Manuale di sopravvivenza per l'esame Fondamenti di Informatica

Una guida pratica per chi deve affrontare questo esame apparentemente insormontabile

Matteo Iervasi

Aprile 2018

Indice

Prefazione **3**

1 Linguaggi regolari **4**

2 Linguaggi liberi dal contesto e grammatiche **8**

3 Linguaggi non liberi dal contesto **14**

4 Esercizi vari **16**

5 Teoria della ricorsione **18**

Prefazione

Questo documento ha lo scopo di dare un'idea al povero studente che deve affrontare il temibile esame di *Fondamenti di Informatica* di come si affrontano gli esercizi.
NON tratterò l'aspetto teorico, per tanto assumo che si abbia già studiato (o almeno tentato di studiare) quella parte. Mi rendo conto che la materia in questione sia piuttosto ostica, ma vi posso assicurare che una volta compresi i concetti base il resto verrà da se. Ovviamente è **fondamentale** fare molti esercizi, in modo da verificare e consolidare l'apprendimento. Cercherò di essere il più chiaro possibile, ma non essendo mai stato bravo a spiegare potrebbero esserci dei punti non chiari, per i quali chiedo scusa in anticipo.

Voglio ringraziare Jenny Bonato per l'immensa pazienza che ha avuto per insegnare a questo somaro le basi di questa materia.

Qualora dovreste trovare degli errori, scrivetemi a matteoiervasi@gmail.com, oppure fate direttamente una "*pull request*" nel repository GitHub.

1 Linguaggi regolari

Anche se molto spesso il primo esercizio non tratta un linguaggio regolare, è sempre utile sapere come si deve procedere.

Come dice il prof. Giacobazzi, la prima cosa da fare quando si osserva un linguaggio è capire *intuitivamente* a che classe appartiene (regolare, CF, ecc.). L'intuizione può essere allenata con la pratica, tuttavia esiste un trucco molto utile: se nel linguaggio è necessario "contare" in qualche modo qualcosa, allora **sicuramente** questo linguaggio non sarà regolare. Dopo aver intuitivamente classificato il linguaggio bisogna procedere con la dimostrazione. Nel caso in cui il linguaggio sia **regolare**, bisogna costruire l'automa e dimostrarne la correttezza.

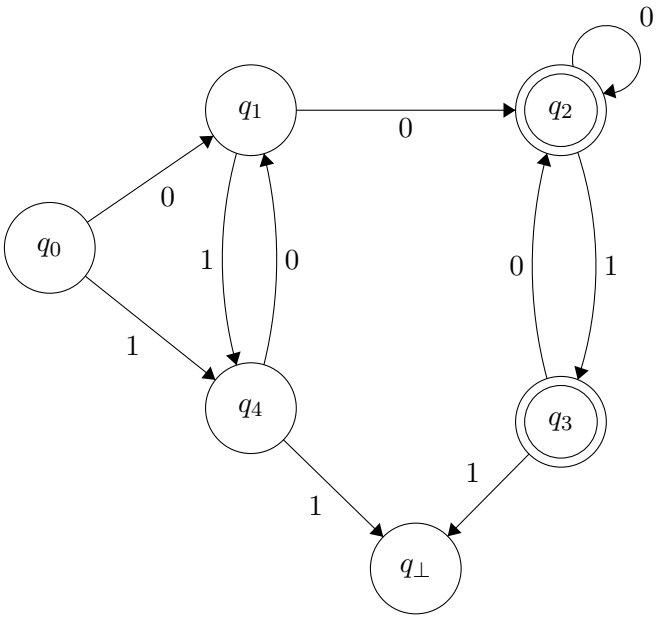
Ricordiamo che un'automa è scrivibile come una quintupla $< Q, \Sigma, \delta, q_0, F >$, dove:

- Q è l'insieme degli stati
- Σ è l'alfabeto di input
- $\delta : Q \times \Sigma \to Q$ è la funzione di transizione di uno stato
- q_0 è lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati finali

Prendiamo in esame il seguente linguaggio:

$L = \{\{0,1\}^*$ t.c. ci sono almeno due 0 consecutivi e non ci sono mai due 1 consecutivi$\}$

Disegniamo l'automa corrispondente (non mi soffermo sul come farlo, questo è argomento del corso di Architettura degli Elaboratori):



Ora è necessario dimostrare la correttezza dell'automa. Dobbiamo dimostrare che $x \in L \Leftrightarrow x \in L(n)$, tuttavia non dimostriamo in maniera diretta la doppia implicazione, ma dimostriamo separatamente le seguenti:

1. $x \in L \Rightarrow x \in L(m)$
2. $x \notin L \Rightarrow x \notin L(m)$

1. $x \in L \Rightarrow x \in L(m)$
   Innanzitutto ci troviamo un **caso base** utile per il **passo induttivo**. Qual'è la stringa più piccola $\in L$? È "00"
   Passo base
   $\delta(q_0, 00) = q_2 \in F$ ✓
   Passo induttivo
   Supponiamo che $\forall x$ t.c. $|x| = n \quad x \in L \Rightarrow x \in L(n)$.
   Prendiamo allora una stringa y t.c. $|y| > n$ e $y = xa$ con $a \in \varepsilon$.
   Visto che la funzione di transizione gode della proprietà della composizione, posso considerare $\delta(\delta(q_0, x), a)$. Nel nostro caso però abbiamo due stati finali, q_2 e q_3, quindi dobbiamo guardare entrambi.
   - $\delta(q_0, x) = q_2$
     Se $a = 0$ allora $\delta(q_2, 0) = q_2 \in F$ ✓
     Se $a = 1$ allora $\delta(q_2, 1) = q_3 \in F$ ✓
   - $\delta(q_0, x) = q_3$
     Se $a = 0$ allora $\delta(q_3, 0) = q_2 \in F$ ✓
     Se $a = 1$ allora $\delta(q_3, 1) = q_\perp \notin F$ ✓ (è giusto visto che in questo caso $y \notin L$, dato che ci sarebbero due 1 consecutivi)

2. $x \notin L \Rightarrow x \notin L(m)$
   In questa caso dobbiamo trovare i casi che **NON** finiscono in stati finali.
   Passo base
   $x = 0 \rightarrow \delta(q_0, 0) = q_1 \notin F$ (non ho almeno due 0 consecutivi) ✓
   $x = 11 \rightarrow \delta(q_0, 11) = q_\perp \notin F$ (ho due 1 consecutivi) ✓
   $x = \varepsilon \rightarrow \delta(q_0, \varepsilon) = q_0 \notin F$ ✓
   $x = 1 \rightarrow \delta(q_0, 1) = q_4 \notin F$ ✓
   Passo induttivo
   Supponiamo che $\forall x$ t.c. $|x| = n \quad x \notin L \Rightarrow x \notin L(n)$.
   Prendiamo allora una stringa y t.c. $|y| > n$ e $y = xa$ con $a \in \varepsilon$. Ricordiamoci che possiamo usare $\delta(\delta(q_0, x), a)$.
   In modo analogo alla precedente dimostrazione, dobbiamo considerare gli stati non finali.

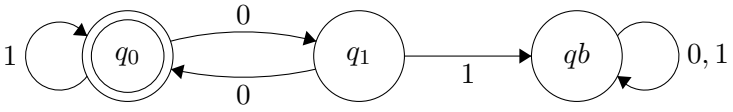
   - $\delta(q_0, x) = q_0$
     Se $a = 0$ allora $\delta(q_0, 0) = q_1 \notin F$ ✓
     Se $a = 1$ allora $\delta(q_0, 1) = q_4 \notin F$ ✓
   - $\delta(q_0, x) = q_1$
     Se $a = 0$ allora $\delta(q_1, 0) = q_2 \in F$ ✓ (è giusto visto che in questo caso $y \in L$, dato che ci sono **almeno** due 0 consecutivi)
     Se $a = 1$ allora $\delta(q_1, 1) = q_4 \notin F$ ✓
   - $\delta(q_0, x) = q_4$
     Se $a = 0$ allora $\delta(q_4, 0) = q_1 \notin F$ ✓
     Se $a = 1$ allora $\delta(q_4, 1) = q_\perp \notin F$ ✓
   - $\delta(q_0, x) = q_\perp$
     Se $a = 0$ allora $\delta(q_\perp, 0) = q_\perp \notin F$ ✓
     Se $a = 1$ allora $\delta(q_\perp, 1) = q_\perp \notin F$ ✓

Abbiamo quindi dimostrato la doppia implicazione e con essa la *correttezza* del nostro automa.

Facciamo un altro esempio, prendiamo come linguaggio

$$L = \{\{0,1\}^* \text{ t.c. gli 0 sono sempre a coppie}\}$$

Ecco l'automa:



Dobbiamo dimostrare che $x \in L \Leftrightarrow x \in L(n)$:

1. $x \in L \Rightarrow x \in L(m)$
2. $x \notin L \Rightarrow x \notin L(m)$

1. $x \in L \Rightarrow x \in L(m)$
   Passo base
   $x = 00 \rightarrow \delta(q_0, 00) = q_0 \in F$ ✓
   Passo induttivo
   Supponiamo che $\forall x$ t.c. $|x| = n \quad x \in L \Rightarrow x \in L(n)$.
   Prendiamo allora una stringa y t.c. $|y| > n$ e $y = xa$ con $a \in \varepsilon$.
   Visto che la funzione di transizione gode della proprietà della composizione, posso considerare $\delta(\delta(q_0, x), a)$.
   $\delta(q_0, x) = q_0$
   Se $a = 0$ allora $\delta(q_0, 0) = q_1 \notin F$ ✓
   Se $a = 1$ allora $\delta(q_0, 1) = q_0 \in F$ ✓

2. $x \notin L \Rightarrow x \notin L(m)$
   Passo base
   $x = 0 \rightarrow \delta(q_0, 0) = q_1 \notin F$ ✓
   $x = 1 \rightarrow \delta(q_0, 1) = q_1 \notin F$ ✓
   Passo induttivo
   Supponiamo che $\forall x$ t.c. $|x| = n \quad x \notin L \Rightarrow x \notin L(n)$.
   Prendiamo allora una stringa y t.c. $|y| > n$ e $y = xa$ con $a \in \varepsilon$. Ricordiamoci che possiamo usare $\delta(\delta(q_0, x), a)$.
   In modo analogo alla precedente dimostrazione, dobbiamo considerare gli stati non finali.

   - $\delta(q_0, x) = q_1$
     Se $a = 0$ allora $\delta(q_1, 0) = q_0 \in F$ ✓
     Se $a = 1$ allora $\delta(q_1, 1) = q_\perp \notin F$ ✓
   - $\delta(q_0, x) = q_\perp$
     Se $a = 0$ allora $\delta(q_\perp, 0) = q_\perp \notin F$ ✓
     Se $a = 1$ allora $\delta(q_\perp, 1) = q_\perp \notin F$ ✓

Abbiamo quindi dimostrato la doppia implicazione e con essa la *correttezza* del nostro automa.

2 Linguaggi liberi dal contesto e grammatiche

Questa categoria è più rognosa di quella dei regolari, ma fortunatamente se dovesse capitarci non dobbiamo costruire un'automa. Dobbiamo però costruire la grammatica, una procedura non sempre immediata purtroppo. Un utile trucco per riconoscere un linguaggio CF è immaginare un'automa con una pila: ad esempio se dobbiamo riconoscere il linguaggio 0^n1^n il nostro automa mano a mano che legge gli 0 li mette in una pila, dopodiché leggendo gli 1 la scarica, e se a fine stringa la pila è vuota allora il linguaggio è riconosciuto.
Nota: l'automa non è costretto a leggere linearmente la stringa, nell'esempio di prima possiamo anche leggere gli 1 a partire dal fondo, facendo valere la stessa regola dello svuotamento della pila.

Prendiamo in esame il seguente linguaggio

$$L = \{x \in \{0,1\}^* \text{ t.c. ci siano tanti 0 quanti 1}\}$$

Intuitivamente notiamo che è necessario "contare" il numero di 0 e far si che sia uguale al numero di 1. Esempi di possibili combinazioni:

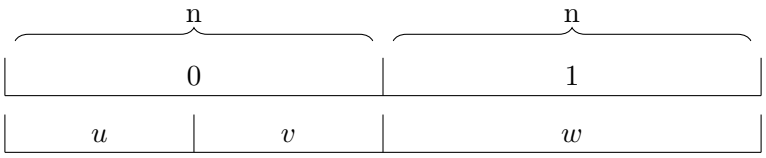
- 01
- 00001111
- 01010101
- 011010

Notiamo che non è necessario che ci sia una determinata sequenza, basta solo che gli 0 totali siano uguali agli 1. Pensandoci un po', viene naturale classificare questo linguaggio nella classe dei *context free*. Per dimostrarlo occorre prima applicare il cosiddetto **pumping lemma**, in modo da mostrare che non può essere regolare. Successivamente dobbiamo scrivere una **grammatica** che genera il linguaggio e **dimostrarne la correttezza**.
Fase 1: Pumping Lemma
In questa fase ci basta prendere una stringa appartenente al linguaggio e dimostrare che "pompandone" una parte noi usciamo dal linguaggio. Ricordiamoci che nell'applicare il pumping lemma dobbiamo sottostare a dei vincoli:

- $z = uvw$

- $|uv| \leq n$
- $|v| > 0$
- $\forall i \in \mathbb{N}. uv^i w \in L$ con $i \geq 0$

Ovviamente non siamo stupidi, quindi scegliamo una stringa facile, in questo caso $z = 0^n 1^n$. L'unica suddivisione possibile è:

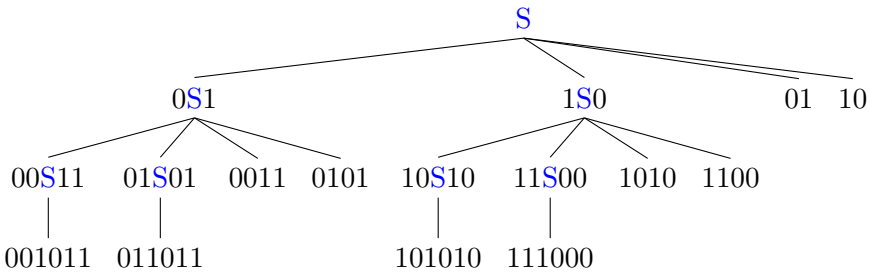


La stringa apparterrà al linguaggio solamente se $0^a 1^b \in L \Leftrightarrow a = b$.
La nostra stringa z sarà quindi composta da $0^{k-|v|}0^{\mathbf{i}|v|}1^k$. Per soddisfare la condizione appena scritta sopra $k - |v| + |v|i = k$, quindi:
$-|v| + |v|i = 0$
$|v|(i-1) = 0$
Ma questo è un assurdo! Infatti $|v|$ non può essere uguale a 0 per i vincoli del pumping lemma. Vediamo che per $i = 0 \wedge i \geq 2$ usciamo dal linguaggio, quindi possiamo dire che è sicuramente non regolare. ✓

Fase 2: Grammatica
Una possibile grammatica per questo linguaggio è:

$$S \to 0S1|01|10$$

Ed ecco una possibile derivazione:



Fase 3: Dimostrazione della grammatica
Per dimostrare la correttezza della nostra grammatica, dobbiamo dimostrare la seguente condizione:

$$x \in L \Leftrightarrow S \Rightarrow_* x$$

Essendo una doppia implicazione, dimostreremo separatamente

1. $x \in L \Rightarrow S \Rightarrow_* x$, che si dimostra per *induzione* sulla lunghezza della stringa
2. $S \Rightarrow_i k \Rightarrow x \in L$, che si dimostra per *induzione* sul numero di passi di derivazione

1. $x \in L \Rightarrow S \Rightarrow_* x$
   Passo base
   $y = 01 \quad y \in L$ ed $\exists S \Rightarrow 01$ ✓ (esiste una derivazione S che porta ad 01).
   $y = 10 \quad y \in L$ ed $\exists S \Rightarrow 10$ ✓ (esiste una derivazione S che porta ad 10).
   Passo induttivo
   Consideriamo $y \in \varepsilon^*$ tale che $|y| < n$ e supponiamo che $\forall y \in L \Rightarrow S \Rightarrow_* y$.
   Allora prendiamo $|x| \geq n$ e sapendo che y è composto come $y = 0^k1^k$, x sarà 0^m1^m con $m > k$. Procediamo prendendo $m = k + 1$:

$$x = 0^{k+1}1^{k+1}$$

   Sappiamo per ipotesi che esiste una derivazione $S \Rightarrow_i 0^k1^k$, quindi

$$\exists S \Rightarrow y \rightarrow \exists S \Rightarrow_i 0^k1^k$$

   ma se esiste questa esiste anche la derivazione precedente, ovvero

$$\exists S \Rightarrow_{i-1} 0^{k-1}S1^{k-1}...$$

   essendo che la produzione $S \rightarrow 0S1$ appartiene all'insieme delle possibili produzioni, posso sostituire:

$$... \Rightarrow_i 0^{k-1}0S11^{k-1} \Rightarrow_{i+1} 0^k011^k = 0^{k+1}1^{k+1} = x \quad ✓$$

   In pratica siamo tornati indietro di un passo e abbiamo applicato una sostituzione che ci conducesse alla stringa x.
   Ripetiamo lo stesso procedimento per $y = 1^k0^k$ e $x = 1^{k+1}0^{k+1}$:

$$\exists S \Rightarrow y \rightarrow \exists S \Rightarrow_i 1^k0^k \Rightarrow_{i-1} 1^{k-1}S0^{k-1}$$

$$\Rightarrow_i 1^{k-1}1S00^{k-1} \Rightarrow_{i+1} 1^k100^k = 1^{k+1}0^{k+1} = x \quad ✓$$

2. $S \Rightarrow_i k \Rightarrow x \in L$
   Passo base
   $S \rightarrow 01 \quad x = 01 \quad x \in L$
   $S \rightarrow 10 \quad x = 10 \quad x \in L$
   Passo induttivo
   $\forall i \leq n \quad S \Rightarrow_i y \Rightarrow y \in L$ (in i passi otteniamo una stringa $\in L$, per ogni $i \leq n$).
   Sappiamo che y sarà nella forma 0^k1^k visto che $\in L$, quindi $S \Rightarrow_i 0^k1^k$. Ma se esiste quella produzione, allora esisterà anche quella precedente. Con lo stesso gioco che abbiamo applicato nella dimostrazione precedente, andiamo indietro per poi andare avanti con una produzione:

$$S \Rightarrow_i 0^k1^k \Rightarrow \exists S \Rightarrow_{i-1} 0^{k-1}S1^{k-1}$$

$$\Rightarrow_i 0^{k-1}0S11^{k-1} \Rightarrow_{i+1} 0^k011^k = 0^{k+1}1^{k+1} \in L \quad ✓$$

Per $y = 1^k0^k$:

$$S \Rightarrow_i 1^k0^k \Rightarrow \exists S \Rightarrow_{i-1} 1^{k-1}S0^{k-1}$$
$$\Rightarrow_i 1^{k-1}1S00^{k-1} \Rightarrow_{i+1} 1^k100^k = 1^{k+1}0^{k+1} \in L \quad \checkmark$$

In pratica mostriamo che andando avanti otteniamo una stringa che appartiene ancora al linguaggio.

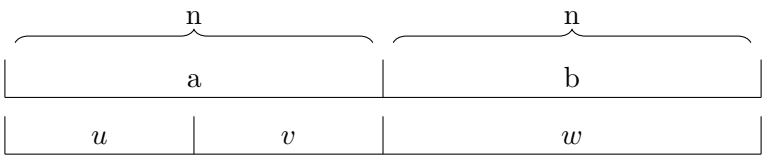
Facciamo un altro esempio, prendiamo come linguaggio

$$L = a^nb^n$$

Sappiamo che è CF perché è necessario "tener conto" di quante a sono state riconosciute per verificare che siano pari al numero di b.
Fase 1: Pumping Lemma
Scegliamo una stringa facile che rispetti i vincoli, come a^nb^n.

- $z = uvw$
- $|uv| \leq n$
- $|v| > 0$
- $\forall i \in \mathbb{N}.uv^iw \in L$ con $i \geq 0$

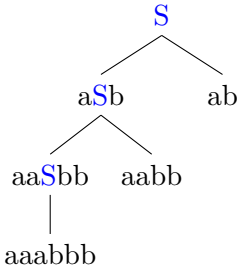
L'unica suddivisione possibile è:



La stringa apparterrà al linguaggio solamente se $a^xb^y \in L \Leftrightarrow x = y$.
La nostra stringa z sarà quindi composta da $a^{k-|v|}a^{\mathbf{i}|v|}b^k$. Per soddisfare la condizione appena scritta sopra $k - |v| + |v|i = k$, quindi:
$-|v| + |v|i = 0$
$|v|(i-1) = 0$
Ma questo è un assurdo! Infatti $|v|$ non può essere uguale a 0 per i vincoli del pumping lemma. Vediamo che per $i = 0 \wedge i \geq 2$ usciamo dal linguaggio, quindi possiamo dire che è sicuramente non regolare. $\checkmark$
Fase 2: Grammatica
Una possibile grammatica per questo linguaggio è:

$$S \to aSb|ab$$

Mentre una possibile derivazione per questa grammatica è:



Fase 3: Dimostrazione della grammatica
Per dimostrare la correttezza della nostra grammatica, dobbiamo dimostrare la seguente condizione:

$$x \in L \Leftrightarrow S \Rightarrow_* x$$

Essendo una doppia implicazione, dimostreremo separatamente

1. $x \in L \Rightarrow S \Rightarrow_* x$, che si dimostra per *induzione* sulla lunghezza della stringa
2. $S \Rightarrow_i k \Rightarrow x \in L$, che si dimostra per *induzione* sul numero di passi di derivazione

1. $x \in L \Rightarrow S \Rightarrow_* x$
   Passo base
   $y = ab \qquad y \in L$ ed $\exists S \Rightarrow ab$ ✓ (esiste una derivazione S che porta ad ab).
   Passo induttivo
   Consideriamo $y \in \varepsilon^*$ tale che $|y| < n$ e supponiamo che $\forall y \in L \Rightarrow S \Rightarrow_* y$.
   Allora prendiamo $|x| \geq n$ e sapendo che y è composto come $y = a^k b^k$, x sarà $x = a^m b^m$ con $m > k$. Procediamo prendendo $m = k + 1$:

   $$x = a^{k+1} b^{k+1}$$

   Sappiamo per ipotesi che esiste una derivazione $S \Rightarrow_i a^k b^k$, quindi

   $$\exists S \Rightarrow y \rightarrow \exists S \Rightarrow_i a^k b^k$$

   ma se esiste questa esiste anche la derivazione precedente, ovvero

   $$\exists S \Rightarrow_{i-1} a^{k-1} S b^{k-1} ...$$

   essendo che la produzione $S \rightarrow aSb$ appartiene all'insieme delle possibili produzioni, posso sostituire:

   $$... \Rightarrow_i a^{k-1} aSbb^{k-1} \Rightarrow_{i+1} a^k abb^k = a^{k+1} b^{k+1} = x \quad ✓$$

   In pratica siamo tornati indietro di un passo e abbiamo applicato una sostituzione che ci conducesse alla stringa x.

2. $S \Rightarrow_i k \Rightarrow x \in L$

   Passo base

   $S \to ab \qquad x = ab \quad x \in L$

   Passo induttivo

   $\forall i \leq n \quad S \Rightarrow_i y \Rightarrow y \in L$ (in i passi otteniamo una stringa $\in L$, per ogni $i \leq n$). Sappiamo che y sarà nella forma $a^k b^k$ visto che $\in L$, quindi $S \Rightarrow_i a^k b^k$. Ma se esiste quella produzione, allora esisterà anche quella precedente. Con lo stesso gioco che abbiamo applicato nella dimostrazione precedente, andiamo indietro per poi andare avanti con una produzione:

$$S \Rightarrow_i a^k b^k \Rightarrow \exists S \Rightarrow_{i-1} a^{k-1} S b^{k-1}$$

$$\Rightarrow_i a^{k-1} a S b b^{k-1} \Rightarrow_{i+1} a^k a b b^k = a^{k+1} b^{k+1} \in L \quad \checkmark$$

3 Linguaggi non liberi dal contesto

Quando un linguaggio richiede più di un singolo conteggio per essere riconosciuto, allora è molto probabile che non sia libero dal contesto. In quel caso, si procede utilizzando il **pumping lemma** per i linguaggi CF, in modo da dimostrare la sua non appartenenza a questa classe. A differenza del pumping lemma per i linguaggi regolari, le suddivisioni possibili sono molte di più, e bisogna dimostrare la fuoriuscita per ognuna di esse.

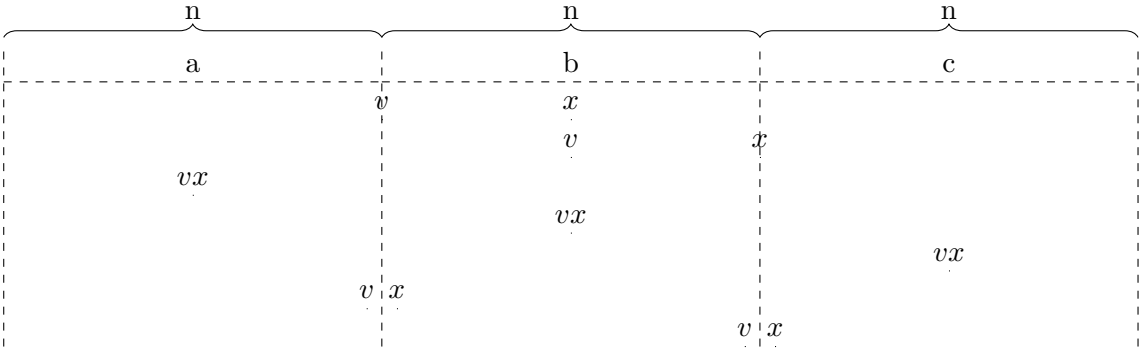
Prendiamo in esempio il linguaggio

$$L = \{a^n b^n c^n\}$$

Esso è chiaramente non-CF, infatti $|a| = |b| = |c|$ e quindi dobbiamo contare due volte, cosa non possibile in un'automa a pila. Posto che $\forall z$ tale che $|z| > n$, dobbiamo soddisfare i seguenti vincoli:

- $\forall z = uvwxy$
- $|vwx| \leq n$
- $|vx| > 0$
- $uv^i wx^i y \in L$

elenchiamo le diverse possibili suddivisioni:



Notiamo che ci sono due casi in cui si ha un'accavallamento, ovvero un caso nel quale o v o x stanno in mezzo alle suddivisioni (ad esempio v è composta da un po' di a e di b). È facile dimostrare come in questi casi il linguaggio "pompato" esca dal linguaggio originale. Facciamo vedere comunque il procedimento, prendendo come esempio il primo caso (d'ora in poi $k = n$). La stringa sarà composta come:

$$a^{k-\frac{|v|}{2}}(ab)^{|v|\mathbf{i}}b^{k-\frac{|v|}{2}-|x|+\mathbf{i}|x|}c^k$$

N.B.: $\frac{|v|}{2}$ è solo indicativo, non è proprio che dobbiamo prendere quel pezzo. Ora abbiamo due casi:

- $|v| = 0$
  $a^k b^{k-|x|+\mathbf{i}|x|} c^k$
  $k + |x|(\mathbf{i} - 1) = k$
  $|x| = 0 \quad \Rightarrow$ impossibile, in quanto si violerebbe il vincolo $|vx| > 0$.

- $|v| > 0$
  $a^{k-\frac{|v|}{2}} (ab)^{|v|\mathbf{i}} b^{k-\frac{|v|}{2}+|x|(\mathbf{i}-1)} c^k$
  con $i = 2 \Rightarrow a^{k-\frac{|v|}{2}} (ab)^{2|v|} b^{k-\frac{|v|}{2}+|x|} c^k$
  $k - \frac{|v|}{2} + 2|v| + k - \frac{|v|}{2} + |x| = 2k$ (la somma della lunghezza di a e b è $2k$)
  $|v| + |x| = 0 \quad \Rightarrow$ impossibile, in quanto si violerebbe il vincolo $|vx| > 0$.

4 Esercizi vari

Classificare le seguenti famiglie di linguaggi al variare di $n, m \in \mathbb{N}$ (N.B. $0 \in \mathbb{N}$), motivando formalmente la risposta:

$$A_{m,n} = \{\sigma \in \{1,0\}^* |\ \sigma = (1^{2n}001^m)^n\}$$
$$B_m = \bigcup_{n \in \mathbb{N}} A_{m,n}$$
$$C_n = \bigcup_{m \in \mathbb{N}} A_{m,n}$$

Notiamo che il linguaggio, al variare di m, n, è finito, ad esempio:

$$A_{0,0} = \varnothing$$
$$A_{1,1} = 11001$$
$$A_{0,1} = 1100$$
$$A_{1,0} = \varnothing$$
$$A_{2,1} = 110011$$
$$A_{2,2} = 1111001111110011$$
$$\dots$$

Essendo i linguaggi finiti sono anche regolari.

Adesso guardiamo l'unione in n del linguaggio $A_{m,n}$:

$$B_m = \bigcup_{n \in \mathbb{N}} A_{m,n}$$

- $B_0 = (1^{2n}00)^n$
- $B_1 = (1^{2n}001)^n$
- $B_2 = (1^{2n}0011)^n$

Notiamo che, al variare di m, otteniamo sempre e comunque un linguaggio non CF. Procediamo quindi con il pumping lemma per i linguaggi non CF.

- $z = uvw$
- $|uv| \leq n$

- $|v| > 0$
- $\forall i \in \mathbb{N}.uv^iw \in L$ con $i \geq 0$

Scegliamo una stringa, $z = (1^{2n}00)^n$, ed elenchiamo le possibili suddivisioni.

Per quanto riguarda l'unione in m del linguaggio $A_{m,n}$:

$$C_n = \bigcup_{m \in \mathbb{N}} A_{m,n}$$

5 Teoria della ricorsione

TODO