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

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 ed Elia Brentarolli per l'immensa pazienza che hanno 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.

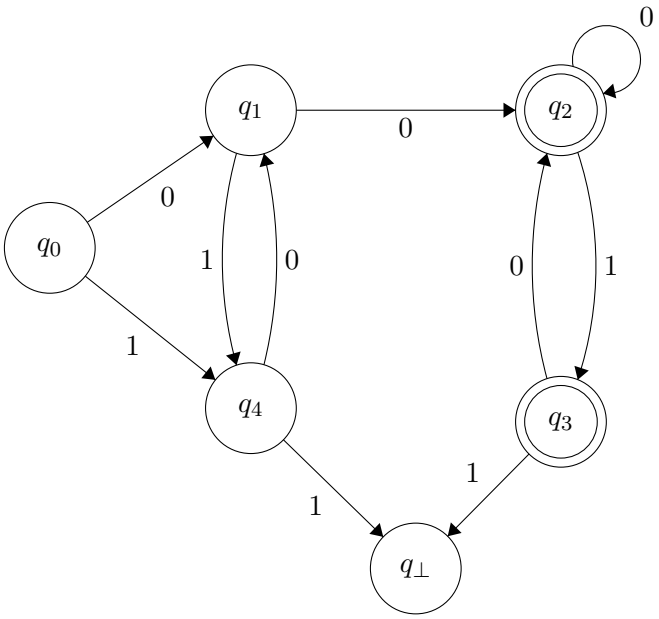
Ricordiamo che un'automa è scrivibile come una quintupla $< Q, \Sigma, \delta, q_0, F >$, dove:

- 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.

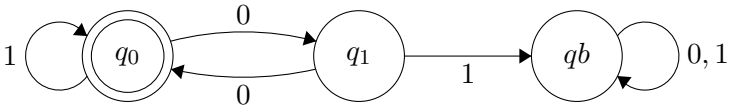
   - $\delta(q_0, x) = q_0$
     Se $a = 0$ allora $\delta(q_0, 0) = q_1 \notin F$ ✓
     Se $a = 1$ allora $\delta(q_0, 1) = q_4 \notin F$ ✓
   - $\delta(q_0, x) = q_1$
     Se $a = 0$ allora $\delta(q_1, 0) = q_2 \in F$ ✓ (è giusto visto che in questo caso $y \in L$, dato che ci sono **almeno** due 0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

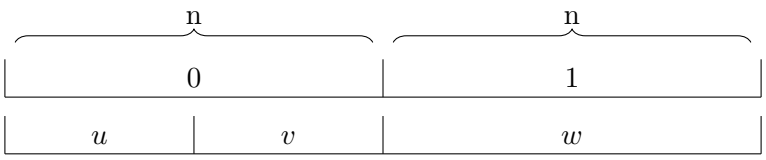
- 01
- 00001111
- 01010101
- 011010

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:

- $z = uvw$

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

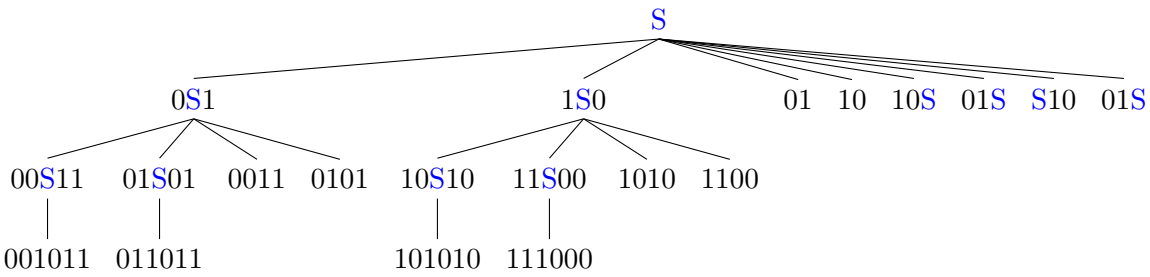


La stringa apparterrà al linguaggio solamente se $0^a 1^b \in L \Leftrightarrow a = b$.
La nostra stringa z sarà quindi composta da $0^{k-|v|}0^{i|v|}1^k$. Per soddisfare la condizione appena scritta sopra $k - |v| + |v|i = k$, quindi:
$-|v| + |v|i = 0$
$|v|(i-1) = 0$
Ma questo è un assurdo! Infatti $|v|$ 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:

$$S \to 1S0|0S1|01|10|10S|01S|S10|01S$$

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:

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

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

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

$$L = a^nb^n$$

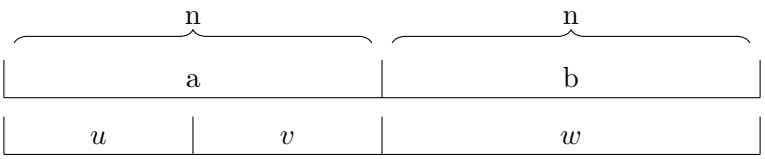
Sappiamo che è CF perché è necessario "tener conto" di quante a sono state riconosciute per verificare che siano pari al numero di b.

Fase 1: Pumping Lemma

Scegliamo una stringa facile che rispetti i vincoli, come a^nb^n.

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

L'unica suddivisione possibile è:



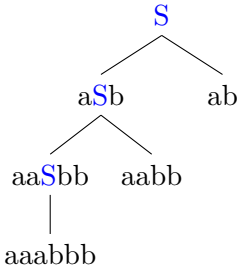
La stringa apparterrà al linguaggio solamente se $a^xb^y \in L \Leftrightarrow x = y$.
La nostra stringa z sarà quindi composta da $a^{k-|v|}a^{i|v|}b^k$. Per soddisfare la condizione appena scritta sopra $k - |v| + |v|i = k$, quindi:
$-|v| + |v|i = 0$
$|v|(i-1) = 0$
Ma questo è un assurdo! Infatti $|v|$ 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. ✓

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, quindi la rottura di palle è decisamente più elevata.

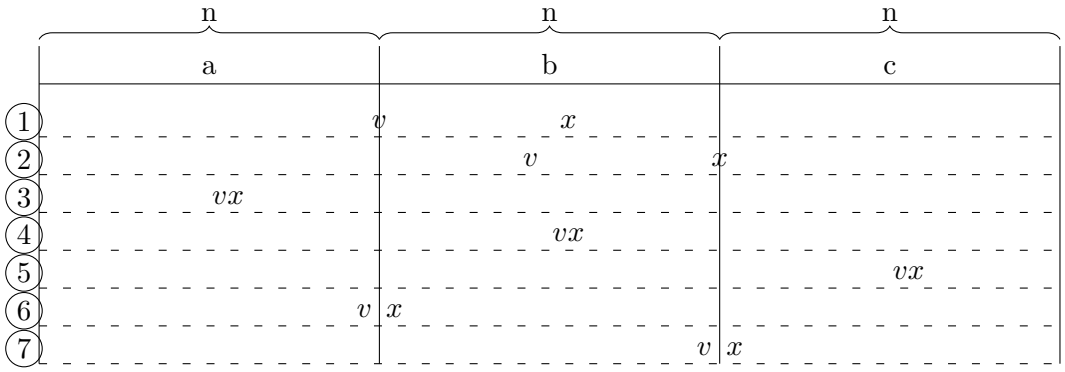
Prendiamo in esempio il linguaggio

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

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:

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

elenchiamo le diverse possibili suddivisioni:



Ora dimostriamo caso per caso che, "pompando", si esce dal linguaggio. Con un po' di pratica si arriva a capire il meccanismo generale.

(1) Notiamo che 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. La stringa sarà composta come:

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

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

- $|v| = 0$
  $a^k b^{k-|x|+i|x|}c^k$
  $k + |x|(i-1) = k$
  con $i = 2 \rightarrow |x| = 0 \quad \Rightarrow$ impossibile, in quanto si violerebbe il vincolo $|vx| > 0$. ✓
- $|v| > 0$
  $a^{k-\frac{|v|}{2}}(ab)^{|v|i}b^{k-\frac{|v|}{2}+|x|(i-1)}c^k$
  con $i = 2 \Rightarrow a^{k-\frac{|v|}{2}}ababb^{k-\frac{|v|}{2}+|x|}c^k \notin L$. ✓

(2) La dimostrazione è analoga al punto precedente.

(3) Ora vx è tutto compreso in a. La stringa sarà quindi composta in questo modo:

$$z = a^{k-|vx|+i|vx|}b^k c^k$$

Affinché sia valida, dobbiamo rispettare il seguente vincolo:

$$\begin{aligned} &k - |vx| + i|vx| = k \\ &|vx|(i-1) = 0 \\ &\text{con } i = 2 \rightarrow |vx| = 0 \quad \text{Impossibile} \end{aligned}$$ ✓

(4) La dimostrazione è analoga al punto precedente.

(5) La dimostrazione è analoga al punto precedente.

(6) v e x si trovano in sottoparti separate. La stringa è così composta:

$$z = a^{k+|v|(i-1)}b^{k+|x|(i-1)}c^k$$

Quindi:

$$\begin{aligned} &k + |v|(i-1) + k + |x|(i-1) = 2k \\ &(i-1)(|v| + |x|) = 0 \\ &\text{con } i = 2 \rightarrow |v| + |x| = 0 \quad \text{Impossibile} \end{aligned}$$ ✓

Abbiamo verificato per ogni caso che il linguaggio "pompato" esce dal linguaggio stesso, dimostrando quindi che non è CF. ✓

4 Esercizi vari sui linguaggi

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.

Unione in n

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

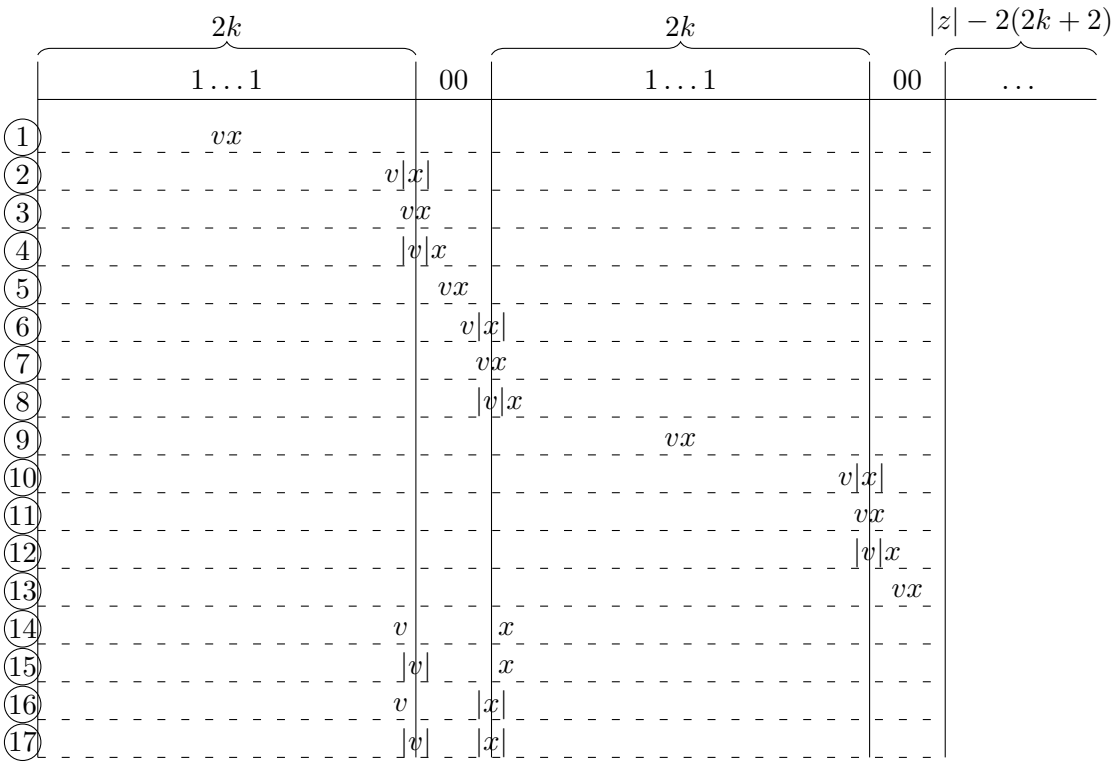
$$A_{m,n} = \{\sigma \in \{1,0\}^* | \ \sigma = (1^{2n}001^m)^n\}$$
$$B_m = \bigcup_{n \in \mathbb{N}} A_{m,n}$$

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

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.

- $\forall z = uvwxy$
- $|z| \geq 2k$
- $|vwx| \leq 2k$
- $|vx| > 0$
- $uv^i wx^i y \in L$

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



Verifichiamo per ogni caso di violare le regole del pumping lemma, in modo da dimostrare che questo linguaggio non è CF.

(1) $z = 1^{2k-|vx|+|vx|i}0^2 1^{2k} 0^2$
Affinché sia valida, deve valere $2k + |vx|(i-1) = 2k$, quindi $|vx|(i-1) = 0$, che però per $i = 2$ viola la condizione $|vx| > 0$. ✓

(2) $z = 1^{2k-|v|+i|v|-\frac{|x|}{2}}(10)^{|x|i}0^{2-\frac{|x|}{2}}1^{2k}0^2$
Distinguiamo due casi:

- $|x| = 0$
  $1^{2k+|v|(i-1)}0^2 1^{2k}0^2 \Rightarrow 2k + |v|(i-1) = 2k$
  per $i \to 2 \Rightarrow |v| = 0$ Impossibile ✓

- $|x| > 0$
  $1^{2k+|v|(i-1)-\frac{|x|}{2}}(10)^{|x|i}0^{2-\frac{|x|}{2}}1^{2k}$
  per $i \to 2 \Rightarrow 1^{2k+|v|-\frac{|x|}{2}}10100^{2-\frac{|x|}{2}}1^{2k} \notin L$ ✓

(3) $z = 1^{2k+|v|(i-1)}0^{2+|x|(i-1)}1^{2k}0^2$
Affinché sia valida, deve valere $2k + |v|(i-1) + 2 + |x|(i-1) = 2k + 2$, quindi $|v|(i-1) + |x|(i-1) = 0$, che per $i \to 2 \Rightarrow |v| + |x| = 0$ viola $|vx| > 0$. ✓

(4) La dimostrazione è analoga al punto 2.

(5) $z = 1^{2k}0^{2+|vx|(i-1)}1^{2k}0^2$
Affinché sia valida, deve valere $2 + |vx|(i-1) = 2$, che implicherebbe $|vx| = 0$ per $i = 2$, quindi non va bene. ✓

(6) La dimostrazione è analoga al punto 2.

(7) La dimostrazione è analoga al punto 3.

(8) La dimostrazione è analoga al punto 2.

(9) La dimostrazione è analoga al punto 1.

(10) La dimostrazione è analoga al punto 2.

(11) La dimostrazione è analoga al punto 3.

(12) La dimostrazione è analoga al punto 2.

(13) La dimostrazione è analoga al punto 5.

(14) $z = 1^{2k+|v|(i-1)}0^2 1^{2k+|x|(i-1)}0^2$
Affinché sia valida, deve valere $2k + |v|(i-1) + 2k + |x|(i-1) = 4k$, quindi per $i \to 2 \Rightarrow |v| + |x| = 0$. Il problema è che andiamo a violare il vincolo $|vx| > 0$, quindi non va bene. ✓

(15) $z = 1^{2k-\frac{|v|}{2}}(10)^{|v|i}0^{2-\frac{|v|}{2}}1^{2k+|x|(i-1)}0^2$
Distinguiamo due casi:

- $|v| = 0$
  $1^{2k}0^2 1^{2k+|x|(i-1)}0^2$
  per $i \to 2 \Rightarrow |v| = 0$ Impossibile ✓
- $|v| > 0$
  $1^{2k-\frac{|v|}{2}}(10)^{|v|i}0^{2-\frac{|v|}{2}}1^{2k+|x|(i-1)}0^2$
  per $i \to 2 \Rightarrow 1^{2k-\frac{|v|}{2}}10100^{2-\frac{|v|}{2}}1^{2k+|x|}0^2 \notin L$ ✓

(16) La dimostrazione è analoga al punto 15.

⑰ $z = 1^{2k-\frac{|v|}{2}}(10)^{|v|i}0^{2-\frac{|v|}{2}-\frac{|x|}{2}}(01)^{|x|i}1^{2k-\frac{|x|}{2}}0^2$
Distinguiamo i seguenti casi:

- $|v| = 0$
  $1^{2k}0^{2-\frac{|x|}{2}}(01)^{|x|i}1^{2k-\frac{|x|}{2}}0^2$
  - $|x| = 0$
    Non possiamo, in quanto violeremmo $|vx| > 0$. ✓
  - $|x| > 0$
    $1^{2k}0^{2-\frac{|x|}{2}}(01)^{|x|i}1^{2k-\frac{|x|}{2}}0^2$
    per $i \to 2 \Rightarrow 1^{2k}0^{2-\frac{|x|}{2}}01011^{2k-\frac{|x|}{2}}0^2 \notin L$. ✓
- $|v| > 0$
  $1^{2k-\frac{|v|}{2}}(10)^{|v|i}0^{2-\frac{|v|}{2}-\frac{|x|}{2}}(01)^{|x|i}1^{2k-\frac{|x|}{2}}0^2$
  - $|x| = 0$
    $1^{2k-\frac{|v|}{2}}(10)^{|v|i}0^{2-\frac{|v|}{2}}1^{2k}0^2$
    per $i \to 2 \Rightarrow 1^{2k-\frac{|v|}{2}}10100^{2-\frac{|v|}{2}}1^{2k}0^2 \notin L$. ✓
  - $|x| > 0$
    $1^{2k-\frac{|v|}{2}}(10)^{|v|i}0^{2-\frac{|v|}{2}-\frac{|x|}{2}}(01)^{|x|i}1^{2k-\frac{|x|}{2}}0^2$
    per $i \to 2 \Rightarrow 1^{2k-\frac{|v|}{2}}10100^{2-\frac{|v|}{2}-\frac{|x|}{2}}01011^{2k-\frac{|x|}{2}}0^2 \notin L$. ✓

Abbiamo dimostrato ogni caso, quindi il nostro linguaggio non è CF.

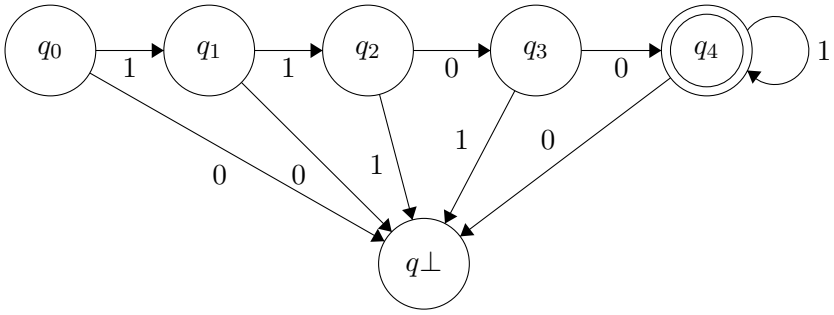
Unione in m

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

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

- $C_0 = \varnothing$
- $C_1 = 11001^m$
- $C_2 = (1111001^m)^2$
- $C_{n>2} = (1^{2n}001^m)^n$

Notiamo che C_0 e C_1 sono linguaggi regolari, il primo è banale, il secondo ha questo automa:



1. $x \in L \Rightarrow x \in L(m)$
   Passo base
   $\delta(q_0, 1100) = q_4 \in F$ ✓
   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 $a \in \varepsilon$.
   Posto che $\delta(q_0, x) = q_4$:
   - se $a = 0$ allora $\delta(q_4, 0) = q_\perp \notin F$ ✓
   - se $a = 1$ allora $\delta(q_4, 1) = q_3 \in F$ ✓

2. $x \notin L \Rightarrow x \notin L(m)$
   Passo base
   $x = 0 \rightarrow \delta(q_0, 0) = q_\perp \notin F$ ✓
   $x = 1 \rightarrow \delta(q_0, 1) = q_1 \notin F$ ✓
   $x = 11 \rightarrow \delta(q_0, 11) = q_2 \notin F$ ✓
   $x = 10 \rightarrow \delta(q_0, 10) = q_\perp \notin F$ ✓
   $x = 111 \rightarrow \delta(q_0, 111) = q_\perp \notin F$ ✓
   $x = 110 \rightarrow \delta(q_0, 110) = q_3 \notin F$ ✓
   $x = 1101 \rightarrow \delta(q_0, 1101) = q_\perp \notin F$ ✓
   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 $\delta(\delta(q_0, x), a)$.
   - $\delta(q_0, x) = q_0$
     Se $a = 0$ allora $\delta(q_0, 0) = q_\perp \notin F$ ✓
     Se $a = 1$ allora $\delta(q_0, 1) = q_1 \notin F$ ✓
   - $\delta(q_0, x) = q_1$
     Se $a = 0$ allora $\delta(q_1, 0) = q_\perp \notin F$ ✓
     Se $a = 1$ allora $\delta(q_1, 1) = q_2 \notin F$ ✓
   - $\delta(q_0, x) = q_2$
     Se $a = 0$ allora $\delta(q_2, 0) = q_3 \notin F$ ✓
     Se $a = 1$ allora $\delta(q_2, 1) = q_\perp \notin F$ ✓

- $\delta(q_0, x) = q_3$
  Se $a = 0$ allora $\delta(q_3, 0) = q_4 \in F$ ✓
  Se $a = 1$ allora $\delta(q_3, 1) = q_\perp \notin F$ ✓

Se consideriamo invece $C_2 = (1111001^m)^2$, notiamo che potrebbe facilmente trattarsi di un linguaggio CF. Procediamo quindi con il pumping lemma.

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

La stringa sarà formata così: $1^4 0^2 1^{k+|v|(i-1)} 1^4 0^2 1^k$. Affinché appartenga al linguaggio, è necessario che $k + |v|(i-1) = k$, che per $i \to 2 \Rightarrow |v| = 0$, che è impossibile. ✓
Una grammatica che genera questo linguaggio è fatta in questo modo:

$$S \to 111100B$$
$$B \to 1B1|111100$$

Mentre una possibile derivazione per questa grammatica è:

$$\begin{array}{c} S \\ | \\ 1^4 0^2 B \\ | \\ 1^4 0^2 1B1 \\ | \\ 1^4 0^2 11B11 \\ | \\ 1^4 0^2 111B111 \\ | \\ 1^4 0^2 1^3 1^4 0^2 1^3 \end{array}$$

Dimostriamo la correttezza 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
   $y = 1^4 0^2 1^4 0^2 \qquad x \in L$ ed $\exists S \Rightarrow 1^4 0^2 1^4 0^2$ ✓
   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 $y = 1^4 0^2 1^k 1^4 0^2 1^k$, x sarà $1^4 0^2 1^m 1^4 0^2 1^m$ con $m > k$. Procediamo prendendo $m = k + 1$:

   $$x = 1^4 0^2 1^{k+1} 1^4 0^2 1^{k+1}$$

   Sappiamo per ipotesi che esiste una derivazione $S \Rightarrow_i 1^4 0^2 1^k 1^4 0^2 1^k$, quindi

   $$\exists S \Rightarrow y \rightarrow \exists S \Rightarrow_i 1^4 0^2 1^k 1^4 0^2 1^k$$

   ma se esiste questa esiste anche la derivazione precedente, ovvero

   $$\exists S \Rightarrow_{i-1} 1^4 0^2 1^k B 1^k \Rightarrow ...$$

   essendo che la produzione $S \rightarrow 1B1$ appartiene all'insieme delle possibili produzioni, posso sostituire:

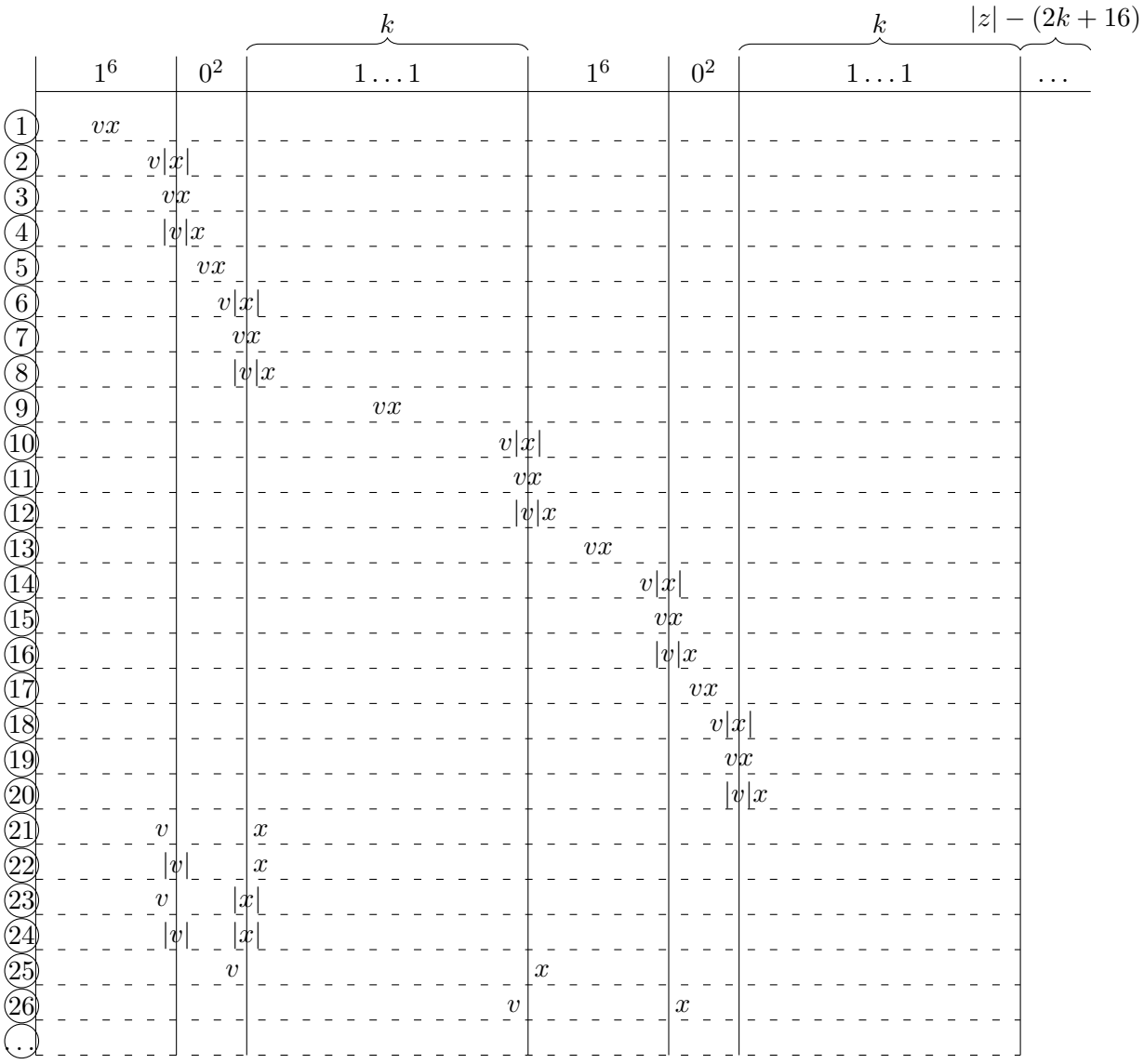
   $$... \Rightarrow_i 1^4 0^2 1^{k+1} B 1^{k+1} \Rightarrow_{i+1} 1^4 0^2 1^{k+1} 1^4 0^2 1^{k+1} = x \quad ✓$$

2. $S \Rightarrow_i k \Rightarrow x \in L$
   Passo base
   $S \rightarrow 1^4 0^2 1^4 0^2 \qquad x = 1^4 0^2 1^4 0^2 \quad x \in L$ ✓
   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 $1^4 0^2 1^k 1^4 0^2 1^k$ visto che $\in L$, quindi $S \Rightarrow_i 1^4 0^2 1^k 1^4 0^2 1^k$. Ma se esiste quella produzione, allora esisterà anche quella precedente:

   $$S \Rightarrow_i 1^4 0^2 1^k 1^4 0^2 1^k \Rightarrow \exists S \Rightarrow_{i-1} 1^4 0^2 1^k B 1^k$$
   $$\Rightarrow_i 1^4 0^2 1^k 1 B 1 1^k \Rightarrow_{i+1} 1^4 0^2 1^{k+1} 1^4 0^2 1^{k+1} \in L \quad ✓$$

Se consideriamo $C_n = (1^{2n} 0 0 1^m)^n$ con $n > 2$, il linguaggio diventa non-CF, per cui dobbiamo dimostrarlo usando il pumping lemma. Facciamo la dimostrazione per

$$n = 3 \rightarrow C_3 = (1^6 0^2 1^m)^3$$



Verifichiamo per ogni caso di violare le regole del pumping lemma, in modo da dimostrare che questo linguaggio non è CF.

(1) $z = 1^{6+|vx|(i-1)}0^2 1^k 1^6 0^2 1^k 1^6 0^2 1^k$
Affinché sia valida, deve valere $6 + |vx|(i-1) = 6$, quindi $|vx|(i-1) = 0$, che però per $i \to 2$ viola la condizione $|vx| > 0$. ✓

(2) $z = 1^{6+|v|(i-1)-\frac{|x|}{2}}(10)^{|x|i}0^{2-\frac{|x|}{2}} 1^k 1^6 0^2 1^k 1^6 0^2 1^k$
Distinguiamo due casi:

- $|x| = 0$
  $1^{6+|v|(i-1)}0^2 1^k 1^6 0^2 1^k 1^6 0^2 1^k \Rightarrow 6 + |v|(i-1) = 6$
  per $i \to 2 \Rightarrow |v| = 0$ Impossibile ✓
- $|x| > 0$
  $1^{6+|v|(i-1)-\frac{|x|}{2}}(10)^{|x|i}0^{2-\frac{|x|}{2}}1^k 1^6 0^2 1^k 1^6 0^2 1^k$
  per $i \to 2 \Rightarrow 1^{6+|v|-\frac{|x|}{2}}10100^{2-\frac{|x|}{2}}1^k 1^6 0^2 1^k 1^6 0^2 1^k \notin L$ ✓

(3) $z = 1^{6+|v|(i-1)}0^{2+|x|(i-1)}1^k 1^6 0^2 1^k 1^6 0^2 1^k$
Affinché sia valida, deve valere $6 + |v|(i-1) + 2 + |x|(i-1) = 8$, quindi $|v|(i-1) + |x|(i-1) = 0$, che per $i \to 2 \Rightarrow |v| + |x| = 0$, che non è possibile. ✓

(4) La dimostrazione è analoga al punto 2.

(5) $z = 1^6 0^{2+|vx|(i-1)}1^k 1^6 0^2 1^k 1^6 0^2 1^k$
Affinché sia valida, deve valere $2 + |vx|(i-1) = 2$, che per $i \to 2 \Rightarrow |vx| = 0$, quindi no. ✓

(6) La dimostrazione è analoga al punto 2.

(7) La dimostrazione è analoga al punto 3.

(8) La dimostrazione è analoga al punto 2.

(9) $z = 1^6 0^2 1^{k-|vx|(i-1)}1^6 0^2 1^k 1^6 0^2 1^k$
Affinché sia valida, deve valere $k - |vx|(i-1) = k$, che per $i \to 2 \Rightarrow |vx| = 0$, quindi no. ✓

(10) La dimostrazione è analoga al punto 2.

(11) La dimostrazione è analoga al punto 3.

(12) La dimostrazione è analoga al punto 2.

(13) La dimostrazione è analoga al punto 1.

(14) La dimostrazione è analoga al punto 2.

(15) La dimostrazione è analoga al punto 3.

(16) La dimostrazione è analoga al punto 2.

(17) La dimostrazione è analoga al punto 5.

(18) La dimostrazione è analoga al punto 2.

(19) La dimostrazione è analoga al punto 3.

(20) La dimostrazione è analoga al punto 2.

(21) $z = 1^{6+|v|(i-1)}0^2 1^{k+|x|(i-1)}1^6 0^2 1^k 1^6 0^2 1^k$
Affinché sia valida, deve valere $6 + |v|(i-1) + k + |x|(i-1) = 6 + k \Rightarrow |v|(i-1) + |x|(i-1) = 0$. Considerando $i \to 2 \Rightarrow |v| + |x| = 0$, che non è possibile. ✓

(22) La dimostrazione è simile al punto 2.

(23) La dimostrazione è simile al punto 2.

(24) La dimostrazione è simile al punto 2.

(25) $z = 1^6 0^{2+|v|(i-1)}1^k 1^{6+|x|(i-1)}0^2 1^k 1^6 0^2 1^k$
Questa suddivisione non si può fare, perché viola $|vwx| < k$. ✓

(26) $<= 1^6 0^2 1^{k+|v|(i-1)}1^6 0^{2+|x|(i-1)}1^k 1^6 0^2 1^k$
Affinché sia valida, deve valere $k + |v|(i-1) + 2 + |x|(i-1) = k + 2 \Rightarrow |v|(i-1) + |x|(i-1) = 0$. Considerando $i \to 2 \Rightarrow |v| + |x| = 0$, che non è possibile. ✓

Riporto un altro interessante esempio. Consideriamo il linguaggio:

$$L = \{0^n | n \text{ è primo}\}$$

Se n fosse fissato, sarebbe un banalissimo linguaggio regolare, perché sarebbe finito. In questo caso però non solo n non è fissato, ma avendo quella particolare condizione di essere primo, fa uscire il nostro linguaggio anche dai CF, relegandolo ai non-CF. Non esiste infatti automa in grado di stabilire se un numero è primo. Procediamo quindi con la dimostrazione tramite il pumping lemma CF. Non ci sono molte suddivisioni possibili, abbiamo solamente che vx è sicuramente dentro 0^n, per cui la nostra stringa sarà formata così:

$$0^{n+|vx|(i-1)}$$

Notiamo che il solito trucco di ridurre $|vx|$ a 0 non funziona, per cui dobbiamo inventarci qualcosa. La soluzione è considerare la somma degli esponenti come un numero n' t.c. è primo, prendere una certa i ad-hoc per poi arrivare ad un assurdo, nella maniera seguente:

$$\begin{aligned} &n + (i-1)|vx| = n', \qquad \text{prendo } i = n+1 \\ &n + (n+1-1)|vx| = n' \\ &n + n|vx| = n' \\ &n(|vx| + 1) = n' \end{aligned}$$

Ma $n(|vx| + 1)$ è divisibile sia per n che per $(|vx| + 1)$, il che contraddice il fatto che n' sia primo, quindi abbiamo trovato un assurdo, che ci costringe a riconoscere che il linguaggio è non-CF.

5 Teoria della ricorsione

Attenzione! Avete presente il dolore di quando sbattete il mignolo del piede contro un angolo del muro? Ecco, questa parte vi farà capire che esistono dolori peggiori. Se come me siete piuttosto pratici, farete una fatica immensa per capire questa roba astrusa (io non l'ho capita nemmeno ora...). Fatta questa doverosa avvertenza, iniziamo subito facendo un ripasso della teoria dietro a questa roba.

Notazione Facciamo una breve panoramica della notazione che andremo ad utilizzare.

$$\varphi_x(y) = z$$

- x Indice del programma, ma viene usato anche per indicare un programma x
- φ_x Programma di indice x
- y Input y
- z Output z
- W_x Insieme degli input per i quali il programma termina (dominio del programma)
- $range(x)$ Insieme degli output del programma (codominio del programma)

Funzioni parziali ricorsive La prima cosa da capire è la definizione di **funzione ricorsiva**. Le funzioni ricorsive sono una classe di funzioni dai numeri naturali ai numeri naturali ($\mathbb{N} \to \mathbb{N}$) che sono "calcolabili" in un qualche senso intuitivo. Possiamo anche dire che le funzioni ricorsive possono essere calcolate da una macchina di Turing. Diciamo che una funzione ricorsiva è **parziale** quando non è definita per tutti gli input.

Insiemi ricorsivi e ricorsivamente enumerabili Dopo aver visto cos'è una funzione parziale ricorsiva, vediamo cos'è un insieme ricorsivo. Dato x un programma e φ_x la funzione parziale ricorsiva calcolata dal programma, definiamo il **dominio** di φ_x come:

$$W_x = \{y | \varphi_x(y) \downarrow\} \quad W_x \subseteq \mathbb{N}$$

Che vuol dire? Significa che il dominio della funziona *calcolata* dal programma è l'insieme degli input per i quali termina (definizione da Capitan Ovvio). Ok, ora invece diciamo che un insieme A è detto **ricorsivamente enumerabile** (r.e.) *se esiste* una funzione parziale ricorsiva ψ tale che

$$W_\psi = A \quad A \subseteq \mathbb{N}$$

In pratica se un certo insieme è l'insieme di input per i quali un programma (che calcola la funzione parziale) termina, allora si dice ricorsivamente enumerabile. Definiamo inoltre **funzione semicaratteristica** di un insieme A r.e. la seguente funzione parziale ricorsiva:

$$\psi_A(x) = \begin{cases} 1 \text{ se } x \in A \\ \uparrow \text{ se } x \notin A \end{cases}$$

Questa funzione ci dice se un certo input $x \in \mathbb{N}$ appartiene all'insieme A, quindi se appartiene all'insieme dei possibili input che fanno terminare il programma. Vediamo che in questo caso è possibile dire di si, ma non di no. Quando è possibile dire anche di no, si ha a che fare con una **funzione caratteristica** di un **insieme ricorsivo**:

$$\psi_A(x) = \begin{cases} 1 \text{ se } x \in A \\ 0 \text{ se } x \notin A \end{cases}$$

Notiamo che la ricorsività è più potente della ricorsiva enumerabilità, infatti se un insieme è ricorsivo allora è anche ricorsivamente enumerabile. Esempi di insiemi noti che sono ricorsivi sono $\mathbb{N}$, $2\mathbb{N}$, $2\mathbb{N}+1$, tutti gli insiemi finiti, infatti per ciascuno di essi io riesco a costruire una funzione caratteristica che mi sa dire di si se un numero gli appartiene o no in caso contrario.

Teorema di Post Il teorema di Post dice che, dato un insieme A ricorsivo, A e $\bar{A}$ sono entrambi ricorsivamente enumerabili.

Teorema di Kleene Questo teorema afferma che le seguenti affermazioni sono equivalenti:

- A è r.e.
- $A = range(\varphi)$ con φ parziale ricorsiva
- $A = \varnothing$ oppure $A = range(f)$ con f totale ricorsiva

Altri teoremi

- A è ricorsivo $\Rightarrow A = \varnothing$ oppure $A = range(f)$ con f non decrescente.
- Ogni insieme r.e. infinito ha un sottoinsieme ricorsivo infinito.
- Se t è una funzione totale ricorsiva allora $\exists e$ t.c. $\varphi_e = \varphi_{t(e)}$.

Proprietà estensionale Consideriamo Π come proprietà sulle MdT, $\Pi \subseteq \mathbb{N}$. Si dice che Π è **estensionale** se

$$\forall (x, y) \in \mathbb{N} : x \in \Pi \wedge \varphi_x = \varphi_y \Rightarrow y \in \Pi$$

Ok sicuramente starete pensando "ma che c'è scritto?". Si capisce meglio con un esempio! Supponiamo che Π sia la proprietà "$\{p|p = 174$ righe$\}$", ovvero che un certo programma p ha esattamente 174 righe di codice. Una proprietà è estensionale quando, preso un altro programma che calcola la stessa funzione, la proprietà si mantiene anche per quel programma. Visto che abbiamo detto che la proprietà è di avere 174 righe, è chiaro che non è estensionale, infatti posso avere un programma più lungo per esempio.

Teorema di Rice Sia Π una proprietà estensionale. Essa è ricorsiva sse $\Pi = \varnothing$ oppure $\Pi = \mathbb{N}$. Detto in soldoni per ogni proprietà **non banale** delle funzioni ricorsive è **indecidibile** il problema di decidere (scusate il gioco di parole) quali funzioni soddisfino tale proprietà e quali no. Per proprietà banale in questo caso si intende una proprietà che non effettua alcuna discriminazione tra le funzioni calcolabili, cioè che vale o per tutte o per nessuna (tipo appunto l'essere vuoto o l'essere tutto $\mathbb{N}$).

L'insieme K e il suo complementare Esistono insiemi che sono ricorsivamente enumerabili ma non sono ricorsivi? Si, e l'insieme che stiamo per scoprire ci accompagnerà sempre nei nostri incubi: sto parlando del famigerato insieme K. Questo insieme è definito come "l'insieme dei programmi che terminano con input sé stessi":

$$K = \{x|\varphi_x(x)\downarrow\} = \{x|x \in W_x\}$$

Non riusciamo infatti a dire che un programma "non termina", per farlo dovremmo essere immortali e aspettare un tempo infinito. Oltre a lui c'è anche suo fratello, e fra i due non saprei dire chi è il peggiore. In ogni caso il suo complementare è $\bar{K}$, che ovviamente è l'insieme di tutti i programmi che **non** terminano con input se stessi. Se vi state chiedendo: "Ma cosa vuol dire dare in input sé stesso", beh sappiate che non l'ho capito nemmeno ora. Forse vuol dire fare l'hash del codice e darlo in pasto a se stesso (e se il programma non prende input?). Boh, robe da informatica teorica...

Riduzione funzionale Questa è una cosa che useremo negli esercizi. Si dice che un insieme si *riduce funzionalmente* ad un altro se:

$$A \preceq_f B \Leftrightarrow \exists f \text{ totale ricorsiva t.c. } x \in A \Rightarrow f(x) \in B$$

Ad esempio, consideriamo l'insieme $K_1 = \{< x, y > |y \in W_x\}$ (l'insieme dei programmi che terminano con input x termina anche con y) e dimostriamo che si riduce funzionalmente a K.

- $x \in K \Rightarrow \varphi_x(x)\downarrow; < x, x > \in W_x$
- $x \notin K \Rightarrow \varphi_x(x)\uparrow; < x, x > \notin K_1$

Un'importante osservazione è che se $A \preceq_f B \iff \bar{A} \preceq_f \bar{B}$.

Teorema fondamentale della riducibilità Il teorema afferma che:

- Se $A \preceq_f B$ e B è r.e. $\Rightarrow A$ è r.e.
- Se $A \preceq_f B$ e B è ricorsivo $\Rightarrow A$ è ricorsivo

Anche questo teorema viene usato negli esercizi.

Facciamo un esempio: consideriamo l'insieme $A = \{x \mid |W_x| < \omega\}$, ovvero tutti i programmi con dominio finito. La condizione di appartenenza ad A è dunque

$$\begin{aligned} x \in A \quad &\iff \quad g(x) \in \{x \mid |W_x| < \omega\} \\ \varphi_x(x)\downarrow \quad &\iff \quad |W_{g(x)}| < \omega \end{aligned}$$

Definiamo la funzione semicaratteristica ψ:

$$\psi_A(x,z) = \begin{cases} 1 \text{ se } \varphi_x(x) \textbf{ non} \text{ converge in meno di } z \text{ passi} \\ \uparrow \text{ altrimenti} \end{cases}$$

- Se $x \in A \Leftrightarrow \exists n.\varphi_x(x)\downarrow^n$ (esiste un n tale per cui $\varphi_x(x)$ termina in n passi). Quindi $\forall z \in [0 \dots n-1]\ \varphi_x(x)\uparrow$.
- Se $x \notin A \Leftrightarrow \varphi_x(x)$ risponde sempre 1 poiché $\forall z \in [n, +\infty)\ \varphi_{g(x)}(z)\uparrow$, che implica $|W_{g(x)}| = \omega$.

Insiemi creativi e produttivi

- Si definisce **insieme produttivo** un insieme A se esiste una funzione totale ricorsiva f t.c.:
$$\forall x \in \mathbb{N}.(W_x \subseteq A \Rightarrow f(x) \in A \setminus W_x)$$
- Si definisce **insieme creativo** un insieme A se è r.e. ed il suo complemento è produttivo:
$$A \in RE \text{ e } \forall x \in \mathbb{N}.(W_x \subseteq \bar{A} \Rightarrow f(x) \in \bar{A} \setminus W_x)$$

Un insieme quindi è produttivo se per ogni tentativo di enumerarlo in modo effettivo mediante un algoritmo di indice x (ovvero per ogni $W_x \subseteq A$), esiste una trasformazione effettiva di di x in un punto di A che sfugge all'enumerazione. Cosa vuol dire nella pratica? Non ne ho idea. Essendo un "praticone" preferisco pensarla in questo modo: dato un insieme A, riesco a pensare ad un algoritmo che converge se $x \in A$? Ad esempio, consideriamo

$$A = \{x | \varphi_x(1) = n\} \quad \text{con } \in \mathbb{N},$$

che è l'insieme dei programmi che, ricevendo in input 1, danno in output il numero n. Riesco a pensare ad un algoritmo che verifica questo? Si, quindi è creativo.
Importante conseguenza di ciò è che:

- A creativo $\Rightarrow \bar{A}$ produttivo
- A produttivo $\Rightarrow A$ non r.e.
- A creativo $\Rightarrow A$ non ricorsivo

Alcune scorciatoie utili Riportiamo di seguito una serie di relazioni utili che si possono utilizzare negli esercizi.

$$f_w = \{f(x) | \varphi_x(f(x)) \downarrow\}, f(x) \text{ totale ricorsiva}$$

Può essere riscritta come

$$f_w = \{f(x) | f(x) \in W_x\} \to RE$$

Mostriamo ora una famiglia di insiemi, le cui composizioni sono ricorsivamente enumerabili:

$$f_w = \{f(x) | \varphi_{g(x)}(h(x)) = C\} \to RE$$

dove $f(x), g(x), h(x)$ sono funzioni e C è una costante (oppure una variabile $\in \mathbb{N}$).

Un'altra famiglia di insiemi è:

$$z = \{x | W_x = A\} \quad \text{e} \quad z = \{x | W_x \neq A\}, \text{ dove } A \text{ è un insieme}$$

Per $z = \{x | W_x = A\}$ valgono le seguenti affermazioni:

- Se A è produttivo $\to z = \varnothing$, quindi ricorsivo
- Se A è r.e. $\to z$ è produttivo

Mentre per $z = \{x | W_x \neq A\}$:

- Se $A = \varnothing \to RE$, mediante la tecnica del *dove tail*
- Se $A = RE \to Z$ è produttivo
- Se A è produttivo $\to Z$ è ricorsivo

6 Esercizi vari sulla teoria della ricorsione

Classificare nella teoria della ricorsione matematica il seguente insieme:

$$A = \{x|\varphi_x(ack(x,x,x)) = 7\}, \text{ dove } ack() \text{ è la funzione di Ackermann.}$$

La prima cosa da fare è assicurarsi di capire bene il testo dell'esercizio. Qui stiamo considerando l'insieme dei programmi che, ricevendo in input l'*output* della funzione di Ackermann, danno in output 7. Per aiutarsi a identificare la tipologia dell'insieme, è buona pratica considerare anche l'insieme complementare. In questo caso, il complementare di A è

$$\bar{A} = \{x|\varphi_x(ack(x,x,x)) \neq 7\}$$

Che è quindi l'insieme dei programmi che ricevendo in input l'output della funzione di Ackermann **non** danno 7. Riesco a pensare ad un algoritmo che prende il nostro programma x, gli passa in input la funzione di Ackermann (che termina sempre) e verifica che il risultato sia 7? Certamente! Quindi è creativo! Di conseguenza il suo complementare è produttivo, per cui possiamo ridurre A a K (e di conseguenza $\bar{A}$ a $\bar{K}$). Definiamo la funzione semicaratteristica ψ_A:

$$\psi_A(x,y) = \psi_{g(x)}(y) = \begin{cases} 7 \text{ se } x \in K \\ \uparrow \text{ altrimenti} \end{cases}$$

- $x \in K . \varphi_{g(x)}(y) = 7 \Rightarrow \varphi_{g(x)}(ack(g(x), g(x), g(x))) = 7 \Rightarrow g(x) \in A$
- $x \notin K . \varphi_{g(x)}(y) \uparrow \Rightarrow \varphi_{g(x)}(ack(g(x), g(x), g(x))) \uparrow \Rightarrow g(x) \notin A$