06-parsing-bottom-up-canonical-LR-and-beyond

06-parsing-bottom-up-canonical-LR-and-beyond

Automi LR(1)

Sono gli automi che riescono a parsare più grammatiche, ma per questo presentano un elevato numero di stati e un'elevata difficoltà computazionale.

LR(1)-items

Gli items hanno la forma $[A \to \beta,\ \Delta]$, con Δ che viene detto *lookahead set*.
La chiusura di un item è chiamata $closure_1$ ed è più raffinata della $closure_0$, infatti dato l'item $\{[A \to \alpha \cdot B\beta,\ \Delta]\}$ la chiusura propaga i simboli seguenti B a tutti gli item aggiunti all'insieme per chiudere B.

$closure_1$ per LR(1)-items

Usiamo questa funzione più raffinata perchè ci permette, oltre che definire un set di partenza, anche di aggiungere elementi.
Questi set "più ampi" ci possono aiutare nel sapere quali caratteri dobbiamo aspettare di vedere per poter fare una `reduce`.

Definizione

Sia P un insieme di LR(1)-items, allora $closure_1(P)$ è identifica il più piccolo insieme di item con il più piccolo *lookahead set* che soddisfa la seguente equazione:

$$closure_1(P) = P \cup \{[B \to \cdot\gamma,\ \Gamma] \mid [A \to \alpha \cdot B\beta,\ \Delta] \in closure_1(P) \land B \to \gamma \in P' \land first(\beta\Delta) \subseteq \Gamma\}$$

Dove $first(\beta\Delta) = \bigcup_{d \in \Delta} first(\beta d)$ e P' è l'insieme delle produzioni con l'aggiunta del nuovo start symbol $S' \to S$.
Ora dopo ver tirato giù il calendario ci sarà passata la paura e potremmo vedere che l'equazione è risolvibile tramite il teorema del punto fisso 😐.

Algoritmo

Con quel formulone pure la dovente si è spaventata quindi vediamola con un linguaggio più familiare a noi, lo pseudocodice:

```
function closure1(P)
        foreach item ∈ P do
                item.unmarked = True;
        while ∃ item ∈ P: item.unmarked==True do
                item.unmarked = False;
                if item has the form [A → α · Bβ, Δ] then
                        Set Δ1 = Set();
                        foreach d∈Δ do
                                Δ1.insert(first(βd));
                        foreach B → γ ∈ P′ do
                                if B → ·γ ∉ presenti_in(P) then
                                        Item nuovo = Item([B → ·γ, Δ1]);
                                        nuovo.unmarked = True;
                                        P.add(nuovo);
                                else
                                        if ([B → ·γ, Γ] ∈ P and Δ1 ⊄ Γ) then
                                        update [B → ·γ, Γ] to [B → ·γ, Γ ∪ Δ1]
in P;
                                        P.get([B → ·γ, Γ ∪ Δ1]).unmarked = True;
        return P ;
```

Sembra ancora uno schifo ma un esempio vale più di mille parole.

Esempio

Prendiamo la grammatica:

$$\mathcal{G}: \begin{cases} S \to aAd|bBd|aBe|bAe \\ A \to c \\ B \to c \end{cases}$$

Calcoliamo $closure_1(\{[S' \to \cdot S,\ \{\$\}]\})$.
Detta a livello brutale dobbiamo "ricalcolare" ogni volta il follow di una produzione in base al contesto un cui ci troviamo.

$$0: \begin{bmatrix} \underline{S' \to \cdot S,\ \{\$\}} \\ S \to \cdot aAd,\ \{\$\} \\ S \to \cdot bBd,\ \{\$\} \\ S \to \cdot aBe,\ \{\$\} \\ S \to \cdot bAe,\ \{\$\} \end{bmatrix}$$

Ora passiamo agli alrti stati, lo stato 1 è il classico di `accept` quindi lo saltiamo.

$$\tau(0,a) = 2: \begin{bmatrix} S \to a \cdot Ad,\ \{\$\} \\ \underline{S \to a \cdot Be,\ \{\$\}} \\ A \to \cdot c,\ \{d\} \\ B \to \cdot c,\ \{e\} \end{bmatrix}$$

Come possiamo vedere abbiamo dovuto ricalcolare i $follow$ di A e B.
Lasciatemi dire Ебать sti cazzo di LR(1)-items.

Costruzione automa LR(1)

L'idea è di popolare gli stati mentre definiamo le funzioni di transizione.
Se lo stato P contiene uno item della forma $[A \to \alpha \cdot Y\beta,\ \Delta]$, allora esiste uno stato $Q = \tau(P, Y)$ tale che conterrà l'item $[A \to \alpha Y \cdot \beta,\ \Delta]$ e tutti gli item di $closure_1(\{[A \to \alpha Y \cdot \beta,\ \Delta]\})$.

Algoritmo

```
Inizializziamo Q per contenere P0 = closure1({[S' → ·S, {$}]});
P0.unmarked = True;
while ∃ P ∈ Q : P.unmarked == True do
        P.unmarked = False;
        foreach Y a destra del marker in un qualsiasi item di P do
                Tmp = τ(P,Y).kernel();
                if ∃ R ∈ Q : R.kernel() == Tmp then
                        τ(P,Y) = R;
                else
                        State nuovo_stato = (closure1(Tmp));
                        nuovo_stato.unmarked = True;
                        Q.add(nuovo_stato);
                        τ(P,Y) = nuovo_stato;
```

Esempio

Prendiamo la grammatica:

$$\mathcal{G}: \begin{cases} S \to aAd|bBd|aBe|bAe \\ A \to c \\ B \to c \end{cases}$$

Se proviamo a fare il parsing SLR(1) questa cosa non viene ma se proviamo a fare quello LR(1).....
Iniziamo con la costruzione degli stati (spero vi piaccia come ho organizzato la scrittura degli stati).

$$0: \left[\begin{array}{l} S' \to \cdot S,\ \{\$\} \\ \hline S \to \cdot aAd,\ \{\$\} \\ S \to \cdot bBd,\ \{\$\} \\ S \to \cdot aBe,\ \{\$\} \\ S \to \cdot bAe,\ \{\$\} \end{array}\right]$$

$$\tau(0, S) = 1 : [S' \to S\cdot,\ \{\$\}]$$

$$\tau(0,a) = 2 : \left[\begin{array}{c} S \to a \cdot Ad,\ \{\$\} \\ S \to a \cdot Be,\ \{\$\} \\ \hline A \to \cdot c,\ \{d\} \\ B \to \cdot c,\ \{e\} \end{array}\right]$$

$$\tau(0,b) = 3 : \left[\begin{array}{c} S \to b \cdot Bd,\ \{\$\} \\ S \to b \cdot Ae,\ \{\$\} \\ \hline B \to \cdot c,\ \{d\} \\ A \to \cdot c,\ \{e\} \end{array}\right]$$

$$\tau(2,A) = 4 : [S \to aA \cdot d,\ \{\$\}]$$

$$\$\tau(2,B) = 5 : [S \to aB \cdot e,\ \{\$\}]$$

$$\tau(2,c) = 6 : \left[\begin{array}{c} A \to c\cdot,\ \{d\} \\ B \to c\cdot,\ \{e\} \end{array}\right]$$

$$\tau(3,B) = 7 : [S \to bB \cdot d,\ \{\$\}]$$

$$\tau(3,A) = 8 : [S \to bA \cdot e,\ \{\$\}]$$

$$\tau(3,c) = 9 : \left[\begin{array}{c} B \to c\cdot,\ \{d\} \\ A \to c\cdot,\ \{e\} \end{array}\right]$$

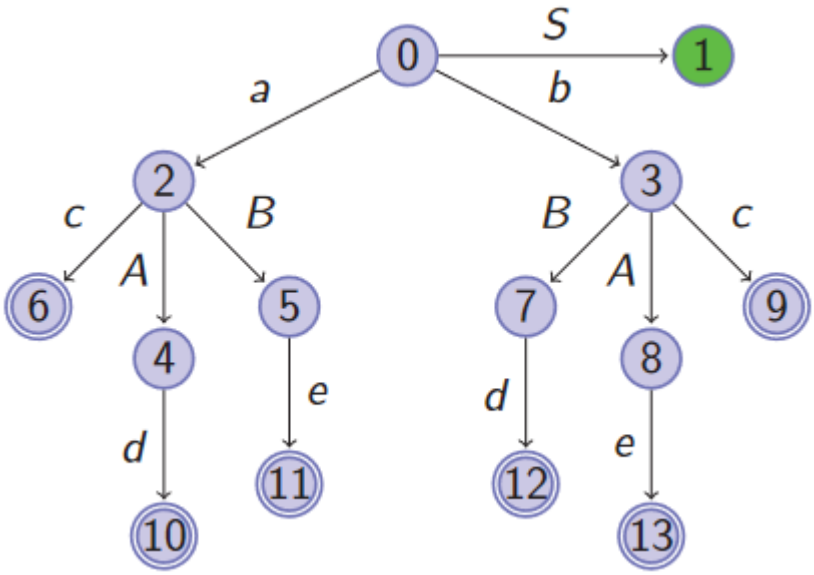
$$\tau(4,d) = 10 : [S \to aAd\cdot,\ \{\$\}]$$

$$\tau(5,e) = 11 : [S \to aBe\cdot,\ \{\$\}]$$

$$\tau(7,d) = 11 : [S \to bBd\cdot,\ \{\$\}]$$

$$\tau(8,e) = 11 : [S \to bAe\cdot,\ \{\$\}]$$

Ora l'automa carrteristico dovrebbe essere questo oppure un grafo isomorfo a questo.



La tabella è troppo lunga, non la riporto, comunque non ci sono conflitti ma il numero di stati è drasticamente aumentato.

Caso studio LR(1)

Proviamo ora a fare il parsing LR(1) della seguente grammatica.

$$\mathcal{G}: \begin{cases} S \to L = R | R \\ L \to *R | id \\ R \to L \end{cases}$$

per mia semplicità di digitazione ometterò le parentesi grafe nei *lookahead set*, l'operatore / indica concatena due elementi quindi $= /\$$ significa $\{=, \$\}$.

$$0: \begin{bmatrix} S' \to \cdot S, \$ \\ \hline S \to \cdot L = R, \$ \\ S \to \cdot R, \$ \\ L \to \cdot * R, = /\$ \\ L \to . id, = /\$ \\ R \to \cdot L, \$ \end{bmatrix}$$

$$\tau(0, S) = 1 : [S' \to S\cdot, \ \$]$$

$$\tau(0, L) = 2 : \begin{bmatrix} S \to L\cdot = R, \$ \\ R \to L\cdot, \$ \end{bmatrix}$$

$$\tau(0, R) = 3 : [S \to R\cdot, \$]$$

$$\tau(0, *) = 4 : \begin{bmatrix} L \to * \cdot R, = /\$ \\ \hline R \to \cdot L, = /\$ \\ L \to \cdot * R, = /\$ \\ L \to \cdot id, = /\$ \end{bmatrix}$$

$$\tau(0, id) = 5 : [L \to id\cdot, \ \ = /\$]$$

$$\tau(2, =) = 6 : \begin{bmatrix} S \to L = \cdot R, \$ \\ \hline R \to \cdot L, \$ \\ L \to \cdot * R, \$ \\ L \to \cdot id, \$ \end{bmatrix}$$

$$\tau(4, R) = 7 : [L \to *R\cdot, = /\$]$$

$$\tau(4, L) = 8 : [R \to L\cdot, = /\$]$$

$$\tau(4, *) = 4$$

$$\tau(4, id) = 5$$

$$\tau(6, R) = 9 : [S \to L = R\cdot, \$]$$

$$\tau(6, L) = 10 : [R \to L\cdot, \$]$$

$$\tau(6, *) = 11 : \begin{bmatrix} L \to * \cdot R, \$ \\ \hline R \to \cdot L, \$ \\ L \to \cdot * R, \$ \\ L \to \cdot id, \$ \end{bmatrix}$$

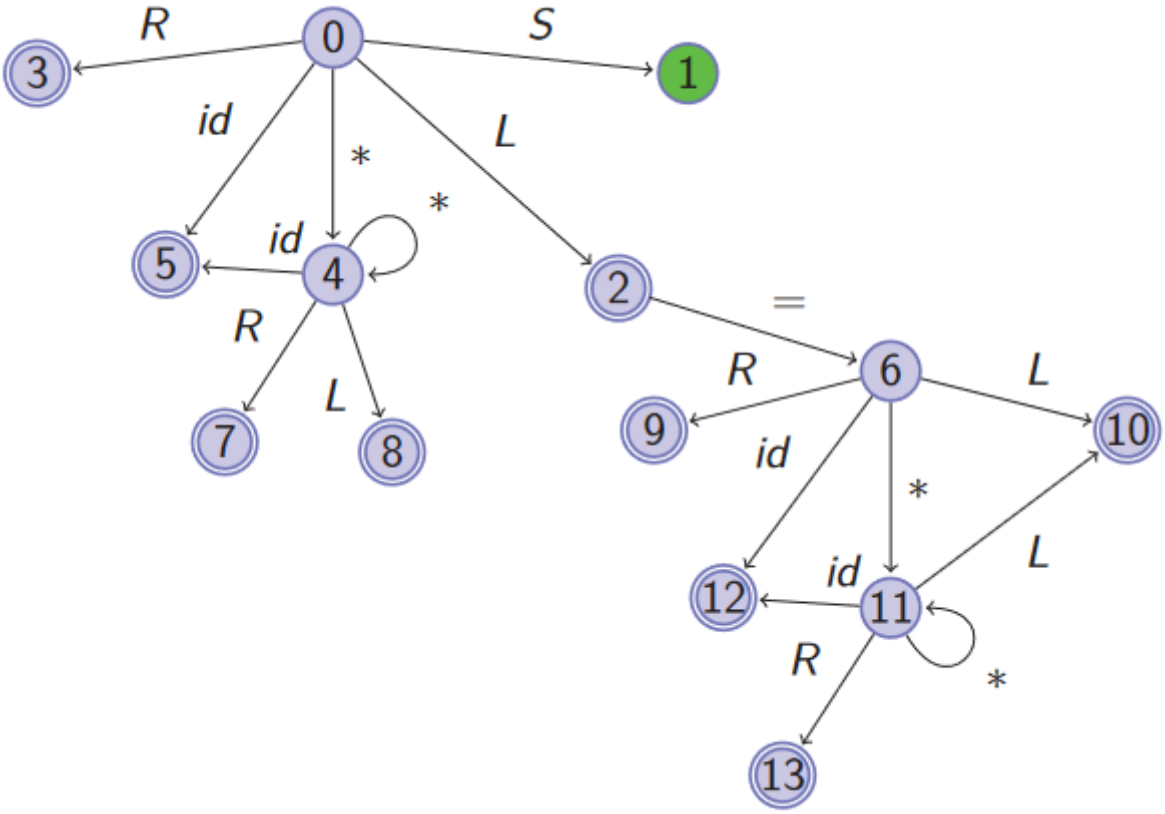
$$\tau(6, id) = 12 : [L \to id\cdot, \$]$$

$$\tau(11, R) = 13 : [L \to *R\cdot, \$]$$

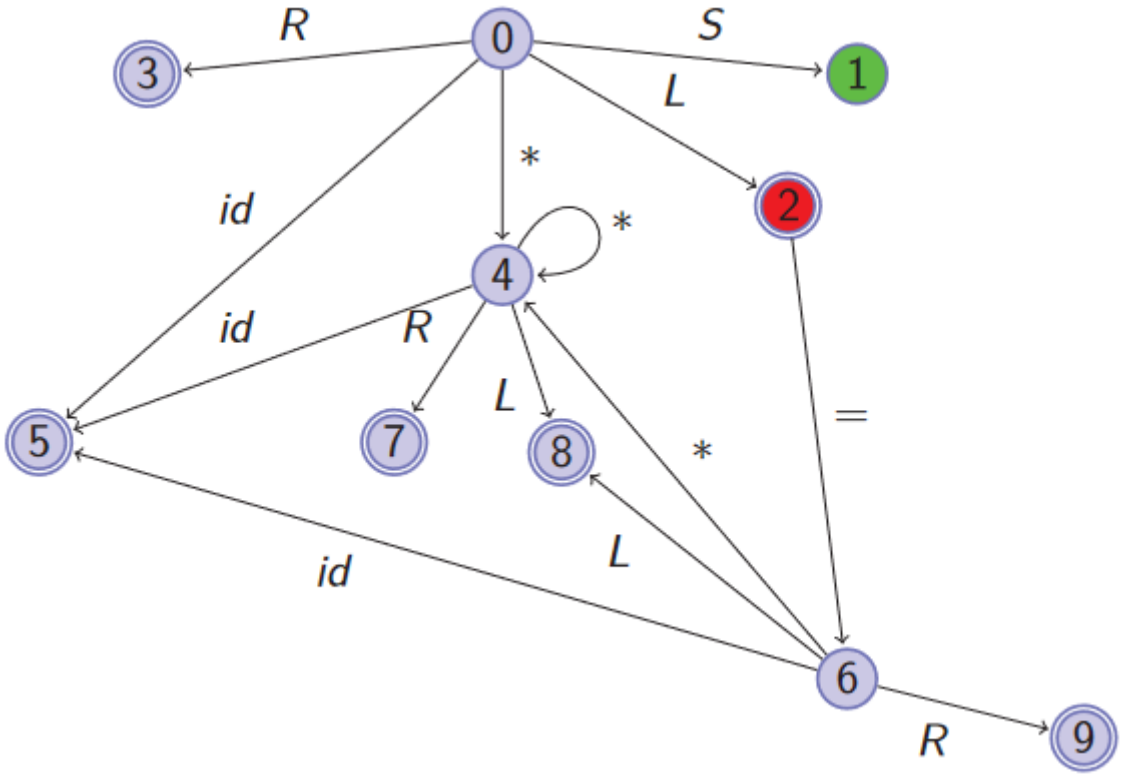
$$\tau(11, L) = 10$$

$$\tau(11, *) = 11$$

$$\tau(11, id) = 12$$



Ma ha un numero spropositato di stati, proviamo allora a vedere se la grammatica è SLR(1), costruiamo l'automa LR(0).



Come si può vedere nello stato 2 (in rosso) è presente un conflitto s/r.
Ora proviamo a fare il parsing della stringa w = "id=id".

Sappiamo che con LR(1) non abbiamo problemi, quindi proviamo il parsing SLR(1) però nel conflitto conserviamo l'istruzione `reduce` ovvero una r5.

Non riesco a impaginare in modo decente il procedimento, fatelo voi, è molto facile tranquilli

Comunque il tutto si conclude nello stato 3 quando dobbiamo leggere un = e ci viene ritornato `error`, zio pera sapevo che dovevo mettere la `reduce` s6 nel conflitto.

Conclusioni

Cerchiamo di capire perchè è LR(1) e non SLR(1).
Abbiamo $follow(L) = \{=, \$\}$ e nello stato 2 possiamo fare la `reduce` $R \to L$ nei casi in cui il prossimo carattere sia un elemento dei $follow(L)$, ma dallo stato 2 possiamo anche dover leggere $=$ perchè abbiamo anche l'item $S \to L \cdot = R$.
Ipotizziamo di leggere una generica stringa $w_1 = w_2$, possiamo osservare che = è generato solo dalla produzione $S \to L = R$.
Dopo aver letto tutta w_1 dovremmo in qualche modo trasformarla in L, le uniche parole derivabili da sono $\{*^n id | n \geq 0\}$ ma le uniche derivazioni di questo insieme coinvolgono R.

- Se abbiamo una parole tipo $id = w_2$ allora leggiamo id e facciamo s5, da qui un r4 $L \to id$ così torniamo a 0 e leggendo L andiamo a 2.
- Se la parola è nella forma $*w_1' = w_2$ facciamo uno s4 e rimaniamo quan finchè non abbiamo finito gli * in input, dopodichè ci sarà sicuramente un id che ci farà fare uno s5, una volta in 5 faremmo un r4 $L \to id$ il quale però ci riporterà in 4 che ci farà rimbalzare in 8 e non in 2.
  Quindi il parsing SLR(1) non riesce a riconoscere tutte le differenze tra gli stati, ma la sua implementazione ha un costo molto minore.

Parsing LALR(1)

Abbiamo detto che LR(1) è il metodo di parsing più ricco di informazioni ma quello più "pesante".
Come si vede nell'esempio precedente ci sono dei sottografi isomorfi, non sarebbe possibile unirli così da semplificare la nostra struttura, si noti che molti nodi hanno produzioni uguali ma *lookahead set* diversi.
Introduciamo quindi il parsing LALR(1) che con una funzione $\mathcal{LA}$ un po' raffinata promette di mantenere l'espressività di LR(1) con la compattezza di SLR(1). jaj
Quest'ultima frase sembra un pubblicità di Mastrota, immaginate voi come sono messo male.

Automi LRm(1)

Quella m sta per "merged".
Vediamo di costruire un automa LRm(1) chiamato $\mathcal{AM}$ partendo da un automa LR(1) detto

$\mathcal{A}$.

- Gli stati di $\mathcal{AM}$ sono costruiti unendo tutti gli stati <$P_1, \dots P_n$> di $\mathcal{A}$ tali che hanno le stesse proiezioni LR(0).
- Se in $\mathcal{A}$ c'è una Y-transizione da P a Q e se P è stato unito in <$P_1, \dots, P_n$> e Q in <$Q_1, \dots, Q_n$>, allora esiste una Y-transizione in $\mathcal{AM}$ da <$P_1, \dots, P_n$> a <$Q_1, \dots, Q_n$>.
  Nell'esempio precedente possiamo fare il merge degli stati 4 e 11 in 4&11 e degli stati 5 e 12 in 5&12, visto che esiste una id-transizione da 4 a 5 e da 11 a 12 allora posso creare una id-transizione da 4&11 a 5&12 nell'automa $\mathcal{AM}$.

Osservazioni

Per costruzione gli stati di un automa LRm(1) possono contenere più item con le stesse produzione LR(0).
Un automa LRm(1) ha lo stesso numero di stati e le stesse transizioni del corrispettivo automa LR(0).

Tabelle di parsing LALR(1)

Dobbiamo prendere:

- L'automa caratteristico LRm(1).
- La funzione di lookahead $\mathcal{LA}(P, A \to \beta) = \bigcup_{[A \to \beta \cdot, \Delta_j]} \Delta_j$
  Come al solito, una grammatica è LALR(1) se e solo se la tabella di parsing LALR(1) non ha conflitti.

Esempio

Prendiamo la nostra amata grammatica che genera i puntatori.

$$\mathcal{G}: \begin{cases} S \to L = R | R \\ L \to *R | id \\ R \to L \end{cases}$$

Proviamo ora a costruire la tabella di parsing LALR(1), non procederò da LR(1) a LRm(1) ma cercherò di fare il merging "a occhio" per velocizzare la scrittura di questa epopea di appunti.

$$0: \left[\begin{array}{c} S' \to \cdot S, \$ \\ \hline S \to \cdot L = R, \$ \\ S \to \cdot R, \$ \\ L \to \cdot * R, = / \$ \\ L \to . id, = / \$ \\ R \to \cdot L, \$ \end{array} \right]$$

$$\tau(0, S) = 1 : [S' \to S \cdot, \ \$]$$

$$\tau(0, L) = 2 : \left[\begin{array}{c} S \to L \cdot = R, \$ \\ R \to L \cdot, \$ \end{array} \right]$$

$$\tau(0, R) = 3 : [S \to R\cdot, \$]$$

$$\tau(0, *) = 4 : \left[\begin{array}{c} L \to * \cdot R, = / \$ \\ \hline R \to \cdot L, = / \$ \\ L \to \cdot * R, = / \$ \\ L \to \cdot id, = / \$ \end{array}\right]$$

$$\tau(0, id) = 5 : [L \to id\cdot, \ \ = / \$]$$

$$\tau(2, =) = 6 : \left[\begin{array}{c} S \to L = \cdot R, \$ \\ \hline R \to \cdot L, \$ \\ L \to \cdot * R, \$ \\ L \to \cdot id, \$ \end{array}\right]$$

$$\tau(4, R) = 7 : [L \to *R\cdot, = / \$]$$

$$\tau(4, L) = 8 : [R \to L\cdot, = / \$]$$

$$\tau(4, *) = 4$$

$$\tau(4, id) = 5$$

$$\tau(6, R) = 9 : [S \to L = R\cdot, \$]$$

$$\tau(6, L) = 8$$

$$\tau(6, *) = 4$$

$$\tau(6, id) = 5$$

Negli ultimi 3 casi avrei che $\Delta = \$$ ma essendo già degli stati con le stesse produzioni LR(0) ed un *lookahead set* più ampio posso fare il merge su quelle.
Quindi la tabella di parsing risulta essere.

	=	*	id	$	S	L	R
0		s4	s5		G1	G2	G3
1				Acc			
2	s6			r5			
3				r2			
4		s4	s5			G8	G7
5	r4			r4			
6		s4	s5			G8	G9
7	r3			r3			
8	r5			r5			
9				r1			

Automa simbolico

Il parsing LALR(1) abbiamo detto essere il miglior compromesso, però la lentezza nella costruzione inizile rimane, infatti dobbiamo prima costruire l'automa LR(1) e poi tradurlo in LRm(1).
Dobbiamo quindi trovare un modo per eliminare la prima scrittura dell'automa LR(1) e partire subito con un utoma LRm(1).
Possiamo usare variabili nei *lokkahead set* degli item nel kernel, per poter gestire i successivi *lookahead set* come un sistema di equazioni.
Quindi l'obbiettivo di una transizione sarà deciso come nelle LR(0), quando un obbiettivo di una transizione è già presente in uno stato il suo contributo al *lookahead set* è inserito nell'equazione associata con il suo item kernel.
Dopo la costruzione dell'automa risolviamo il sistema di equazioni ed il gioco è fatto.

Esempio

Prendiamo una grammatica che ormai dovremmo iniziare a considerare come una sorella ✝️🐷, la grammatica dei puntatori:

$$\mathcal{G}: \begin{cases} S \to L = R | R \\ L \to *R | id \\ R \to L \end{cases}$$

Iniziamo instanziando lo stato 0, ovvero

$$0: \left[\begin{array}{c} S' \to \cdot S, x_0 \\ \hline S \to \cdot L = R, x_0 \\ S \to \cdot R, x_0 \\ L \to \cdot * R, = / x_0 \\ L \to . id, = / x_0 \\ R \to \cdot L, x_0 \end{array}\right]$$

Aggiungiamo quindi al sistema $x_0 = \{\$\}$.
Ora calcoliamo $\tau(0, S)$ provenendo da $[S' \to S\cdot, x_0]$

$$1: [S' \to S\cdot,\ x_1]$$

Aggiungiamo al sistema di equazioni $x_1 = x_0$.
Calcoliamo $\tau(0, L)$ provenendo da $[S \to \cdot L = R, x_0]$ e $[R \to \cdot L, x_0]$

$$2: \left[\begin{array}{c} S \to L\cdot = R, x_2 \\ R \to L\cdot, x_3 \end{array}\right]$$

Aggiungiamo al sietma di equazioni $x_2 = x_0$ e $x_3 = x_0$.
Calcoliamo $\tau(0, R)$

$$3: [S \to R\cdot, x_4]$$

Aggiungiamo al sistema di equazioni $x_4 = x_0$.
Calcoliamo $\tau(0, *)$ provenendo da $[S \to \cdot R, x_0]$

$$4 : \begin{bmatrix} \overline{L \to * \cdot R, x_5} \\ R \to \cdot L, x_5 \\ L \to \cdot * R, x_5 \\ L \to \cdot id, x_5 \end{bmatrix}$$

Aggiungiamo al sistema di equazioni $x_5 = \{=, x_0\}$
Calcoliamo $\tau(0, id)$ provenendo da $[L \to \cdot * R, \{=, x0\}]$

$$5 : [L \to id\cdot,\ x_6]$$

Aggiungiamo al sistema di equazioni $x_6 = \{=, x_0\}$.
Calcoliamo $\tau(2, =)$ provenendo da $[S \to L\cdot = R, x_2]$

$$6 : \begin{bmatrix} \overline{S \to L = \cdot R, x_7} \\ R \to \cdot L, x_7 \\ L \to \cdot * R, x_7 \\ L \to \cdot id, x_7 \end{bmatrix}$$

Aggiungiamo al sistema di equazioni $x_7 = x_2$.
Calcoliamo $\tau(4, R)$ provenendo da $[L \to * \cdot R, x_5]$

$$7 : [L \to *R\cdot, x_8]$$

Aggiungiamo al sistema di equazioni $x_8 = x_5$.
Calcoliamo $\tau(4, L)$ provenendo da $[R \to \cdot L, x_5]$

$$8 : [R \to L\cdot, x_9]$$

Aggiungiamo al sistema di equazioni $x_9 = x_5$.
Calcoliamo $\tau(4, *)$ provenendo da $[L \to \cdot * R, x_5]$

$$4 : \begin{bmatrix} \overline{L \to * \cdot R, x_5} \\ R \to \cdot L, x_5 \\ L \to \cdot * R, x_5 \\ L \to \cdot id, x_5 \end{bmatrix}$$

Aggiungiamo al sistema di equazioni $x_5 = \{=, x_0\} \cup x_5$.
Calcoliamo $\tau(4, id)$ provenendo da $[L \to \cdot id, x_5]$

$$5 : [L \to id\cdot,\ x_6]$$

Aggiungiamo al sistema di equazioni $x_6 = \{=, x_0\} \cup x_5$.
Calcolo $\tau(6, R)$ provenendo da $[S \to L = \cdot R, x_7]$

$$9 : [S \to L = R\cdot, x_{10}]$$

Aggiungiamo al sistema di equazioni $x_{10} = x_7$.
Calcoliamo $\tau(6, L)$ provenendo da $[R \to \cdot L, x_7]$

$$8 : [R \to L\cdot, x_9]$$

Aggiungo al sistema di equazioni $x_9 = x_5 \cup x_7$.
Calcoliamo $\tau(6, *)$ provenendo da $[L \to \cdot * R, x_7]$

$$4 : \begin{bmatrix} \dfrac{L \to * \cdot R, x_5}{R \to \cdot L, x_5} \\ L \to \cdot * R, x_5 \\ L \to \cdot id, x_5 \end{bmatrix}$$

Aggiungiamo al sistema di equazioni $x_5 = \{=, x_0\} \cup x_5 \cup x_7$.
Calcoliamo $\tau(6, id)$ provenendo da $[L \to \cdot id, x_7]$

$$5 : [L \to id \cdot,\ x_6]$$

Aggiungiamo al sistema i equazioni $x6 = \{=, x_0\} \cup x_5 \cup x_7$.
Quindi ora il sistema dovrebbe risultare:

$$\begin{cases} x_0 = \{\$\} \\ x_1 = x_0 \\ x_2 = x_0 \\ x_3 = x_0 \\ x_4 = x_0 \\ x_5 = \{=, x_0\} \cup x_5 \cup x_7 \\ x_6 = \{=, x0\} \cup x_5 \cup x_7 \\ x_7 = x_2 \\ x_8 = x_5 \\ x_9 = x_5 \cup x_7 \\ x_{10} = x_7 \end{cases}$$

Ora risolvendolo:

- $x_0, x_1, x_2, x_3, x_4, x_7, x_{10} = \{\$\}$
- $x_5, x_6, x_8, x_9 = \{=, \$\}$