

Stefan Sokołowski

JĘZYKI FORMALNE I METODY KOMPILACJI

wykład 15 — powtórka

Inst. Informatyki Stosowanej, ANS
Elbląg, 2025/2026

Wykład 15 — powtórka, str. 1

Jak **porządnie** opisać język nieskończony?

- $\{\lambda, ab, aabb, aaabbb, \dots\}$ — co oznaczają kropeczki?
- $\{a^n b^n \mid n \geq 0\}$ — lepiej, ale są niepożądane elementy obce (liczby, zmienne, relacje, ...)

Język definiujemy na dwa sposoby. Przez

- **generator** — cokolwiek **wyprodukuje**, uważamy za należące do języka
- **akceptor** — cokolwiek **zatwierdzi**, uważamy za należące do języka

Opis języka programowania przeznaczony dla programisty powinien być **generatorem** — ma wyjaśniać, jak pisać poprawne programy.

Kompilator języka programowania powinien zawierać **akceptor** — sprawdzający, czy program na jego wejściu należy do języka.

Generatory: gramatyki bezkontekstowe

DEFINICJA: Gramatyka bezkontekstowa G — czwórka $\langle \Sigma, N, P, S \rangle$

- alfabet *terminalny* Σ ;
- alfabet *nieterminalny* lub *pomocniczy* N (rozłączny z Σ);
- skończony zbiór P *produkcji* postaci $A \rightarrow w$, gdzie
 $A \in N$ (czyli jest pojedynczym nieterminaliem),
 $w \in (N \cup \Sigma)^*$ (czyli jest słowem z terminali i nieterminali);
- wybrany nieterminal $S \in N$ nazywany *nieterminaliem początkowym* albo *aksjomatem* gramatyki.

PRZYKŁAD:

Terminale: a, b

Nieterminale: X

Aksjomat: X

Produkcje:

$X \rightarrow \lambda$
 $X \rightarrow aXa$
 $X \rightarrow bXb$

Jak gramatyka definiuje język?

DEFINICJA: Wywodliwość —

$uXv \Rightarrow_G uvw$ czyli uwv jest *bezpośrednio wywodliwe* z uXv $\iff$ w P jest produkcja $X \rightarrow w$

$w_1 \Rightarrow_G^* w_n$ czyli w_n jest *wywodliwe* z w_1 $\iff$ $w_1 \Rightarrow_G w_2 \Rightarrow_G w_3 \Rightarrow_G \dots \Rightarrow_G w_{n-1} \Rightarrow_G w_n$

DEFINICJA: Język $L(G)$ *generowany przez gramatykę* G —

składa się z tych słów wywodliwych z **aksjomatu**, które już nie zawierają nieterminali: $L(G) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$.

PRZYKŁAD:

Gramatyka:

$X \rightarrow \lambda$
 $X \rightarrow aXa$
 $X \rightarrow bXb$

$X \Rightarrow_G^* bbaabb \in L(G)$

LEMAT:

$L(G) = \{ww^R \mid w \in \{a, b\}^*\}$

$X \Rightarrow bXb \Rightarrow bbXbb \Rightarrow bbaXabb \Rightarrow bbaabb$

Powtarzanie, a rekursja w produkcjach

Gramatyka dla $\{a^n \mid n \geq 0\}$?

$$\begin{cases} A \rightarrow \lambda \\ A \rightarrow \underline{A}a \end{cases} \text{rekursja lewostronna}$$

$$A \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow \dots \Rightarrow aaaa \dots a$$

albo

$$\begin{cases} A \rightarrow \lambda \\ A \rightarrow a\underline{A} \end{cases} \text{rekursja prawostronna}$$

$$A \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow \dots \Rightarrow a \dots aaa$$

Gramatyki prawoliniowe

DEFINICJA:

Gramatyka bezkontekstowa $G = \langle \Sigma, N, P, S \rangle$ jest *prawo-
wostronnie liniowa* jeśli każda produkcja w P ma po-
stać:

- albo $A \rightarrow x$ (czyli: żadnych nieterminali)
- albo $A \rightarrow xB$ (czyli: tylko jeden nieterminal po prawej)

gdzie $x \in \Sigma^*$ a $A, B \in N$.

praw. liniowe

bezkontekstowe

PRZYKŁAD:

$$\begin{aligned} E &\rightarrow aC \\ E &\rightarrow bC \\ C &\rightarrow \lambda \\ C &\rightarrow aC \\ C &\rightarrow bC \\ C &\rightarrow 0C \\ C &\rightarrow 1C \end{aligned}$$

prawoliniowa

Więc w prawoliniowych rekursja
może być tylko prawostronna.

$$\begin{aligned} E &\rightarrow a \\ E &\rightarrow (E) \\ E &\rightarrow E + E \end{aligned}$$

bezkontekstowa
ale nie prawoliniowa

Rodzaje gramatyk

PRZYKŁAD:

G_1 :

$E \rightarrow aC$
$E \rightarrow bC$
$C \rightarrow \lambda$
$C \rightarrow aC$
$C \rightarrow bC$
$C \rightarrow 0C$
$C \rightarrow 1C$

G_2 :

$I \rightarrow a$
$I \rightarrow b$
$I \rightarrow Ia$
$I \rightarrow Ib$
$I \rightarrow I0$
$I \rightarrow I1$

$$L(G_1) = \text{język identyfikatorów} = L(G_2)$$

G_1 jest prawostronnie liniowa
 G_2 jest bezkontekstowa
 ale nie prawostronnie liniowa

} dwie gramatyki **różnych typów** mogą określać ten sam język

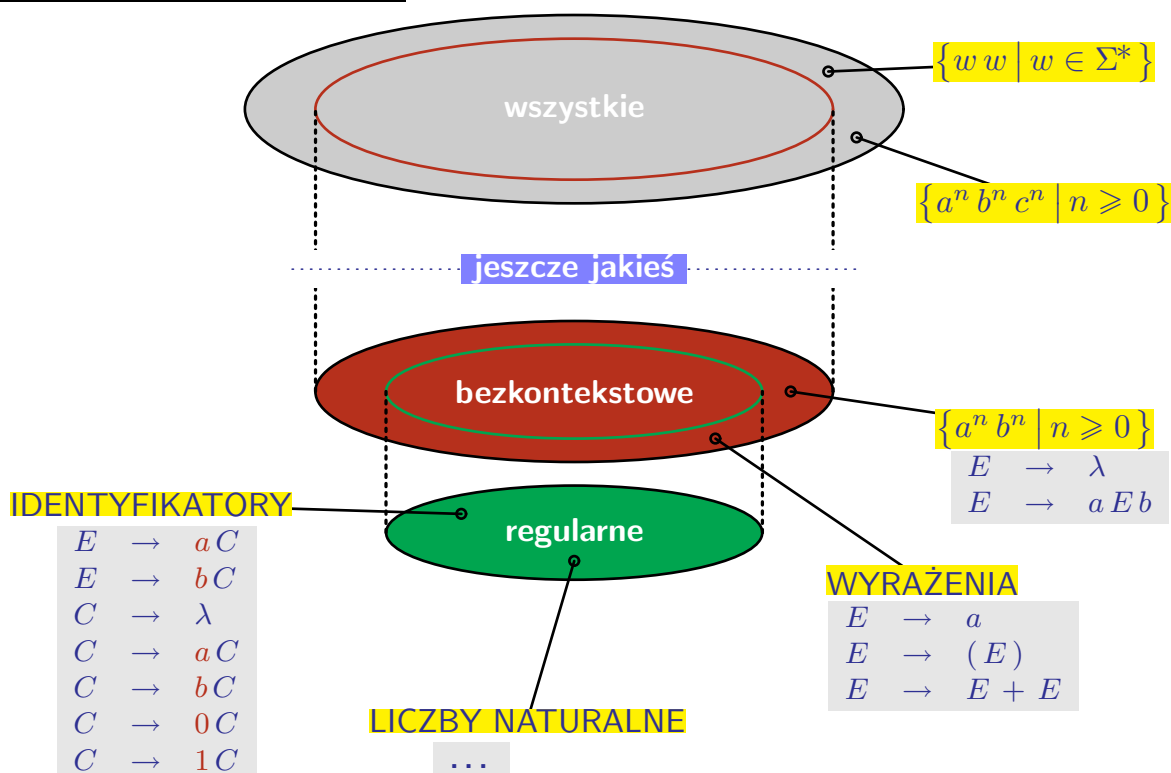
DEFINICJA:

Język L jest

- **regularny** jeśli ist. gram. **prawostronnie liniowa** G taka, że $L = L(G)$;
- **bezkontekstowy** jeśli ist. gram. **bezkontekstowa** G taka, że $L = L(G)$.

PRZYKŁAD: Język identyfikatorów $L(G_2)$ jest regularny.

Hierarchia języków



Czy dany język jest bezkontekstowy lub regularny?

Żeby dowieść, że język **jest** bezkontekstowy [ew. regularny], należy znaleźć dla niego gramatykę bezkontekstową [prawoliniową].

Żeby dowieść, że język **nie jest** bezkontekstowy [regularny], należy udowodnić, że nie posiada gramatyki bezkontekstowej [prawoliniowej] — to jest **na ogół trudne**.

Bardzo nieformalne kryteria negatywne:

1. Rozpoznanie słowa języka **regularnego** nie wymaga porównywania liczby liter.
Np. język $\{a^n b^n \mid n \geq 0\}$ nie jest regularny.
2. Rozpoznanie słowa języka **bezkontekstowego** może wymagać porównania liczby liter ale tylko w dwóch blokach.
Np. $\{a^n b^n \mid n \geq 0\}$ jest a $\{a^n b^n a^n \mid n \geq 0\}$ nie jest bezkontekstowy.
3. Rozpoznanie słowa języka **bezkontekstowego** może wymagać **porównania** liczby liter j.w., ale nie może wymagać **obliczeń** arytmetycznych.
Np. $\{a^n b^{n^2} \mid n \geq 0\}$ nie jest bezkontekstowy.
4. Rozpoznanie słowa języka **bezkontekstowego** nie może wymagać pamiętania słów bez ograniczenia długości.
Np. $\{ww^R \mid w \in \Sigma^*\}$ jest a $\{ww \mid w \in \Sigma^*\}$ nie jest bezkontekstowy.

Definiowanie języka przez wyrażenie regularne (WR)

DEFINICJA: (wyrażenia regularne)

(specjalne znaki nie należące do alfabetu: $\{, \}, |, *$)

- **litery** alfabetu są WR
- **złożenie (konkatenacja)** WR jest WR:
jeśli W_1 i W_2 są WR, to $\{W_1 W_2\}$ jest WR
- **suma** WR jest WR:
jeśli W_1 i W_2 są WR, to $\{W_1 \mid W_2\}$ jest WR
- **domk. Kleenego (iteracja)** WR jest WR:
jeśli W jest WR, to $\{W\}^*$ jest WR

Umowa notacyjna: opuszczamy zbędne klamry, przyjmując, że

- najsilniej łączy $*$ a najsłabiej $|$,
- złożenie i suma łączą do lewej.

Np. zamiast

$$\left\{ \left\{ \left\{ \{ab\} a \right\} \mid \{a \{c\}^* \right\} \right\} \mid \{cb\} \right\}$$

można napisać

$$\underline{\underline{\underline{aba \mid ac^* \mid cb}}}$$

Definiowanie języka przez wyrażenie regularne (WR)

Każdemu wyrażeniu regularnemu **odpowiada** język:

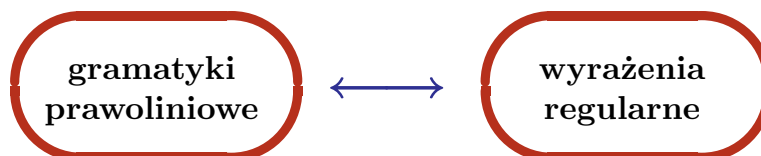
- pojedynczej literze σ **odpowiada** jednowyrazowy język $\{\sigma\}$
- jeśli wyrażeniu W_1 **odpowiada** język L_1
a wyrażeniu W_2 **odpowiada** język L_2 ,
to wyrażeniu $\{W_1 W_2\}$ **odpowiada** język złożenia (konkatenacji)
$$\{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$$
- jeśli wyrażeniu W_1 **odpowiada** język L_1
a wyrażeniu W_2 **odpowiada** język L_2 ,
to wyrażeniu $\{W_1 \mid W_2\}$ **odpowiada** suma języków
$$L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$$
- jeśli wyrażeniu W **odpowiada** język L ,
to wyrażeniu $\{W\}^*$ **odpowiada** nieskończona suma języków
$$\{\lambda\} \cup L \cup LL \cup LLL \cup LLLL \cup \dots$$

Wyrażenia regularne a języki regularne

TWIERDZENIE:

Każdy **język regularny** odpowiada jakiemuś **wyrażeniu regularnemu**.

Każdy język, który **odpowiada** jakiemuś **wyrażeniu regularnemu**, jest **regularny**.



Czyli:

Języki regularne można definiować

- albo przy pomocy **gramatyk prawoliniowych**,
- albo przy pomocy **wyrażeń regularnych**

— to wychodzi na jedno.

Wyrażenia regularne a języki regularne

PRZYKŁAD:

Gramatyka prawoliniowa:

$$\begin{aligned} S &\rightarrow \lambda \\ S &\rightarrow abS \\ S &\rightarrow cS \end{aligned}$$


Wyrażenie regularne:

$$\{ab \mid c\}^*$$

PRZYKŁAD:

Wyrażenie regularne:

$$a^*c^* \mid ba$$


Gramatyka prawoliniowa:

$$\begin{aligned} S &\rightarrow A \\ S &\rightarrow ba \\ A &\rightarrow C \\ A &\rightarrow aA \\ C &\rightarrow \lambda \\ C &\rightarrow cC \end{aligned}$$

Proste przejście między gramatyką regularną a wyrażeniem regularnym.

Umowa notacyjna dla gramatyk bezkontekstowych

- ponieważ **nieterminale** oznaczają pojęcia — sugestywne nazwy
np. „wyrażenie”, „liczba”
- nieterminale** **ujmujemy** w nawiasy kątowe: $\langle \dots \rangle$
np. $\langle \text{wyrażenie} \rangle$, $\langle \text{liczba} \rangle$
- wszystkie produkcje z pojedynczego nieterminalu **grupujemy**; np.:

piszemy	zamiast
$\langle \text{cyfra} \rangle ::= 0 \mid 1 \mid \dots \mid 9$	$\langle \text{cyfra} \rangle \rightarrow 0$ $\langle \text{cyfra} \rangle \rightarrow 1$ $\dots$ $\langle \text{cyfra} \rangle \rightarrow 9$
$\langle \text{liczba} \rangle ::= \langle \text{cyfra} \rangle \mid \langle \text{liczba} \rangle \langle \text{cyfra} \rangle$	$\langle \text{liczba} \rangle \rightarrow \langle \text{cyfra} \rangle$ $\langle \text{liczba} \rangle \rightarrow \langle \text{liczba} \rangle \langle \text{cyfra} \rangle$

Drzewa wyvodu

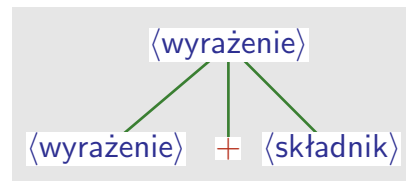
DEFINICJA:

Drzewo wyvodu słowa $w \in \Sigma^*$ w gramatyce bezkontekstowej $G = \langle \Sigma, N, P, \langle S \rangle \rangle$ to takie drzewo, które ma:

- w liściach symbole terminalne składające się na słowo w ,
- w pozostałych (wewnętrznych) węzłach nieterminale rozmieszczone wg takich zasad:
 - jeśli w jakimś węźle znajduje się nieterminał $\langle A \rangle$ a w jego dzieciach od lewej do prawej symbole $a_1, a_2, \dots, a_n \in \Sigma \cup N$, to $\langle A \rangle \rightarrow a_1 a_2 \dots a_n$ musi być produkcją z P ,
 - w korzeniu znajduje się nieterminał początkowy $\langle S \rangle$.

PRZYKŁAD:

Fragment drzewa



jest OK jeśli

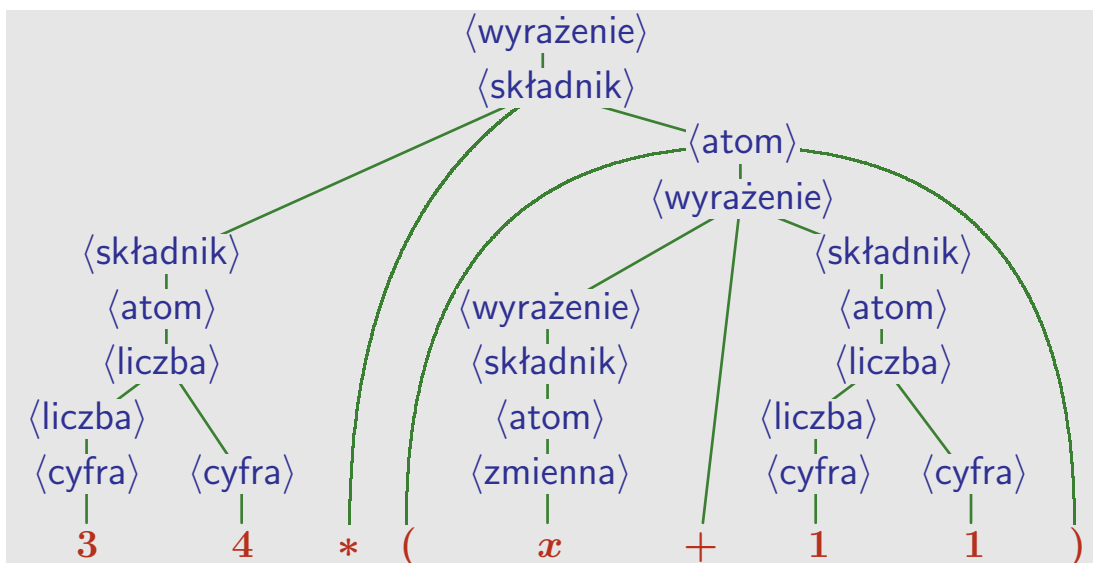
$\langle \text{wyrażenie} \rangle \rightarrow \langle \text{wyrażenie} \rangle + \langle \text{składnik} \rangle$ jest produkcją z gramatyki.

Drzewa wyvodu

ciąg terminali:

34 * (x + 11)

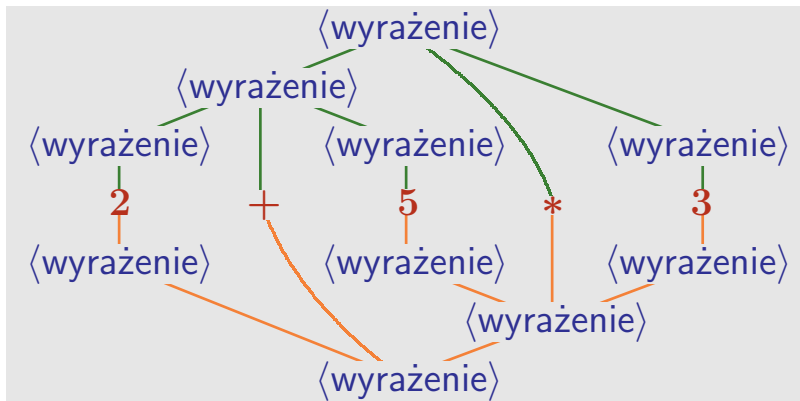
$\langle \text{zmienna} \rangle ::= a \mid b \mid \dots \mid z$
$\langle \text{cyfra} \rangle ::= 0 \mid 1 \mid \dots \mid 9$
$\langle \text{liczba} \rangle ::= \langle \text{liczba} \rangle \langle \text{cyfra} \rangle \mid \langle \text{cyfra} \rangle$
$\langle \text{atom} \rangle ::= \langle \text{zmienna} \rangle \mid \langle \text{liczba} \rangle \mid (\langle \text{wyrażenie} \rangle)$
$\langle \text{wyrażenie} \rangle ::= \langle \text{składnik} \rangle \mid \langle \text{wyrażenie} \rangle + \langle \text{składnik} \rangle$
$\langle \text{składnik} \rangle ::= \langle \text{atom} \rangle \mid \langle \text{składnik} \rangle * \langle \text{atom} \rangle$



Drzewo wyvodu a struktura słowa

$$\langle \text{wyrażenie} \rangle ::= 0 \mid 1 \mid \dots \mid 9 \mid$$

$$\langle \text{wyrażenie} \rangle + \langle \text{wyrażenie} \rangle \mid$$

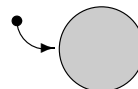
$$\langle \text{wyrażenie} \rangle * \langle \text{wyrażenie} \rangle$$


Niejednoznaczność wyvodu: istnieją dwa różne drzewa wyvodu danego słowa. W tym przypadku sugerują one kolejność działań:

- **drzewo górne** odpowiada wykonaniu dodawania przed mnożeniem
- **drzewo dolne** odpowiada wykonaniu mnożenia przed dodawaniem

Konstrukcja automatu skończonego

- dokładnie jeden stan *początkowy*



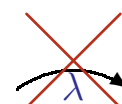
- dowolnie wiele stanów *końcowych* (*akceptujących*)



- między stanami *strzałki* (*tranzycje*)

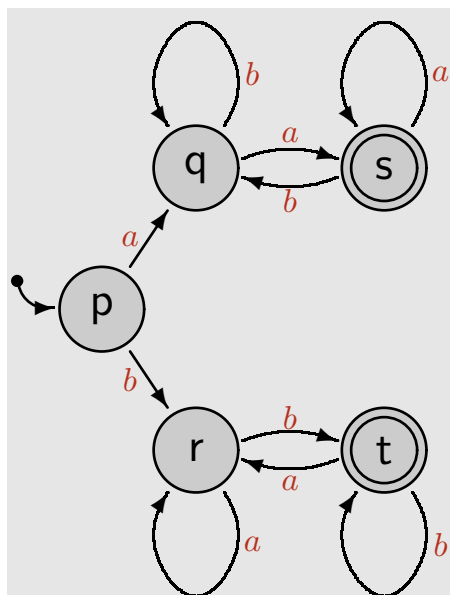


- na strzałkach stoją **pojedyncze** litery alfabetu



- z każdego stanu wychodzi po **jednej** strzałce dla **każdej** litery alfabetu

Akceptory: automaty skończone



Funkcja przejścia automatu:

δ	a	b
p	q	r
q	s	q
r	r	t
s	s	q
t	r	t

$$\begin{aligned}
 \delta^*\langle p, baab \rangle &= \delta^*\langle \delta\langle p, b \rangle, aab \rangle \\
 &= \delta^*\langle r, aab \rangle = \delta^*\langle \delta\langle r, a \rangle, ab \rangle \\
 &= \delta^*\langle r, ab \rangle = \delta^*\langle \delta\langle r, a \rangle, b \rangle \\
 &= \delta^*\langle r, b \rangle = \delta^*\langle \delta\langle r, b \rangle, \lambda \rangle \\
 &= \delta^*\langle t, \lambda \rangle = t
 \end{aligned}$$

Akceptacja słowa w : jeśli $\delta^*\langle p, w \rangle$ jest stanem akceptującym.

Automaty skończone

DEFINICJA: Deterministyczny automat skończony M —

- skończony zbiór Q *stanów*,
- skończony zbiór Σ *liter*,
- funkcja *przejścia* $\delta : Q \times \Sigma \rightarrow Q$,
- wyróżniony stan *początkowy* $q_0 \in Q$,
- wyróżniony zbiór stanów *końcowych* (*akceptujących*) $F \subseteq Q$.

DEFINICJA: Uogólniona funkcja przejścia — $\delta^* : Q \times \Sigma^* \rightarrow Q$:

$$\delta^*\langle q, w \rangle \stackrel{\text{def}}{=} \begin{cases} q & \text{jeśli } w = \lambda \\ \delta^*\langle \delta\langle q, \sigma \rangle, w' \rangle & \text{jeśli } w = \sigma w' \\ & \text{dla pewnych } \sigma \in \Sigma \text{ i } w' \in \Sigma^* \end{cases}$$

DEFINICJA: Automat M akceptuje słowo w jeśli $\delta^*\langle q_0, w \rangle \in F$

DEFINICJA: Język akceptowany przez automat $M = \langle Q, \Sigma, \delta, q_0, F \rangle$:

$$L(M) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid M \text{ akceptuje } w\} = \{w \in \Sigma^* \mid \delta^*\langle q_0, w \rangle \in F\}$$

Automaty skończone a gramatyki prawoliniowe

TWIERDZENIE:

Dla każdego automatu skończonego M język $L(M)$ jest *regularny*.

Dla każdego języka regularnego L istnieje automat skończony M taki że $L = L(M)$.

PRZYKŁAD:

gramatyka prawoliniowa	wyrażenie regularne	automat skończony
$\langle S \rangle ::= \lambda \mid abb \langle S \rangle$	$\{abb\}^*$	

Automaty skończone a gramatyki prawoliniowe

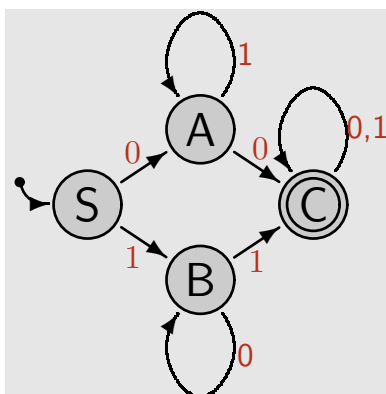
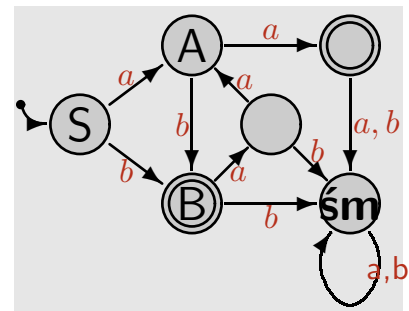
PRZYKŁAD:

Gramatyka prawoliniowa G :

$$\langle S \rangle ::= a \langle A \rangle \mid b \langle B \rangle$$

$$\langle A \rangle ::= a \mid b \langle B \rangle$$

$$\langle B \rangle ::= \lambda \mid aa \langle A \rangle$$



PRZYKŁAD:

Gramatyka prawoliniowa:

$$\langle S \rangle ::= 0 \langle A \rangle \mid 1 \langle B \rangle$$

$$\langle A \rangle ::= 0 \langle C \rangle \mid 1 \langle A \rangle$$

$$\langle B \rangle ::= 0 \langle B \rangle \mid 1 \langle C \rangle$$

$$\langle C \rangle ::= \lambda \mid 0 \langle C \rangle \mid 1 \langle C \rangle$$

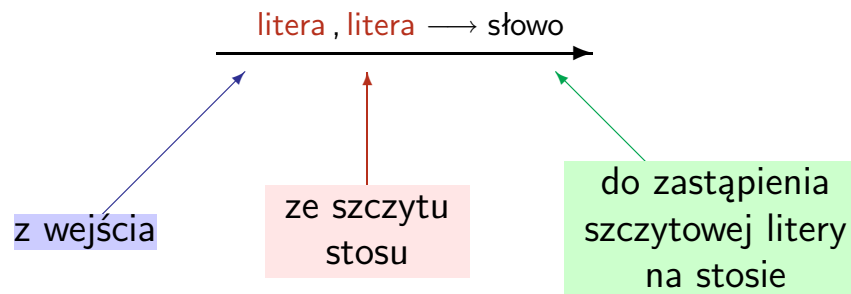
Maszyna ze stosem

— jakby automat, ale z **nieskończonym stosem**, na którym można przechowywać litery wg zasady

$$\left\{ \begin{array}{c} \text{Last} \\ \text{In} \\ \text{First} \\ \text{Out} \end{array} \right\} = \left\{ \begin{array}{c} \text{kto ostatni} \\ \text{przyszedł,} \\ \text{pierwszy} \\ \text{wychodzi} \end{array} \right\}$$

b
a
b
b
a
Z

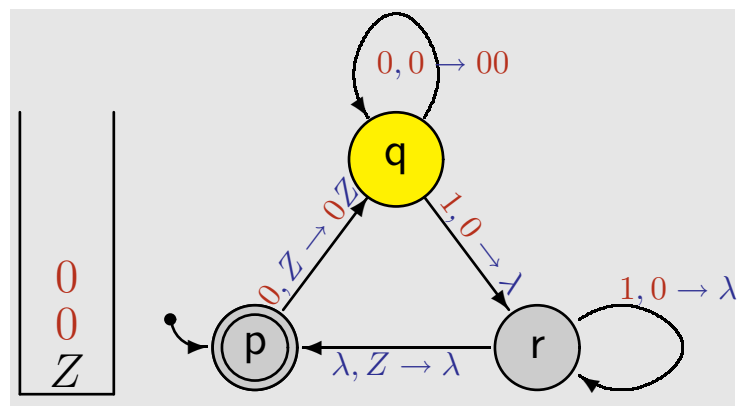
Strzałki mają postać:



Maszyna ze stosem

Słowo na wejściu:

0 0 1 1
 └─┬─┘
 część
nieprzeczytana



Kiedy wędrówka po maszynie się „zatnie”, czyli nie ma już żadnej możliwości pójścia dalej, sprawdzamy, czy słowo zostało **zaakceptowane**; t.zn., czy spełnione są następujące warunki:

- na **wejściu** jest słowo puste (wszystko już przeczytane);
- na **stosie** jest słowo puste;
- sprawdzanie zakończyło się w **stanie** końcowym (akceptującym).

Maszyna ze stosem

DEFINICJA: Maszyna ze stosem M —

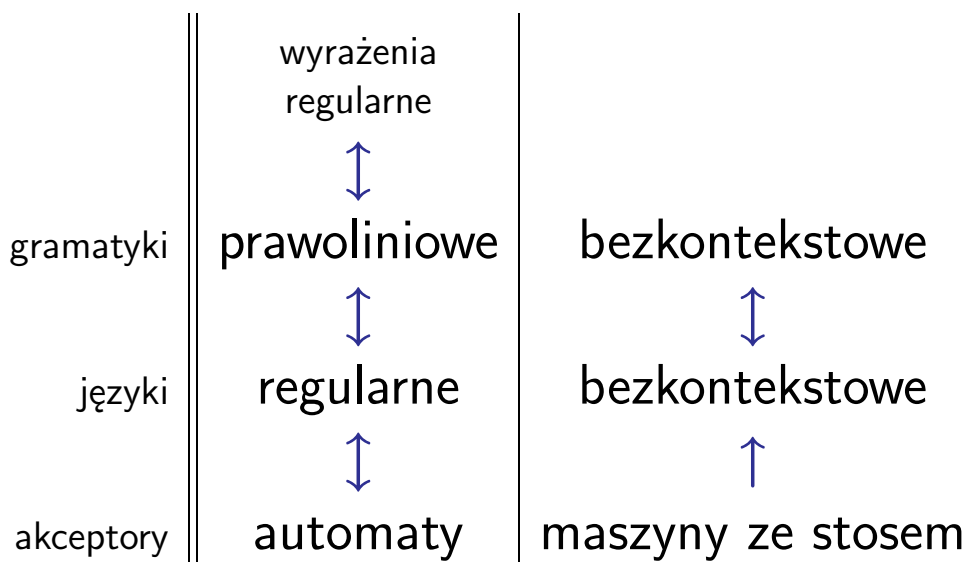
- skończony zbiór Q *stanów*,
- skończony zbiór Σ *liter wejściowych*,
- skończony zbiór Γ *symboli stosowych*,
- $Z \in \Gamma$ — wyróżniony symbol stosowy oznaczający dno stosu,
- częściowa funkcja *przejścia* $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightrightarrows Q \times \Gamma^*$
(częściowa — t.zn. nie musi być określona dla wszystkich argumentów),
- wyróżniony stan *początkowy* $q_0 \in Q$,
- wyróżniony zbiór stanów *końcowych* (*akceptujących*) $F \subseteq Q$.

DEFINICJA: Język akceptowany przez M —

$$L(M) \stackrel{\text{def}}{=} \{w \in \Sigma^* \mid \delta^* \langle q_0, w, Z \rangle = \langle q', \lambda, \lambda \rangle \text{ dla pewnego } q' \in F\}$$

gdzie δ^* jest uogólnioną funkcją przejścia.

Gramatyki, języki, maszyny



Działania na językach

— takie jak podstawowe konstrukcje wyrażeń regularnych:

1. Suma:

$$L \mid M \stackrel{\text{def}}{=} \{w \mid w \in L \text{ lub } w \in M\}$$

2. Złączenie:

$$LM \stackrel{\text{def}}{=} \{uv \mid u \in L \text{ i } v \in M\}$$

3. Iteracja (domknięcie Kleenego):

$$\begin{aligned} L^* &\stackrel{\text{def}}{=} \{u_1 u_2 \dots u_n \mid u_1, u_2, \dots, u_n \in L \text{ i } n \geq 0\} \\ &= \{\lambda\} \mid L \mid L^2 \mid L^3 \mid \dots \end{aligned}$$

Działania na językach

Własności działań:

- Łączność:

$$K \mid (L \mid M) = (K \mid L) \mid M \qquad K(LM) = (KL)M$$

- Przemienność:

$$L \mid M = M \mid L$$

Uwaga: złączenie nie jest przemienne!

- Elementy neutralne:

$$M \mid \emptyset = M \qquad \Lambda M = M \Lambda = M$$

$$\emptyset \stackrel{\text{def}}{=} \{\} \text{ — język pusty}$$

$$\Lambda \stackrel{\text{def}}{=} \{\lambda\} \text{ — język zawierający tylko słowo puste}$$

- Rozdzielność (wchodzenie pod nawias, wyłączanie przed/za nawias):

$$K(L \mid M) = KL \mid KM \qquad (L \mid M)K = LK \mid MK$$

Przekształcenia gramatyk

Gramatyka jako układ równań na językach

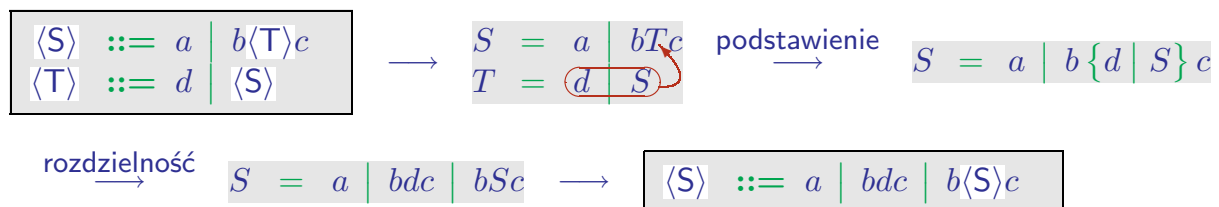
Interpretujemy:

- *nieterminale* jako języki złożone z tych słów, które dają się z nich wyprowadzić;
- *terminale* jako języki 1-literowe;
- *znak ::=* jako równość języków.

Wtedy gramatyka staje się *układem równań* na językach, przy czym nieterminale są jego *niewiadomymi*.

Przekształcanie gramatyk do postaci równoważnych odpowiada wykonywaniu działań na językach, określonych tymi układami.

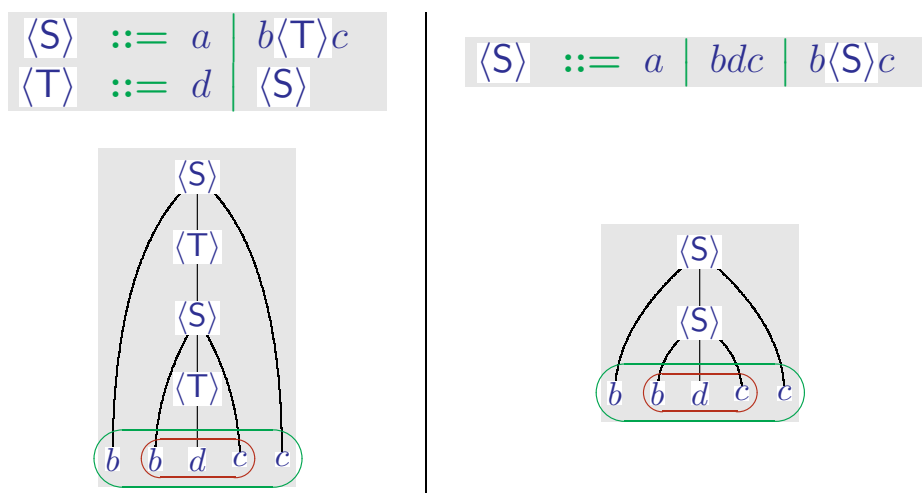
Przykład:



Przekształcenia gramatyk

Wpływ przekształceń na kształt drzewa wyvodu

Przykład: słowo *bbdcc*



Wniosek:

Drzewa są różne, ale ustalają takie same pierwszeństwa.

Dbajmy o drzewa!

$\langle X \rangle ::= L \mid \langle X \rangle M$	$\begin{cases} \langle X \rangle ::= L \langle Y \rangle \\ \langle Y \rangle ::= \lambda \mid M \langle Y \rangle \end{cases}$
$\langle X \rangle \Rightarrow \langle X \rangle M \Rightarrow \langle X \rangle MM \Rightarrow \langle X \rangle MMM \Rightarrow LMMM$	$\langle X \rangle \Rightarrow L \langle Y \rangle \Rightarrow LM \langle Y \rangle \Rightarrow LMM \langle Y \rangle \Rightarrow LMMM \langle Y \rangle \Rightarrow LMMM$

- Gramatyki **niebieska** i **czerwona** generują **te same języki**, ale drzewa wyvodu ich słów dają różne pierwszeństwa.
- Z punktu widzenia struktury logicznej słów, **nie można zastąpić** jednej przez drugą.
- Ostrzeżenie:** ChatGPT o tym nie wie.

Rekursja lewostronna

— kiedy jakaś produkcja z nieterminalu $\langle X \rangle$ zaczyna się od $\langle X \rangle$.

Niektóre rodzaje parserów nie radzą sobie z rekursją lewostronną (będzie o tym dalej). Rozwiązaniem jest przekształcenie gramatyki:

$$\langle X \rangle ::= L \mid \langle X \rangle M \longrightarrow \langle X \rangle ::= LM^*$$

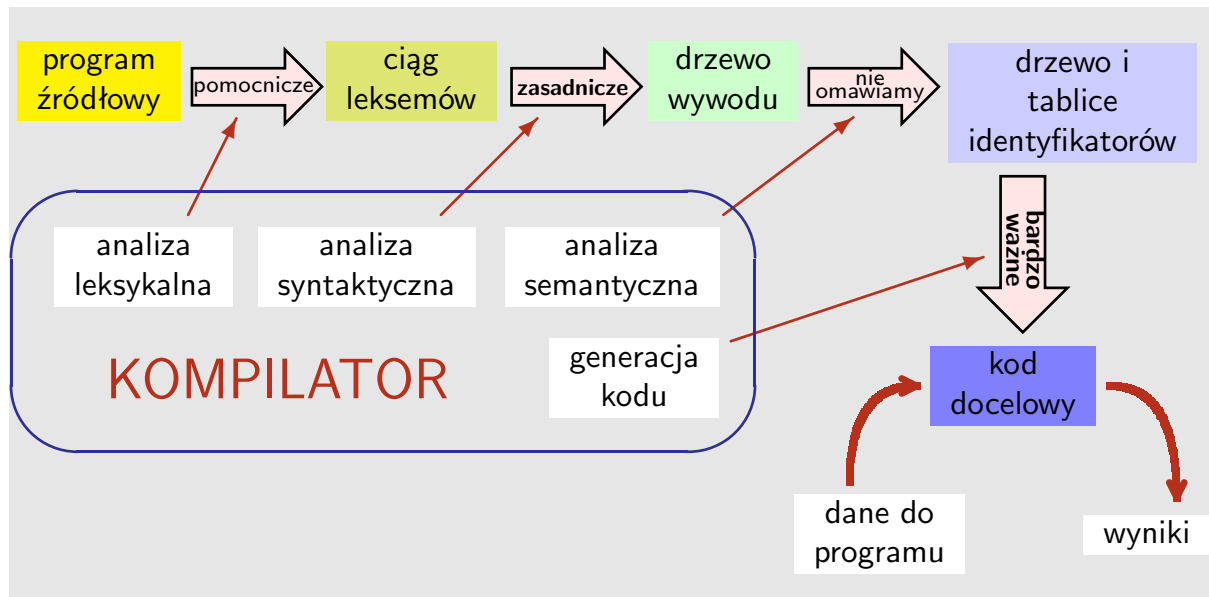
Czy język LM^* spełnia **niebieskie** równanie?

$$\begin{aligned} X &= LM^* = L \{ \Lambda \mid M \mid M^2 \mid M^3 \mid \dots \} \\ &= L \mid L \{ M \mid M^2 \mid M^3 \mid \dots \} = L \mid L \{ \Lambda \mid M \mid M^2 \mid \dots \} M \\ &= L \mid LM^*M = L \mid XM \end{aligned}$$

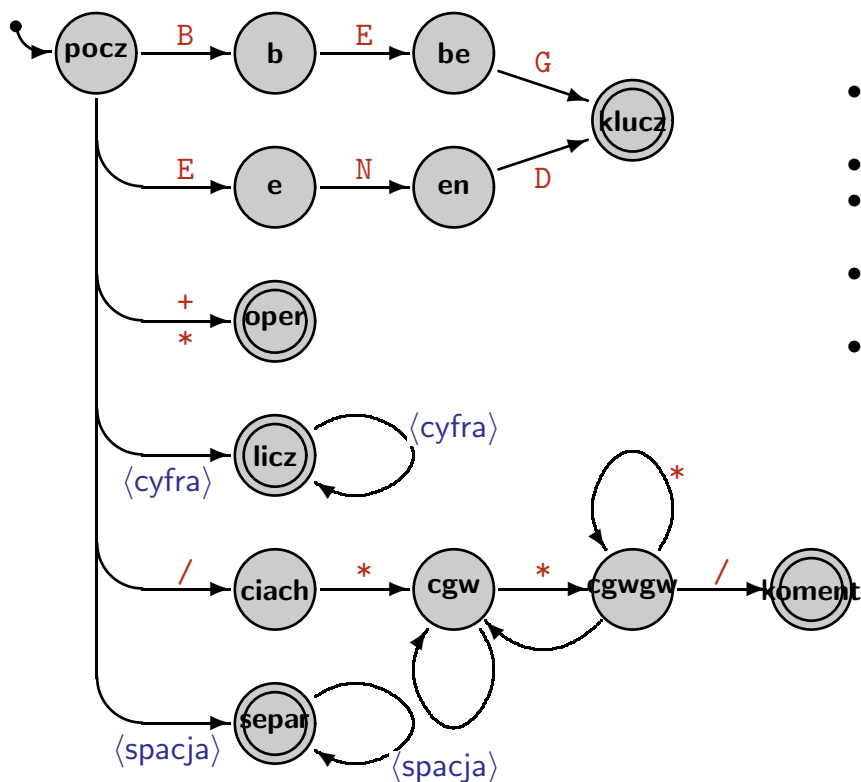
Konstruując parser należy więc

- zastąpić w gramatyce lewostronną rekursję „gwiazdkowaniem”,
- do fragmentów „gwiazdkowanych” budować drzewo **z łąčeniem do lewej**.

Uproszczony schemat kompilacji



Analizator leksykalny czyli skaner — przykład



- słowa **kluczowe**: **BEG**, **END**
(wielkość liter nieistotna)
- znaki **operacji**: **+**, *****
- **liczby** całkowite postaci
 $\langle \text{cyfra} \rangle \langle \text{cyfra} \rangle^*$
- **komentarze** postaci
 $\langle * \langle \text{cokolwiek} \rangle * \rangle$
- **separatory** postaci
 $\langle \text{spacja} \rangle \langle \text{spacja} \rangle^*$

Analizator leksykalny czyli skaner — przykład

Funkcja przejścia:

	b	e	g	n	d	pl	gw	cyf	ciach	sp	inny
pocz	b	e				oper	oper	licz	ciach	separ	
b		be									
be			klucz								
e				en							
en					klucz						
licz								licz			
ciach							cgw				
cgw	cgw	cgw	cgw	cgw	cgw	cgw	cgw	cgw	cgw	cgw	cgw
cgw	cgw	cgw	cgw	cgw	cgw	cgw	cgw	cgw	koment	cgw	cgw
separ										separ	

- to jest funkcja przejścia automatu skończonego stany $\times$ znaki $\rightarrow$ stany
- zbiór znaków jest rozszerzony o znak **inny** : „jakikolwiek inny”
- puste miejsca oraz brakujące wiersze oznaczają „stan śmietnikowy”

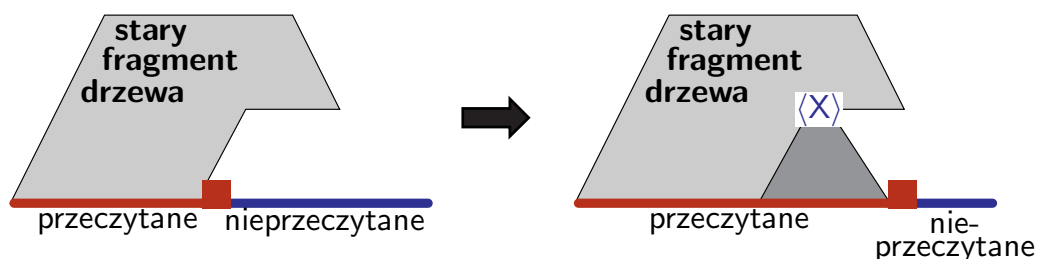
iis.ans-elblag.pl/~stefan/Dydaktyka/JezForm/Wyklady/06-leksyk.c

Analiza *top-down* — rekursywny parser zstępujący

(*recursive descent parser*)

Zawiera po jednej funkcji rekursywnej dla każdego nieterminala. Funkcja dla nieterminala $\langle X \rangle$

- konstruuje nad leksemami z wejścia drzewo wyvodu z nieterminala $\langle X \rangle$ w korzeniu:

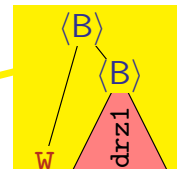


- przed — znany (wczytany) jest pierwszy liść (leksem) tego drzewa;
po — znany jest pierwszy leksem *spoza* korony tego drzewa;
- może zasygnalizować błąd i wtedy program ją wywołujący stara się znaleźć inną produkcję do zastosowania.

Rekursywny parser zstępujący

$$\langle B \rangle ::= \underline{w \langle B \rangle} \mid w$$
CZY SIĘ
UDAŁOKONSTRUOWANE
DRZEWO

```
Boolean B(drzewo* drz) {
    drzewo drz1;
    if (nowyleks('w'))
        if (B(&drz1)) {
            *drz = ...;
            return true;
        }
    else {
        *drz = ...;
        return true;
    }
    else return false;
}
```

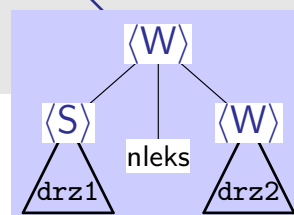
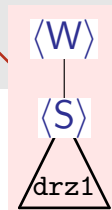


Programowanie parsera zstępującego

$$\langle W \rangle ::= \langle S \rangle \mid \langle S \rangle + \langle W \rangle \mid \langle S \rangle - \langle W \rangle$$

$$\langle W \rangle ::= \langle S \rangle \{ + \langle W \rangle \mid - \langle W \rangle \mid \lambda \}$$

```
Boolean W(drzewo* drz) {
    drzewo drz1, drz2;
    if (S(&drz1))
        if (nowyleks('+', &nleks) || nowyleks('-', &nleks))
            if (W(&drz2)) { *drz = ...; return TRUE; }
            else blad("po + lub - nie ma <W>");
        else { *drz = ...; return TRUE; }
    else return FALSE;
}
```



Uwaga: To jeszcze nie jest dokładnie to, co trzeba. Patrz dalej.

Parser zstępujący — problemy

Lewostronna rekursja w gramatyce powoduje nieskończone obliczenie.

PRZYKŁAD: inna gramatyka: $\langle W \rangle ::= \langle S \rangle \mid \langle W \rangle + \langle S \rangle$

```
Boolean W(drzewo* drz) {
    drzewo drz1, drz2;
    if (S(&drz1)) { *drz = ...; return TRUE; }
    else
        if (W(&drz1)) {
            if (nowyleks('+', &nleks))
                if (S(&drz2)) { *drz = ...; return TRUE; }
            else blad("po + brakuje <S>");
            else return FALSE;
        }
    else return FALSE;
}
```

— wewnętrzne W (w ciele funkcji) wywoływane jest dla tego samego stanu wejścia, co zewnętrzne — **ślepa rekursja**.

Leczenie lewostronnej rekursji

Lewostronna rekursja (lub lewostronna rekursja wzajemna) w gramatyce — powoduje zapętlenie parsera zstępującego

Co na to można zrobić?

Zastąpić $\langle X \rangle ::= L \mid \langle X \rangle M$ przez $\langle X \rangle ::= LM^*$

Nieformalnie:

M^* oznacza „ciąg M -ów grupowany do lewej”.

Leczenie lewostronnej rekursji

Gramatyka oryginalna z lewostronną rekursją:

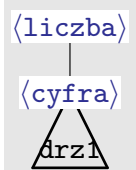
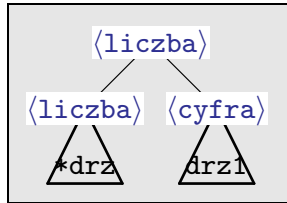
$$\begin{aligned}
 \langle W \rangle &::= \langle S \rangle \mid \langle W \rangle + \langle S \rangle \\
 &\quad \mid \langle W \rangle - \langle S \rangle \\
 \langle S \rangle &::= \langle C \rangle \mid \langle S \rangle * \langle C \rangle \\
 &\quad \mid \langle S \rangle / \langle C \rangle \\
 \langle C \rangle &::= \langle L \rangle \mid (\langle W \rangle) \\
 \langle L \rangle &::= 0 \mid 1 \mid \langle L \rangle 0 \mid \langle L \rangle 1
 \end{aligned}$$

Gramatyka wyleczona z iteracją:

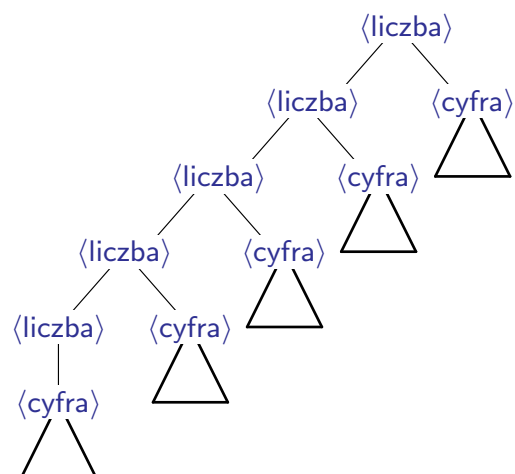
$$\begin{aligned}
 \langle W \rangle &::= \langle S \rangle \left\{ \left\{ + \mid - \right\} \langle S \rangle \right\}^* \\
 \langle S \rangle &::= \langle C \rangle \left\{ \left\{ * \mid / \right\} \langle C \rangle \right\}^* \\
 \langle C \rangle &::= (\langle W \rangle) \mid \langle L \rangle \\
 \langle L \rangle &::= \{ 0 \mid 1 \} \{ 0 \mid 1 \}^*
 \end{aligned}$$

Leczenie lewostronnej rekursji

```

Boolean liczba (drzewo* drz) {
    if (cyfra(&drz1)) {
        *drz = ;
        while (cyfra(&drz1))
            *drz = ;
        return TRUE;
    }
    else return FALSE;
}

```



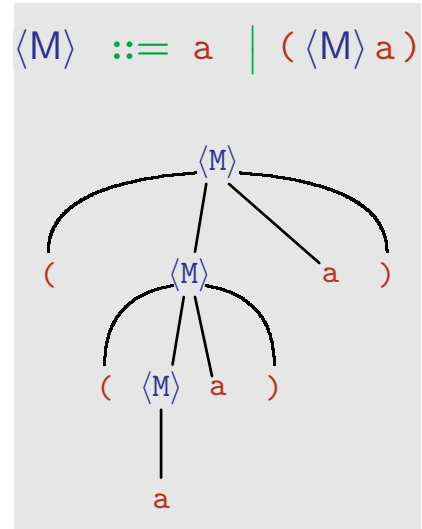
Rozbiór wstępny — gramatyki z pierwszeństwem

— systematyczny sposób przypasowywania prawych stron produkcji do terminali na wejściu.

W ciągu symboli wejściowych znaleźć *podstawę redukcji* — podciąg wejścia zgodny **znak ze znakiem** z *prawą stroną* produkcji.

Dokonać *redukcji* — zastąpić *podstawę* nie-terminalem z *lewej strony* tej produkcji.

Te kroki powtarzać, aż powstanie całe drzewo.



W jaki sposób znajdujemy podstawę redukcji?

Rozbiór wstępny — gramatyki z pierwszeństwem

W jaki sposób znajdujemy podstawę redukcji?

Relacje wpisane w tablicę związane są z kolejnością redukowania poszczególnych symboli:

- jeśli $x \doteq y$ to symbole x i y będą się redukować razem (jeśli stoją koło siebie i jeden należy do podstawy, to drugi też);
- jeśli $x < y$ to symbol x powinien poczekać, aż y zostanie zredukowany (jeśli stoją koło siebie i y należy do podstawy, to x nie należy do podstawy);
- jeśli $x > y$ to symbol y powinien poczekać, aż x zostanie zredukowany (jeśli stoją koło siebie i x należy do podstawy, to y nie należy do podstawy);
- jeśli nie ma między nimi żadnej relacji, to nie powinny stać obok siebie (jeśli stoją koło siebie, to jest to błąd).

$\langle M \rangle ::= a \mid (\langle M \rangle a)$

Tablica pierwszeństwa redukcji:

	(	$\langle M \rangle$	a	)
(	<	$\doteq$	<	
$\langle M \rangle$			$\doteq$	
a			>	$\doteq$
)			>	

Konstrukcja tablic pierwszeństwa

Niech $G = \langle \Sigma, N, P, S \rangle$ będzie gramatyką bezkontekstową.

Definicja:

Niech $A \in N$.

$$fst A \stackrel{\text{def}}{=} \{x \in \Sigma \cup N \mid \text{istnieje w } P \text{ jakaś produkcja } A \rightarrow xv\}$$

$$fst^+ A \stackrel{\text{def}}{=} fst A \cup \bigcup \{fst^+ B \mid B \in N \wedge B \in fst A\}$$

$$lst A \stackrel{\text{def}}{=} \{x \in \Sigma \cup N \mid \text{istnieje w } P \text{ jakaś produkcja } A \rightarrow wx\}$$

$$lst^+ A \stackrel{\text{def}}{=} lst A \cup \bigcup \{lst^+ B \mid B \in N \wedge B \in lst A\}$$

Przykład:

$$\langle M \rangle ::= a \mid (\langle M \rangle a) \quad \text{—} \quad fst \langle M \rangle = \{a, (\}$$

Przykład:

$$\left. \begin{array}{l} \langle M \rangle ::= a \mid \langle Q \rangle a \\ \langle Q \rangle ::= (\langle M \rangle \end{array} \right\} \quad \text{—} \quad \begin{cases} fst \langle M \rangle = \{a, \langle Q \rangle\} \\ fst^+ \langle M \rangle = \{a, \langle Q \rangle\} \cup fst^+ \langle Q \rangle \\ = \{a, \langle Q \rangle, (\} \end{cases}$$

Konstrukcja tablic pierwszeństwa

Niech $G = \langle \Sigma, N, P, S \rangle$ będzie gramatyką bezkontekstową.

Definicja:

Relacje $\doteq, \triangleleft, \triangleright$ w $\Sigma \cup N$ określone są następująco:

$$x \doteq y \stackrel{\text{def}}{\iff} \text{w } P \text{ istnieje produkcja } A \rightarrow wxyv$$

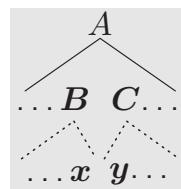
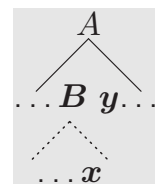
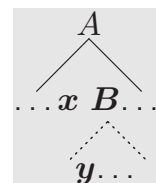
$$x \triangleleft y \stackrel{\text{def}}{\iff} \text{w } P \text{ istnieje produkcja } A \rightarrow wxBv \text{ taka że } y \in fst^+ B$$

$$x \triangleright y \stackrel{\text{def}}{\iff}$$

w P istnieje produkcja $A \rightarrow wByv$
taka że $x \in lst^+ B$

lub

w P istnieje produkcja $A \rightarrow wBCv$
taka że $x \in lst^+ B$ i $y \in fst^+ C$



Konstrukcja tablic pierwszeństwa

	(	$\langle M \rangle$	a	)
(	$\leq$	$\doteq$	$\leq$	
$\langle M \rangle$			$\doteq$	
a			$\geq$	$\doteq$
)			$\geq$	

Uwaga:

- nie jest prawdą, że zawsze $x \doteq x$
(np. nie zachodzi $a \doteq a$);
- nie jest prawdą, że zawsze z $x \doteq y$ wynika $y \doteq x$
(np. zachodzi $a \doteq)$ ale nie zachodzi $) \doteq a$);
- nie jest prawdą, że zawsze z $x \leq y$ wynika $y \geq x$
(np. zachodzi $(\leq a$ ale nie zachodzi $a \geq ($);
- nie jest prawdą, że zawsze z $x \leq y$ i $y \doteq z$ wynika $x \leq z$
(np. zachodzi $(\leq a$ i $a \doteq)$ ale nie zachodzi $(\leq)$).

Zakres stosowalności gramatyk z pierwszeństwem

Przykład:

$\langle W \rangle ::= \langle S \rangle \mid \langle W \rangle + \langle S \rangle$
 $\langle S \rangle ::= \ell \mid (\langle W \rangle)$

Ponieważ $($ i $\langle W \rangle$ sąsiadują w tej samej produkcji, więc $(\doteq \langle W \rangle$.

Ponieważ $\langle W \rangle \in \text{fst}^+ \langle W \rangle$, więc $(\leq \langle W \rangle$.

Dla tej gramatyki nie da się więc skonstruować tablicy pierwszeństwa.

Przykład:

$\langle A \rangle ::= a \mid \langle B \rangle a$
 $\langle B \rangle ::= a$

Nie wiadomo, do jakiego nieterminalu zredukować a .

Dla tej gramatyki nie da się zastosować analizy z pierwszeństwem.

Definicja:

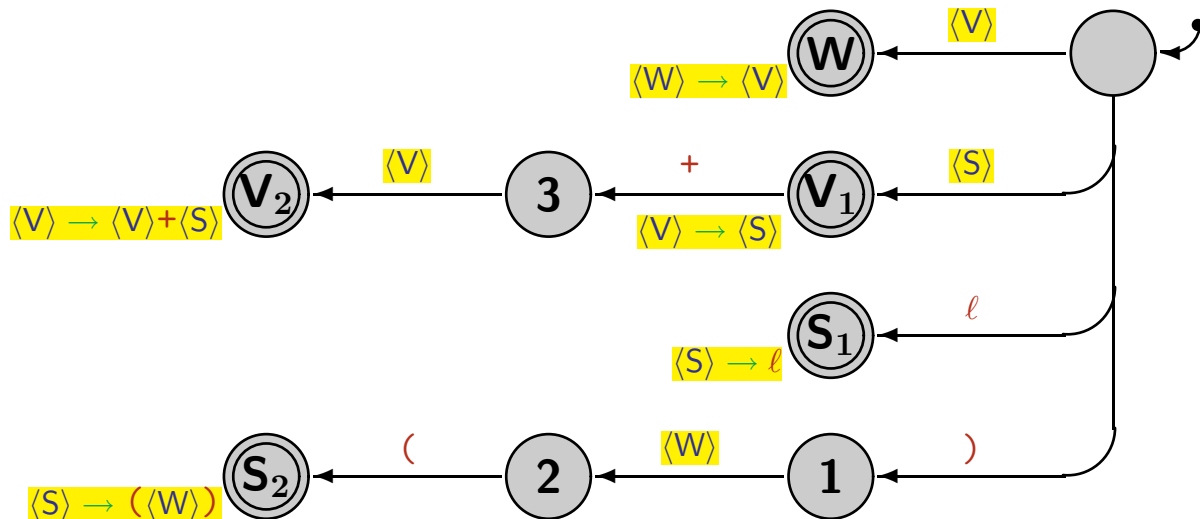
Gramatyka bezkontekstowa $G = \langle \Sigma, N, P, S \rangle$ jest *gramatyką z pierwszeństwem* jeśli

- dla każdych dwóch symboli $x, y \in \Sigma \cup N$, zachodzi najwyżej jeden ze związków $x \doteq y$, $x \leq y$, $x \geq y$ (może nie zachodzić żaden),
- nie istnieją dwie różne produkcje o tej samej prawej stronie, ani produkcja o pustej prawej stronie: $\langle A \rangle \rightarrow \lambda$.

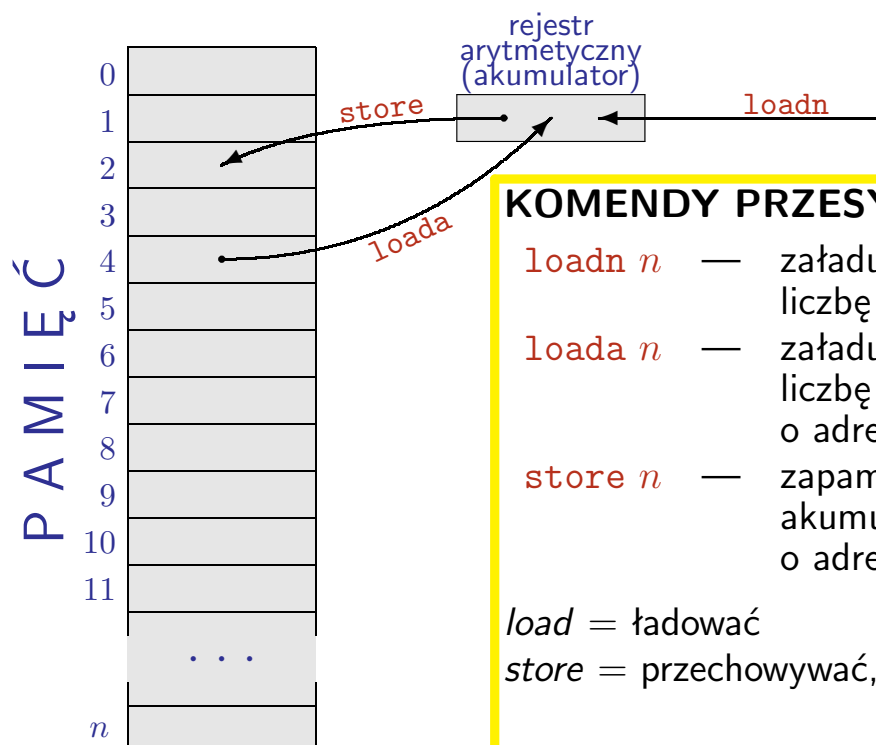
Wykonanie redukcji

$$\begin{array}{lcl} \langle W \rangle & ::= & \langle V \rangle \\ \langle V \rangle & ::= & \langle S \rangle \mid \langle V \rangle + \langle S \rangle \\ \langle S \rangle & ::= & \ell \mid (\langle W \rangle) \end{array}$$

- zdejmujemy ze stosu symbole między $\leq$ and $\geq$;
- wyszukujemy właściwą produkcję za pomocą automatu skończonego zbudowanego wg gramatyki;
- wkładamy na stos nieterminal z lewej strony tej produkcji.



Programowanie niskiego poziomu



KOMENDY PRZESYŁANIA:

`loadn  $n$` — załaduj do akumulatora
liczbę n

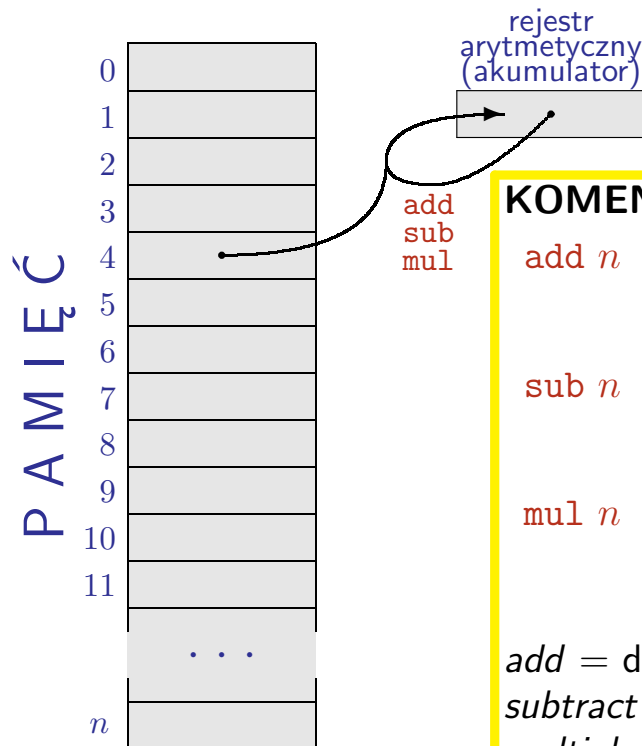
loada n — załaduj do akumulatora
liczbę z komórki
o adresie n

store n — zapamiętaj zawartość
akumulatora w kom.
o adresie n

load = ładować

store = przechowywać, magazynować

Programowanie niskiego poziomu



KOMENDY ARYTMETYCZNE:

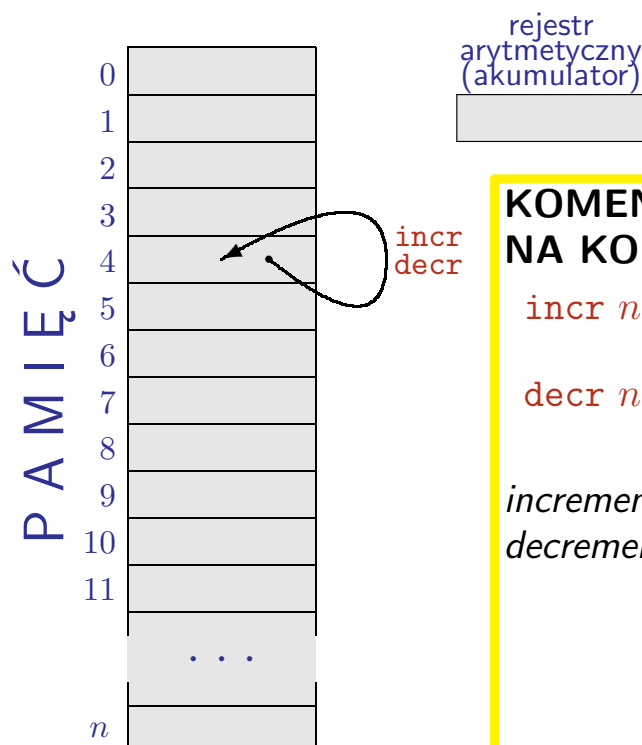
- add n** — dodaj do akumulatora liczbę z kom. o adresie n ; wynik w akumulatorze
- sub n** — odejmij od akumulatora liczbę z kom. o adresie n ; wynik w akumulatorze
- mul n** — pomnóż akumulator przez liczbę z kom. o adresie n ; wynik w akumulatorze

add = dodawać

subtract = odejmować

multiply = mnożyć

Programowanie niskiego poziomu



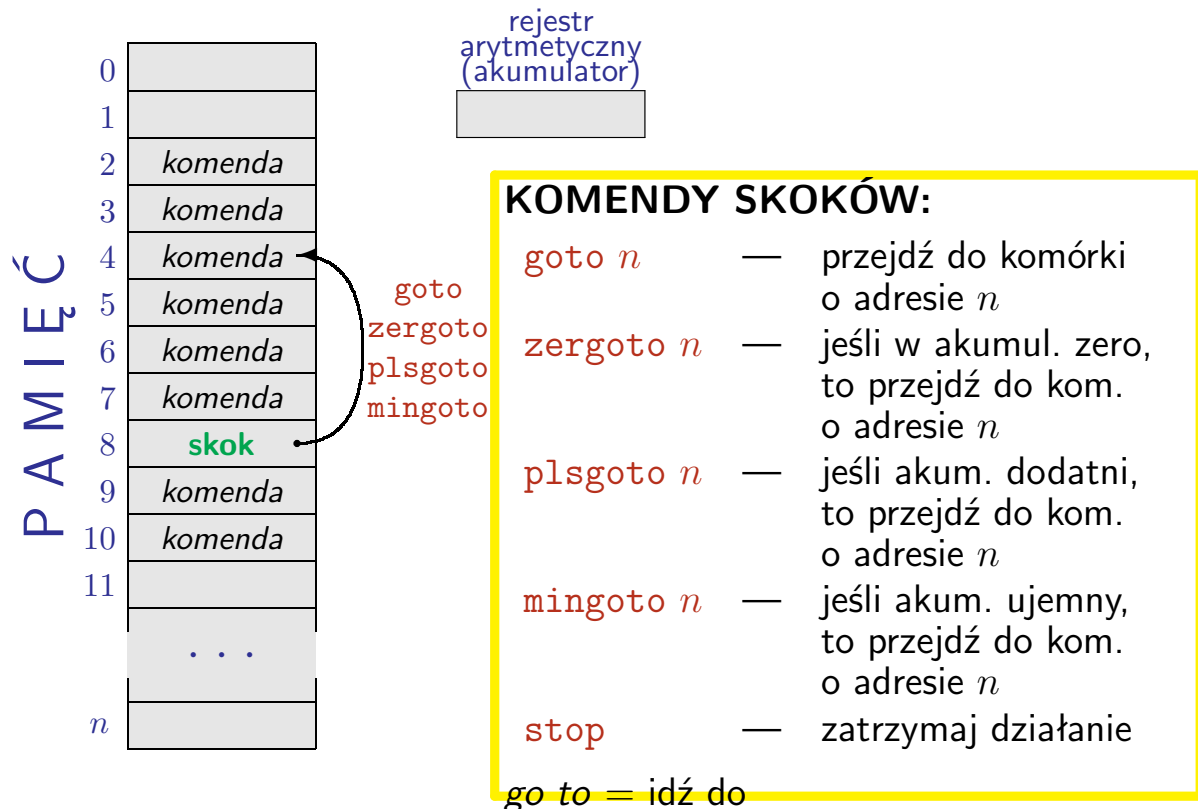
KOMENDY DZIAŁAŃ NA KOMÓRKACH:

- incr n** — powiększ o 1 zawartość kom. o adresie n
- decr n** — zmniejsz o 1 zawartość kom. o adresie n

increment = zwiększać

decrement = zmniejszać

Programowanie niskiego poziomu



Adresy

1 Prosty adres komendy:

liczba całkowita oznaczająca

- albo samą liczbę,
np. `loadn 500` oznacza: załaduj do akumulatora liczbę 500;
- albo adres komórki, której dotyczy komenda,
np. `loada 500` oznacza: załaduj do akumulatora liczbę z komórki 500.

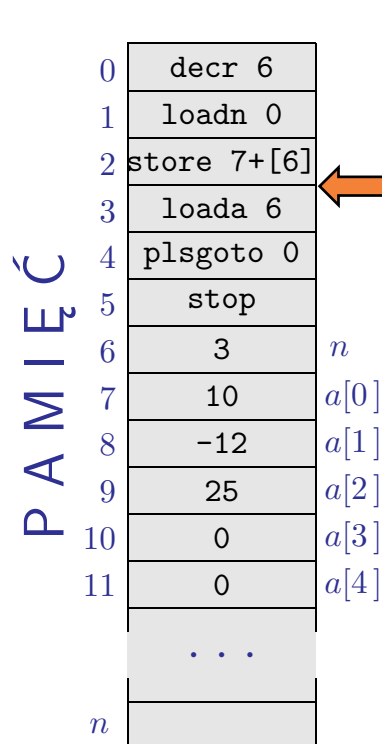
2 Modyfikowany adres komendy:

postaci `n+[k]` — adres n modyfikowany zawartością komórki o adresie k , należy tę zawartość do adresu n dodać.

Np. jeśli pod adresem 500 stoi liczba 1, to komenda `add 200+[500]` oznacza: dodaj do akumulatora zawartość komórki 201;

a jeśli pod adresem 500 stoi liczba 5, to ta sama komenda `add 200+[500]` oznacza: dodaj do akumulatora zawartość komórki 205.

Adresy



rejestr
arytmetyczny
(akumulator)

0

PRZYKŁAD:

Zerowanie n komórek

```
do
  n--;
  a[n]=0;
while (n>0);
```

Wielkość n obszaru zerowanego — kom. 6;
obszar zerowany zaczyna się w kom. 7;
program uruchomić od kom. 0;

Duży program tworzony po kawałku

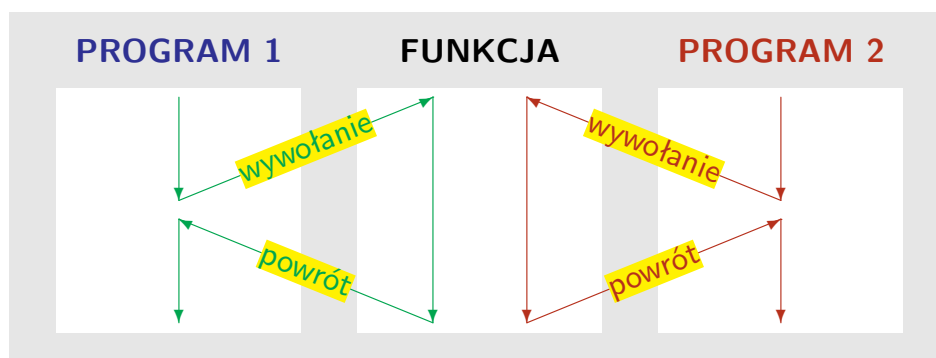
Duże programy **muszą** składać się ze **współpracujących** kawałków/podprogramów/procedur/funkcji/modułów/...

dla języka wewnętrznego
to są synonimy

Jak program współpracuje z funkcją?

- funkcja współdziała z **różnymi** programami,
- program podaje funkcji argumenty oraz **adres powrotu**,
- funkcja oblicza wartość wyniku i oddaje programowi,
- po zakończeniu działania funkcji, sterowanie wraca tam, skąd przyszło, czyli dla każdego wywołującego programu gdzie indziej.

skok pod adres zapisany w komórce x : goto $0+[x]$



W realnych komputerach...

- adresowane są nie **komórki** tylko **bajty**; bajt zawiera 8 bitów — to wystarcza na liczbę całkowitą z przedziału $[0..255]$ ($2^8 = 256$), albo z przedziału $[-128..127]$
- na ogół operujemy na danych znacznie wykraczających poza 1 bajt

MAŁO!

Zwykle (choć nie zawsze):

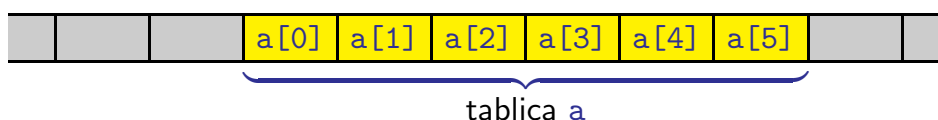
typ	bajty	
unsigned int	4	zakres: $[0..2^{32}-1] \simeq [0..4 \text{ mld}]$
int	4	zakres: $[-2^{31}..2^{31}-1] \simeq [-2 \text{ mld}..2 \text{ mld}]$
float	4	niska precyzja, 6–7 cyfr znaczących
double	8	15–16 cyfr znaczących
tablice	...	
struktury	...	
...	...	

UNIKAĆ!

Organizacja pamięci

W czasie kompilacji:

- na każdą *zadeklarowaną zmienną* rezerwuje się *blok pamięci* wielkości zależnej od *typu* tej zmiennej (np. na całkowitą zawsze 4 bajty, na rzeczywistą zawsze 8 bajtów, itp.);
- instrukcje *jęz. źródłowego* dotyczące tej zmiennej są tłumaczone na *instrukcje jęz. wewnętrznego* dotyczące tego bloku (odnoszące się do adresów zawartych w tym bloku);
- tablice zajmują spójne obszary pamięci o kolejnych adresach:



Organizacja pamięci

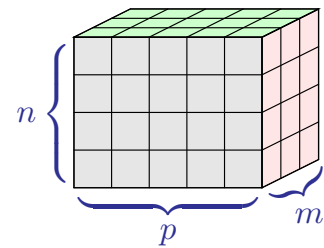
Tablice wielowymiarowe:

Deklaracja: `T tab[n][m][p];`

czyli n warstw po

m wierszy w warstwie po

p elementów typu T w wierszu



Założmy, że pojedynczy element typu T zajmuje b bajtów; a początek tablicy jest w bajcie a .

Które adresy należą do elementu `tab[i][j][k]` ?

Poprzedza go:

i warstw po $m \cdot p$ w warstwie : $i \cdot m \cdot p$

j wierszy własnej warstwy po p w wierszu : $j \cdot p$

k elementów własnego wiersza : k

deklaracja: `tab[n][m][p]`



element: `tab[i][j][k]`

Razem : $i \cdot m \cdot p + j \cdot p + k = (i \cdot m + j) \cdot p + k$

czyli $((i \cdot m + j) \cdot p + k) \cdot b$ bajtów

więc zaczyna się w bajcie $((i \cdot m + j) \cdot p + k) \cdot b + a$

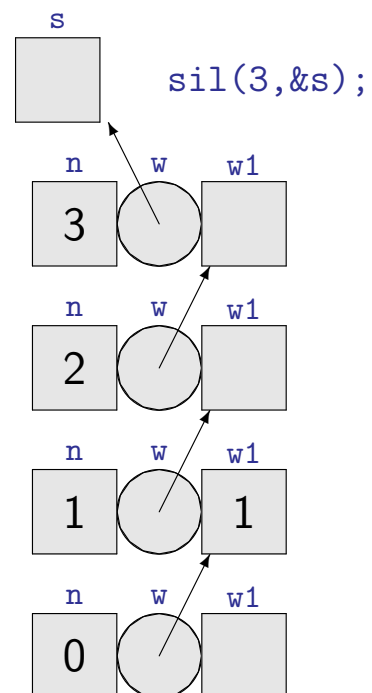
Organizacja pamięci — funkcje, parametry, zm. lokalne

```
void sil(int n, int* w) {
    int w1;
    if (n==0) *w = 1;
    else {sil(n-1,&w1); *w = n*w1; }
}
```

```
void sil(int n, int* w) {
    int w1;
    if (n==0) *w = 1;
    else {sil(n-1,&w1); *w = n*w1; }
}
```

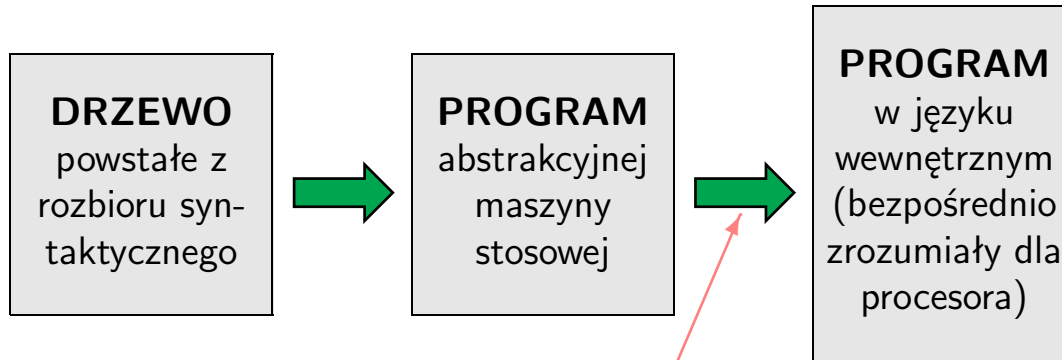
```
void sil(int n, int* w) {
    int w1;
    if (n==0) *w = 1;
    else {sil(n-1,&w1); *w = n*w1; }
}
```

```
void sil(int n, int* w) {
    int w1;
    if (n==0) *w = 1;
    else { sil(n-1,&w1); *w = n*w1; }
}
```



Dwuetapowe tłumaczenie programu

Obliczenia skompilowanego programu odbywają się **na stosie**. Dlatego tłumaczenie programu odbywa się w dwóch etapach:

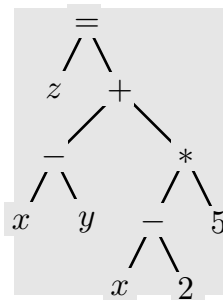


Abstrakcyjna maszyna stosowa jest niezależna od platformy. Przy przenoszeniu kompilatora na inną platformę trzeba przeprogramować **tylko** drugi etap.

Tłumaczenie komend

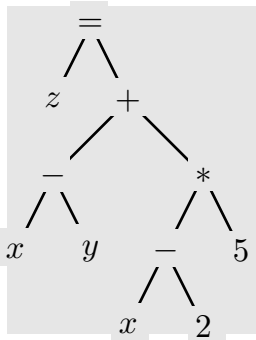
Po analizie składniowej komendy są w postaci drzew; np.

$$z = (x - y) + (x - 2) * 5;$$



przekład stałej s	na stos liczbę s
przekład zmiennej o adr. x po prawej str. przypisania	na stos liczbę spod adr. x
przekład wyrażenia	przekład T_1 ; przekład T_2 ; odejmij od przedostatniego na stosie ostatni na stosie
przekład przypisania	na stos adres z ; przekład T ; pod adres przedostatni na stosie włóż wartość ostatnią na stosie

Tłumaczenie komend



przekład st. <i>s</i>	na stos liczbę <i>s</i>
przekład zmiennej o adr. <i>x</i> po prawej str. przypisania	na stos liczbę spod adr. <i>x</i>
przekład wyr. 	przekład <i>T</i> ₁ ; przekład <i>T</i> ₂ ; odejmij od przedost. na stosie ost. na stosie
przekład przyp. 	na stos adres <i>z</i> ; przekład <i>T</i> ; pod adres przedost. na stosie włóż wartość ost. na stosie

PROGRAM:

```
na stos adres 52;  
na stos liczbę spod adr. 50;  
na stos liczbę spod adr. 51;  
odejmowanie;  
na stos liczbę spod adr. 50;  
na stos liczbę 2;  
odejmowanie;  
na stos liczbę 5;  
mnożenie;  
dodawanie;  
przypisanie
```

PAMIĘĆ:

	50	51	52	
...	15	3	-4	...
	<i>x</i>	<i>y</i>	<i>z</i>	

Język stosowy → język wewnętrzny

PROGRAM STOSOWY:

```
E1: na stos wart.zm. a;  
na stos wart.zm. b;  
nierówność;  
jeśli na stosie false  
to idźdo E2;  
na stos wart.zm. a;  
na stos wart.zm. b;  
większość;  
jeśli na stosie false  
to idźdo E3;  
na stos adres a;  
na stos wart.zm. a;  
na stos wart.zm. b;  
odejmowanie;  
przypisanie;  
idźdo E4;  
E3: na stos adres b;  
na stos wart.zm. b;  
na stos wart.zm. a;  
odejmowanie;  
przypisanie;  
E4: idźdo E1;  
E2:
```

		324
		320
	11	316
	9	312
<i>n</i>	4	308
<i>b</i>	11	304
<i>a</i>	20	300

PROGRAM W JĘZ.WEWN.:

```
zwiększ [308] o 4  
pobierz spod adr. 300  
włóż pod adres 308+[308]  
zwiększ [308] o 4  
pobierz spod adr. 304  
włóż pod adres 308+[308]  
zmniejsz [308] o 4  
pobierz spod adr. 308+[308]  
odejmij spod adr. 312+[308]  
włóż pod adres 308+[308]  
zmniejsz [308] o 4  
czy zero pod adr. 312+[308]  
jeśli tak to idźdo E2  
zwiększ [308] o 4  
pobierz spod adr. 300  
włóż pod adres 308+[308]  
zwiększ [308] o 4  
pobierz spod adr. 304  
włóż pod adres 308+[308]  
zmniejsz [308] o 4  
pobierz spod adr. 308+[308]  
odejmij spod adr. 312+[308]  
włóż pod adres 308+[308]  
...
```

Gramatyki atrybutowe

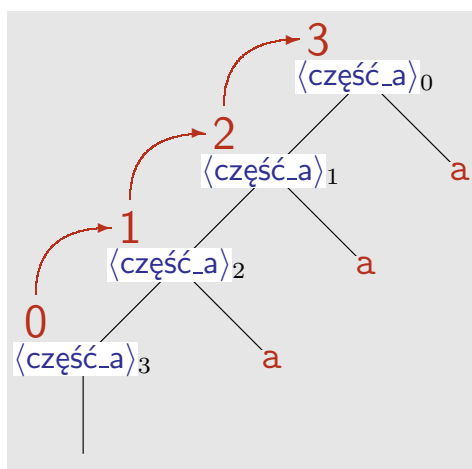
Gramatyka atrybutowa — gramatyka bezkontekstowa wzbogacona o

- **atrybuty** symboli w drzewie wyvodu (terminali i nieterminali); atrybuty mogą mieć wartości;
- **funkcje obliczające** wartości atrybutów.



Gramatyki atrybutowe

Funkcje obliczające powodują propagowanie wartości atrybutu po drzewie wyvodu:



$$\langle \text{część_a} \rangle_0.\text{ile} \leftarrow \langle \text{część_a} \rangle_1.\text{ile} + 1$$

$$\langle \text{część_a} \rangle.\text{ile} \leftarrow 0$$

Działanie BISONa — rozpoznawanie $a^n b^n c^n$

```
slovo:  czesc_a czesc_b czesc_c '.'
```

```
{ if ($1 == $2 && $1 == $3) printf("\n  NALEZY.\n\n");
  else printf("\n  NIE NALEZY.\n\n");
  exit(1);
} ;
```

```
czesc_a:  /* puste */
          |  czesc_a 'a'
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

```
czesc_b:  /* puste */
          |  czesc_b 'b'
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

```
czesc_c:  /* puste */
          |  czesc_c 'c'
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

— gramatyka

— działania na atrybutach

iis.ans-elblag.pl/~stefan/Dydaktyka/JezForm/Wyklady/13-bizony/1-aabbcc.y

Analiza leksykalna

LEX



FLEX (= Fast LEX)

www.cs.princeton.edu/~appel/modern/c/software/flex

wyszukuje wzorce **regularne** i wykonuje akcje;
w najprostszym przypadku zastępuje

Dane użytkownika
— co robić z tekstem

definicje

%%

reguły

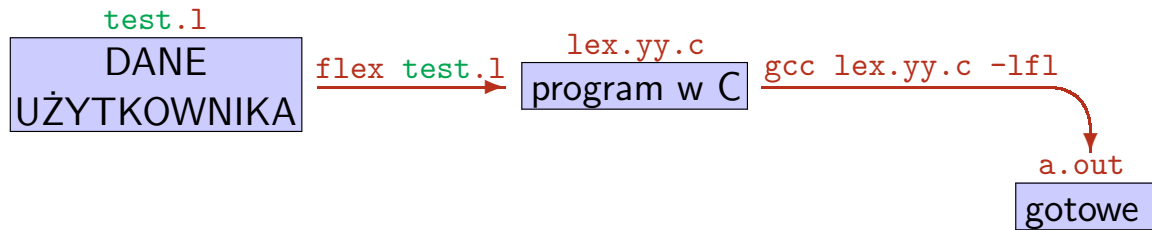
%%

kod użytkownika

Najważniejsza część: **reguły** składające się z

- **wzorca** — jego napotkanie w tekście wejściowym ma spowodować jakąś akcję;
- **akcji** — czynności, które należy podjąć po rozpoznaniu wzorca w tekście wejściowym (opisane w C).

Analiza leksykalna



Przykład:

Jeśli dane użytkownika:

test.l

[0-9]+	printf("LICZBA");
[+\-]	printf("OP");
[A-Z] [a-z]+	printf("KTOŚ");

to

tekst	wynik
kto wie, ile jest (15 - 8) + 6 ? Jasio wie, Szymek nie wie.	kto wie, ile jest (LICZBA OP LICZBA) OP LICZBA ? KTOŚ wie, KTOŚ nie wie.

./a.out < tekst

iis.ans-elblag.pl/~stefan/Dydaktyka/JezForm/JezForm/Wyklady/14-fleksy/1-FlexSam