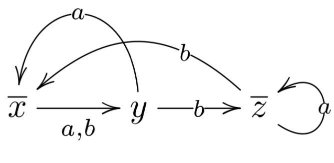
Esempi:

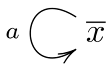
- A Alfabeto latino

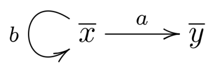


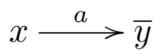
- $A = \{a, b\}$











Automi non-deterministici

Se un automa non è deterministico, la relazione di transizione T può non essere univalente, quindi anche le relazioni derivate T_a (per $a \in A$) e T_ω (per $\omega \in A^*$) possono non essere univalenti.

Ricordiamo il linguaggio accettato da x è

$$\ll x \gg = \{\omega \in A^* \mid \exists y \in F \,.\, (x, y) \in T_\omega\}$$

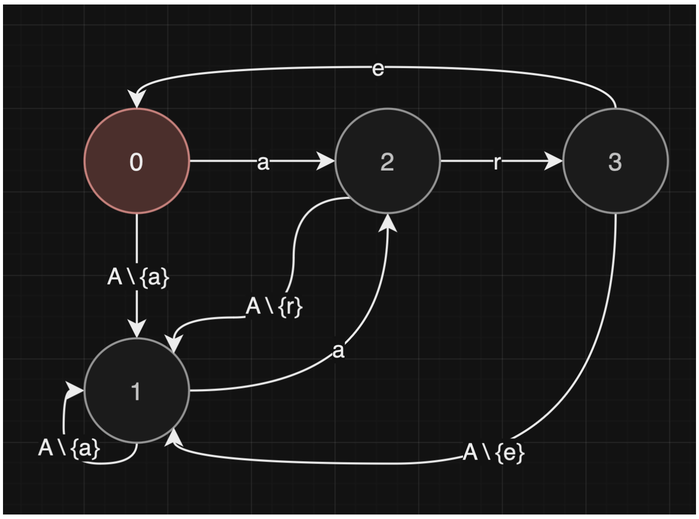
Quindi basta che ci sia uno stato di accettazione raggiungibile da x con la stringa ω perché questa sia accettata

Esercizio: progettare un automa che riconosca le parole italiane che finiscono con "are"

$$A = \{a, b, c, \ldots, y, z\} \quad \mathcal{A} = (S, T, F) \quad S = \{0, 1, 2, 3\} \quad F = \{2\}$$

$$T = \begin{array}{ll} \{((a,0),2), ((r,2),3), ((e,3),0)\} & \cup \\ \{((i,0),1) \mid i \in A \backslash \{a\}\} & \cup \\ \{((i,1),1) \mid i \in A \backslash \{a\}\} & \cup \\ \{((i,2),1) \mid i \in A \backslash \{r\}\} & \cup \\ \{((i,3),1) \mid i \in A \backslash \{e\}\} & \end{array}$$

$$L = \{\omega \in A^* \mid \text{tutte le stringhe che finiscono con } are\}$$



Grammatiche libere da contesto

Approccio generativo per descrivere linguaggi, utilizzato per specificare linguaggi di programmazione, linguaggi logici e altro.

Esempio: espressioni aritmetiche sull'alfabeto $A = \{1, 2, \ldots, 9\} \cup \{+, -, \times, \div\}$

- Solo alcune delle stringhe in A^* sono espressioni aritmetiche, per esempio

$$5 \times 3 + 1 \div 3 \text{ sintatticamente corretta}$$
$$5 + \times \div 3 \text{ sintatticamente scorretta}$$

Introduciamo quindi una grammatica per generare le espressioni corrette

Definizione formale di Grammatica

Una grammatica $\mathcal{G}$ su un alfabeto A è una coppia (S, P) dove

- S è l'insieme dei simboli non terminali ($S \cap A = \emptyset$)
- P un insieme di produzioni della forma $\langle X \rangle \rightsquigarrow \omega$
  dove
  $\langle X \rangle \in S$ e $\omega \in (S \cup A)^*$, nota che ω può essere ε

A volte scriviamo più produzioni per lo stesso simbolo non terminale separando i corpi delle produzioni con "|": $\langle X \rangle \rightsquigarrow \omega_1 \mid \omega_2 \mid \cdots \mid \omega_k$

Grammatica delle espressioni aritmetiche

Ogni riga è una produzione

- $\langle ExA \rangle, \langle Num \rangle$ simboli non terminali, o categorie sintattiche: rappresentano linguaggi
- $\rightsquigarrow$ metasimbolo ("può essere")
- $+, -, \times, \div$ simboli terminali
- Le produzioni hanno forma $Testa \rightsquigarrow Corpo$
  Dove
  $Testa$ è un simbolo non terminale, $Corpo$ una sequenza di simboli

La Grammatica completa con produzioni per $\langle Num \rangle$ e $\langle Cif \rangle$: un $\langle Num \rangle$ è una sequenza non vuota di $\langle Cif \rangle$

$$\begin{aligned}
&\langle Cif \rangle \rightsquigarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
&\langle Num \rangle \rightsquigarrow \langle Cif \rangle \mid \langle Num \rangle\langle Cif \rangle \\
&\langle ExA \rangle \rightsquigarrow \langle Num \rangle \mid \langle ExA \rangle + \langle ExA \rangle \mid \langle ExA \rangle - \langle ExA \rangle \mid \langle ExA \rangle \times \langle ExA \rangle \mid \langle ExA \rangle \div \langle ExA \rangle
\end{aligned}$$

Arricchiamo la nostra grammatica per generare anche espressioni con variabili come $x + y \times 3$

- Alfabeto $A = \{+, -, \times, \div\} \cup \{1, 2, \dots, 9\} \cup \{a, b, c, \dots, y, z\}$
- Nuove categorie sintattiche $\langle Ide \rangle, \langle Car \rangle, \langle OpA \rangle$

$$\begin{aligned}
&\langle Cif \rangle \rightsquigarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
&\langle Num \rangle \rightsquigarrow \langle Cif \rangle \mid \langle Num \rangle\langle Cif \rangle \\
&\langle ExA \rangle \rightsquigarrow \langle Num \rangle \mid \langle ExA \rangle\langle OpA \rangle\langle ExA \rangle \mid \langle Ide \rangle \\
&\langle OpA \rangle \rightsquigarrow + \mid - \mid \times \mid \div \\
&\langle Ide \rangle \rightsquigarrow \langle Car \rangle \mid \langle Ide \rangle\langle Car \rangle \\
&\langle Car \rangle \rightsquigarrow a \mid b \mid c \mid \cdots \mid y \mid z
\end{aligned}$$

Backus-Naur Form (BNF)

Le grammatiche libere da contesto vengono spesso presentate in *Backus-Naur Form*

Per la precedente grammatica

```
Cif ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Num ::= Cif | Num Cif
ExA ::= Num | ExA + ExA | ExA – ExA | ExA × ExA | ExA ÷ ExA
```

Albero di derivazione sintattica

Vediamo in che senso la grammatica vista genera le espressioni

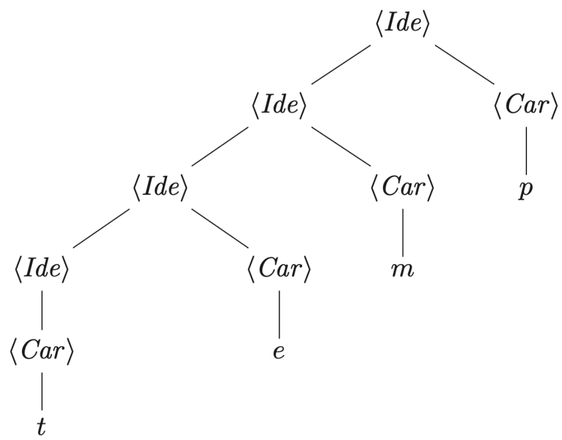
Sia $\mathcal{G} = (S, P)$ una grammatica libera da contesto sull'alfabeto A

Un albero di derivazione sintattica o parse tree è un albero radicato dove:

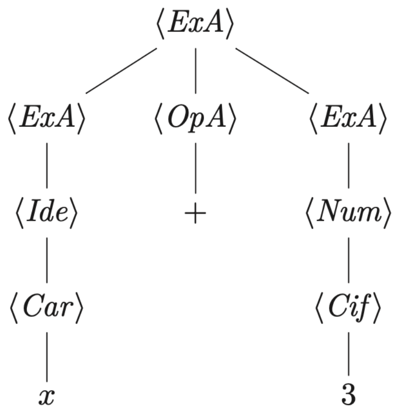
- Ogni nodo interno è etichettato con un simbolo non terminale (in S)
- Ogni foglia è etichettata da un simbolo terminale (in A) o da ε
- Se nodo interno x è etichettato con $\langle X \rangle$ allora deve esistere una produzione $\langle X \rangle \rightsquigarrow \omega$ in P tale che ω è la sequenza di etichette dei figli di x, da sinistra a destra

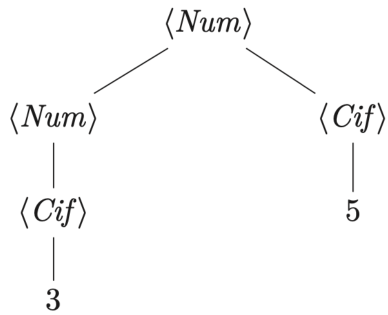
  Sia $v \in A^*$ la sequenza di etichette delle foglie (ignorando ε). Allora l'albero è un albero di derivazione per v

Consideriamo la stringa $temp$ usando la grammatica delle espressioni aritmetiche



Invece gli alberi di derivazione per $x + 3$ e 35 sono rappresentati come





Linguaggio generato da un non terminale

Sia $\mathcal{G} = (S, P)$ una grammatica libera da contesto sull'alfabeto A. Sia $\langle X \rangle \in S$ un simbolo non terminale.

Il linguaggio generato da $\langle X \rangle$ è l'insieme delle stringe $\omega \in A^*$ tali che esiste un albero di derivazione per ω avente come radice un nodo etichettato con $\langle X \rangle$

- Denoteremo questo linguaggio come $\ll \langle X \rangle \gg$

Esempi di linguaggi generati da non terminali, usando la grammatica delle espressioni aritmetiche

- $\ll \langle Num \rangle \gg$? $\{1, 2, 30, 40, 100, 245\}$
- $\ll \langle Ide \rangle \gg$? $\{aba, c, ciao\}$
- $35 \in \ll \langle Num \rangle \gg$? Si
- $35 \in \ll \langle Ide \rangle \gg$? No
- $35 \in \ll \langle ExA \rangle \gg$? Si, $\langle ExA \rangle \rightsquigarrow \langle Num \rangle$

Grammatica per $\mathcal{F}$-termini

Sia $\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}$ una segnatura. Gli $\mathcal{F}$-termini sono definiti induttivamente come

[Clausola Base] Per ogni costante $c \in \mathcal{F}_0,\ c \in \mathcal{F}Term$

[Clausola Induttiva] Per ogni $n \geq 1$ e per ogni simbolo $f \in \mathcal{F}_n$, se $t_1, t_2, \dots, t_n \in \mathcal{F}Term$, allora $f(t_1, t_2, \dots, t_n) \in \mathcal{F}Term$

- Introduciamo una categoria sintattica $\langle FIde_n \rangle,\ \forall n \in \mathbb{N} \mid \ll \langle FIde_n \rangle \gg = \mathcal{F}_n$
  Aggiungiamo una produzione
  $\langle FIde_n \rangle \rightsquigarrow f,\ \forall f \in \mathcal{F}_n$
- Definiamo $\langle Term \rangle$ con le produzioni

$$\langle Term \rangle \rightsquigarrow \langle FIde_0 \rangle \mid \langle FIde_n \rangle (\overbrace{\langle Term \rangle, \dots, \langle Term \rangle}^{n}), n \in \mathbb{N}$$

- Abbiamo che $\ll \langle Term \rangle \gg = \mathcal{F}Term$

Grammatica per Alberi binari

Applichiamo la definizione precedente alla segnatura per $\mathcal{BT}$

$$\mathcal{BT}_0 = \{\lambda\} \quad \mathcal{BT}_1 = \emptyset \quad \mathcal{BT}_2 = \{N\} \quad \mathcal{BT}_n = \emptyset,\ \forall n \geq 3$$

- $\langle \mathcal{BT}Ide_0 \rangle \rightsquigarrow \lambda$
- $\langle \mathcal{BT}Ide_2 \rangle \rightsquigarrow N$
- $\langle Term \rangle \rightsquigarrow \langle \mathcal{BT}Ide_0 \rangle \mid \langle \mathcal{BT}Ide_2 \rangle (\langle Term \rangle, \langle Term \rangle)$

Stringhe con più alberi di derivazione

Considerando la grammatica per le espressioni aritmetiche, vediamo come per ad esempio la stringa $y \times 5 + x$ possa avere due alberi di derivazione, uno che da la precedenza a $y \times 5$ e uno che la da a $5 + x$

Questi due alberi valutano l'espressione in modo diverso, dando risultati diversi

- Per le espressioni aritmetiche si risolve introducendo una precedenza sugli operatori ($\times$ ha la precedenza su $+$)
- Per poter valutare prima $+$ si introducono le parentesi nell'alfabeto e una produzione

$$\langle ExA \rangle \rightsquigarrow (\langle ExA \rangle)$$

Ambiguità di grammatiche

Una grammatica libera da contesto $\mathcal{G} = (S, P)$ è ambigua se esiste una stringa $\omega \in A^*$ e almeno due diversi alberi di derivazione per ω che hanno la stessa etichetta nella radice

Confronto tra automi e grammatiche

- Un automa su A accetta un linguaggio $\ll x \gg$ per ogni stato $x \in S$
- Una grammatica su A genera un linguaggio $\ll \langle X \rangle \gg$ per ogni $\langle X \rangle \in S$

I due formalismi sono equivalenti? Caratterizzano gli stessi linguaggi?

1. Per ogni automa a stati finiti $\mathcal{A}$ con stato x, esiste una grammatica libera da contesto $\mathcal{G}$ con categoria sintattica $\langle X \rangle$ tale che

$$\ll \langle X \rangle \gg = \ll x \gg \ ? \text{ Si}$$

2. Per ogni grammatica libera da contesto $\mathcal{G}$ con categoria sintattica $\langle X \rangle$, esiste un automa a stati finiti $\mathcal{A}$ con stato x tale che

$$\ll x \gg = \ll \langle X \rangle \gg \ ? \text{ No}$$

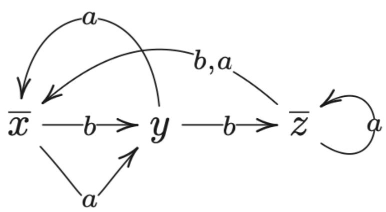
Estrazione di una grammatica da un automa

Sia A un alfabeto e $\mathcal{A} = (S, T, F)$ un automa a stati finiti su A, definiamo $\mathcal{G}_{\mathcal{A}} = (S_{\mathcal{A}}, P_{\mathcal{A}})$ come

$S_{\mathcal{A}} = \{\langle x \rangle \mid x \in S\}$

$P_{\mathcal{A}} = \{\langle x \rangle \rightsquigarrow \varepsilon \mid x \in F\} \cup \{\langle x \rangle \rightsquigarrow a\langle y \rangle \mid ((a, x), y) \in T\}$

- $\langle x \rangle \rightsquigarrow b\langle y \rangle \mid a\langle y \rangle \ \mid \varepsilon$
- $\langle y \rangle \rightsquigarrow a\langle x \rangle \mid b\langle z \rangle$
- $\langle z \rangle \rightsquigarrow a\langle z \rangle \mid a\langle x \rangle \mid b\langle x \rangle \mid \varepsilon$



Dato un automa $\mathcal{A} = (S, T, F)$ e $x \in S$

$$\forall \omega \in A^* \ . \ \omega \in \ll x \gg \Leftrightarrow \omega \in \ll \langle x \rangle \gg$$

Grammatica per un dato linguaggio

Per $n \in \mathbb{N}$ e $\omega \in A^*$ scriviamo ω^n per la stringa $\omega\omega \dots \omega, n$ volte

Sia $A = \{a, b\}$ e consideriamo il linguaggio $L = \{a^n b^n \mid n \in \mathbb{N}\}$

Mostriamo che il linguaggio generato da $\langle X \rangle \rightsquigarrow \varepsilon \mid a\langle X \rangle b$

- $aaabbb \in L$
- $aabbb \notin L$
- $abab \notin L$

Grammatiche che nessun automa a stati finiti può accettare

Nessun automa può riconoscere il linguaggio $L = \{a^n b^n \mid n \in \mathbb{N}\}$ sull'alfabeto $A = \{a, b\}$

Dimostrazione per assurdo

Supponiamo esista $\mathcal{A} = (S, T, F)$ con uno stato $x \in S$, tale che $\ll x \gg = L$ e $|S| = m$, mostriamo che otteniamo una contraddizione

1. Dato m, certamente $a^m b^m \in L$, quindi $a^m b^m \in \ll x \gg$
2. Ma allora esistono $y \in S$ e $z \in F$ tali che $(x, y) \in T_{a^m}$ e $(y, z) \in T_{b^m}$
3. $(x, y) \in T_{a^m}$ vuol dire che esiste un walk

4. Il walk passa per $m + 1$ stati. Per il principio delle buche e dei piccioni

$$\exists i, j \ . \ i \neq j \land x_i = x_j$$

5. Quindi esiste un walk di lunghezza $n < m$ da x_0 a x_m, e quindi $a^n b^m \in \ll x \gg$, non accettato dalla grammatica