

1 Concetti

Alfabeto, stringa, lunghezza di una stringa x (si indica con $|x|$), linguaggio (insieme di stringhe), cardinalità di un linguaggio (numero stringhe di un linguaggio).

Linguaggio universale Insieme di tutte le stringhe che possono essere scritte in un dato linguaggio, compresa la stringa vuota.

2 Operazioni sulle stringhe

Concatenamento Associativo. Non commutativo. Unisce due stringhe. Il concatenamento di due stringhe x e y si scrive xy oppure $x.y$.

Stringa vuota Si indica con ε $|\varepsilon| = 0$ $\varepsilon x = x\varepsilon = x$

Riflessione Lettura da destra a sinistra. Si indica con x^R .

Potenza n-esima $x^n = \underbrace{x.x.\dots.x}_{n \text{ volte}}$ $x^0 = \varepsilon$

3 Operazioni sui linguaggi

3.1 Operatori di grado 1

Riflessione Riflessione applicata a tutte le stringhe del linguaggio.

Prefissi

$Pref(L) = \{y|x = yz \wedge x \in L \wedge y \neq \varepsilon \wedge z \neq \varepsilon\}$ Questo è l'insieme dei prefissi *propri*, perchè non include quelli coincidenti con la stringa stessa.

Complemento $\neg L = L_{univ} \setminus L$

3.2 Operatori di grado 2

Analoghi agli operatori sulle stringhe dove non altrimenti specificato.

Operazioni insiemistiche Mantengono il significato usuale. \cup \cap \setminus \subset \subseteq

Concatenamento

Potenza n-esima

Stella di Kleene Operatore per il passaggio al limite dell'elevamento a potenza. $L^* = \bigcup_{h=0}^{\infty} L^h = L^0 \cup L^1 \cup L^2 \cup \dots$

Proprietà Monotonicità, chiusura rispetto al complemento, idempotenza, commutatività di $*$ e R (riflessione).

Operatore croce Come la stella, ma implica almeno una ripetizione. $L^+ = \bigcup_{h=1}^{\infty} L^h = L^1 \cup L^2 \cup \dots$

4 Linguaggi regolari

Def: Un linguaggio è regolare (appartiene alla famiglia REG) se può essere definito usando le **espressioni regolari**, cioè solo con le operazioni:

- concatenamento
- unione
- stella

applicare un numero finito di volte ai linguaggi unitari $\{a\}$, $\{b\}$, ... e al linguaggio vuoto.

Def: dato Σ alfabeto terminale, r è una espressione regolare (o *r.e.* o *regexp*) se:

1. $r = \emptyset$
2. $r = a \quad a \in \Sigma$
3. $r = (s \cup t)$ con s, t espressioni regolari
4. $r = (s.t)$
5. $r = (s)^*$

Ulteriori scritture utilizzabili Per comodità, sono definite ulteriori scritture per le regexp, derivabili dalle precedenti.

(0..9) Una cifra decimale

(a..z) Una lettera alfabetica minuscola

(A..Z) Un carattere alfabetico maiuscolo

Espressioni regolari estese Introducono gli operatori $+$ (croce), \neg (complemento) e \cap (intersezione), non presenti nelle regexp ma derivabili dagli operatori base.

Precedenze tra gli operatori $*$ oppure $+$, \cdot , \cup

Chiusura La famiglia REG è chiusa rispetto alle operazioni: \cdot \cup $*$ $+$ \cap \neg

4.1 Grammatica delle espressioni regolari

$G_{ER} : \text{espr} \rightarrow \varepsilon$
 $\text{espr} \rightarrow (\text{espr}.\text{espr})$
 $\text{espr} \rightarrow (\text{espr})^*$
 $\text{espr} \rightarrow (\text{espr} \cup \text{espr})$

4.2 Linguaggi finiti

I linguaggi appartenenti alla famiglia FIN sono i linguaggi che hanno un numero finito di frasi.

$$\text{FIN} \subseteq \text{REG}$$

5 Automi

Def: Uno stato di un automa si dice *accessibile* se è raggiungibile dallo stato iniziale e *post-accessibile* se da esso è raggiungibile uno stato finale. Uno stato accessibile e post-accessibile è *utile*. Un automa si dice *pulito* se ogni suo stato è utile.

6 Derivazione

$e' \Rightarrow e''$ se esistono fattorizzazioni $e' = \alpha.\gamma.\beta$ e $e'' = \alpha.\delta.\beta$ tali che δ è una scelta di γ , cioè è una delle possibile stringhe ottenibili da gamma per sostituzione di un carattere non-terminale.

Derivazione a n passi $e' \xRightarrow{n} e''$

Indica che è possibile passare dalla stringa e' alla stringa e'' tramite n derivazioni successive.

Se $n = 0$ si tratta di un'identità e quindi $e' = e''$.

Se il numero di passi di derivazione non è specificato (ed è almeno uguale a 1) si può scrivere $e' \xRightarrow{+} e''$.

Ambiguità Se due o più percorsi di derivazione diversi (alberi sintattici) portano alla stessa stringa, la frase è ambigua, così come la grammatica che l'ha generata.

Possono esistere ambiguità: di ricorsione bilaterale, di unione non disgiunta, di concatenamento, di codifica, di incertezza sull'ordine di applicazione delle regole (vedere pag 49 del libro¹ per info sulla rimozione delle ambiguità).

L'ambiguità può anche essere inerente al linguaggio: un linguaggio può cioè essere definito solo da grammatiche ambigue. Si parla quindi di un linguaggio **inerentemente ambiguo**.

Un linguaggio è ambiguo se e solo se la grammatica che lo genera e l'automa corrispondente sono a loro volta ambigui.

Verificare l'ambiguità

- E' possibile verificare l'ambiguità di una regexp numerandola (cioè assegnando un differente numero ad ogni suo carattere terminale) e poi verificando se possa o meno generare due stringhe diverse x_1 e x_2 tali che, rimuovendo i numeri, diventano uguali. Se lo può fare, è ambigua, altrimenti no.
- Se l'automa associato è deterministico, la grammatica (e la regexp) non sono ambigue
- Se il linguaggio da verificare è riconducibile ad un linguaggio noto sicuramente non ambiguo, allora non è ambiguo.

7 Grammatiche

Def (Grammatica): Insieme di regole che, attraverso applicazioni ripetute permettono di definire tutte e sole le frasi di un linguaggio. Ogni grammatica comprende oltre alle regole, un alfabeto non-terminale V , un alfabeto terminale Σ e un particolare terminale $S \in V$ detto assioma. Ogni regola è una coppia ordinata $X \rightarrow \alpha$ con $X \in V$ e $\alpha \in (V \cup \Sigma)^*$

Def (grammatica lineare): grammatica con al più un nonterminale nella parte destra, cioè della forma:

$$A \rightarrow uBv \text{ dove } u, v \in \Sigma^*, B \in (V \cup \epsilon)$$

7.1 Grammatiche libere (Linguaggi context-free)

Anche chiamati linguaggi *Type 2* di Chomsky o *BNF* (Backus Normal Form, o Backus Naur Form).

Servono a superare i limiti dei linguaggi REG, come l'impossibilità di avere strutture annidate all'infinito.

Le grammatiche libere possono essere estese, per ragioni di leggibilità, con l'uso di espressioni regolari nella parte destra delle regole. Le grammatiche libere estese vengono dette *EBNF* e hanno pari potenza alle grammatiche *BNF*.

Linguaggi di Chomsky

Type 0 Macchina di Turing (ricorsivamente enumerabili)

Type 1 Context sensitive

Type 2 Context-free (BNF)

Type 3 REG

¹Stefano Crespi Reghizzi, "Linguaggi formali e compilazione", ed. Pitagora

7.1.1 Composizioni regolari di linguaggi liberi

Applicando le operazioni dei linguaggi regolari (unione, concatenamento, ecc) a linguaggi liberi si ottengono ancora linguaggi liberi.

7.2 Linguaggi infiniti

Sono composti da un numero infinito di frasi.

Condizione necessaria e sufficiente affinché un linguaggio sia infinito è che la sua grammatica ammetta derivazioni ricorsive.

7.3 Linguaggi di Dyck

Per linguaggi di Dyck si intendono i linguaggi che seguono il paradigma a parentesi (aperte e chiuse, concatenate e annidate). I loro alfabeti terminali sono caratterizzati da una o più coppie di parentesi aperte e chiuse.

Le frasi di Dyck sono caratterizzate dalla cosiddetta *regola di cancellazione*: applicando più volte, ad esempio ad un linguaggio caratterizzato dalle coppie di parentesi $()$ e $[]$, le regole

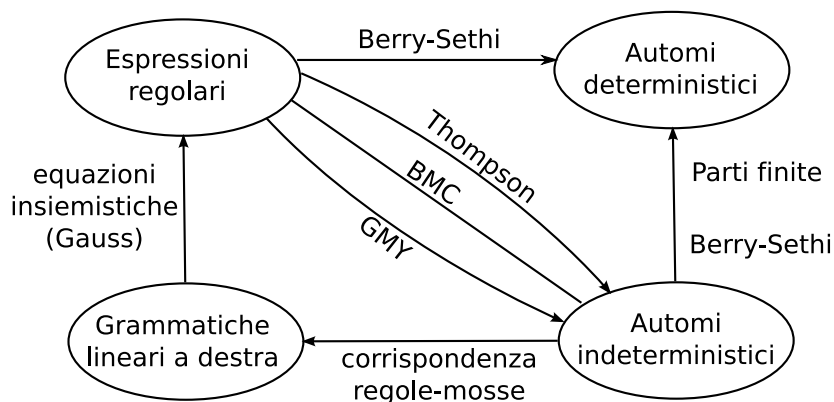
$$() \Rightarrow \varepsilon \quad [] \Rightarrow \varepsilon$$

se si ottiene la stringa vuota, la frase appartiene a un linguaggio di Dyck.

Preso l'alfabeto $\Sigma = \{a, a', b, b'\}$, dove a e b sono parentesi aperte e a' e b' parentesi chiuse, la grammatica del relativo linguaggio di Dyck è:

$$S \rightarrow aSa'S|bSb'S|\varepsilon$$

8 Trasformazioni



8.1 Da automa ad automa riflesso

Per passare dall'automa che riconosce il linguaggio $L(A)$ all'automa che riconosce il linguaggio $L(A^R)$ si possono seguire due metodi:

1. Trasformazione diretta dell'automa

- Invertire il verso delle frecce
- Scambiare stati iniziali e finali
- Creare un unico stato iniziale (e collegarlo agli altri tramite ε -mosse)
- (Opzionale) Determinizzare l'automa con i procedimenti appositi

2. Trasformazione utilizzando espressioni regolari

- Calcolare l'espressione regolare di $L(A)$ (metodo BMC)
- Ricavare l'espressione regolare di $L(A^R)$

Vedere paragrafo 8.2.

- (c) Ricavare l'automa relativo a $L(A^R)$ a partire dalla regexp (metodo GMY - potenzialmente indeterministico - o metodo Berry-Sethi - sicuramente deterministico)

8.2 Da espressione regolare a espressione regolare del linguaggio riflesso

L'espressione regolare del linguaggio riflesso si ottiene riflettendo tutti i concatenamenti.

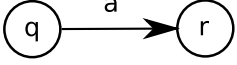
Quindi, se il linguaggio (o una sua sottostringa) è composto da più parti $L(A) = P_1 \dots P_n$, l'espressione regolare ottenuta sarà $L(A^R) = P_n \dots P_1$.

Solo il concatenamento è toccato da questa trasformazione. L'unione (\cup oppure $|$), la stella $*$, e tutti gli altri operatori non ne sono influenzati.

Esempio $L(A) = a(ba|bca)^*(\varepsilon|b) = P_1 P_2 P_3$
 $P_1 = a \Rightarrow P_1^R = a^R = a$
 $P_2 = (ba|bca)^* \Rightarrow P_2^R = ((ba^R)|(bca)^R)^* = (ab|acb)^*$
 $P_3 = (\varepsilon|b) \Rightarrow P_3^R = (\varepsilon^R|b^R) = (\varepsilon|b)$
 $L(A^R) = (P_1 P_2 P_3)^R = P_3^R P_2^R P_1^R = (\varepsilon|b)(ab|acb)^* a$

8.3 Da automa deterministico a grammatica (lineare a destra)

La grammatica ha come **nonterminali** i nomi degli stati dell'automa, come **assioma** lo stato iniziale, come **regole**, per ogni

mossa  si ha $q \rightarrow ar$. Se q è **finale**, si ha anche la regola $q \rightarrow \varepsilon$.

Successivamente, se la grammatica contiene regole vuote (cioè che finiscono in ε) la si può convertire nella forma non annullabile.

8.4 Da grammatica lineare a sinistra ad automa finito

1. Costruire la grammatica riflessa G^R cambiando tutte le regole della forma $A \rightarrow Ba$ in $A \rightarrow aB$.
2. Costruire l'automa finito corrispondente alla grammatica lineare a destra così ottenuta.
3. Scambiare gli stati iniziali e finali e invertire tutte le frecce dell'automa.

8.5 Da grammatica annullabile a non annullabile

Def: Un nonterminale A è annullabile se esiste una derivazione $A \xRightarrow{+} \varepsilon$.

Def: Una grammatica è in *forma normale senza regole vuote* (o *non annullabile*) se nessun nonterminale diverso dall'assioma è annullabile.

Per trasformare la grammatica in non annullabile, per ogni regola R tale che contiene un nonterminale annullabile nella parte destra:

$$R \rightarrow aA | \dots$$

$$A \rightarrow bB | \varepsilon$$

si rimuove il nonterminale annullabile e si aggiungono tra le possibili regole alternative quelle che si avrebbero annullando il nonterminale:

$$R \rightarrow aA | \dots | a \text{ perchè } aA \Rightarrow a\varepsilon \Rightarrow a$$

$$A \rightarrow bB$$

8.6 Da grammatica con copiatore a grammatica priva di copiatore

Def (copiatore): una regola $A \rightarrow B$ di una grammatica, dove B è un simbolo nonterminale.

Data una grammatica contenente regole:

$$A \rightarrow B$$

$$B \rightarrow \alpha$$

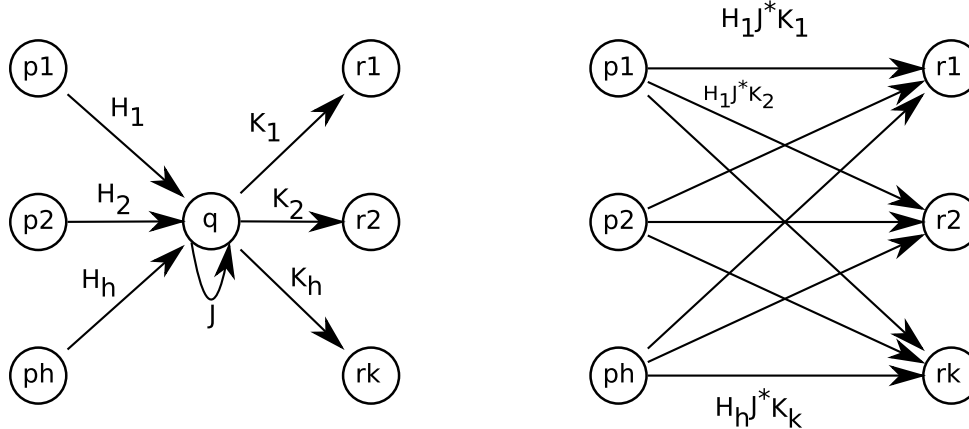
eliminare le copiatore significa modificare tutte le regole di tipo $A \rightarrow B$ in modo tale da ottenere regole:

$$A \rightarrow \alpha \text{ con } \alpha \in ((\Sigma \cup V)^* \setminus V).$$

Per fare ciò, può essere utile calcolare gli insiemi delle copie, cioè gli insiemi di nonterminali che si possono ottenere a partire da ogni nonterminale senza generare alcun terminale.

8.7 Da automa indeterministico a espressione regolare - Metodo Brozowski-McCluskey (BMC)

1. **Normalizzazione:** deve essere presente un solo stato iniziale privo di archi entranti e un solo stato finale privo di archi uscenti. Ciò può essere ottenuto, dove necessario, aggiungendo uno stato iniziale e finale che si collegano al resto dell'automa tramite ε -mosse.
2. **Eliminazione:** uno alla volta, si scelgono tutti gli stati interni dell'automa e li si eliminano, sostituendo le etichette degli archi uscenti con le rispettive espressioni regolari, fino ad ottenere l'espressione regolare completa, corrispondente all'etichetta dell'unico arco che collega lo stato iniziale allo stato finale.



Tra ogni coppia di stati p_i e r_j vi sarà l'arco $p_i \xrightarrow{H_i J^* K_j} r_j$.

8.8 Da automa finito indeterministico ad automa finito deterministico

Algoritmo delle parti finite

1. Rendere pulito l'automa (vedere 5)
2. Eliminazione delle mosse spontanee ed eliminazione delle regole di copiatura (vedere 8.5 e 8.6). *NB:* quando ci sono più ε -mosse collegate "in cascata" una dopo l'altra, conviene eliminarle in ordine inverso, a partire dall'ultima (ad esempio, in: $a \xrightarrow{\varepsilon_1} b \xrightarrow{\varepsilon_2} c$ conviene eliminare prima ε_2). In tal modo si ottiene un automa più semplice
3. Sostituzione di più transazioni non deterministiche con una sola che porta in un nuovo stato (costruzione delle parti finite). Sia N un automa indeterministico e M' l'automa deterministico equivalente. Se in N vi sono le mosse

$$p \xrightarrow{a} p_1, p \xrightarrow{a} p_2, \dots, p \xrightarrow{a} p_k$$

perchè lo stato successore di p è scelto in modo non deterministico, si crea in M' un nuovo stato collettivo $[p_1, p_2, \dots, p_k]$ tale che $p \xrightarrow{a} [p_1, p_2, \dots, p_k]$.

Si costruiscono quindi gli archi uscenti dallo stato collettivo in questo modo: se uno stato collettivo contiene gli stati p_1, p_2, \dots, p_k per ognuno di essi si considerano in N gli archi uscenti, etichettati con la stessa lettera b :

$$p_1 \xrightarrow{b} [q_1, q_2, \dots], p_2 \xrightarrow{b} [r_1, r_2, \dots]$$

e si uniscono gli stati di arrivo ottenendo lo stato collettivo di arrivo della transizione

$$[p_1, p_2, \dots, p_k] \xrightarrow{b} [q_1, q_2, \dots, r_1, r_2, \dots]$$

Se tale stato ancora non esiste in M' , lo si crea.

Se un macrostato contiene uno stato che in N era finale, anche il macrostato è finale.

4. Si prosegue finchè non sono state sostituite tutte le transazioni di N .

Algoritmo di Berry-Sethi

1. Numerare le etichette degli archi non spontanei di N ottenendo l'automa N' .
2. Calcolare gli insiemi locali *Ini*, *Fin*, *Seg* per N' .
3. Applicare la costruzione di BS per ottenere l'automa deterministico M (vedere 8.12, BS, punto 3).

8.9 Da automa deterministico ad automa complemento

1. Completare l'automa deterministico aggiungendo uno stato p detto pozzo. Detta $\delta(q, a)$ la funzione di transizione dell'automa originario che determina lo stato di destinazione cui si giunge da q leggendo a , si definisce la nuova funzione di transizione $\bar{\delta}(q, a)$ come segue:
 $\bar{\delta}(q, a) = \delta(q, a)$ dove $\delta(q, a)$ è definita (le transizioni già definite non cambiano)
 $\bar{\delta}(q, a) = p$ altrimenti (tutte le transizioni non definite ora vanno nello stato pozzo)
 $\bar{\delta}(p, a) = p \quad \forall a \in \Sigma$ (qualunque carattere venga letto, il successore del pozzo è il pozzo stesso).
2. Scambiare gli stati finali e non finali della macchina. Il pozzo è quindi uno stato finale (lo stato iniziale non cambia).

8.10 Da espressione regolare a grammatica libera

Si decompone ripetutamente la e.r. data r nelle sue sottoespressioni. I casi possibili sono i seguenti:

Sottoespressione	Regola grammaticale
$r = r_1 \cdot r_2 \cdot \dots \cdot r_k$	$E \rightarrow E_1 E_2 \dots E_k$
$r = r_1 \cup r_2 \cup \dots \cup r_k$	$E \rightarrow E_1 \cup E_2 \cup \dots \cup E_k$
$r = (r_1)^*$	$E \rightarrow E E_1 \varepsilon$ oppure $E \rightarrow E_1 E \varepsilon$
$r = (r_1)^+$	$E \rightarrow E E_1 E_1$ oppure $E \rightarrow E_1 E E_1$
$r = b \in \Sigma$	$E \rightarrow b$
$r = \varepsilon$	$E \rightarrow \varepsilon$

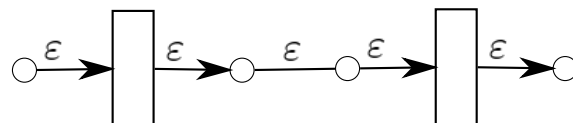
8.11 Da espressione regolare ad automa indeterministico

Metodo strutturale (di Thompson) Una e.r. può essere trasformata in automa a partire dalle sue sottoespressioni atomiche, unite e concatenate secondo le seguenti regole di corrispondenza (in figura).

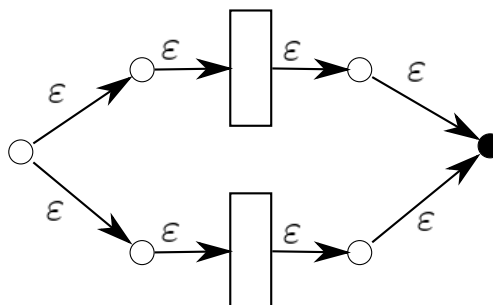
Espressioni regolari atomiche



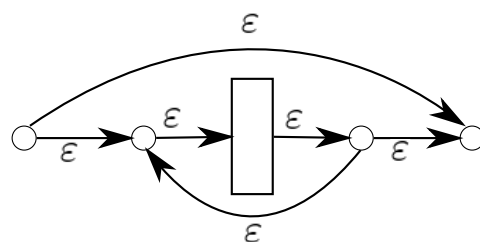
Concatenamento di due e.r.



Unione di due e.r.



Stella di una e.r.



Algoritmo di Glushkov, Mc Naughton e Yamada (GMY)

Def (Insieme degli inizi): $Ini(L) = \{a \in \Sigma | a\Sigma^* \cap L \neq \emptyset\}$

Def (Insieme delle fini): $Fin(L) = \{a \in \Sigma | \Sigma^*a \cap L \neq \emptyset\}$

Def (Insieme dei digrammi): $Dig(L) = \{x \in \Sigma^2 | \Sigma^*x\Sigma^* \cap L \neq \emptyset\}$.

Def (linguaggio locale): un linguaggio L è locale se è completamente determinato da $Ini(L)$, $Fin(L)$, $Dig(L)$.

Def (e.r. lineare): una e.r. è detta lineare se nessun carattere è ripetuto. Il linguaggio definito da una e.r. lineare è locale.

1. Numerare l'e.r. ottenendo una e.r. lineare e' .
2. Calcolare per e' l'annullabilità e gli insiemi locali Ini , Fin , Dig .
3. Costruire il riconoscitore del linguaggio caratterizzato da tali insiemi:
 - (a) Esiste un unico stato iniziale q_0 , con nessuna freccia entrante.
 - (b) Da q_0 partono frecce verso gli stati indicati da Ini (ognuna marcata con il nome dello stato di arrivo), gli stati finali sono quelli indicati da Fin , le frecce del diagramma vanno dallo stato corrispondente al primo carattere a quello corrispondente al secondo per ogni digramma presente in Dig .
 - (c) tutte le frecce etichettate con la stessa lettera a entrano nello stesso stato, quello denominato a .
 - (d) gli stati non iniziali sono in corrispondenza biunivoca con l'alfabeto della e.r. numerata.
4. Cancellare la numerazione dalle etichette degli archi.

8.12 Da espressione regolare ad automa deterministico

Si potrebbe usare Thompson o GMY e poi determinizzare, altrimenti esiste un algoritmo apposito.

Algoritmo di Berry-Sethi (BS) Utilizza gli insiemi Ini e Fin , già definiti. Invece di Dig , usa Seg .

Sia e la e.r. e sia e' la sua corrispettiva numerata.

$Def(Insieme\ dei\ seguiti)$: l'insieme dei seguiti di un carattere è definito come

$$Seg(a_i) = \{b_j | a_i b_j \in Dig(e')\}$$

Per convenzione, si considera anche il carattere terminatore $\neg \in Seg(a_i)$ per ogni $a_i \in Fin(e')$.

1. Numerare l'e.r. ottenendo una e.r. lineare e' .
2. Calcolare $Ini(e')$ e Seg di tutti i caratteri che compaiono nella e.r.
3. A partire dall'insieme Ini (il primo macrostato), si compongono nuovi macrostati (ove necessario) o ci si ricollega a macrostati già esistenti nel seguente modo: per ogni macrostato, si considera l'unione dei seguiti di tutti gli stati contenuti nel macrostato aventi la stessa lettera x . Tale unione, è il macrostato in cui si giunge a partire dal primo tramite un arco marcato x . Ad esempio, dal macrostato $[a_1, b_2, a_4]$, con l'arco a si giunge nel macrostato $Seg(a_1) \cup Seg(a_4)$, mentre con l'arco b in $Seg(b_2)$.
4. I macrostati contententi \neg sono finali.

9 Grafi di flusso

Con “calcolare le equazioni di flusso” (siano esse per l'analisi degli intervalli di vita o per le definizioni raggiungenti) si intende dire che bisogna scrivere le equazioni $in(p)$ e $out(p)$ specifiche per ogni singolo nodo p .

Trovare i valori da essi assunti viene detto “calcolare le soluzioni delle equazioni di flusso”.

9.1 Analisi degli intervalli di vita (liveness analysis)

Def : una variabile a è viva nel punto p del programma se esiste un cammino $p \rightarrow \dots \rightarrow q$ tale che q è un uso di a e non esiste una definizione (assegnamento) di a nel tratto c del cammino.

Def : $In(p)$ è l'insieme delle variabili vive all'ingresso del nodo p .

Def : $Out(p)$ è l'insieme delle variabili vive all'uscita del nodo p .

Def : $Use(p)$ è l'insieme delle variabili usate dal nodo p , cioè il cui valore viene letto (ad esempio perchè compaiono in un'espressione nella parte destra di un assegnamento).

Def : $Def(p)$ è l'insieme delle variabili definite dal nodo p , cioè a cui viene attribuito un valore tramite un'istruzione di assegnamento o di lettura.

Def : $Succ(p)$ è l'insieme dei successori immediati del nodo p , cioè dei nodi in cui entra una freccia uscente da p .

Esempio Dato $p : a := a \oplus b$, dove \oplus è un generico operatore: $def(p) = \{a\}$, $use(p) = \{a, b\}$

Equazioni di flusso per l'analisi degli intervalli di vita Condizione iniziale per il calcolo: $\forall p \ In(p) = Out(p) = \emptyset$

$$In(p) = Use(p) \cup (Out(p) \setminus Def(p))$$

$$Out(p) = \bigcup_{s \in Succ(p)} In(s)$$

Si itera sostituendo alle incognite la soluzione all'iterazione (i-1)-esima, fino ad ottenere valori costanti in due iterazioni successive.

Nota Per calcolare le equazioni di flusso può essere comodo calcolare prima una tabella che indichi gli insiemi $Def(p)$ e $Use(p)$ per tutti i nodi p del grafo di flusso.

9.2 Definizioni raggiungenti

Def: Si dice che la definizione di a nell'istruzione q , detta a_q , raggiunge l'ingresso di un'istruzione p (non necessariamente distinta da q) se esiste un cammino da q a p che non passa per un nodo che definisce a (escluso il nodo q stesso).

Def: $pred(n)$ = predecessori di n : tutti quei blocchi da cui parte una freccia entrante in n .

Def: $def(n)$ = insieme delle variabili definite nel blocco n

Def: $sopp(n)$ = insieme delle variabili la cui definizione precedente viene annullata nel blocco n , cioè di tutte le variabili definite in n e che erano già state definite in precedenza in un blocco diverso da n .

Def: $in(n)$ è l'insieme insieme delle definizioni che raggiungono l'entrata del nodo n .

Def: $out(n)$ è l'insieme insieme delle definizioni che raggiungono l'uscita del nodo n .

Equazioni di flusso delle definizioni raggiungenti Per il nodo 1 iniziale: $in(1) = \emptyset$ (supponendo che non ci siano parametri in entrata al sottoprogramma)

$$in(n) = \bigcup_{p \in pred(n)} out(p)$$

$$out(n) = def(n) \cup (in(n) \setminus sopp(n))$$

Si itera sostituendo alle incognite la soluzione all'iterazione (i-1)-esima, fino ad ottenere valori costanti in due iterazioni successive.

Nota Per calcolare le equazioni di flusso, è utile preparare prima la tabella delle definizioni e soppressioni in ogni nodo.

10 Analisi sintattica

10.1 Analisi sintattica discendente deterministica

Def(Parsificatore $LL(k)$): parsificatore discendente che legge il testo da sinistra a destra (Left) con derivazione sinistra (Leftmost). k precisa il numero di caratteri della prospezione.

Condizione $LL(k)$ Un parsificatore discendente è $LL(k)$ se in ogni bivio della rete di macchine è possibile distinguere con una prospezione di k caratteri il percorso da scegliere, cioè se gli insiemi guida di lunghezza k sono disgiunti per gli archi del bivio.

In modo più formale:

Def(Condizione $LL(1)$): Una macchina M_A della rete soddisfa la condizione $LL(1)$ nello stato q se, per ogni coppia di frecce uscenti dallo stato, gli insiemi guida sono disgiunti.

Una regola soddisfa la condizione $LL(1)$ se ogni stato della macchina corrispondente la soddisfa.

Proprietà Ogni linguaggio regolare è generato da una grammatica $LL(1)$.

10.1.1 Grammatica $LL(1)$

Una grammatica gode della proprietà $LL(1)$ se ogni stato delle macchine della rete soddisfa la condizione $LL(1)$.

Per essere $LL(1)$, una grammatica deve essere non ambigua (rimedi a pag 49 del libro) e senza ricorsioni sinistre (pag 64).

Se, pur rispettando tali due condizioni la grammatica non è $LL(1)$, può essere necessario applicare una *fattorizzazione sinistra*.

La grammatica lineare a destra ricavata da un'automa a stati finiti deterministico è sicuramente $LL(1)$.

10.1.2 Definizioni utili

Insieme degli inizi

1. $a \in Ini(q)$ se \exists arco $q \xrightarrow{a} r$.
2. $a \in Ini(q)$ se \exists arco $q \xrightarrow{A} r$ e $a \in Ini(q_{A,0})$ dove $q_{A,0}$ è lo stato iniziale della macchina M_A .
3. $a \in Ini(q)$ se \exists arco $q \xrightarrow{A} r$ e $L(q_{A,0})$ è annullabile e $a \in Ini(r)$.

Insieme dei seguiti

1. $\neg \in Seg(S)$
2. $a \in Seg(A)$ se \exists arco $q \xrightarrow{A} r$ e $a \in Ini(r)$
3. $a \in Seg(A)$ se \exists arco $q \xrightarrow{A} r$ nella macchina M_B , $B \neq A$ e il linguaggio $L(r)$ è annullabile e $a \in Seg(B)$
4. $a \in Seg(A)$ se \exists arco $q \xrightarrow{A} r \rightarrow e$ e r è uno stato finale della macchina M_B , $B \neq A$ e $a \in Seg(B)$

NB: se in una rete vi sono più comparse di un nonterminale, i seguiti di ogni comparsa vanno uniti.

Insieme guida (o di prospezione) dall'automa Serve da selezionatore per la scelta della mossa negli stati dove si presenta un bivio.

E' definito per ogni freccia (mossa o accettazione) della rete.

1. Per un arco $q \xrightarrow{b} r$ con etichetta terminale $b \in \Sigma$ si ha: $Gui(q \xrightarrow{b} r) = \{b\}$
2. Per un arco $q \xrightarrow{A} r$ della macchina M_B , etichettato dal nonterminale A, si ha:
 - (a) $Gui(q \xrightarrow{A} r) = Ini(L(q_{A,0})L(r))$ se il concatenamento $L(q_{A,0})L(r)$ non è annullabile
 - (b) $Gui(q \xrightarrow{A} r) = Ini(L(q_{A,0})L(r)) \cup Seg(B)$, altrimenti
3. Per la freccia $q \rightarrow$ di uno stato finale q della macchina M_B si ha $Gui(q \rightarrow) = Seg(B)$

10.1.3 Insieme guida (o di prospezione) dalla grammatica non estesa (BNF)

$$\begin{cases} Gui(A \rightarrow \alpha) = Ini(\alpha) & \text{se } \alpha \text{ non è annullabile} \\ Gui(A \rightarrow \alpha) = Ini(\alpha) \cup Seg(A) & \text{altrimenti} \end{cases}$$

inoltre

$$Ini(\alpha) = \{a \in \Sigma \mid \alpha \xrightarrow{*} ay \wedge y \in \Sigma^*\}$$

$$Seg(A) = \{a \in \Sigma \cup \{\neg\} \mid S \xrightarrow{*} yAaz \wedge y, z \in \Sigma^*\}$$

10.2 Analisi sintattica ascendente deterministica

Viene utilizzata quando l'analisi LL(k) non è sufficiente, cioè quando esiste stato da cui escono almeno due frecce con insiemi guida sovrapposti.

LR(k) permette di posticipare la decisione della strada da prendere finché non si hanno informazioni a sufficienza. LR indica che la scansione del testo è da sinistra a destra (Left) e che la derivazione calcolata è destra (Rightmost).

E' possibile usare LR(0), perché in taluni casi è sufficiente osservare lo stato in cui si trova l'automa.

Un linguaggio non deterministico non è LR(k). Richiede l'uso del metodo di Earley.

Tipi di stati Gli stati contenuti in un macrostato possono essere:

- di spostamento da esso, nella macchina della rete, esce un arco verso un altro stato
- di riduzione è uno stato finale

Tipi di macrostati

- di *riduzione*: contiene soltanto stati di riduzione
- di *spostamento*: contiene soltanto stati di spostamento
- *misto*: contiene stati di spostamento e stati di riduzione

10.2.1 LR(0)

Un parsificatore LR(0) interpreta una stringa basandosi solo sul proprio stato e sui caratteri già letti. Non esegue prospezione.

Si trovano pochi linguaggi LR(0) perché sono necessariamente privi di prefissi.

Se una grammatica contiene regole annullabili del tipo $A \rightarrow \varepsilon|\alpha$ o regole con prefissi $A \rightarrow a\alpha$, non può essere LR(0).

La grammatica di Dyck non è LR(0).

Condizione LR(0) Una grammatica soddisfa la condizione LR(0) se ogni macrostato della macchina pilota verifica entrambe le condizioni:

1. non è misto
2. se è un macrostato di riduzione, contiene un solo stato

10.2.2 LR(1)

Un parsificatore LR(1) interpreta una stringa basandosi sulle stesse informazioni di un LR(0) e, in più, sulla conoscenza del carattere successivo (detto *prospezione*).

Una grammatica ambigua non può essere riconosciuta da un parsificatore LR(1).

Condizione LR(1) Una grammatica soddisfa la condizione LR(1) se per ogni macrostato della macchina pilota valgono entrambe le condizioni:

1. *assenza di conflitto riduzione-spostamento*: ogni candidata di riduzione² ha un insieme di prospezione disgiunto dall'insieme dei simboli terminali di spostamento (ossia dall'insieme delle etichette terminali delle frecce uscenti dal macrostato)
2. *assenza di conflitto riduzione-riduzione*: se vi sono due candidate di riduzione, i loro insiemi di prospezione sono disgiunti

Una grammatica gode della proprietà LR(1) se ogni macrostato della macchina pilota soddisfa la condizione LR(1).

Condizione LR(k) Data una grammatica, non è decidibile se esista un intero $k > 0$ per cui essa risulti LR(k); di conseguenza non è decidibile se il linguaggio generato da una grammatica libera è deterministico.

Se però il valore di k è fissato, si può decidere se la grammatica è LR(k) per quel particolare k costruendo l'automa pilota e verificando che non presenti macrostati inadeguati.

10.2.3 LALR(1)

LALR(1) = Look Ahead LR(1).

È una condizione di determinismo intermedia tra LR(0) e LR(1), molto diffusa a causa della scarsità di memoria richiesta rispetto a un parsificatore LR(1).

L'automa pilota LALR(1) si può ottenere ignorando le informazioni sulla prospezione dell'automa LR(1) e fondendo insieme gli stati che in questo modo risultano identici. Si ottiene così un automa pilota isomorfo a quello LR(0). Le informazioni sulla prospezione, tuttavia, non vengono eliminate, ma mantenute all'interno dell'unico stato rimasto (che ha quindi informazioni doppie, nei casi in cui queste non sono coincidenti).

La **condizione LALR(1)** è, per il resto, analoga alla condizione LR(1).

10.3 Metodo di Earley

È un algoritmo generale di analisi sintattica che permette di trattare tutte le grammatiche libere, anche se ambigue.

Quando possibile non viene utilizzato a causa delle sue grandi richieste computazionali.

Il metodo di Earley prevede l'uso di un vettore E di insiemi di coppie (stato, puntatore), dove "stato" è lo stato (della rete di macchine definita dalla grammatica) in cui ci si trova attualmente e "puntatore" è il puntatore all'ultimo carattere letto prima di generare la coppia. Il vettore è definito come $E[0..n]$ con n pari al numero di caratteri della stringa da analizzare. Sono inoltre disponibili la stringa da analizzare x , il cui n -esimo carattere è x_n e i , l'indice dell'ultimo carattere letto ($i = 0$ indica che non sono ancora stati letti caratteri).

Sono definite tre operazioni:

²Una *candidata di riduzione* è un macrostato con una regola marcata completata. Ciò significa che, rappresentando la grammatica del linguaggio come rete di macchine, una di queste si trova nel suo stato finale.

Predizione applicabile quando dallo stato attuale (rappresentato come rete di macchine) esce un'etichetta nonterminale A . Aggiunge a $E[i]$ la coppia formata dallo stato iniziale della macchina che è attivata dal **nonterminale** $(q_{A,0})$ e il puntatore i : $\langle q_{A,0}, i \rangle$.

Scansione applicabile quando dallo stato attuale esce una freccia marcata da un carattere **terminale** a . Se $x_{i+1} = a$ si aggiunge lo stato di arrivo q' della freccia a $E[i+1]$ come $\langle q', i \rangle$.

Completamento si applica a una coppia $\langle s, j \rangle \in E[i]$ il cui stato s è finale per una macchina M_A della rete (si è appena finito di elaborare un nonterminale A , quindi). L'elaborazione ritorna sull'insieme $E[j]$. Ricerca al suo interno una coppia $\langle q, k \rangle$ tale che da q esca l'arco $q \xrightarrow{A} r$. Aggiunge a $E[i]$ lo stato di arrivo r con il valore k del puntatore.

Inizializzazione ed esecuzione

1. $E[0] = \{\langle s = 0, p = 0 \rangle\}$; $E[i] = \emptyset$ per $i = 1, \dots, n$

2. $i = 0$

3. ripeti

(a) esegui predizioni finchè possibile

(b) esegui scansioni finchè possibile

(c) esegui completamenti finchè possibile

(d) $i = i + 1$

4. finchè la costruzione dell'insieme $E[n]$ non è finita.

Se $E[n]$ contiene almeno una coppia completata $\langle q_{term} \equiv S \rightarrow \alpha\bullet, 0 \rangle$ dove q_{term} è lo stato finale della macchina dell'assioma, la stringa sorgente è accettata.

Se in un qualunque passaggio, dopo la scansione, $E[i+1]$ è vuoto e $i < n$, la stringa è rifiutata.