

Metody Realizacji Języków Programowania

Analiza składniowa wstępująca

Marcin Benke

MIM UW

9 stycznia 2015

Analiza wstępująca — metoda LR

- Od Lewej, pRawostronne wyprowadzenie (w odwrotnej kolejności)
- Automat ze stosem, na stosie ciąg terminali i nieterminali
- Jeśli na stosie jest prawa strona produkcji, możemy ją zastąpić symbolem z lewej (redukcja)
- Pytanie, kiedy to robić — poznamy różne techniki.
- Automat startuje z pustym stosem i akceptuje, gdy całe wejście zredukuje do symbolu startowego.

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5: $F \rightarrow n$

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5: $F \rightarrow n$
E + F	* 3	reduce 4: $T \rightarrow F$

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5: $F \rightarrow n$
E + F	* 3	reduce 4: $T \rightarrow F$
E + T	* 3	shift (dlaczego?)

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5: $F \rightarrow n$
E + F	* 3	reduce 4: $T \rightarrow F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5: $F \rightarrow n$
E + F	* 3	reduce 4: $T \rightarrow F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift
E + T * 3	#	reduce 5: $F \rightarrow n$

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5: $F \rightarrow n$
E + F	* 3	reduce 4: $T \rightarrow F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift
E + T * 3	#	reduce 5: $F \rightarrow n$
E + T * F	#	reduce 3: $T \rightarrow T * F$

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5: $F \rightarrow n$
E + F	* 3	reduce 4: $T \rightarrow F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift
E + T * 3	#	reduce 5: $F \rightarrow n$
E + T * F	#	reduce 3: $T \rightarrow T * F$
E + T	#	reduce 1

Przykład

Gramatyka: 1 : $E \rightarrow E + T$ 2 : $E \rightarrow T$, 3 : $T \rightarrow T * F$, 4 : $T \rightarrow F$,
5 : $F \rightarrow n$

Stos	Wejście	Akcja
(pusty)	1 + 2 * 3	shift (przesuń z wejścia na stos)
1	+ 2 * 3	reduce 5 (redukcja produkcji 5: $F \rightarrow n$)
F	+ 2 * 3	reduce 4: $T \rightarrow F$
T	+ 2 * 3	reduce 2: $E \rightarrow T$
E	+ 2 * 3	shift
E +	2 * 3	shift
E + 2	* 3	reduce 5: $F \rightarrow n$
E + F	* 3	reduce 4: $T \rightarrow F$
E + T	* 3	shift (dlaczego?)
E + T *	3	shift
E + T * 3	#	reduce 5: $F \rightarrow n$
E + T * F	#	reduce 3: $T \rightarrow T * F$
E + T	#	reduce 1
E	#	accept

Gramatyki LR(k)

Nieformalnie: jeśli dla formy zdaniowej αw mamy już na stosie α , to para $\langle \alpha, k : w \rangle$ wyznacza jednoznacznie co zrobić, a w szczególności:

- czy na szczycie stosu jest prawa strona jakiejś produkcji?
(łatwe, ale może być więcej niż jedna)
- czy należy redukować, a jeśli tak, to którą produkcję?
(trudne, podglądamy k symboli z wejścia)

W praktyce ograniczamy się do $k \leq 1$.

Własności gramatyk LR(k)

Twierdzenie (Knuth)

Dla każdej gramatyki LR(k) istnieje równoważny deterministyczny automat ze stosem.

Własności gramatyk LR(k)

Twierdzenie (Knuth)

Dla każdej gramatyki LR(k) istnieje równoważny deterministyczny automat ze stosem.

Twierdzenie (Knuth)

Każdy język deterministyczny L ma gramatykę $G \in LR(k)$ taką, że $L = L(G)$.

Własności gramatyk LR(k)

Twierdzenie (Knuth)

Dla każdej gramatyki LR(k) istnieje równoważny deterministyczny automat ze stosem.

Twierdzenie (Knuth)

Każdy język deterministyczny L ma gramatykę $G \in LR(k)$ taką, że $L = L(G)$.

Twierdzenie

Każda gramatyka LL(k) jest też LR(k). Istnieją gramatyki LR(0), które nie są LL(n) dla żadnego n.

Kiedy redukować?

Różne metody:

LR(0) — redukujemy kiedy się tylko da
w praktyce za słaba

LR(1) — precyzyjnie wyliczamy dla jakich terminali na wejściu redukować
bardzo silna metoda, ale koszt generowania rzędu 2^{n^2} .

SLR(1) — Simple LR(1): LR(0) + prosty pomysł: redukujemy $A \rightarrow \alpha$ jeśli terminal z wejścia należy do FOLLOW(A).

LALR(1) — Look Ahead LR(1): zgrubnie wyliczamy (budujemy automat LR(1) i sklejamy podobne stany).
*w praktyce dostatecznie silna metoda,
tyle samo stanów co w automacie LR(0)*

Zależności między klasami gramatyk

Pomiędzy klasami gramatyk zachodzą inkluzje

$$LR(0) \subset SLR(1) \subset LALR(1) \subset LR(1)$$

Wszystkie powyższe inkluzje są ostre.

Ponadto

$$LL(1) \subset LR(1)$$

Uwaga: istnieją gramatyki $LL(1)$, które nie są $LALR(1)$, czyli

$$LL(1) \not\subset LALR(1)$$

Dla każdego DCFL istnieje gramatyka $SLR(1)$. [Mickunas, Lancaster, Schneider 1976]

Dla każdego k istnieje DCFL, dla którego nie ma gramatyki $LL(k)$.

Problemy

- 1 Czy na szczycie stosu jest prawa strona jakiejś produkcji?
(łatwe, ale może być więcej niż jedna)
- 2 Czy należy redukować, a jeśli tak, to którą produkcję?

Tworzymy deterministyczny automat ze stosem, symulujący prawostronne wyprowadzenie. Automat wykrywa uchwyty (produkcje wraz z miejscem wystąpienia).

Definicja (uchwyt)

$A \rightarrow \beta$ jest *uchwytem* w prawostronnej formie zdaniowej $\alpha\beta w$, jeśli

$$S \rightarrow^R \alpha A w \rightarrow^* \alpha \beta w$$

dla pewnych $\alpha, \beta \in (N \cup T)^*$, $w \in T^*$.

Jak rozpoznać uchwyt?

Zbudujemy automat skończony rozpoznający wiele wzorców
(możliwe prawe strony produkcji)

Sytuacja LR(0)

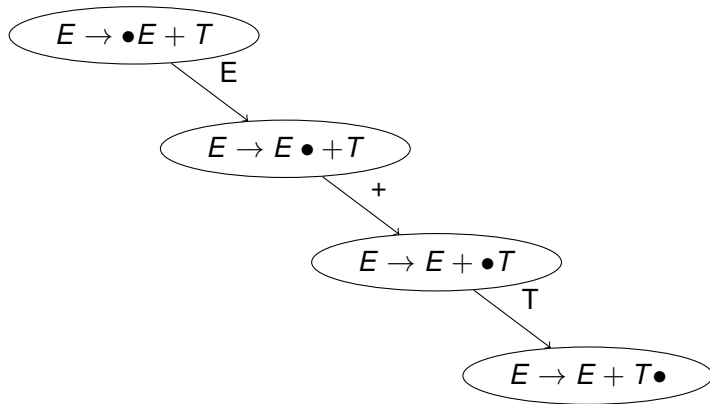
$$A \rightarrow \alpha \bullet \beta$$

czyli produkcja z wyróżnionym miejscem.

*Jesteśmy w trakcie rozpoznawania $A \rightarrow \alpha\beta$,
na stosie jest już α , trzeba jeszcze rozpoznać β*

Przykład (fragment automatu)

Jeśli mamy sytuację $A \rightarrow \alpha \bullet X \beta$, to po rozpoznaniu X mamy sytuację $A \rightarrow \alpha X \bullet \beta$



Stany i przejścia automatu LR

- Ponieważ trzeba równocześnie rozpoznawać wiele uchwytów, to stanami automatu będą **zbiory sytuacji**.
- Jeśli jesteśmy w sytuacji $B \rightarrow \alpha \bullet A\beta$, to jesteśmy też w sytuacji $A \rightarrow \bullet \gamma$ dla każdego $A \rightarrow \gamma \in P$
- Zbiór sytuacji musi być domknięty zwn tę implikacją:
- $Closure(Q)$ – najmniejszy zbiór zawierający Q oraz taki, że jeśli $B \rightarrow \alpha \bullet A\beta \in Closure(Q)$, to

$$\forall A \rightarrow \gamma \in P \quad A \rightarrow \bullet \gamma \in Closure(Q)$$

- Jeśli $A \rightarrow \alpha \bullet X\gamma \in Q$ dla pewnego $X \in N \cup T$, to ze stanu Q jest przejście (po X) do stanu $Closure(A \rightarrow \alpha X \bullet \gamma)$

Przykład

$S \rightarrow E, E \rightarrow E + T \mid E - T \mid T, T \rightarrow T * F \mid F, F \rightarrow n$

Domknięciem stanu $S \rightarrow \bullet E$ jest stan zawierający również

Przykład

$S \rightarrow E, E \rightarrow E + T \mid E - T \mid T, T \rightarrow T * F \mid F, F \rightarrow n$

Domknięciem stanu $S \rightarrow \bullet E$ jest stan zawierający również

$$E \rightarrow \bullet E + T, E \rightarrow \bullet E - T, E \rightarrow \bullet T$$

Przykład

$S \rightarrow E, E \rightarrow E + T \mid E - T \mid T, T \rightarrow T * F \mid F, F \rightarrow n$

Domknięciem stanu $S \rightarrow \bullet E$ jest stan zawierający również

$E \rightarrow \bullet E + T, E \rightarrow \bullet E - T, E \rightarrow \bullet T$

$T \rightarrow \bullet T * F, T \rightarrow \bullet F$

Przykład

$S \rightarrow E, E \rightarrow E + T \mid E - T \mid T, T \rightarrow T * F \mid F, F \rightarrow n$

Domknięciem stanu $S \rightarrow \bullet E$ jest stan zawierający również

$E \rightarrow \bullet E + T, E \rightarrow \bullet E - T, E \rightarrow \bullet T$

$T \rightarrow \bullet T * F, T \rightarrow \bullet F$

$F \rightarrow \bullet n$

Przykład

$S \rightarrow E, E \rightarrow E + T \mid E - T \mid T, T \rightarrow T * F \mid F, F \rightarrow n$

Domknięciem stanu $S \rightarrow \bullet E$ jest stan zawierający również

$$E \rightarrow \bullet E + T, E \rightarrow \bullet E - T, E \rightarrow \bullet T$$

$$T \rightarrow \bullet T * F, T \rightarrow \bullet F$$

$$F \rightarrow \bullet n$$

Z tego stanu po rozpoznaniu E przejdziemy do stanu zawierającego domknięcie zbioru

$$S \rightarrow E \bullet, E \rightarrow E \bullet + T, E \rightarrow E \bullet - T$$

(ale już bez $E \rightarrow \bullet T$).

Działanie automatu LR

- Dwie tablice indeksowane stanami i symbolami: ACTION (dla terminali) i GOTO (dla nieterminali)
- Stos zawiera stany przetykane symbolami gramatyki
- Automat startuje ze stosem zawierającym symbol początkowy
- Niech na stosie stan s , na wejściu terminal a :
 - ACTION[s, a] = **shift** p
przenosi a z wejścia na stos i nakrywa stanem p
 - ACTION[s, a] = **reduce**($A \rightarrow \alpha$)
zdejmuje $|\alpha|$ par ze stosu
odsłoni się stan q (zawierał sytuację $\dots \bullet A \dots$)
wkłada na stos A , GOTO[q, A].
 - Specjalne akcje: **error**, **accept**

Konstrukcja automatu LR

- 1 Rozszerzamy gramatykę o produkcję $Z \rightarrow S\#$
(nowy symbol początkowy)
- 2 Budujemy automat skończony:
 - stanami są zbiory sytuacji
 - stan początkowy: $Closure(\{Z \rightarrow \bullet S\#\})$
 - dla stanu p przejście po symbolu X do stanu

$$\delta(p, X) = Closure(\{A \rightarrow \alpha X \bullet \gamma : A \rightarrow \alpha \bullet X \gamma \in p\})$$

- stanem akceptującym jest $\{Z \rightarrow S\# \bullet\}$
- 3 Wypełniamy tablicę sterującą automatu ze stosem.

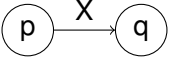
Przykład na tablicy (NB na razie bez redukcji)

$$L \rightarrow L; s \mid \epsilon \quad S \rightarrow a \mid \epsilon$$

Wypełnianie tablic sterujących

Numerujemy stany, numerujemy produkcje.

Jednolicie dla wszystkich klas automatów wpisujemy akcje **shift** (przepisujemy przejścia automatu skończonego) i **accept**:

Dla przejścia  wpisujemy:

- jeśli X jest *terminalem* to

$$\text{ACTION}[p, x] = \mathbf{shift\ q}$$

- jeśli X jest *nieterminalem* to

$$\text{GOTO}[p, x] = \mathbf{q}$$

- Jeśli stan p zawiera $S' \rightarrow S \bullet \#$, to $\text{ACTION}[p, \#] = \mathbf{accept}$

Redukcje

Tu postępujemy różnie dla różnych klas automatów.

Jeśli stan p zawiera $A \rightarrow \alpha \bullet$, to:

LR(0) wpisujemy **reduce**($A \rightarrow \alpha$) do ACTION[p, a]
dla wszystkich a

SLR(1) wpisujemy **reduce**($A \rightarrow \alpha$) do ACTION[p, a]
dla $a \in \text{FOLLOW}(A)$

Miejsca nie wypełnione oznaczają **error**

Jeśli gdzieś zostanie wpisana więcej niż jedna akcja, to źle:
gramatyka nie jest odpowiedniej klasy (konflikt shift-reduce lub
reduce-reduce).

Przykład na tablicy

Gdy SLR(1) zawodzi...

Rozważmy gramatykę

$$S \rightarrow L = R \mid R$$

$$L \rightarrow * R \mid \mathbf{a}$$

$$R \rightarrow L$$

Próba budowy automatu SLR(1) doprowadzi nas do stanu

$$S \rightarrow L\bullet = R$$

$$R \rightarrow L\bullet$$

W którym dla $=$ na wejściu możliwe jest zarówno przesunięcie jak i redukcja (do R). Okazuje się przy tym, że $= \in \text{FOLLOW}(R)$ i konfliktu nie da się usunąć metodą SLR(1).

Potrzebujemy silniejszego narzędzia.

Sytuacje LR(1)

Sytuacja LR(1)

$$[A \rightarrow \alpha \bullet \beta, a]$$

czyli para zawierająca sytuację LR(0) i terminal.

Jesteśmy w trakcie rozpoznawania $A \rightarrow \alpha\beta$,

na stosie jest już α , trzeba jeszcze rozpoznać β .

Sytuacja $[A \rightarrow \alpha \bullet, a]$ oznacza, że na stosie mamy całą prawą stronę produkcji; możemy zredukować gdy na wejściu jest a .

Notacja

$$[A \rightarrow \alpha \bullet X\beta, a, b, \dots]$$

oznacza zbiór sytuacji

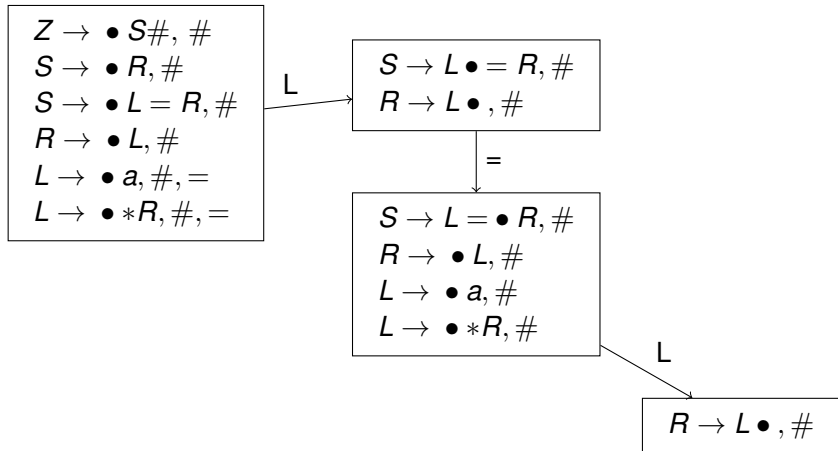
$$\{[A \rightarrow \alpha \bullet X\beta, a], [A \rightarrow \alpha \bullet X\beta, b], \dots\}$$

Ponadto, jeśli nie powoduje to niejasności, opuszczamy nawiasy i piszemy

$$A \rightarrow \alpha \bullet X\beta, a, b, \dots$$

Przykład (fragment automatu)

Jeśli mamy sytuację $[A \rightarrow \alpha \bullet X\beta, a]$, to po rozpoznaniu X mamy sytuację $[A \rightarrow \alpha X \bullet \beta, a]$



Stany i przejścia automatu LR(1)

- Stanami automatu są zbiory sytuacji LR(1).
- Jeśli jesteśmy w sytuacji $[B \rightarrow \alpha \bullet A\beta, a]$, to w wyprowadzeniu po A może wystąpić symbol z $\text{FIRST}(\beta a)$. Jesteśmy zatem też w sytuacji $[A \rightarrow \bullet \gamma, b]$ dla każdego $A \rightarrow \gamma \in P$ oraz $b \in \text{FIRST}(\beta a)$.
- Stan musi być domknięty zwn tę implikację:
- $\text{Closure}(Q)$ – najmniejszy zbiór zawierający Q oraz taki, że jeśli $[B \rightarrow \alpha \bullet A\beta, a] \in \text{Closure}(Q)$, to

$$\forall A \rightarrow \gamma \in P, b \in \text{FIRST}(\beta a) \quad [A \rightarrow \bullet \gamma, b] \in \text{Closure}(Q)$$

- Jeśli $[A \rightarrow \alpha \bullet X\gamma, a] \in Q$ dla pewnego $X \in N \cup T$, to ze stanu Q jest przejście (po X) do stanu $\text{Closure}(\{[A \rightarrow \alpha X \bullet \gamma, a]\})$.

Redukcje LR(1)

Jeśli stan p zawiera $[A \rightarrow \alpha \bullet, a]$, to: wpisujemy **reduce**($A \rightarrow \alpha$) do ACTION[p, a]

Jeśli gdzieś zostanie wpisana więcej niż jedna akcja, to źle: gramatyka nie jest klasy LR(1) (konflikt shift-reduce lub reduce-reduce).

W naszym przykładzie, w stanie

$$\begin{array}{l} S \rightarrow L \bullet = R, \# \\ R \rightarrow L \bullet, \# \end{array}$$

wpiszemy redukcję dla $\#$ a shift dla $=$. Nie ma konfliktu.

Usprawnianie metody LR(1)

W automacie LR(1) istnieje zwykle wiele podobnych stanów, np

$$[R \rightarrow L \bullet, \#, =]$$

oraz

$$[R \rightarrow L \bullet, \#]$$

Często możemy zmniejszyć automat, sklejając podobne stany.

Definicja

Jądro zbioru sytuacji LR(1) to następujący zbiór sytuacji LR(0):

$$\text{kernel}(p) = \{A \rightarrow \alpha \bullet \beta : \exists a \in T. [A \rightarrow \alpha \bullet \beta, a] \in p\}$$

Konstrukcja automatu LALR(1)

- Budujemy automat ze zbiorów sytuacji LR(1).
- Sklejamy równoważne stany (sumujemy stany mające identyczne jądra).
- Dalej postępujemy jak w metodzie LR(1).
- Jeśli nie powstaną nowe konflikty, to gramatyka jest LALR(1).

Zauważmy, że:

- Względem LR(1) mogą powstać tylko konflikty reduce-reduce, bo gdyby był konflikt shift-reduce, to istniałby i przy metodzie LR(1).
- automat LALR(1) ma tyle samo stanów co w metodzie LR(0)

Przykład na tablicy

Dalsze usprawnienia

- Wchodzenie do stanu, w którym jest tylko jedna sytuacja typu $A \rightarrow \alpha$ • nie ma sensu, bo tam zawsze redukujemy.
- Wprowadzamy nową akcję *shift-reduce j*: połączenie shift i reduce j .
- Usuwanie takich stanów — zysk rzędu 30%
- W tablicy mogą występować miejsca nieosiągalne — nieistotne.
- Informację o błędach można przesunąć do osobnej tablicy *Err* — bitowej.
- Teraz możemy skleić wiersze różniące się tylko na miejscach pustych i nieistotnych.

Generatory parserów

- Ręczne tworzenie automatu LR jest w praktyce zbyt żmudne
- Może to z powodzeniem wykonać specjalny program
- Wejście: gramatyka translacyjna (gramatyka + akcje)
- Wyjście: analizator składniowy LR w języku docelowym
- Istnieją dla wielu języków: C,C++ (Yacc,Bison), Java (Cup), Haskell (Happy), Ocaml (Ocamlyacc), ...

Bison

Generator parserów dla C/C++

Format wejścia:

deklaracje

```
%%
```

gramatyka: produkcje z akcjami postaci

```
a: b c { $$ .x = f($1.y, $2.z); }
```

```
| d { $$ .x = $1.x; }
```

```
;
```

```
%%
```

dodatkowy kod (w C/C++)

Wyjście: funkcja `int yyparse()` — w wyniku 0 jeśli sukces.

Wywołuje `yylex()` dla pobierania kolejnych leksemów.

Przykład — kalkulator dla ONP

```
%token NUM
%%
input:      /* empty */ | input line ;

line:      '\n' | exp '\n' { printf("%d\n", $1); } ;

exp:       NUM                { $$ = $1;          }
          | exp exp '+'      { $$ = $1 + $2;    }
          | exp exp '-'      { $$ = $1 - $2;    }
          | exp exp '*'      { $$ = $1 * $2;    }
          | exp exp '/'      { $$ = $1 / $2;    }

%%
```

Happy

Generator parserów dla Haskell

```
%name calc
%tokentype { Token }
%token
    int                { TokenInt $$ }
    '+'                { TokenPlus }
    '*'                { TokenTimes }
%%
Exp
: int                { EInt $1 }
| Exp Exp '+'        { EAdd $1 $2 }
| Exp Exp '*'        { EMul $1 $2 }
```


Zależności między klasami gramatyk

Pomiędzy klasami gramatyk zachodzą inkluzje

$$LR(0) \subset SLR(1) \subset LALR(1) \subset LR(1)$$

Wszystkie powyższe inkluzje są ostre.

Ponadto

$$LL(1) \subset LR(1)$$

Uwaga: istnieją gramatyki $LL(1)$, które nie są $LALR(1)$, czyli

$$LL(1) \not\subset LALR(1)$$

Dla każdego DCFL istnieje gramatyka $SLR(1)$. [Mickunas, Lancaster, Schneider 1976]

Dla każdego k istnieje DCFL, dla którego nie ma gramatyki $LL(k)$.

Gramatyki niejednoznaczne

Rozważmy następującą (niejednoznaczną) gramatykę:

$$E \rightarrow E + E \mid E * E \mid a$$

Budując dla niej automat (np. LR(0)) natkniemy się na stan:

$$E \rightarrow E + E \bullet$$

$$E \rightarrow E \bullet + E$$

$$E \rightarrow E \bullet * E$$

Mamy tu konflikty shift-reduce dla + i * na wejściu.

Jeżeli wybierzemy: dla + reduce, a dla * shift (i podobnie dla drugiego stanu z konfliktami) — uzyskamy tradycyjne priorytety operatorów.

Jak to sformalizować?

- Niektórym terminalom i produkcjom przypisujemy *priorytety*.
- Domyślnie produkcja otrzymuje priorytet ostatniego terminala.
- W sytuacji konfliktu: shift terminala kontra redukcja produkcji — wybieramy wyższy priorytet.
- Przy równych priorytetach wybieramy w/g kierunku wiązania operatora.

Przykład w Bisonie

```
%token NUM
%left '-' '+'
%left '*' '/'
%left NEG      /* leksem-widmo: minus unarny */
%right '^'     /* potęgowanie          */
%%
exp:          NUM          { $$ = $1;          }
    | exp '+' exp        { $$ = $1 + $3;      }
    | exp '-' exp        { $$ = $1 - $3;      }
    | exp '*' exp        { $$ = $1 * $3;      }
    | exp '/' exp        { $$ = $1 / $3;      }
    | '-' exp %prec NEG { $$ = -$2;          }
    | exp '^' exp        { $$ = pow ($1, $3); }
    | '(' exp ')'        { $$ = $2;          }

;
%%
```

Niejednoznaczność **if-then-else**

Rozważmy gramatykę

$$\begin{aligned} stmt &\rightarrow \mathbf{if\ expr\ then\ stmt\ else\ stmt} \\ &| \mathbf{if\ expr\ then\ stmt} \\ &| \mathit{other} \end{aligned}$$

Jest ona niejednoznaczna; próba budowy parsera prowadzi do konfliktu przesunięcie/redukcja.

Wybór przesunięcia odpowiada konwencji, że **else** łączy się z najbliższym **then**.

To jest standardowa metoda rozwiązania konfliktu.

Generalized LR

Metoda oryginalnie wymyślona przez Tomitę dla analizy języków naturalnych.

- Budowa automatu LR dla gramatyki niejednoznacznej prowadzi do konfliktów (kilka możliwych akcji w jednej sytuacji).
- Każdy element tablicy automatu GLR może zawierać zbiór akcji.
- Jeśli w danym momencie mamy zbiór akcji (> 1), automat *rozmnaża* się na odpowiednią liczbę kopii.
- Przy napotkaniu błędu kopia ginie
- Efekt: zbiór możliwych rozbiórów danego tekstu.
- Niektóre generatory (Bison, Happy) potrafią generować parsery GLR.