

Denti's Frequently Asked Questions

Linguaggi e Modelli Computazionali LS

a cura di Silvia Cereda
silvia.cereda@studio.unibo.it

1. Sia $S \rightarrow aA$ e $A \rightarrow bS$. Di quanti vocaboli è formato il linguaggio?

Il linguaggio è vuoto perché non è presente una produzione che contenga solo simboli terminali e che pertanto permetta di terminare una frase. A differenza di ciò che si potrebbe pensare, cioè che la frase creata avrà lunghezza infinita perché vale $S \rightarrow aA \rightarrow abS \rightarrow ababS \rightarrow abababS \rightarrow \dots$ e che quindi il linguaggio sia infinito, esso è vuoto perché non permette di creare neanche una frase.

Qui è possibile cominciare a parlare anche dei teoremi che definiscono quando un linguaggio è vuoto e quando è infinito, ma normalmente non lo chiede. Occhio però che se sono io la prima a formulare il teorema, allora il prof mi chiederà il perché vale, e poi finisce che se gli ho enumerato un teorema, lui mi chiede anche l'altro...

2. Data la grammatica dire quali parole può generare

$$S \rightarrow aS|bB$$

$$B \rightarrow bS|aB$$

Il linguaggio è vuoto in quanto non ci sono simboli terminali che fanno terminare la sequenza di sostituzioni.

3. Classificare il seguente linguaggio:

- $L = \{11(0|1)^{m_1}, m_1 \geq 0\}$

Questo linguaggio è infinito in quanto non vi è un limite superiore alle ripetizioni di n , ed è di tipo 3, non c'è self-embedding.

- $L2 = \{(11)^n(0|1)^m(1)^n, n \geq 1, m \geq 0\}$

Qui la m è fuorviante e non interessante. Sono invece da notare gli 1^n all'inizio e alla fine di ogni parola del linguaggio: c'è self-embedding e si tratta quindi di un linguaggio di tipo 2. Anche questo linguaggio è infinito.

- $L = \{a^{2n}b^n, n \geq 1, n \leq N\}$
Si potrebbe pensare che questa grammatica sia di tipo 2, ma il self-embedding è falso dato che i cicli possibili sono limitati dalla regola $n \leq N$, pertanto tale linguaggio, finito, può essere rappresentato da un automa e quindi da una grammatica di tipo 3.
- $L = \{a^{2m}b^n, n \geq 1, m \geq 1\}$
Questo linguaggio è infinito perchè m e n non sono limitate superiormente, ed è di tipo 3 in quanto m e n non sono in alcun modo correlate tra di loro.
- $L = \{a^n b^m, n \geq 1, m \geq 1\}$
Il linguaggio è di tipo 3 perché non c'è self-embedding e basta un ASF per riconoscerlo. n e m non sono legate in nessun modo.
- $L = \{a^n b^n c^n, n \geq 1\}$
Questo linguaggio è di tipo 1: una grammatica di tipo 2 non è in grado di esprimere le ripetizioni di tutti e tre i caratteri, ma solamente di entrambi. E' pertanto necessaria una grammatica di tipo 1. Può essere utile, se te la ricordi, scrivergli questa grammatica poiché così puoi mostrare che la differenza sostanziale che distingue una grammatica di tipo 1 da una di tipo 2 è la cosiddetta "inversione", che nel tipo 2, con la libertà dal contesto e un solo simbolo nella parte sinistra della produzione non è possibile fare.

$$S \rightarrow aBC|aSBC$$

$$CB \rightarrow BC$$

$$aB \rightarrow ab, bB \rightarrow bb, cB \rightarrow bc, cC \rightarrow cc$$

- $L = \{a^r b^n c^{2n}, n \geq 1, r \geq 1\}$
Questo linguaggio è di tipo 2: il termine elevato alla r non mi interessa e non mi deve fuorviare: deve essere notato invece che per quante b incontro, troverò un numero doppio di c , inoltre tale numero non è limitato (non esiste alcun $n \leq N$).
- $L = \{a^n b^m c^{n+m}, n \geq 0, m \geq 0\}$
Questo linguaggio è di tipo 2, ha 2 volte il self-embedding poiché esternamente si ripetono n volte le a e le c , mentre internamente si ripetono m volte delle b e delle c . Una possibile grammatica potrebbe essere questa (ovviamente da ridurre perchè le ϵ -rules non possono stare che nello scopo:

$$S \rightarrow aSc|A$$

$$A \rightarrow bAc|\epsilon$$

4. Se ci sono vincoli di realizzazione, come si fa per far diventare la grammatica di tipo 3?

$$L = \{begin^n istrend^n, n \geq 1\}$$

Semplicemente, se ci sono vincoli sulle risorse, io vincolo lo spazio del linguaggio $1 \leq n \leq N$ e in questo modo il linguaggio da tipo 2 diventa di tipo 3.

5. Se io ho un linguaggio formato da regole di un solo simbolo, di che tipo è il linguaggio?

Se il linguaggio è formato da un solo simbolo terminale (esempio $S \rightarrow aSa|a$), il linguaggio può essere solo di tipo 3. Infatti è possibile trasformare la produzione in $S \rightarrow aaS|a$ senza perdere di generalità: questa grammatica è ovviamente regolare e di tipo 3.

6. Che differenza c'è tra le grammatiche di tipo 1 e tipo 2?

Le grammatiche di tipo 2 sono libere dal contesto e si avranno delle produzioni del tipo $A \rightarrow B$, mentre le grammatiche di tipo 1 sono legate al contesto e una tipica regola di produzione del tipo 1 è $BC \rightarrow CB$: il concetto di inversione è una peculiarità che non è catturabile avendo un simbolo solo.

7. Di che tipo è questa grammatica?

$$S \rightarrow (A)$$

$$A \rightarrow a$$

Bisogna guardare se le parentesi sono ripetute oppure appaiono una sola volta o un numero finito di volte: nel primo caso siamo in presenza di self-embedding è la grammatica è di tipo 2, in caso contrario si tratta di una grammatica di tipo 3.

8. (*Domanda semplice*) Sia $S \rightarrow dA$ (più altre regole). E' una forma normale di Chomsky?

No, una forma normale di Chomsky prevede che una produzione sia delle forma $A \rightarrow BC|d$: non può esserci una produzione con simboli terminali e non nella parte destra (e ovviamente non possono neanche esserci produzioni che semplicemente rinominano un simbolo terminale, es. $A \rightarrow B$), quindi la produzione data deve essere splittata in due: $S \rightarrow DA, D \rightarrow d$.

9. Ci sia una applet/lampadina/sistema di supporto al corso di reti logiche/... di cui conosco solo il numero di stati N e che $A = \{a, b\}$. Non conosco nient'altro. Come so cosa fa questo sistema?
Faccio esperimenti per conoscere come è fatto questo insieme. Comincio partendo dalla stringa vuota, poi da tutti i vocaboli di lunghezza 1, poi da quelli di lunghezza 2, ecc fino a lunghezza $2N$ finché non trovo lo stato finale. So infatti che con i vocaboli di $N - 1$ lettere devo raggiungere lo stato finale, ma provo fino a $2N$ per controllare la presenza di anelli o autoanelli. Al caso peggiore, il sistema non ha stato finale.
10. Spiegare i due teoremi che definiscono un linguaggio non vuoto o infinito sulla base della lunghezza delle stringhe che compongono il linguaggio. Si tratta dei due teoremi che dicono che un linguaggio di tipo 3 è non vuoto se esiste una stringa la cui lunghezza è $< N$, con N il numero degli stati dell'ASF; l'altro teorema invece dice che un linguaggio è infinito se esiste una stringa di lunghezza compresa tra N e $2N$. Le dimostrazioni sono solamente intuitive, non formali come le dimostrazioni che ci sono sulle slides.
11. Sia data l'espressione regolare, viene chiesto di disegnare l'automa.
- $a(a^*b + c)^*b$
 - $a(a^*b + c)^*d$
 - $(a^*b + c)^*b$
 - $(ab^* + c)^*a$
 - $(ab^* + c^*a)^*b$

Il prof. non chiede di disegnare direttamente l'automa minimo o deterministico, anzi, non è richiesto. Normalmente conviene cominciare a disegnare l'automa con gli archi epsilon, ricordando che:

- * equivale ad un autoanello
- + equivale ad un'alternativa
- () disegno prima il pezzo tra parentesi, poi collego anche gli stati esterni
- ()* disegno prima delle transizioni epsilon, poi ragiono su come eliminarle
- una transizione epsilon viene eliminata sostituendola con altrettante transizioni verso le stesse destinazioni dello stato a cui si sarebbe giunti con la transizione epsilon.

Dopo questo si proseguirà ad eliminare gli archi epsilon. Se al termine di questa operazione l'ASF ottenuto è non-deterministico non è un problema: il prof non chiede di renderlo deterministico, ma se ti offri di farlo, sii sicura di trasformarlo bene.

12. Dato l'ASF, determinare la grammatica regolare a destra e a sinistra.

- Per la grammatica regolare a destra è semplice: si parte dallo scopo $S = I$ e si scrivono le produzioni seguendo gli archi dell'automa, ovvero guardando "in avanti" fino ad arrivare allo stato finale, che ovviamente non viene scritto (cioè se dallo stato B con il carattere c vado in F, scriverò $B \rightarrow c$). Tutte le produzioni che scriverò avranno la forma del tipo *stato attuale* \rightarrow *carattere*, *stato futuro*.
- Per la grammatica regolare a sinistra devo leggere l'automa dalla fine verso l'inizio, quindi parto dallo scopo $S = F$ e pian piano torno indietro fino alla "fine" che è il mio stato I. Le regole di produzione che scrivo sono "ribaltate" rispetto alle regole per la grammatica regolare a destra, e hanno una forma del tipo *stato precedente*, *carattere* \leftarrow *stato attuale*.

13. Date le regole regolari a destra definire le corrispettive regole regolari a sinistra.

Niente paura, semplicemente si ridisegna l'automa derivante dalle regole regolari a destra e si calcolano quelle regolari a sinistra.

14. Dato l'ASF, scrivere le espressioni regolari relative.

- (a) Dall'automa si calcolano le regole di produzione, poi si trova l'equazione sintattica e infine si determina l'espressione regolare \Leftarrow è un passaggio matematicamente corretto, ma lungo, e possono facilmente verificarsi errori di calcolo nei passaggi che portano a trovare espressioni regolari errate
- (b) la soluzione più veloce e sicura è quella di minimizzare l'automa e scrivere l'espressione regolare sapendo che ogni diramazione è un $+$ e ogni anello o autoanello un $*$.

15. Data una grammatica, scrivere le espressioni regolari relative.

- (a) Riscrivo tutte le produzioni sapendo che ogni \rightarrow diventa un $=$ e ogni OR ($|$) un $+$

- (b) Raccolgo a destra tutti i simboli non terminali
 - (c) La ricorsione diventa un * (esempio: $S = aA + b = a*b$)
16. Spiegare pro e contro dell'analisi top-down e di quella bottom-up.
 17. In Javascript esiste l'ereditarietà? E come può essere resa?
 18. Parlare di Javascript di `__proto__` e `prototype`
 19. In JavaScript come si aggiungono funzioni a oggetti già creati?
 Per aggiungere funzioni a un oggetto con l'idea che tali funzioni siano disponibili anche per gli oggetti che sono già stati creati è necessario aggiungerle al prototipo di costruzione esistente, che è già associato agli oggetti esistenti, ovvero `oggetto.prototype.mia_funzione = function(etc...)`. Al contrario, per aggiungere delle funzioni o proprietà solamente agli oggetti che verranno creati d'ora in poi, senza toccare gli oggetti che ho già creato, devo definire il costruttore per l'oggetto che deve essere il mio nuovo prototipo e assegnarlo alla proprietà `prototype` del costruttore `oggetto.prototype = new NuovoCostruttore()`.
 20. Parlare dei riconoscitori LR
 21. Spiegare a livello di principio la differenza tra SLR e LR. Perché l'SLR è semplificato? Descrivere una tipica situazione in cui SLR non ci riesce mentre l'LR sì. (Se $SLR \supseteq LR$, quel po' di più genera il conflitto)
 L'analisi LR(1) è lunga e complicata perché come l'analisi LR(0) cerca di costruire l'automa caratteristico calcolando le espressioni regolari per i contesti LR(1), lavoro già lungo e complicato nel caso in cui $k = 0$. L'analisi semplificata SLR invece calcola i contesti SLR(1) come la concatenazione dei contesti LR(0) della produzione voluta con tutti i simboli che possono seguire il non terminale della produzione da studiare. Per questo motivo si può dire che l'analisi SLR ha il costo dell'analisi LR(0) e quasi tutta l'espressività dell'LR(1). Quest'analisi però non è esatta perché inserisce tutte le possibili soluzioni da cui può derivare una produzione, mentre la grammatica LR potrebbe essere più stringente e vincolata, pertanto una grammatica LR(1) potrebbe non essere SLR(1) proprio perché quelle regole "in più" che potrebbero saltare fuori nell'analisi semplificata genera conflitto.
 22. Parlare delle Reti di Petri.
 23. Quando una rete di Petri è assimilabile a un ASF?
 Una rete di Petri è assimilabile a un ASF se

- (a) ogni transizione ha un solo posto d'ingresso e un solo posto di uscita;
 - (b) tutti gli archi hanno peso unitario;
 - (c) la rete è k -limitata: tutte le marcature prevedono un unico token
24. Data una rete di Petri dire se può essere considerata equivalente ad un automa a stati finiti.
- Controllo che ogni transizione abbia solamente un posto d'entrata e uno d'uscita, e soprattutto che non ci siano posti che accumulano token senza avere uscite: in questo modo la rete non è k -limitata dato che accumula infinitamente i token.

25. Come è possibile collocare le reti di Petri sotto il profilo dell'espressività tra le macchine che abbiamo visto? Quali sono le proprietà delle reti di Petri? Cosa non può fare la rete di Petri rispetto alla piena espressività?

Le reti di Petri si collocano a metà strada tra gli ASF e le macchine di Turing. Le proprietà di analizzabilità delle reti di Petri sono raggiungibilità e copertura, limitatezza e vitalità, e sono tutte decidibili negli ASF. Tuttavia, aumentando l'espressività delle reti di Petri, il formalismo diventa Turing-equivalente e le proprietà diventano indecidibili. Le reti di Petri non possono in generale risolvere il problema del *test a zero*, ovvero non è possibile abilitare una transizione in assenza di token in un dato posto d'ingresso. E' però possibile estendere le reti di Petri per renderle più espressive, aggiungendo degli archi inibitori (che quindi condiziona il test a zero) oppure trasformando le reti in reti colorate o con priorità.