Języki formalne i kompilatory — materiały pomocnicze do nauki do egzaminu

ver. 2.01

16 września 2011

Spis treści

1 Teoria

1.1 Wstęp

1.1.1 Język

Język to system zorganizowanych znaków służący do utrwalania rezultatów działalności myślowej a także wzajemnej komunikacji.

Podział języków:

naturalne (etniczne) — powstałe w procesie ewolucji biologicznej lub społecznej (np. polski, gestowo-mimiczny). Ich reguły powstają ewolucyjnie i są trudne lub wręcz niemożliwe do ścisłego określenia,

formalne (sztuczne) — ściśle określone systemy znaków (symboli) wraz z regułami postępowania z tymi znakami a także regułami ich interpretowania (np. kody, języki zapisu dokumentów, język logiki, języki programowania); reguły zostają określone (zazwyczaj) przed rozpoczęciem użytkowania w sposób ścisły.

Tradycyjnie przyjmuje się, iż każdy język opisywany jest przez trzy podstawowe aspekty (**składowe języka**):

składnia — reguły konstrukcji języka pozwalające ne zbudowanie wszystkich poprawnych zdań danego języka; abstrahuje od znaczenia wygenerowanych zdań reguły syntaktyczne dzielimy na:

formujące — pozwalają określić sposoby poprawnego tworzenia wyrażeń

transformujące — pozwalają określić sposoby poprawnego przekształcania wyrażeń tak, by zachowana była pewna ich własność, np. tautologiczność.

semantyka — opis znaczenia przypisywanego poprawnym zdaniom, odniesienie wygenerowanych zdań do rzeczywistości,

pragmatyka — zespół reguł mówiących jak należy stosować składnię i semantykę języka, bada stosunki między językiem a jego użytkownikami, bliskoznaczna ze stylistyką — mówi, którą z form o zbliżonym znaczeniu lepiej wybrać (= bo jest „piękniejsza").

Najłatwiej formalizować składnię (gramatyki generacyjne, gramatyki dwupoziomowe, składnia abstrakcyjna (np. wiedeńska metoda definiowania składni VDL)), trudniej semantykę (metoda operacyjna, denotacyjna oraz aksjomatyczna), nie ma za to zadowalających metod formalizacji pragmatyki.

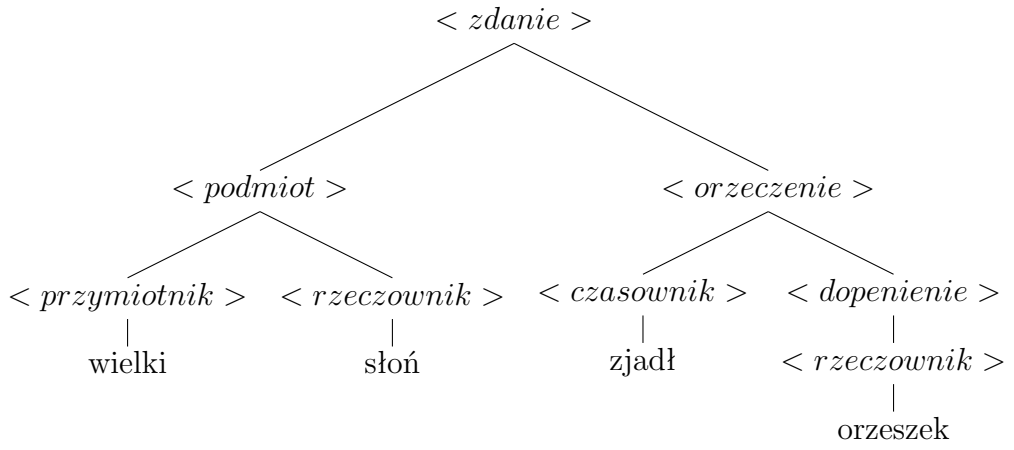
Metajęzyk to język służący do opisu innego języka (jest potrzebny, co udowadnia antynomia Grellinga [1]).

Symbole metajęzyka nazywamy **nieterminalnymi** zaś symbole właściwego języka **terminalnymi**.

Metody opisu składni:

- drzewo syntaktyczne:

[1] Niech orzeczniki autosemantyczne orzekają o sobie (np. rzeczownik), zaś heterosemantyczne nie orzekają o sobie; czy orzecznik „heterosemantyczny" jest heterosemantyczny czy autosemantyczny?



- notacja BNF

```
<zdanie> ::= <podmiot><orzeczenie>
<podmiot> ::= <przymiotnik><rzeczownik>
<przymiotnik> ::= wielki
<orzeczenie> ::= <czasownik><dopełnienie>
<czasownik> ::= zjadł
<dopełnienie> ::= <rzeczownik>
<rzeczownik> ::= słoń
<rzeczownik> ::= orzeszek
```

Dwa ostatnie zdania możemy też zapisać jako

```
<rzeczownik> ::= słoń | orzeszek
```

1.2 Języki formalne — definicje

Alfabetem V pewnego języka nazywamy dowolny skończony ciąg symboli $a_1, a_2, a_3, \ldots, a_n$.

Słowem (łańcuchem) nad alfabetem V nazywamy dowolny skończony ciąg z powtórzeniami elementów zbioru V. Zbiór wszystkich słów nad alfabetem V oznaczamy V^*. Zachodzi $V \subseteq V^*$.

Konkatenacja . Niech $x, y \in V^*, x = x_1 \ldots x_m, y = y_1 \ldots y_n$. Słowo z takie, że $z = x_1 \ldots x_m y_1 \ldots y_n$ nazywamy złożeniem słów x i y i oznaczamy $z = x \cdot y$ lub $z = xy$. Złożenie nie jest przemienne, ale jest łączne. Przykład: $a =$ 'alama', $b =$ 'kota', $ab =$ 'alamakota'.

Symbol pusty to łańcuch nie zawierający żadnego symbolu, oznaczamy go ε. Zakładamy, że zawsze należy do V^*. Dla każdego słowa x zachodzi $x\varepsilon = \varepsilon x = x$.

Potęga słowa . Niech $x \in V^*$ będzie słowem. Określamy potęgę słowa $X^n, n \geqslant 0$

$$x^n = \begin{cases} \varepsilon & \text{jeśli } n = 0 \\ xx^{n-1} & \text{jeśli } n \geqslant 0 \end{cases}$$

Przykład: $'alamakota'^3 = 'alamakotaalamakotaalamakota'$

Długością słowa nazywamy liczbę symboli występującą w łańcuchu, indukcyjnie:

$$|\varepsilon| = 0$$
$$|ax| = |x| + 1$$
$$a \in V, x \in V^*$$

Przykład: $|'alamakota'| = 9$.

Podsłowo . Niech $x, y \in V^*$. Słowo x nazywamy podsłowem słowa y jeśli istnieją słowa z, w takie, że $y = zxw$, na przykład słowo 'ma' jest podsłowem słowa 'alamakota' (wtedy $z =$ 'ala', a $w =$ 'kota'). Jeśli $z = \varepsilon$ to x nazywamy **prefiksem (podsłowem początkowym)**. Jeśli $w = \varepsilon$ to x nazywamy **sufiksem (podsłowem końcowym)**. Jeśli x jest prefiksem (sufiksem) słowa y i $x \neq y$ to x nazywamy **prefiksem (sufiksem) właściwym**.

Język . Jeśli V jest dowolnym alfabetem, to każdy podzbiór L zbioru V^* nazywamy językiem nad alfabetem V.

Złożenie języków . Jeśli L_1, L_2 są językami, to ich złożenie oznaczamy L_1L_2 i definiujemy

$$L_1L_2 \equiv \{xy : x \in L_1 \wedge y \in L_2\}$$

Przykład: $L_1 = \{a, b\}, L_2 = \{c, d\}$, wtedy $L_1L_2 = \{ac, ad, bc, bd\}$.

Potęga złożeniowa języka . Analogicznie do potęgi słowa definiujemy potęgę języka L:

$$L^n = \begin{cases} \{\varepsilon\} & \text{jeśli } n = 0 \\ LL^{n-1} & \text{jeśli } n \geqslant 0 \end{cases}$$

Domknięciem (nieskończonym) języka L nazywamy język

$$L^* \equiv \bigcup_{i=0}^{\infty} L^i$$

Domknięciem dodatnim języka L nazywamy język

$$L^+ \equiv \bigcup_{i=1}^{\infty} L^i$$

Jedyną różnicą między tymi dwoma domknięciami jest początkowe i. Zachodzą związki:

$$L^* = L^0 \cup L^+ = \{\varepsilon\} \cup L^+$$
$$L^* = LL^* = L^*L$$

1.3 Gramatyki formalne (generacyjne) — definicje

Pojęcie **gramatyki formalnej** w dzisiejszym rozumieniu powstało przez połączenie pojęć **gramatyki struktur frazowych** N. Chomsky'ego i **semisystemu (semigramatyki) Thuego**

Gramatyką formalną G nazywamy czwórkę $G = \langle N, V, P, S\rangle$ gdzie:

- N — alfabet nieterminalny (symboli metajęzyka),

- V — alfabet terminalny (symbole języka tworzące słowa języka określanego przez gramatykę),
- P — zbiór skończony reguł produkcji (lista produkcji, zbiór reguł zastępowania), tj. relacja $P \subset (N \cup V)^+ \times (N \cup V)^*$,
- S — symbol początkowy (aksjomat języka).

Przyjmujemy, że:

- $N \cap V = \emptyset$
- Reguły produkcji możemy też zapisać równoważnie

$$P \subset ((N \cup V)^* \setminus \{\varepsilon\}) \times (N \cup V)^*$$

- $S \in N$

Przykład gramatyki generującej liczbę binarną:

$$\begin{aligned} G &= \langle N, V, P, S \rangle \\ N &= \{\langle liczba\ binarna \rangle, \langle cyfra \rangle\} \\ V &= \{0, 1\} \\ P &= \left\{ \begin{array}{l} \langle liczba\ binarna \rangle, \langle liczba\ binarna \rangle \langle cyfra \rangle \\ \langle liczba\ binarna \rangle, \langle cyfra \rangle \\ \langle cyfra \rangle, 0 \\ \langle cyfra \rangle, 1 \end{array} \right\} \\ S &= \langle liczba\ binarna \rangle \end{aligned}$$

Wyprowadzenie bezpośrednie. Mówimy, że napis v **wyprowadza bezpośrednio** napis w, albo że w jest **bezpośrednim wyprowadzeniem** v, albo też że w **redukuje się bezpośrednio do** v (oznaczamy $v \Rightarrow w$), jeśli istnieją napisy x, y (jeden z nich lub oba mogą być puste) oraz produkcja $U ::= u$, takie, że spełnione są warunki

$$v = xUy$$
$$w = xuy$$

Wyprwadzenie pośrednie. Mówimy, że v wyprowadza (pośrednio) napis w albo że w jest wyprowadzeniem (pośrednim) v albo że w redukuje się (pośrednio) do v (oznaczamy $v \stackrel{+}{\Rightarrow} w$), jeśli istnieje ciąg napisów $u_0, u_1, \ldots, u_n, n > 0$ taki, że

$$v = u_0 \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_n = w$$

przy czym mówimy, że jest to wyprowadzenie o długości n. Dodatkowo oznaczamy $v \stackrel{*}{\Rightarrow} w$ gdy zachodzi $v \stackrel{+}{\Rightarrow} w$ lub $v = w$.

Formą zdaniową nazywamy pewien napis x taki, że x można wprowadzić z symbolu początkowego gramatyki, tzn gdy $S \stackrel{*}{\Rightarrow} x$. **Zdaniem** nazywamy formę zdaniową zwierającą wyłącznie symbole terminalne.

Językiem definiowanym (generowanym) przez gramatykę G której symbolem początkowym jest S nazywamy zbiór

$$L(G) = \{x : x \in V^* \wedge S \stackrel{*}{\Rightarrow} x\}$$

Słowa alfabetu nieterminalnego nie pojawiają się w języku $L(G)$, choć pojawiają się w wyprowadzeniach.

Równoważność gramatyk. Gramatyki G_1, G_2 nazywamy **równoważnymi** gdy $L(G_1) = L(G_2)$.

U-fraza. Jeżeli w jest formą zdaniową, przy czym $w = xuy$, to u nazywamy **U-frazą** lub krótko **frazą** jeśli $S \overset{*}{\Rightarrow} xUy$ oraz $U \overset{+}{\Rightarrow} u$. Napis u nazywamy **U-frazą prostą** lub krótko **frazą prostą** jeżeli $S \overset{*}{\Rightarrow} xUy$ oraz $U \Rightarrow u$.

$U \overset{+}{\Rightarrow} u$ to za mało, aby u było U-frazą, gdyż u lub U może nie być podsłowem żadnej formy zdaniowej.

Argumentem dowolnej formy zdaniowej nazywamy jej najbliższą z lewej strony frazę prostą.

1.4 Klasyfikacja Chomsky'ego

0 — gramatyki struktur frazowych zawierają produkcje postaci

$$\alpha \to \beta$$

gdzie $\alpha \in (N \cup V)^+$, $\beta \in (N \cup V)^*$. Nie ma żadnych ograniczeń na α i β.

Gramatyki te są zbyt ogólne do opisy składni języków programowania, a jednocześnie nie są wystarczające do opisu języków naturalnych.

1 — gramatyki kontekstowe zawierają produkcje postaci

$$\alpha A \beta \to \alpha \pi \beta$$

gdzie $\alpha, \beta \in (N \cup V)^*$, $A \in N$, $\pi \in (N \cup V)^+$. Mówiąc ogólnie $P \subset (N \cup V)^+ \times (N \cup V)^+$ przy czym po lewej stronie produkcji występuje przynajmniej jeden symbol nieterminalny.

2 — gramatyki bezkontekstowe zawierają produkcje postaci

$$A \to \pi$$

gdzie $A \in N$, $\pi \in (N \cup V)^*$, czyli $P \subset N \times (N \cup V)^*$, a więc lewa strona produkcji jest pojedynczym symbolem nieterminalnym.

Większość języków programowania należy do tej klasy, także notacja BNF odpowiada definicji języków klasy 2.

Klasa 2 **nie jest** podklasą klasy 1, gdyż w gramatykach klasy 2 po prawej stronie produkcji może znajdować się symbol pusty.

3 — gramatyki (prawostronnie) regularne zawierają produkcje postaci

$$A \to a$$
$$A \to aB$$

gdzie $A, B \in N$, $a \in V^*$. Prawa strona produkcji zawiera albo tylko symbol terminalny, albo symbol terminalny po którym następuje symbol nieterminalny.

Analogicznie można określić gramatyki lewostronnie regularne.

Kolejne klasy gramatyk są coraz mniejsze, np. istnieją gramatyki struktur frazowych, które nie są kontekstowe.

Jeżeli gramatyka generująca język jest kontekstowa (bezkontekstowa, regularna) to **język nazywamy także kontekstowym (bezkontekstowym, regularnym)**.

Gramatyką ε-wolną nazywamy gramatyki nie zawierające symbolu pustego po prawej stronie.

Dla każdej gramatyki bezkontekstowej G można skonstruować gramatykę ε-wolną G' taką, że $L(G') = L(G) \setminus \{\varepsilon\}$.

Jeżeli języki L_1, L_2 są regularne, to regularnymi są także języki:

$$L_1 \cup L_2, L_1 \cap L_2, L_1 \setminus L_2, L_1 L_2, L^*$$

1.5 Ważne gramatyki

$a^{n_1}b^{n_1}a^{n_2}b^{n_2}\ldots a^{n_k}b^{n_k}$, $n_i \geqslant 1$	$N = \{A, B\}$ $V = \{a, b\}$ $P = \{A \to B, A \to AB, B \to ab, B \to aBb\}$ $S = A$
$(noswsos)^k$, $k \geqslant 1$	$N = \{Q, X, Y, Z\}$ $V = \{n, o, s, w\}$ $P = \{Z \to nY, Y \to osX, X \to wsQ, Q \to osZ, Q \to os\}$ $S = Z$
$a^{2^n}b^{2^n}$, $n \geqslant 0$	$N = \{L, Q, R\}$ $V = \{a, b\}$ $P = \{R \to ab, R \to LRQ, La \to aaL, bQ \to Qbb, LLQQ \to LQ, aLQb \to ab\}$ $S = R$
$a^{2^n-1}b^{2^n-1}c^{2^n-1}$, $n \geqslant 1$	$N = \{K, X, Y\}$ $V = \{a, b, c\}$ $P = \{K \to abc, K \to aXKY, Xa \to aaX, Xb \to bbX, XY \to bc, cY \to Ycc\}$ $S = K$
$a^k b^{k+l} a^l : k, l > 0$	$N = \{B, U, K\}$ $V = \{a, b\}$ $P = \{B \to UK, U \to ab, U \to aUb, K \to ba, K \to bKa\}$ $S = B$

1.6 Rozbiór gramatyczny

Rozbiór. Jeśli forma α wyprowadza formę β, to rozbiorem formy β do α nazywamy ciąg produkcji stosowanych w kolejnych krokach aby zredukować formę β do α.

Rozbiorem kanonicznym (wyprowadzeniem prawostronnym) nazywamy rozbiór który zawsze rozpoczyna się od lewej strony redukowanej formy zdaniowej, tzn. w pierwszej kolejności redukuje lewostronne symbole formy.

Podstawa (uchwyt, osnowa) to podciąg symboli, który redukowany jest podczas rozbioru kanonicznego.

Rozbiór zdania 0110 przyjmując numerację symboli od 1 do 4 (patrz sekcja 1.3):

- do symbolu $<$ *liczba binarna* $>$: 3,4,4,3,2,1,1,1
- kanoniczny: 3,2,4,1,4,1,3,1

Zdanie wieloznaczne. Zdanie języka generowanego przez gramatykę nazywamy wieloznacznym, jeżeli istnieje więcej niż jedno drzewo syntaktyczne wprowadzania tego zdania.

Gramatyka wieloznaczna to taka gramatyka, że język przez nią generowany zawiera przynajmniej jedno zdanie wieloznaczne. Gramatykę, która nie jest wieloznaczna nazywamy **jednoznaczną**.

Uwagi:

- Jedno drzewo syntaktyczne może odpowiadać wielu różnym wyprowadzeniom. Każde wyprowadzenie ma dokładnie jedno drzewo syntaktyczne. Dwa wywody są **równoważne** gdy ich drzewa syntaktyczne są identyczne.
- Własność jedno- lub wieloznaczności przypisuje się gramatyce, a nie językowi przez nią generowanemu.
- Dla tego samego języka często można sformułować gramatykę zarówno jednoznaczną jak i wieloznaczną.
- Języki, dla których nie można zdefiniować gramatyki jednoznacznej (istnieją takie) nazywamy **istotnie wieloznacznymi**.
- Problem jednoznaczności i wieloznaczności jest w ogólnym przypadku nierozstrzygalny.

Jeżeli w produkcji gramatyki ten sam symbol nieterminalny występuje zarówno po lewej jak i po prawej stronie symbolu produkcji to sytuację taką nazywamy **rekursją**.

Gramatyka bezpośrednio rekursywna lewostronnie to gramatyka zawierająca produkcje postaci

$$A \to A\alpha$$

Gramatyka bezpośrednio rekursywna prawostronnie to gramatyka zawierająca produkcje postaci

$$A \to \alpha A$$

Gramatyka bezpośrednio rekursywna to gramatyka zawierająca produkcje postaci

$$A \to \alpha A \beta$$

Podobnie definiujemy gramatyki **pośrednio** rekursywne (rekursywne lewostronnie, rekursywne prawostronnie).

Metody rozbioru gramatycznego

Analiza prawostronna — analizowane są symbole z prawej strony

Analiza lewostronna — analizowane są symbole poczynając od lewej strony, bardziej naturalna [2].

Metoda generacyjna — rozpoczynając od symbolu wyróżnionego gramatyki stosujemy kolejne produkcje od najbardziej na lewo położonego symbolu nieterminalnego aktualnej formy zdaniowej.

Metoda redukcyjna — poprzez kolejne redukcje analizowanej formy zdaniowej dochodzimy do symbolu wyróżnionego gramatyki.

[2] Arabowie pewnie mają inne zdanie na ten temat ...

Wyprowadzenie bezpośrednie $xUy \Rightarrow xuy$ jest nazywane **kanonicznym** jeśli napis y zawiera tylko symbole terminalne.

Wyprowadzenie $w \stackrel{+}{\Rightarrow} u$ nazywane jest kanonicznym, jeśli każde wchodzące w jego skład wyprowadzenie bezpośrednie jest kanoniczne.

Każde zdanie (lecz nie każda forma zdaniowa) ma przynajmniej jedno wyprowadzenie kanoniczne.

Forma zdaniowa dla której istnieje wyprowadzenie kanoniczne jest nazywana **kanoniczną formą zdaniową**.

1.7 Automaty

Automatem deterministycznym A_D (lub A) nazywamy piątkę $A_D = \langle N, V, M, S, Z \rangle$, gdzie

- N — zbiór stanów (alfabet nieterminalny automatu),
- V — zbiór wejść (alfabet terminalny automatu),
- M — funkcja przejścia automatu, $M : (N \times V) \to N$, określa dla danego stanu automatu i wejścia do jakiego stanu automat ma przejść,
- S — stan początkowy automatu,
- Z — zbiór stanów końcowych (niepusty podzbiór symboli nieterminalnych).

Funkcja przejścia w sposób jednoznaczny określa nowy stan $M(Q, x) = R$. Można określić działanie automatu dla danego napisu wejściowego t rozszerzając definicję funkcji M:

- $M(Q, \varepsilon) = Q$ dla każdego $Q \in N$,
- $M(Q, Tt) = M(M(Q, T), t)$ dla $t \in V^*, T \in V$.

Mówimy, że napis t jest **akceptowany przez automat** jeżeli $M(S, t) \in Z$ (po „przetworzeniu" wejścia automat znajduje się w jednym ze stanów końcowych).

Językiem akceptowanym przez automat (ozn. $L(A)$) nazywamy zbiór

$$L(A) = \{x \in V^* : M(S, x) \in Z\}$$

Stan nieosiągalny danego automatu to stan dla którego nie istnieje ścieżka prowadząca od stanu początkowego do niego.

Stany nierozróżnialne (dwa lub więcej) to stany, które można zastąpić pojedynczym stanem wraz z odpowiednią modyfikacją wejść tak, by nowy automat był równoważny wyjściowemu.

Automat zredukowany to automat nie zawierający stanów nieosiągalnych i nierozróżnialnych.

Automatem niedeterministycznym A_N (lub A) nazywamy piątkę $A_N = \langle N, V, M, S, Z \rangle$, gdzie

- N — zbiór stanów (alfabet nieterminalny automatu),
- V — zbiór wejść (alfabet terminalny automatu),
- M — funkcja przejścia automatu, $M : (N \times V) \to 2^N$, określa dla danego stanu automatu i wejścia do jakich stanów automat może przejść,

- S — stan początkowy automatu,
- Z — zbiór stanów końcowych (niepusty podzbiór symboli nieterminalnych).

Dla automatu niedeterministycznego inaczej rozszerzamy funkcję M:

- $M(Q, \varepsilon) = \{Q\}$ dla każdego $Q \in N$,
- $M(Q, Tt) = \bigcup_{P \in M(Q,T)} M(P, t)$ dla $t \in V^*, T \in V$.

Przyjmujemy też

$$M(\{P_1, P_2, \ldots P_n\}, t) = \bigcup_{i=1}^{n} M(P_i, t)$$

Mówimy, że napis t jest **akceptowany** przez automat, jeżeli $M(S, t) \cap Z \neq \emptyset$.

1.7.1 Przejście od automatu niedeterministycznego do deterministycznego

Jeśli $A_N = \langle N, V, M, S, Z \rangle$ jest automatem niedeterministycznym akceptującym język L, to $A_D = \langle N', V', M', S', Z' \rangle$ będzie automatem deterministycznym dla którego $L(A_N) = L(A_D)$ jeśli określimy go następująco:

- Jeśli $N = \{a_1, a_2, \ldots, a_n\}$, to $N' = 2^N = \{a'_1, a'_2, \ldots, a'_{2^n}\}$, gdzie $a'_1 = \emptyset, a'_2 = \{a_1\}, a'_3 = \{a_2\}, \ldots, a'_{2^n} = N = \{a_1, a_2, \ldots, a_n\}$,
- $V' = V$,
- $M'(\{a_1, a_2, \ldots, a_i\}, t) = M(\{a_1, a_2, \ldots, a_i\}, t) = \bigcup_{k=1}^{i} M(a_k, t)$,
- $S' = S$.
- $Z' = Z$.

Krótko mówiąc, każdy zbiór stanów automatu niedeterministycznego reprezentujemy pojedynczym stanem automatu deterministycznego.

Otrzymany automat deterministyczny będzie na ogół zawierał stany nieosiągalne i nierozróżnialne.

1.7.2 Przejście od gramatyki liniowej jednostronnej do automatu akceptującego język tej gramatyki

Jeśli $G_L = \langle N, V, P, S \rangle$ jest gramatyką liniową lewostronną generującą język L, to $A_N = \langle N', V', M', S', Z' \rangle$ jest automatem niedeterministycznym dla którego $L(G_L) = L(A_N)$ jeśli określimy go następująco:

1. Każdą produkcję $Q ::= Rt_1t_2 \ldots t_j$ gdzie $\forall_{i \in \{1,2,\ldots j\}} t_i \in V$, jeżeli $j > 1$, to rozbijamy taką produkcję na następujące [3]:
   - $Q ::= Q_{j-1}t_j$
   - $Q_{j-1} ::= Q_{j-2}t_{j-1}$
     $\vdots$
   - $Q_2 ::= Q_1t_2$

[3] Krok potrzebny, choć u Klimka go brak. Dodane za `http://wazniak.mimuw.edu.pl/index.php?title=J%C4%99zyki%2C_automaty_i_obliczenia/Wyk%C5%82ad_8:_Dalsze_algorytmy_dla_j%C4%99zyk%C3%B3w_regularnych._Problemy_rozstrzygalne`

- $Q_1 ::= Rt_1$

2. Operując na poprawionej gramatyce budujemy automat:
   - $N' = N \cup \{pocz.\}$,
   - $V' = V$,
   - jeśli $Q, R \in N$ oraz $t \in V$, to
     - $(Q ::= t) \in P \Rightarrow M'(pocz., t) = Q$
     - $(Q ::= Rt) \in P \Rightarrow M'(R, t) = Q$
     - produkcje $(Q ::= \varepsilon) \in P$ ignorujemy [4].
   - $S' = \{pocz.\}$,
   - $Z' = S$.

Krótko mówiąc automat „czyta" to co jest po prawej stronie produkcji i przechodzi do lewej strony produkcji.

1.7.3 Redukowanie automatu

$x \in V^*$ rozróżnia stany q_1, q_2 automatu $A = \langle Q, V, \delta, q_0, F\rangle \overset{df}{\Leftrightarrow}$

$$\begin{cases} (q_1, x) \overset{*}{\vdash} (q_3, \varepsilon) \\ (q_1, x) \overset{*}{\vdash} (q_4, \varepsilon) \\ (q_3 \in F \wedge q_4 \notin F) \vee (q_3 \notin F \wedge q_4 \in F) \end{cases}$$

[5]

k-nierozróżnialność. Dwa stany q_1, q_2 są k-nierozróżnialne, co oznaczamy $q_1 \overset{k}{\equiv} q_2$ wtedy i tylko wtedy gdy nie istnieje słowo długości co najwyżej k, które je rozróżnia ($\neg\exists_{x \in V^*, |x| \leqslant k}$: x rozróżnia q_1 i q_2).

Nierozróżnialność. Dwa stany q_1, q_2 są nierozróżnialne (ozn. $q_1 \equiv q_2$) wtedy i tylko wtedy gdy $\forall_{k \geqslant 0} : q_1 \overset{k}{\equiv} q_2$.

Nieosiągalność. Stan q jest nieosiągalny wtedy i tylko wtedy gdy $\neg\exists_{x \in V^*} : (q_0, x) \overset{*}{\vdash} (q, \varepsilon)$ (gdzie q_0 to stan początkowy automatu).

Automat jest zredukowany wtedy i tylko wtedy gdy nie ma stanów nieosiągalnych i żadne dwa jego stany nie są nierozróżnialne. [6]

---

[4] Zgodnie z `http://wazniak.mimuw.edu.pl/index.php?title=J%C4%99zyki%2C_automaty_i_obliczenia/Wyk%C5%82ad_8:_Dalsze_algorytmy_dla_j%C4%99zyk%C3%B3w_regularnych._Problemy_rozstrzygalne` dla gramatyk liniowych prawostronnych dodawalibyśmy Q do stanów końcowych automatu. Ale mamy gramatykę lewostronną. Niby drobiazg, ale u Klimka nie ma, a moim zdaniem warto wspomnieć.

[5] Z tego co rozumiem to $\overset{*}{\vdash}$ oznacza przeczytanie przez automat zadanego słowa. W definicji chodzi o to, że istnieje słowo, w przypadku jednego stanu prowadzi do jakiegoś stanu końcowego, a dla drugiego stanu nie.
e ze slajdów z wykładu powinno być epsilonem, błąd jest w wykładzie.

[6] Ktoś mógłby chcieć powiedzieć, że automat jest zredukowany gdy wszystkie stany są osiągalne i każde dwa są rozróżnialne, ale nie definiowaliśmy osiągalności i rozróżnialności tylko nieosiągalność i nierozróżnialność!

Jeżeli automat ma n stanów ($\overline{\overline{Q}} = n$), to $(n-2)$-nierozróżnialność jest równoważna nierozróżnialności ($q_1 \equiv q_2 \Leftrightarrow q_1 \overset{n-2}{\equiv} q_2$).

Równoważna (rekurencyjna) definicja k-nierozróżnialności:

$$q_1 \overset{0}{\equiv} q_2 \Leftrightarrow (q_1 \in F \wedge q_2 \in F) \vee (q_1 \notin F \wedge q_2 \notin F)$$
$$q_1 \overset{k}{\equiv} q_2 \Leftrightarrow q_1 \overset{k-1}{\equiv} q_2 \wedge \forall_{a \in V} : \delta(q_1, a) \overset{k-1}{\equiv} \delta(q_2, a)$$

Teoriomnogościowe własności relacji nierozróżnialności:

- $(\overset{0}{\equiv}) \supset (\overset{1}{\equiv}) \supset \cdots \supset (\overset{n-2}{\equiv}) = (\equiv)$, gdzie $n = \overline{\overline{Q}}$.
- $(\overset{i}{\equiv}) = (\overset{i+1}{\equiv}) \Longrightarrow (\overset{i}{\equiv}) = (\equiv)$.
- Relacja $\overset{k}{\equiv}$ jest relacją równoważności (jest zwrotna, symetryczna i przechodnia).
- Klasa równoważności stanu q: $[q]_{\equiv} = \{q' \in Q : q' \equiv q\}$.

Niech dany będzie automat deterministyczny $A = \langle Q, V, \delta, q_0, F\rangle$ Równoważny mu automat zredukowany A' określamy następująco: $A' = \langle Q', V, \delta', q_0', F'\rangle$ gdzie

- $Q' = Q/_{\equiv}$ — zbiór klas abstrakcji relacji nierozróżnialności.
- $\delta'([p], a) = [\delta(p, a)]$
- $q_0' = [q_0]$
- $F' = \{[q] : q \in F\}$

Jak w praktyce sprawdzamy nierozróżnialność? Tworzymy tabelkę $n \times n$ w której będziemy wpisywać liczby 0, 1, 2, ... gdy odpowiednie stany będą 0-, 1-, ... nierozróżnialne. k-nierozróżnialność będziemy sprawdzać z definicji rekurencyjnej, tylko dla stanów $(k-1)$-nierozróżnialnych. Będziemy to kontynuować do $(n-2)$-nierozróżnialności lub stwierdzenia, że zwiększenie k nic nie wnosi (z drugiej własności relacji nierozróżnialności to wystarczy).

1.8 Wyrażenia regularne

Jeżeli $S = \{s_1, s_2, \ldots, s_n\}$ jest zbiorem pewnych symbolu należących do Vm a $\emptyset$ jest zbiorem pustym, zaś ε napisem pustym, to regułami tworzenia wyrażeń regularnych W_R są następujące zdania:

- symbole $\emptyset$, ε oraz $s_1, s_2, \ldots, s_n$ są wyrażeniami regularnymi,
- jeżeli p oraz q są wyrażeniami regularnymi, to są nimi też $pq, (p|q), (p)^*, (p)^+$.

1.8.1 Przejście od wyrażenia regularnego do automatu niedeterministycznego

Jeżeli W_R jest wyrażeniem regularnym generującym pewien ustalony język L, to $A_N = \langle N', V', M', S', Z'\rangle$ jest automatem niedeterministycznym dla którego zachodzi $L(W_R) = L(A_N)$, jeżeli zostanie od określony następująco:

- jeżeli $R_0 = \varepsilon$
  to wtedy $A_0 = \langle \{q\}, V, \delta : N \times V \ni (n, v) \to \emptyset \in 2^N, q, \{q\}\rangle$,

  Krótko mówiąc, mamy jeden jedyny stan, automat niezależnie od tego, co jest na wejściu przechodzi w maliny (w pusty zbiór stanów, nie akceptujemy wejścia).

- jeżeli $R_0 = a, a \in V$
  to wtedy $A_0 = \langle \{q_1, q_2\}, V, \delta_0, q_1, \{q_2\} \rangle$
  przy czym $\delta_0(q_1, a) = \{q_2\}$, a dla pozostałych argumentów funkcja przejścia przyjmuje jako wartość zbiór pusty.

  Krótko: automat zaczyna w jednym stanie, jak na wejściu pojawi się a, to przechodzi do drugiego. Jak pojawi się coś innego niż a lub więcej niż jeden symbol, to trafiamy w maliny i uznajemy, że wejście jest fuj.

  Klimek w wykładach ma funkcję przejścia postaci zbioru pustego, ale to jest zdecydowanie niepoprawny zapis. Funkcja z definicji musi być określona na całym zbiorze argumentów, który u nas jest niepusty, więc nie może ona być zbiorem pustym (choć zbiór pusty jest jej wartością na każdym elemencie; to może nie być do końca intuicyjne, ale takie są podstawowe prawidła teorii mnogości).

- jeżeli $R_0 = R_1|R_2$ przy czym $A_1 = \langle Q_1, V, \delta_1, q_1, F_1 \rangle : L(A_1) = L(R_1)$ oraz $A_2 = \langle Q_2, V, \delta_2, q_2, F_2 \rangle : L(A_2) = L(R_2)$ oraz $Q_1 \cap Q_2 = \emptyset$
  to wtedy $A_0 = \langle Q_1 \cup Q_2 \cup \{q_0\}, V, \delta_0, q_0, F_0 \rangle$
  przy czym

  - $\delta_0 = \delta_1 \cup \delta_2 \cup \bigcup_{a \in V : \delta_1(q_1,a) \neq \emptyset \vee \delta_2(q_2,a) \neq \emptyset} \{((q_0, a), \delta_1(q_1, a) \cup \delta_2(q_2, a))\}$ [7]
  - jeśli $q_1 \notin F_1 \wedge q_2 \notin F_2$ to $F_0 = F_1 \cup F_2$, a w przeciwnym przypadku $F_0 = F_1 \cup F_2 \cup \{q_0\}$

  Tutaj zaczyna być ciekawie. Chcemy zrobić automat akceptujący to co akceptuje jeden z automatów A_1, A_2. W tym celu zbieramy wszystkie stany obu automatów i gratis dorzucamy nowy początkowy. Tyle nam będzie potrzebnych. Automat startuje w tym dorzuconym i można powiedzieć „kradnie" przejścia stanom q_1, q_2 początkowym dla automatów wyjściowych. Dzięki temu otrzymujemy taki automat-hybrydę, który może zachowywać się zarówno jak pierwszy (wykorzystując przejścia ze stanu q_1), jak również i tak jak drugi. Co istotne, nie zabieramy przejść wiodących do stanów q_1, q_2 — gdy raz opuścimy stan q_0, zachowujemy się zupełnie jak jeden z danych automatów.

  Jest tu tylko jeden haczyk: jeśli któryś z automatów akceptował słowo puste, to my też powinniśmy, dlatego w takim przypadku musimy dodać nasz sztuczny stan początkowy do stanów końcowych — w końcu jak dostaniemy słowo puste na wejściu, to w ogóle się nie ruszymy z niego!

- jeżeli $R_0 = R_1 R_2$ przy czym $A_1 = \langle Q_1, V, \delta_1, q_1, F_1 \rangle : L(A_1) = L(R_1)$ oraz $A_2 = \langle Q_2, V, \delta_2, q_2, F_2 \rangle : L(A_2) = L(R_2)$ oraz $Q_1 \cap Q_2 = \emptyset$
  to wtedy $A_0 = \langle Q_1 \cup Q_2, V, \delta_0, q_0, F_0 \rangle$
  przy czym

  -

$$\delta_0(q, a) = \begin{cases} \delta_1(q, a) \cup \delta_2(q_2, a) & \text{jeśli } q \in F_1 \\ \delta_1(q, a) & \text{jeśli } q \notin F_1 \wedge q \in Q_1 \\ \delta_2(q, a) & \text{jeśli } q \in Q_2 \text{ (a co za tym idzie także } q \notin F_1) \end{cases}$$

[7] Dość mocno przeredagowane, oryginalna notacja pewnie wystraszyłaby przeciętnego matematyka bardziej niż niejeden horror. Generalnie chodzi o to, że bierzemy wszystkie przejścia z obu maszyn i dodajemy przejścia od nowego stanu początkowego do tego, do czego mogliśmy dojść ze stanów początkowych dwóch maszyn wyjściowych.

–

$$F_0 = \begin{cases} F_2 & \text{jeśli } q_2 \notin F_2 \\ F_1 \cup F_2 & \text{jeśli } q_2 \in F_2 \end{cases}$$

Tutaj potrzebujemy zbudować automat który najpierw będzie się zachowywał jak A_1, potem zaś jak A_2. Co zatem robimy? Jeśli jesteśmy w jakimś stanie pierwszego automatu to postępujemy tak jak on — chyba, że jest to stan końcowy tego automatu, co umożliwia nam jego opuszczenie i przejście do automatu A_2 na podobnej zasadzie co przy powyższym punkcie, po prostu przejmujemy przejścia ze stanu początkowego drugiej maszyny. Taki standardowy trick. Działanie w obrębie drugiej maszyny już jest proste.

Jeśli chodzi o stany końcowe, to generalnie chcemy kończyć w stanach końcowych drugiej maszyny, chyba, że akceptuje ona słowo puste — wtedy również w stanach końcowych maszyny pierwszej.

- jeżeli $R_0 = R_1^*$ przy czym $A_1 = \langle Q_1, V, \delta_1, q_1, F_1 \rangle : L(A_1) = L(R_1)$,
  to wtedy $A_0 = \langle Q_1 \cup \{q_0\}, V, \delta_0, q_0, F_1 \cup \{q_0\} \rangle$
  przy czym
  $$\delta_0(q,a) = \begin{cases} \delta_1(q_1, a) & \text{jeśli } q = q_0 \\ \delta_1(q,a) \cup \delta_1(q_1,a) & \text{jeśli } q \in F_1 \\ \delta_1(q,a) & \text{jeśli } q \neq q_0 \wedge q \notin F_1 \end{cases}$$

- jeżeli $R_0 = R_1^+$ przy czym $A_1 = \langle Q_1, V, \delta_1, q_1, F_1 \rangle : L(A_1) = L(R_1)$,
  to wtedy $A_0 = \langle Q_1, V, \delta_0, q_1, F_1 \rangle$
  przy czym
  $$\delta_0(q,a) = \begin{cases} \delta_1(q,a) \cup \delta_1(q_1,a) & \text{jeśli } q \in F_1 \\ \delta_1(q,a) & \text{jeśli } q \notin F_1 \end{cases}$$

  Te dwa przypadki są generalnie podobne do tego co było wcześniej, ich analizę pozostawiam jako dobre ćwiczenie przed egzaminem :).

1.9 Gramatyki $LL(k)$

Analizatory $LL(k)$ są analizatorami generacyjnymi ze stosem.

- Pierwsza litera L odnosi się do kierunku analizowania wejścia (od lewej do prawej).
- Druga litera L odnosi się do wywodu (lewostronny).
- k wskazuje ile symboli wprzód musi być przeglądanych, aby jednoznacznie określić produkcję niezbędną do poprowadzenia wywodu.
- Analizatory pracują w sposób deterministyczny, co pozwala przeanalizować poprawny program w czasie liniowo zależnym od jego długości, nie wykonując przy tym nawrotów.
- Błąd składniowy rozpoznawany jest bezpośrednio po natrafieniu na symbol, który nie należy do programu poprawnego.
- $LL(0)$ oznacza, że analizator nie patrzy w ogóle na wejście.
- O analizatorach $LL(k)$ mówimy, że nie znają historii.

Funkcja $PRFX_k(\alpha)$ określa zbiór prefiksów symboli podstawowych o długości co najwyżej k wyprowadzalnych z danego α:

$$PRFX_k(\alpha) = \left\{x \in V^* : \left(\alpha \overset{*,l}{\Rightarrow} x\beta \wedge |x| = k\right) \vee \left(\alpha \overset{*,l}{\Rightarrow} x \wedge |x| < k\right)\right\}$$

[8]

Przykładowo dla produkcji $A \to aB|bB|cD$ $PRFX_1(A) = \{a, b, c\}$.

Funkcja $FOLLOW_k(\beta)$ określa zbiór możliwych następników β (dla przypomnienia — S to symbol początkowy gramatyki generującej dany język):

$$FOLLOW_K(\beta) = \left\{\omega : S \overset{*}{\Rightarrow} \alpha\beta\gamma \wedge \omega \in PRFX_k(\gamma)\right\}$$

W szczególności $FOLLOW_1(\beta)$ określa zbiór symboli mogących pojawić się po łańcuchu β.

Gramatyka $LL(k)$ Jeżeli gramatyka G jest bezkontekstowa oraz dla każdych dwóch wywodów zachodzi

$$S \overset{*,l}{\Rightarrow} \omega A\alpha \overset{l}{\Rightarrow} \omega\beta\alpha \overset{*,l}{\Rightarrow} \omega x$$
$$S \overset{*,l}{\Rightarrow} \omega A\alpha \overset{l}{\Rightarrow} \omega\gamma\alpha \overset{*,l}{\Rightarrow} \omega y$$

to gramatykę nazywamy **gramatyką** $LL(k)$ gdy

$$PRFX_k(x) = PRFX_k(y) \Longrightarrow \beta = \gamma$$

gdzie $x, y \in V^*, \alpha, \beta, \gamma \in (N \cup V)^*, A \in N$.

Opisałbym to tak: powiedzmy, że przeprowadzamy wywód lewostronny. Dochodzimy do momentu zastępowania nieterminala A. Możemy go jednak zastąpić na kilka sposobów. No to rozważamy wszystkie po kolei aż uzyskamy napisy w których wszystko na prawo od omegi nie ma już nieterminalni. Gramatyka jest $LL(k)$, jeśli na podstawie k symboli mogących wystąpić przed x bądź y możemy wywnioskować, którą produkcję z A wybraliśmy.

Gramatyki należące do klasy LL(k) oznaczamy $G_{LL(k)}$, natomiast języki generowane przez te gramatyki jako $L_{LL(k)}$.

1.9.1 Sprawdzanie, czy gramatyka jest klasy $LL(k)$

Jeśli $G = \langle N, V, P, S\rangle$ jest gramatyką bezkontekstową, to G jest gramatyką LL(k) wtedy i tylko wtedy gdy dla każdej pary produkcji $A \to \beta, A \to \gamma \in P$ oraz dla $wA\alpha$ takiego, że $A \overset{*,l}{\Rightarrow} wA\alpha$ gdzie $w \in V^*, A \in N, \alpha \in (N \cup V)^*$ zachodzi

$$PRFX_K(\beta\alpha) \cap PRFX_k(\gamma\alpha) = \emptyset$$

Uproszczona definicja gramatyki $LL(1)$:

Bezkontekstowa gramatyka G jest $LL(1)$ wtedy i tylko wtedy gdy dla każdej produkcji $A \to \alpha_1|\alpha_2|\ldots|\alpha_n$ zachodzą warunki

- $PRFX_1(\alpha_1), PRFX_1(\alpha_2), \ldots, PRFX_1(\alpha_n)$ są parami rozłączne

[8] W $\overset{*,l}{\Rightarrow}$ l nad strzałką oznacza wywód lewostronny (z wykładu 9).

- $\alpha_i \stackrel{*}{\Rightarrow} \varepsilon \Longrightarrow \forall_{j\in\{1,2,\ldots,n\}\setminus\{i\}} PRFX_1(\alpha_j) \cap FOLLOW_1(A) = \emptyset$

Fakty o gramatykach $LL(k)$:

- Gramatyki $LL(k)$ są jednoznaczne.
- Klasa $LL(k-1)$ jest istotną podklasą języków $LL(k)$. Jeżeli język jest klasy $LL(k-1)$, to jest też klasy $LL(k)$.
- Dla każdej klasy $LL(k)$ istnieje język tej klasy nie należący do klasy $LL(k-1)$ ($G_{LL(k-1)} \subsetneq G_{LL(k)}$ oraz $L_{LL(k-1)} \subsetneq L_{LL(k)}$).
- Istnieje język L' bezkontekstowy i zdeterminowany, taki, że nie istnieje gramatyka $LL(k)$ generująca ten sam język.
- Jeżeli gramatyka G jest lewostronnie rekursywna, to nie jest gramatyką $LL(k)$ (trzeba usunąć lewostronną rekursję).

1.9.2 Automat rozkładu gramatyki $LL(k)$

Automat rozkładu dla gramatyki $LL(k)$ jest automatem ze stosem, czytającym k kolejnych symboli z wejścia. Po przyłożeniu do łańcucha wejściowego w daje jako wynik ciąg produkcji niezbędnych do przeprowadzenia wywodu. Zapisujemy to jako $A(w) = \Pi$, gdzie $w \in V^*, \Pi = p_1 p_2 \ldots p_n$.

Oznaczenia:

w — łańcuch symboli na wejściu,

x — nieprzeanalizowana (niewykorzystana) część łańcucha wejściowego,

Π — wyjście (ciąg numerów produkcji),

Γ — stos z odkładanymi symbolami nieterminalnymi i terminalnymi,

\$ — dno stosu,

X — symbol z wierzchołka stosu,

α — zawartość stosu (bez wierzchołka),

M — funkcja (tablica) przejścia automatu.

Funkcja sterująca:

$$M : (\Gamma \cup \{\$\} \times V^k) \rightarrow \left\{ \begin{array}{ll} (\beta, i) & -\beta \in \Gamma, \text{i - numer produkcji} \\ pop & -\text{zdejmij ze stosu} \\ accept & -\text{akceptuj} \\ error & -\text{błąd} \end{array} \right\}$$

Konfiguracja automatu: $(x, X\alpha, \Pi)$ gdzie x - część wejścia, $X\alpha$ to stos z symbolem na wierzchołku (X) i resztą stosu (α), Π - ciąg numerów produkcji.

Opis automatu:

Jeżeli $M(X, U) = (\beta, i)$ to stan zmienia się następująco: $(x, X\alpha, \Pi) \longmapsto (x, \beta\alpha, \Pi i)$
gdzie $U \in PRFX_k(x), i$ to numer produkcji postaci $p_i = (X \rightarrow \beta) \in P$
(zastępujemy na stosie symbol nieterminalny prawą stroną odpowiedniej produkcji — tu następuje wypełnianie stosu. Dla przypomnienia początkową zawartością stosu jest symbol początkowy gramatyki),

Jeżeli $M(a,a) = pop$ to stan zmienia się następująco: $(ax', X\alpha, \Pi) \longmapsto (x', \alpha, \Pi)$
gdzie $x = ax', a \in V$
(zdejmujemy ze stosu symbol terminalny i przesuwamy głowicę na wejściu o jedno miejsce w prawo),

Jeżeli $M(\$, \varepsilon) = accept$ to stan musi mieć postać $(\varepsilon, \$, \Pi)$, kończymy działanie akceptując wejście, $w \in L(G)$, a Π jest wywodem w,

Jeżeli $M(x, \cdot) = error$ kończymy działanie nie akceptując wejścia, $w \notin L(G)$.

Działanie automatu można schematycznie opisać w następująco sposób:

$$(w, S\$, \varepsilon) \overset{*}{\longmapsto} (\varepsilon, \$, \Pi) \iff A(w) = \Pi$$

co możemy rozumieć jako „automat rozpoczyna ze słowem w na wejściu, Symbolem początkowym gramatyki na stosie i pustym ciągiem produkcji, następnie przetwarza podane słowo kończąc w stanie z całym słowem przeczytanym, pustym stosem i określonym ciągiem produkcji, zwracanym z automatu." Tak przynajmniej się dzieje, gdy słowo na wejściu istotnie należy do języka akceptowanego przez automat.

Budowa tablicy automatu (dla $LL(1)$):

- jeżeli $p_i = (A \to \alpha) \in P$ to $\forall_{a \in PRFX_1(\alpha), a \neq \varepsilon} : M(A,a) = (\alpha, i)$
  jeżeli $\varepsilon \in PRFX_1(\alpha)$ to $\forall_{b \in FOLLOW_1(A)} : M(A,b) = (\alpha, i)$
- $\forall_{a \in A} : M(a,a) = pop$
- $M(\$, \varepsilon) = accept$
- w pozostałe pola wpisać error

Jeżeli podczas analizowania danej produkcji następuje próba zapisania w już zapełnionej kratce nowej wartości, to oznacza to, iż nie mamy do czynienia z gramatyką $LL(1)$ [9]

1.9.3 Usuwanie lewostronnej rekursji i lewostronna faktoryzacja

Jeżeli gramatyka $G = \langle N, V, P, S \rangle$ jest bezkontekstowa i zawiera produkcje

$$\{A \to A\alpha_1 | \ldots | A\alpha_m | \beta_1 | \ldots | \beta_n\} \subset P$$

to gramatyka G' będzie gramatyką nie zawierającą lewostronnej rekursji generującą ten sam język, jeśli określimy ją następująco: $G' = \langle N \cup \{A'\}, V, P', S \rangle$ gdzie

$$P' = P \setminus \{A \to A\alpha_1 | \ldots | A\alpha_m | \beta_1 | \ldots | \beta_n\} \cup \begin{cases} A \to \beta_1 | \ldots | \beta_n | \beta_1 A' | \ldots | \beta_n A' \\ A' \to \alpha_1 | \ldots | \alpha_m | \alpha_1 A' | \ldots | \alpha_m A' \end{cases}$$

Obrazowo: w gramatyce G produkcje z A generowały po prawej stronie od siebie te różne alfy aż w końcu musieliśmy wybrać pojedynczą betę trafiającą na lewy kraniec (ewentualnie mogliśmy w ogóle zrezygnować z alf). W gramatyce G' usuwamy problematyczną produkcję i zastępujemy ją dwoma nowymi: jedna ma dać lewą betę i ewentualnie coś jeszcze, druga ma rozwinąć to „coś jeszcze" do ciągu alf, już korzystając z nieproblematycznej dla gramatyk $LL(k)$ rekursji prawostronnej.

[9] Ewentualnie zrobiliśmy błąd :P.

Przykład: likwidacja lewostronnej rekursji w gramatyce z produkcjami

$$E \to E + T|T$$
$$T \to T * F|F$$
$$F \to (E)|a$$

Po likwidacji mamy produkcje:

$$E \to T|TE' \quad E' \to +T| + TE'$$
$$T \to F|FT' \quad T' \to *F| * FT'$$
$$F \to (E)|a$$

Otrzymana gramatyka dalej jednak nie jest $LL(k)$. Musimy jeszcze zastosować tzw. **lewostronną faktoryzację**, coś jakby wyłączanie przed nawias. Kiedy to robimy? Zawsze, kiedy dwie produkcje z tego samego symbolu nieterminalnego zaczynają się tak samo (mają wspólny, niepusty prefiks). Schematycznie:

$$A \to \alpha\beta_1| \ldots |\alpha\beta_n|\gamma_1| \ldots |\gamma_r \Longrightarrow \begin{cases} A \to \alpha A'|\gamma_1| \ldots |\gamma_r \\ A' \to \beta_1| \ldots |\beta_n \end{cases}$$

gdzie α nie jest prefiksem żadnego słowa z $\gamma_1, \ldots, \gamma_r$. [10]

Po faktoryzacji mamy następujące produkcje:

$$E \to TT'' \quad T'' \to \varepsilon|E' \quad E' \to +TT''$$
$$T \to FF'' \quad F'' \to \varepsilon|T' \quad T' \to *FF''$$
$$F \to (E)|a$$

W analizatorach typu $LL(k)$ ε-produkcje są dopuszczalne, warto jednak produkcje takie zastąpić produkcją pojedynczą. Można również w ogóle usunąć ε-produkcję.

Dla każdej gramatyki G można skonstruować gramatykę G' bez ε-produkcji taką, że $L(G') = L(G) \setminus \{\varepsilon\}$. Jeżeli gramatyka G jest $LL(k)$, to G' jest $LL(k+1)$. Jeżeli gramatyka G jest $LL(k)$ bez ε-produkcji i $k \geqslant 2$, to istnieje G' będąca $LL(k-1)$ taka, że $L(G) = L(G')$.

1.9.4 Usuwanie ε-produkcji

[źródło: http://eli.thegreenplace.net/2010/02/08/removing-epsilon-productions-from-context-free-grammars/]

Aby usunąć z gramatyki ε-produkcje, wykonujemy następujące kroki do momentu zatrzymania (dla przykładu będę je od razu wykonywał dla podanej gramatyki o symbolu początkowym A):

- $A \to B|BcB|Ab$
- $B \to \varepsilon|Ba$
- $C \to \varepsilon$
- $D \to Cf$

1. Wybieramy wszystkie ε-produkcje z gramatyki. W naszym przypadku jest tylko jedna: $B \to \varepsilon$.

[10] Trochę to przerobiłem względem wykładów Klimka, na wzór `http://web.cs.wpi.edu/~gpollice/cs544-f05/CourseNotes/pdf/CS544%20Class%205.pdf`. W ten sposób wydaje mi się to jaśniej opisane, widać, że mimo występowania niefaktoryzowalnych produkcji faktoryzujemy te, które się da.

2. Usuwamy wszystkie znalezione $\varepsilon - produkcje$ za wyjątkiem tej z symbolem początkowym po lewej stronie. Zmienia to język generowany przez gramatykę, ale tym zajmiemy się za chwilę.

3. Tam, gdzie usunęliśmy ε-produkcje, wprowadzamy je ponownie, ale w miejscach występowania danego nieterminala po prawej stronie produkcji. Jeśli dany nieterminal występuje po prawej stronie więcej niż raz, to pokrywamy wszystkie możliwe kombinacje ($A \to BcB$ zamienia się w $A \to BcB|\varepsilon cB|Bc\varepsilon|\varepsilon c\varepsilon$). Ale uwaga: jeżeli dany nieterminal przechodzi tylko w ε, to nie dodajemy produkcji mających ten nieterminal po prawej stronie, a jedynie te z ε (patrz: nieterminal C) W naszym przypadku otrzymujemy gramatykę:

   - $A \to B|BcB|Ab|\varepsilon|c|Bc|cB$
   - $B \to Ba|a$
   - $D \to f$

4. Jeśli dalej jest jakaś ε-produkcja, to wracamy do kroku 1. Ale jest tu pewien problem. Istnieją gramatyki, które zapętlają nasz algorytm albo wprowadzają pewne efekty utrudniające określenie warunku stopu, albo w ogóle gramatyka tylko eksploduje produkcjami ($A \to A|AA|B|BB|a|\varepsilon, B \to A|AA|B|BB|a|\varepsilon$). Nie udało mi się odszukać dyskusji o tym traktującej, ale algorytm często działa. W naszym przypadku została już tylko ε-produkcja od symbolu początkowego. Została dodana w tym kroku, więc należy wykonać jeszcze jeden, aby się upewnić, że jej usunięcie nie spowoduje zmiany gramatyki. Na koniec powinniśmy dostać

   - $A \to B|BcB|Ab|c|Bc|cB|b$ (dodajemy ostanie b, usuwamy ε)
   - $B \to Ba|a$
   - $D \to f$

2 Zadania egzaminacyjne

2.1 Rok 2007

2.1.1 Termin ???

1. Podać gramatykę bezkontekstową generującą liczby dziesiętne ze znakiem lub bez znaku podzielne przez 5.

2. Podać gramatykę klasy 0, która produkuje te i tylko te słowa, które należą do zbioru $\{a, a^2, a^4, a^8 \dots\}$

3. Jaki język generuje gramatyka zawierająca produkcje: $H \to QRX, Q \to fQ, Q \to fR, RR \to X, fX \to f$. Symbol początkowy: H.

2.1.2 Termin ???

1. Definicja stanów nierozróżnialnych automatów.

2. Opisz:

   1) Formalna definicja gramatyki LL(k)

   2) interpretacja werbalna

3) interpretacja graficzna gramatyki

3. Gramatyka o produkcjach:

   1) $S \to LS'$
   2) $S' \to *S'$
   3) $S' \to \varepsilon$
   4) $L \to aL'$
   5) $L' \to (S)$
   6) $L' \to \varepsilon$

   Zbudować tablice parsera LL(1) i przeprowadzić parsing dla ciągu a(a*)*

2.2 Rok 2008

1. Podać gramatykę generującą liczby zmiennoprzecinkowe ze znakiem lub bez znaku tak, aby część całkowita nie zaczynała się zerem i była podzielna przez 5, a część ułamkowa nie kończyła się zerem (jedyny przypadek to 0,0 lub 0,543...).

2. Definicja normy zdaniowej, rozbioru kanonicznego i uchwytu oraz podać po prostym ich przykładzie.

3. Podać jaki język generuje gramatyka zawierająca produkcję...wychodziło $a^(3n)$.

4. Dla gramatyki o produkcjach:

   1) $S \to LS'$
   2) $S' \to *S'$
   3) $S' \to \varepsilon$
   4) $L \to aL'$
   5) $L' \to (S)$
   6) $L' \to \varepsilon$

   zbudować tablice parsera LL(1) i przeprowadzić parsing dla ciągu a(a*)*.

5. Definicja stanów nierozróżnialnych.

6. Podać gramatykę generującą dany język... chyba: $a * b^{n+m} * c^m, n \geqslant 1, m \geqslant 1$ albo coś w tym stylu.