Teoretyczne podstawy informatyki

Wykład 12: Opis wzorców: gramatyki

Opis wzorców polegający na na wykorzystaniu modelu definicji rekurencyjnych, nazywamy gramatyką bezkontekstową (ang. contex-free grammar). Jednym z ważnych zastosowań gramatyk są specyfikacje języków programowania. Gramatyki stanowią zwięzłą notacje opisu ich składni. Istnieje możliwość mechanicznego przekonwertowania gramatyki typowego języka programowania na *analizator składniowy (ang. parser)*, który stanowi jeden z *kluczowych elementów kompilatora takiego języka*. Analizator składniowy pozwala na zidentyfikowanie struktury programu źródłowego, często w postaci drzewa wyrażeń dla każdej instrukcji programu.

Gramatyki bezkontekstowe

Wyrażenia arytmetyczne można w naturalny sposób zdefiniować rekurencyjnie.
Weźmy pod uwagę wyrażenia arytmetyczne zawierające:

(a) Cztery operatory dwuargumentowe +, -, *, /
(b) Nawiasy służące do grupowania podwyrażeń
(c) Operandy które są liczbami

Tradycyjna definicja takich wyrażeń stanowi indukcje:

Podstawa:
Liczba jest wyrażeniem.

Indukcja:
Jeżeli E oznacza dowolne wyrażenie, to wyrażeniami są także wszystkie z poniższych elementów:

(1) **(E)**. Oznacza to ze wyrażenie można umieścić w nawiasach w wyniku czego otrzymuje się nowe wyrażenie.
(2) **E + E**. Oznacza to ze dwa wyrażenie połączone znakiem plus stanowią wyrażenie.
(3) **E-E**.
(4) **E*E**.
(5) **E/E**.

Powyższa indukcja indukuje język czyli zbiór ciągów znaków. Podstawa określa że każda liczba należy do tego języka. Reguła (1) określa, że jeżeli s jest ciągiem znaków należącym do omawianego języka, to także ciąg znaków objęty nawiasami należy do tego języka. Taki ciąg s jest poprzedzony znakiem nawiasu otwierającego, zaś po nim występuje znak nawiasu zamykającego. Reguły (2),(3),(4),(5) określają, że jeżeli s i t są dwoma ciągami znaków należącymi do języka, to należą do niego również ciągi znaków s+t, s-t, s*t, s/t.

Gramatyki pozwalają na zapisywanie takich reguł w sposób zwięzły i precyzyjny. Tak by wyglądał zapis definicji wyrażeń arytmetycznych

<Wyrażenie> → liczba
<Wyrażenie> → (< Wyrażenie>)
<Wyrażenie> → <Wyrażenie> + <Wyrażenie>
<Wyrażenie> → <Wyrażenie> - <Wyrażenie>
<Wyrażenie> → <Wyrażenie> * <Wyrażenie>
<Wyrażenie> → <Wyrażenie> / <Wyrażenie>

<Wyrażenie>

Ten symbol nosi nazwę kategorii syntaktycznej (ang. syntactic cathegory). Zastępuje on dowolny ciąg znaków należący do języka wyrażeń arytmetycznych.

→

Ten symbol należy traktować jako zwrot „może się składać z"

liczba

To pewien abstrakcyjny symbol zastępczy dla dowolnego ciągu znaków.

Terminologia

Istnieją trzy rodzaje symboli wykorzystywanych w gramatykach

metasymbol - sam w sobie nie ma żadnego znaczenia, oddziela definiowaną kategorię syntaktyczna od opisu sposobu w jaki ciąg znaków może tworzyć dana kategorię.

kategoria syntaktyczna - reprezentuje zbiór definiowanych ciągów znaków.

symbol terminalny - może być znakami, np. (, +, -) lub symbolem abstrakcyjnym

Gramatyka składa się z jednej lub większej liczby produkcji (ang. productions). Każda produkcja składa się z trzech części:

(1) Części nagłówkowej (ang. head), która jest kategoria syntaktyczna umieszczona po lewej stronie strzałki

(2) Metasymbolu (np. strzałki)

(3) Części zasadniczej (ang. body)

<Cyfra> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Liczba> → <Cyfra>

<Liczba> → <Liczba> <Cyfra>

<Wyrażenie> → <Liczba>

<Wyrażenie> → (< Wyrażenie>)

<Wyrażenie> → <Wyrażenie> + <Wyrażenie>

<Wyrażenie> → <Wyrażenie> - <Wyrażenie>

<Wyrażenie> → <Wyrażenie> * <Wyrażenie>

<Wyrażenie> → <Wyrażenie> / <Wyrażenie>

Gramatyka wyrażeń w której liczby zdefiniowano przy pomocy konstrukcji gramatycznych.

Gramatyka ciągów znaków o zbilansowanej liczbie nawiasów.

<Zbilansowany> → ε
<Zbilansowany> → (<Zbilansowany>) <Zbilansowany>

Gramatyka ciągów znaków o zbilansowanej liczbie nawiasów zdefiniowana na podstawie gramatyki wyrażeń arytmetycznych.

<ZbilansowaneW> → ε
<ZbilansowaneW> → (<ZbilansowaneW>)
<ZbilansowaneW> → <ZbilansowaneW> <ZbilansowaneW>

Jeżeli kategorie syntaktyczna <Wyrażenie> zastąpi się bardziej odpowiednią nazwą <ZbilanowaneW>, to otrzymuje się nową gramatykę, opisującą ciągi znaków o zbilansowanej liczbie nawiasów. Produkcje są naturalne. Gramatyki przedstawione powyżej definiują ten sam zbiór ciągów znaków.

Produkcje definiujące cześć instrukcji języka C

<Instrukcja> → while (warunek) <Instrukcja>
<Instrukcja> → if (warunek) <Instrukcja>
<Instrukcja> → if (warunek) <Instrukcja> else <Instrukcja>
<Instrukcja> → {<ListaInstr>};
<Instrukcja> → prostaInstr;
<ListaInstr> → ε
<ListaInstr> → <ListaInstr> <Instrukcja>

Można opisywać gramatycznie strukturę przebiegu sterowania występującą w językach takich jak C. Załóżmy istnienie abstrakcyjnych symboli terminalnych warunek oraz instrProsta. Pierwszy z nich oznacza wyrażenie warunkowe i można go zastąpić kategorią syntaktyczna <Warunek>. Symbol terminalny instrProsta określa instrukcję nie zawierającą zagnieżdżonych struktur sterujących, takich jak instrukcja przypisania, wywołania funkcji, odczytu, zapisu i skoku. Można zastąpić symbol terminalny kategorią syntaktyczną oraz rozszerzającymi ją produkcjami. Jako kategorii syntaktycznej instrukcji języka C będziemy używać kategorii <Instrukcja>.

Języki gramatyk

Gramatyka to definicja indukcyjna zawierająca zbiory znaków. W przypadku gramatyk jest rzeczą normalną definiowanie kilku kategorii syntaktycznych za pomocą jednej gramatyki. Dla każdej kategorii syntaktycznej <S> danej gramatyki, definiuje się język L(<S>) w sposób opisany poniżej.

Podstawa:

W przypadku każdej kategorii syntaktycznej <S> danej gramatyki, język L(<S>) jest zbiorem pustym.

Indukcja:

Załóżmy, że gramatyka posiada produkcję $\langle S \rangle \rightarrow X_1X_2...X_m$ danej gramatyki, gdzie każdy element X_i , dla i=1,2,....n, jest albo kategorią syntaktyczną, albo symbolem terminalnym. Dla każdego i=1, 2, ... , n wybieramy ciąg znaków si dla Xi w sposób następujący.

(1) Jeżeli Xi jest symbolem terminalnym, to Xi można użyć jedynie jako ciągu znaków si.

(2) Jeżeli Xi jest kategorią syntaktyczną, to wybieramy jako si dowolny ciąg znaków, o którym wiadomo już że należy do języka L(Xi).

Wówczas złożenie s1s2...sn stanowi ciąg znaków należący do języka L(<S>).

Jednym z metodycznych sposobów zaimplementowania takiej definicji jest wykonanie sekwencyjnego przebiegu przez produkcję gramatyki. W każdym przebiegu następuje uaktualnienie języka każdej kategorii syntaktycznej przy użyciu reguły indukcyjnej na wszystkie możliwe sposoby, tzn. dla każdego Xi będącego kategorią syntaktyczną wybieramy ciągi znaków ze zbioru L(<Xi>) na wszystkie możliwe sposoby.

(1) <Instrukcja> → while (warunek) <Instrukcja>
(2) <Instrukcja> → {<ListaInstr>};
(3) <Instrukcja> → prostaInstr;

(4) <ListaInstr> → <ListaInstr> <Instrukcja>
(5) <ListaInstr> → ε

Uproszczona gramatyka instrukcji

<I> → w c <I>
<I> → {<L>}
<I> → s;

<L> → <L> <I>
<L> → ε

Uproszczona notacja

Uproszczona gramatyka instrukcji

<I> → w c <I>
<I> → {<L>}
<I> → s;

<L> → <L> <I>
<L> → ε

Język definiowany przez gramatykę może być nieskończony, czyli nie ma możliwości wypisania wszystkich należących do niego znaków.

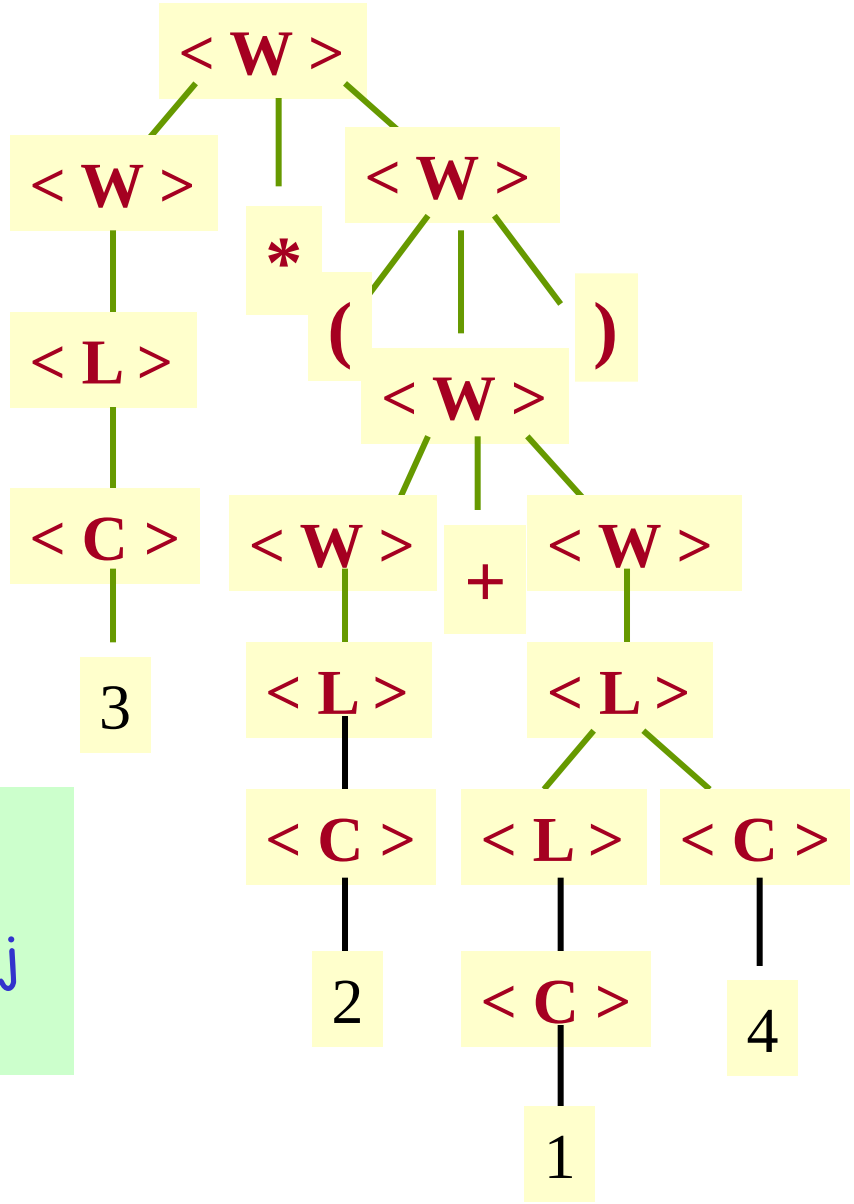
Nowe ciągi znaków dodawane w pierwszych trzech przebiegach

	I	L
Przebieg 1.	s ;	ε
Przebieg 2.	wcs ; {}	s ;
Przebieg 3.	wcwcs ; ws{} {s;}	wcs ; {} s ; s ; s ; wcs ; s ; {}

Drzewa rozbioru

Można pokazać ze ciąg s należy do języka L(<S>) pewnej kategorii syntaktycznej <S> w wyniku *powtarzalnego stosowania produkcji*. Rozpoczyna się od pewnych ciągów znaków wynikających z *produkcji bazowych*, czyli tych, które w swojej części zasadniczej nie posiadają żadnych kategorii syntaktycznych. Następnie należy „zastosować" produkcję wobec ciągów znaków które już otrzymano dla różnych kategorii syntaktycznych. Każde takie zastosowanie polega na podstawieniu ciągów znaków za występujące w części zasadniczej produkcji różne kategorie syntaktyczne, stąd skonstruowanie ciągu znaków należącego do kategorii syntaktycznej stanowiącej cześć nagłówkową. W końcu ciąg s konstruuje się poprzez zastosowanie produkcji posiadającej w części nagłówkowej kategorię <S>. Możemy ilustrować przynależność s do L(<S>) w formie drzewa, zwanego *drzewem rozbioru* lub *drzewem analizy składniowej* (ang. parse tree). *Wierzchołki drzewa rozbioru* etykietuje się albo symbolami terminalnymi, albo kategoriami syntaktycznymi, albo symbolem ε. *Liście* są etykietowane jedynie symbolami terminalnymi lub symbolem ε, zaś *wierzchołki wewnętrzne* są etykietowane jedynie kategoriami syntaktycznymi. Każdy wierzchołek wewnętrzny ν reprezentuje zastosowanie produkcji. Tzn. kategoria syntaktyczna etykietująca ν wierzchołek stanowi cześć nagłówkową produkcji. Etykiety potomków wierzchołka ν, od strony lewej do prawej, tworzą cześć zasadniczą tej produkcji.

<Cyfra> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<Liczba> → <Cyfra>
<Liczba> → <Liczba> <Cyfra>
<Wyrażenie> → <Liczba>
<Wyrażenie> → (< Wyrażenie>)
<Wyrażenie> → <Wyrażenie> + <Wyrażenie>
<Wyrażenie> → <Wyrażenie> - <Wyrażenie>
<Wyrażenie> → <Wyrażenie> * <Wyrażenie>
<Wyrażenie> → <Wyrażenie> / <Wyrażenie>



Drzewo rozbioru dla ciągu znaków
3 * (2 + 14)
przy użyciu gramatyki zdefiniowanej powyżej.

Konstruowanie drzew rozbioru

Każde drzewo rozbioru reprezentuje *ciąg symboli terminalnych s*, który nosi nazwę *wyniku* (ang. yield) drzewa. *Ciąg s składa się z etykiet liści drzewa* ułożonych w kolejności od strony lewej do prawej. Jeżeli drzewo posiada jeden wierzchołek, wierzchołek ten jest etykietowany symbolem terminalnym lub symbolem ε, ponieważ jest liściem. Jeżeli drzewo posiada więcej niż jeden wierzchołek, to korzeń zostaje zaetykietowany kategorią syntaktyczną, gdyż *korzeń drzewa* posiadającego dwa lub więcej wierzchołków *jest zawsze wierzchołkiem wewnętrznym*. Ta kategoria syntaktyczna zawsze zawiera wśród swoich ciągów znaków także wynik drzewa.

Definicja indukcyjna drzew rozbioru dla danej gramatyki

Podstawa:

Dla każdego symbolu terminalnego, np. x, danej gramatyki istnieje drzewo posiadające tylko jeden wierzchołek zaetykietowany jako x. Wynikiem takiego drzewa jest x.

Indukcja:

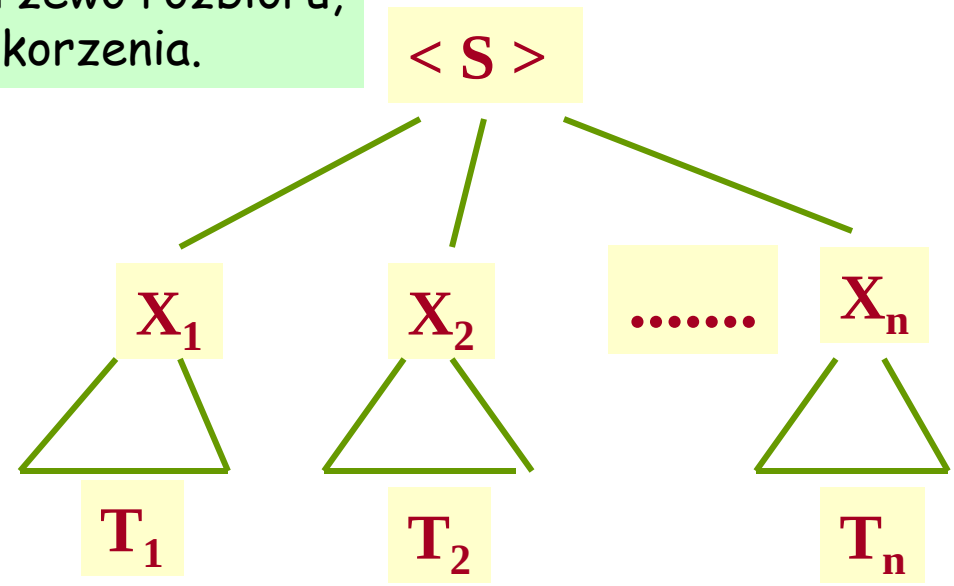
Załóżmy, ze istnieje produkcja $\langle S \rangle \rightarrow X_1X_2....X_m$, gdzie każdy z symboli X oznacza albo symbol terminalny, albo kategoria syntaktyczna.
Jeżeli n=0, czyli produkcja ma postać $\langle S \rangle \rightarrow \varepsilon$, to istnieje drzewo którego wynikiem jest ε, a korzeniem $\langle S \rangle$. Z uwagi na te produkcje ciąg ε należy do języka $L(\langle S \rangle)$.

Teraz załóżmy, ze <S> $\rightarrow X_1X_2.....X_n$ a $n \geq 1$. Dla każdego Xi możemy wybrać drzewo T_i w sposób następujący:

(1) Jeśli Xi jest symbolem terminalnym, musimy wybrać dla każdego wystąpienia tego symbolu terminalnego 1-wierzchołkowe drzewo zaetykietowane jako Xi. Jeżeli jeden lub więcej symboli X jest tym samym symbolem terminalnym, to musimy wybrać różne jednowierzchołkowe drzewa o tej samej etykiecie

(2) Jeżeli Xi jest kategorią syntaktyczną, możemy wybrać dowolne już skonstruowane drzewo rozbioru, takie, że posiada on Xi jako etykietę korzenia.

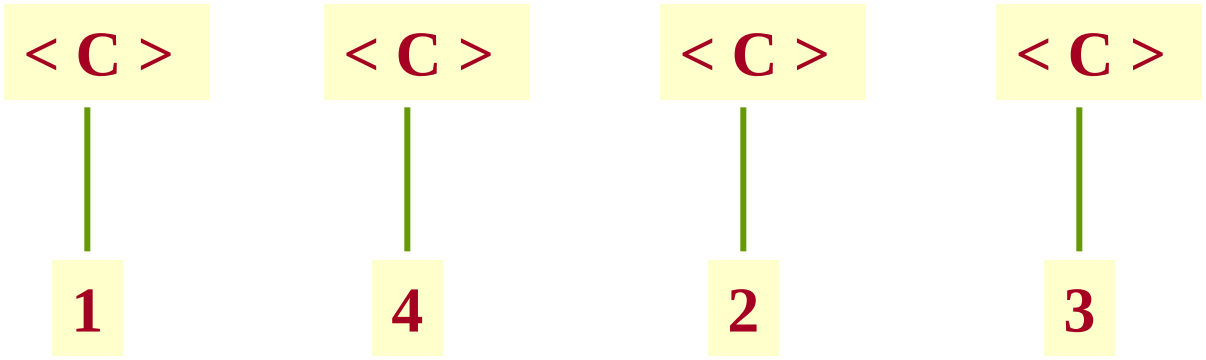
Następnie konstruujemy drzewo, tworząc korzeń zaetykietowany kategorią syntaktyczną <S> umieszczoną w części nagłówkowej produkcji oraz przypisujemy mu jako potomków korzenie drzew wybranych dla X1, X2, ... Xn.



Etapy konstruowania drzewa rozbioru 3* (2+14)

(a) Konstruujemy jednowierzchołkowe drzewo dla każdego symbolu terminalnego w drzewie. Używamy produkcji

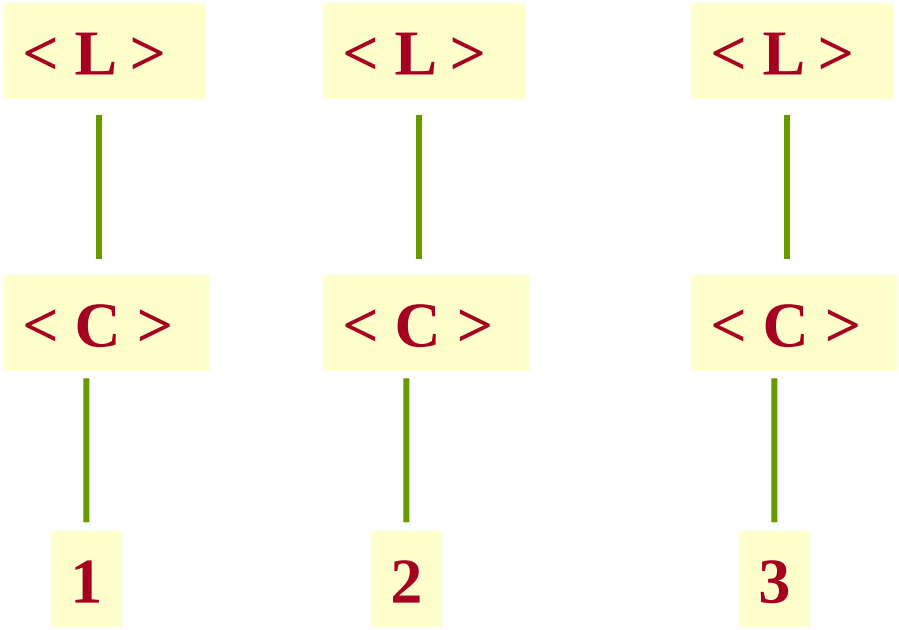
<Cyfra> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9



Etapy konstruowania drzewa rozbioru 3* (2+14)

(b) Określamy że cyfry są liczbami. Używamy produkcji

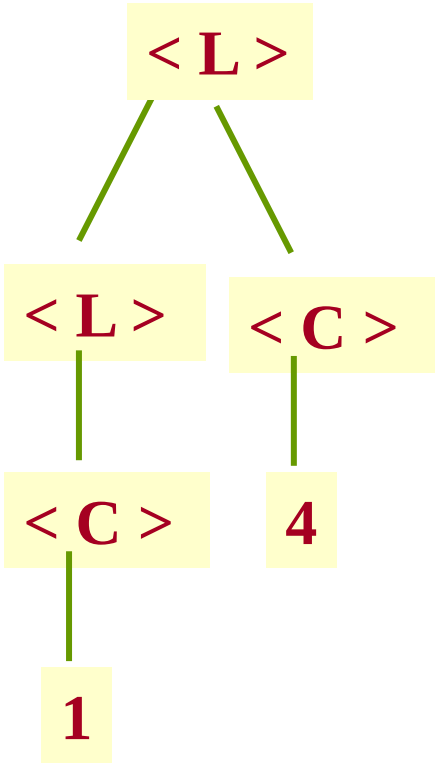
<Liczba> → <Cyfra>



Etapy konstruowania drzewa rozbioru 3* (2+14)

(c) Określamy że cyfr i liczba jest liczba. Wynikiem drzewa jest 14. Używamy produkcji

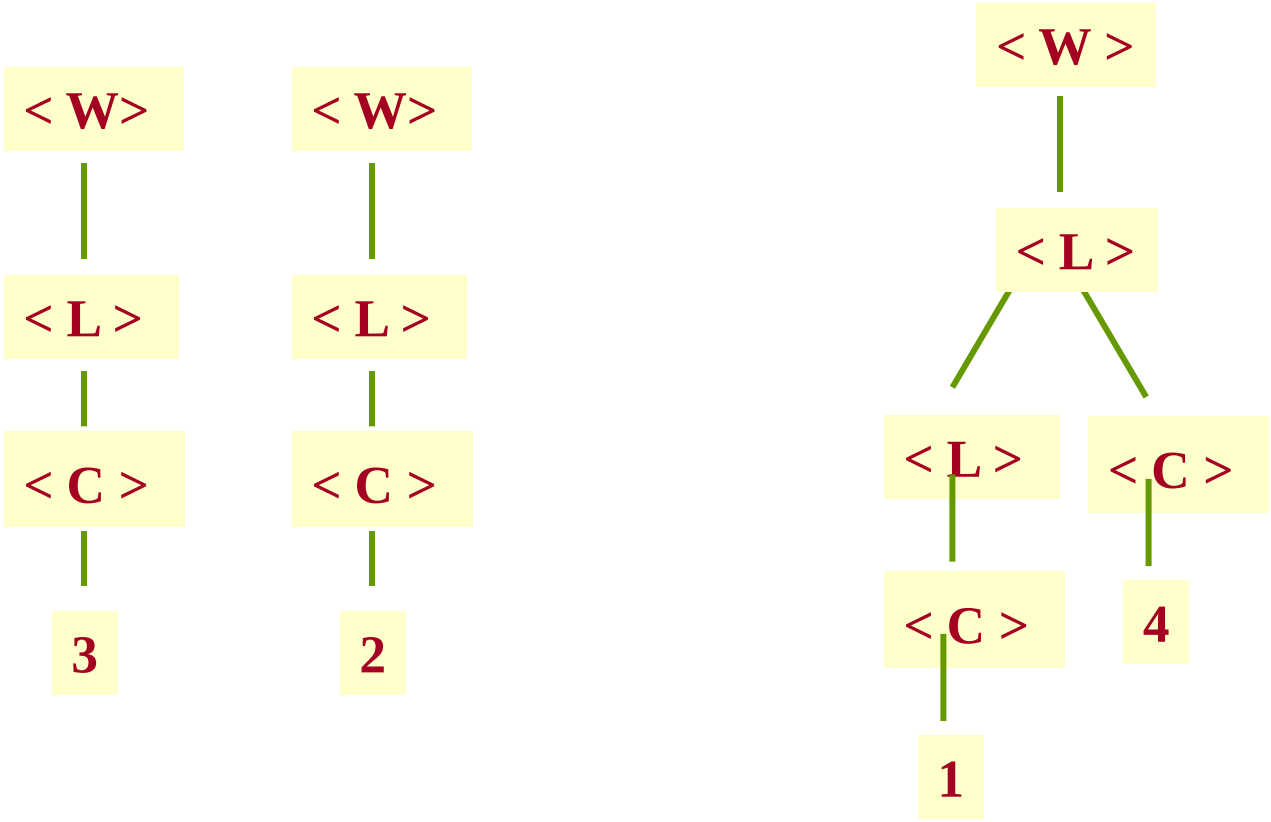
<Liczba> → <Liczba> <Cyfra>



Etapy konstruowania drzewa rozbioru 3* (2+14)

(d) Tworzymy drzewa rozbioru dla wyrażeń 3, 2, 14. Używamy produkcji

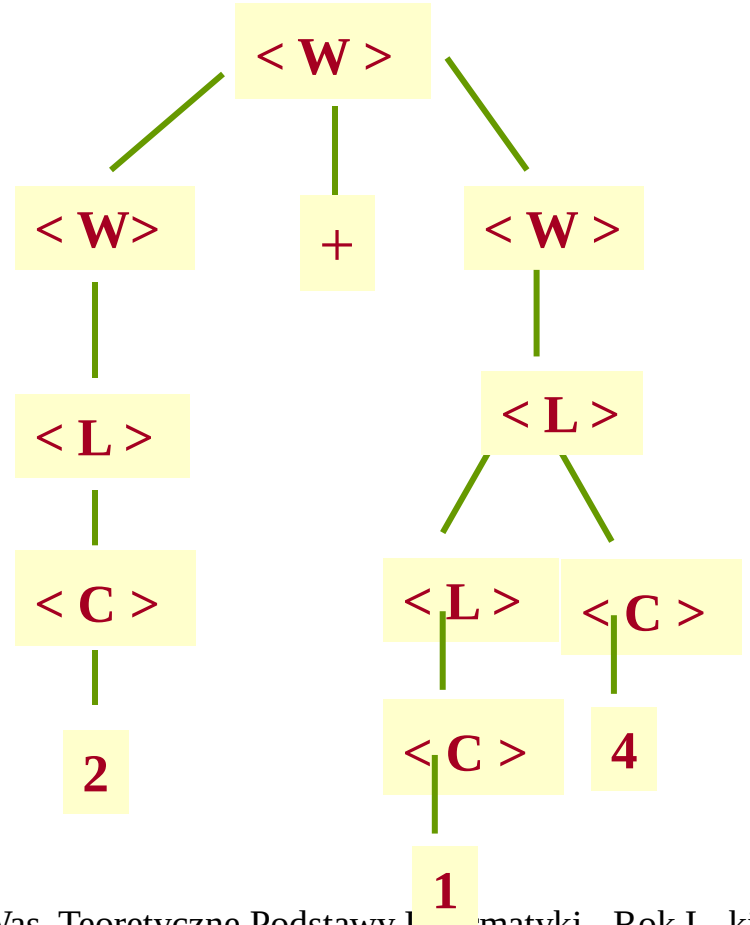
<Wyrażenie> → <Liczba>



Etapy konstruowania drzewa rozbioru 3* (2+14)

(e) Tworzymy drzewo dla sumy 2 + 14. Używamy produkcji

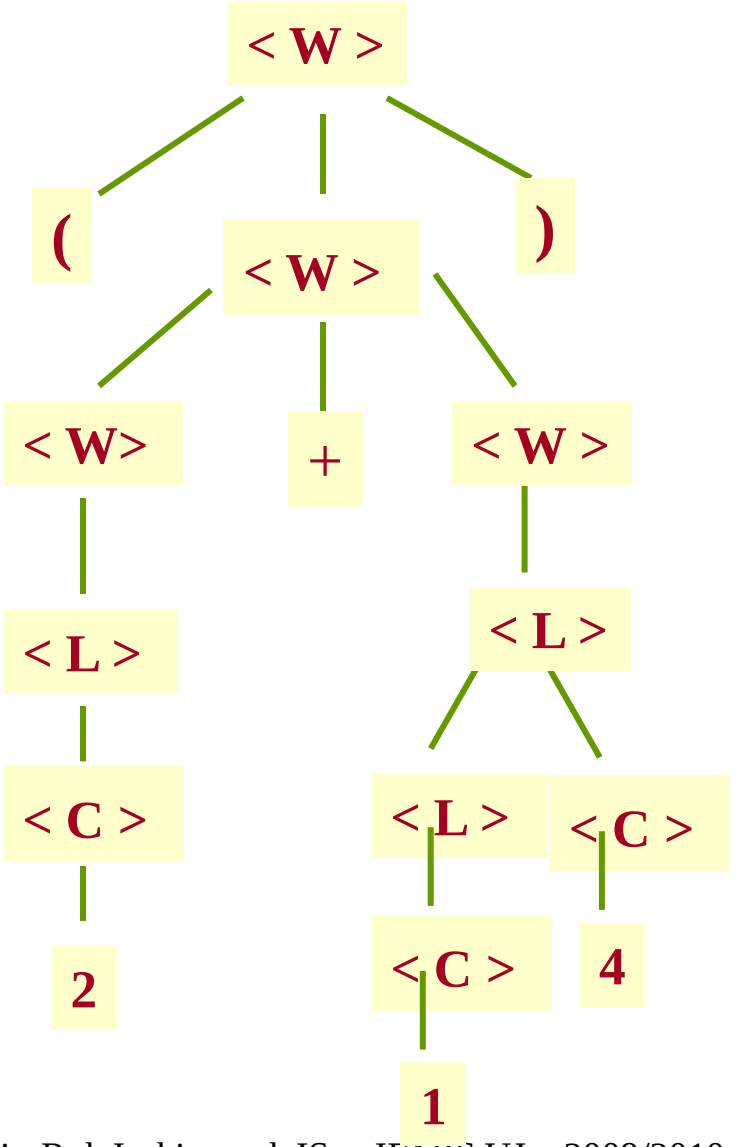
<Wyrażenie> → <Wyrażenie> + <Wyrażenie>



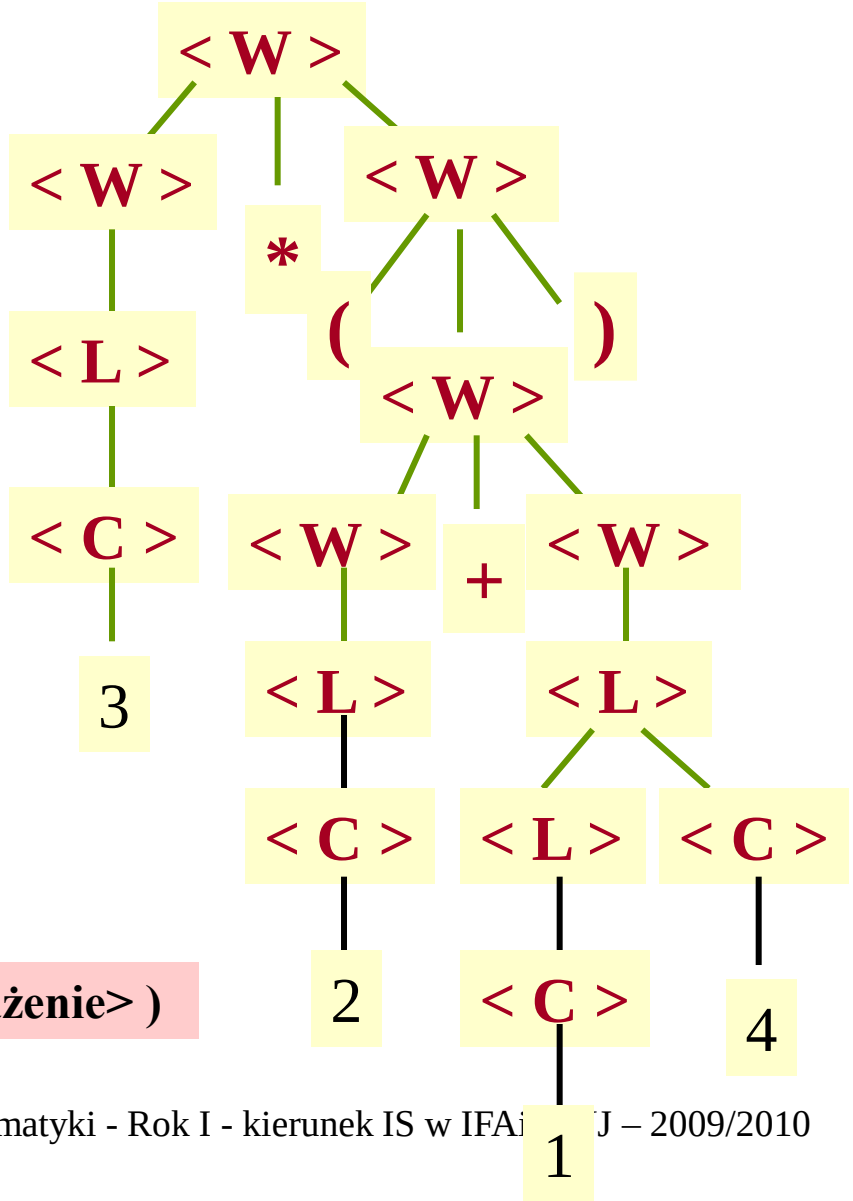
Etapy konstruowania drzewa rozbioru 3* (2+14)

(f) Tworzymy drzewo
dla sumy (2 + 14). Używamy produkcji

<Wyrażenie> → (<Wyrażenie>)



Etapy konstruowania drzewa rozbioru 3* (2+14)



(g) Tworzymy drzewo 3* (2 + 14).
Używamy produkcji

<Wyrażenie> → <Wyrażenie> * (<Wyrażenie>)

Uzasadnienie poprawności konstrukcji drzew rozbioru

Można udowodnić za pomocą dwóch prostych indukcji, że wyniki drzew rozbioru o korzeniu <S> odpowiadają dokładnie ciągom znaków języka L(<S>) dla dowolnej kategorii syntaktycznej <S>.
To znaczy:

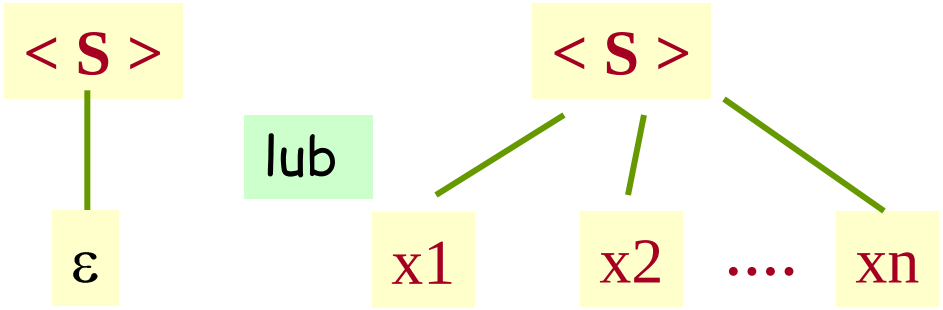
(1) Jeżeli T jest drzewem rozbioru o korzeniu zaetykietowanym przez <S> i daje ono wynik s, to ciąg s należy do języka L(<S>).

(2) Jeżeli ciąg s należy do języka L(<S>), to istnieje drzewo rozbioru dające wynik s oraz posiadające korzeń zaetykietowany przez <S>.

Drzewa rozbioru są tworzone z mniejszych drzew rozbioru w ten sam sposób, w jaki dłuższe ciągi znaków składa się z krótszych, przy użyciu podstawień za kategorie syntaktyczne w częściach zasadniczych produkcji.
Dowód rozpoczynamy od części (1), której dowodzimy za pomocą indukcji zupełnej względem wysokości drzewa.

Podstawa:

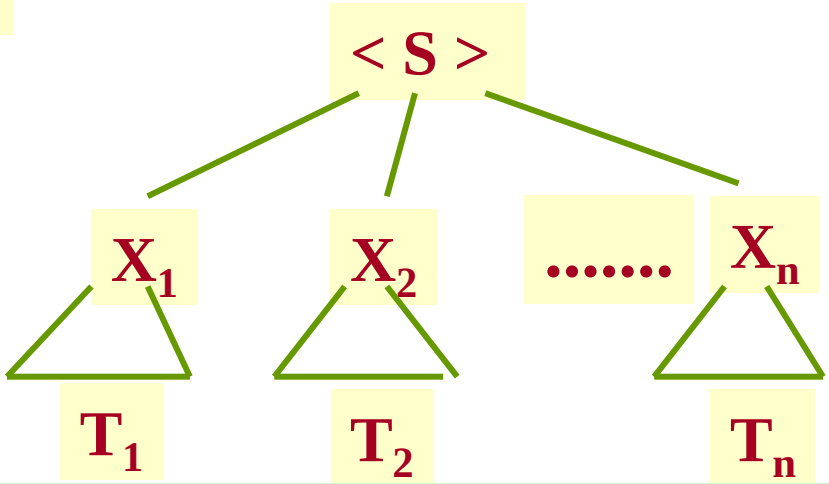
Zakładamy że wysokość drzewa rozbioru wynosi 1. Wówczas drzewo ma postać



gdzie każde x jest symbolem terminalnym. Zatem x1x2...xn stanowi ciąg znaków należący do języka L(<S>)

Indukcja:

Zakładamy, ze (1) jest prawdziwe dla wszystkich drzew o wysokości k lub mniejszej.
Bierzemy pod uwagę drzewo o wysokości k+1. Każde poddrzewo T ma wysokość co najwyżej k. Jeżeli Xi jest kategoria syntaktyczna, to wynik drzewa Ti, na przykład si, należy do języka L(Xi). Jeżeli Xi jest symbolem terminalnym, ciąg si definiujemy jako Xi. Wówczas wynikiem drzewa jest s1s2...sn.
Na podstawie reguły indukcyjnej dla języka kategorii <S> , s1s2...sn należy do L(<S>).



Teraz należy udowodnić część (2) mówiącą, że każdy ciąg s kategorii syntaktycznej <S> posiada drzewo rozbioru o korzeniu <S> oraz wyniku s.
Na początku należy zauważyć, ze dla każdego symbolu terminalnego x istnieje drzewo rozbioru, w którym x jest zarówno korzeniem jak i wynikiem.

Podstawa:

Zakładamy, że s wymaga jednego zastosowania kroku indukcyjnego w celu wykazania, że s należy do L(<S>). Wówczas musi istnieć produkcja postaci <S> → x1x2....xn, gdzie wszystkie symbole x są symbolami terminalnymi a s=x1x2....xn. Wiadomo, że istnieje jednowierzchołkowe drzewo rozbioru z etykietą x1, dla i=1,2,...n. A zatem istnieje drzewo rozbioru o wyniku s i korzeniu zaetykietowanym przez <S>. (Patrz drzewa na str. 24.)

Indukcja:

Zakładamy, że dowolny ciąg znaków t należący do języka dowolnej kategorii syntaktycznej <T> w wyniku zastosowania kroku indukcyjnego k lub mniejsza liczbę razy posiada drzewo rozbioru z t jako wynikiem oraz <T> jako korzeniem.
Bierzemy pod uwagę ciąg s należący do języka kategorii syntaktycznej <S> w wyniku zastosowania kroku indukcyjnego k+1 lub mniejsza liczbę razy.

Indukcja (cont):

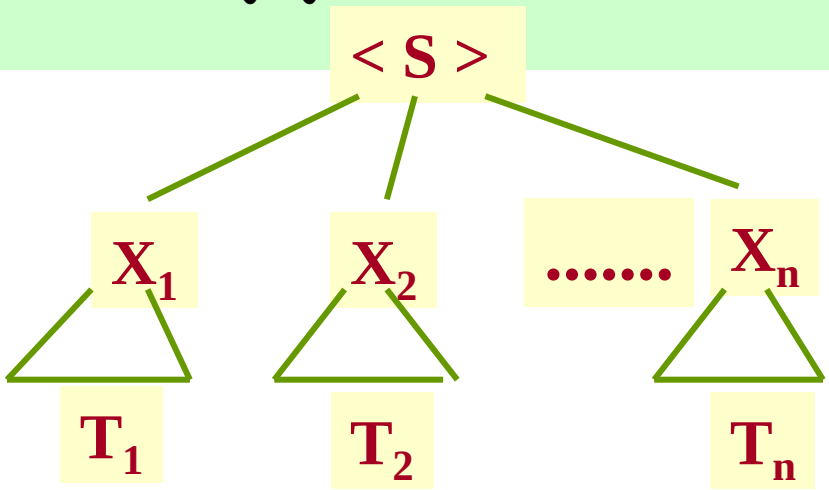
Wówczas istnieje produkcja ‹S› → X1X2....Xn oraz zachodzi s=s1s2.....sn, gdzie każdy podciąg si jest:

(1) Równy Xi, jeżeli Xi jest symbolem terminalnym.

(2) Pewnym ciągiem, o którym wiadomo, że należy do języka L(Xi) w wyniku zastosowania reguły indukcyjnej co najwyżej k razy, jeżeli Xi jest kategorią syntaktyczną.

A zatem dla każdego i można określić drzewo Ti posiadające wynik si oraz korzeń zaetykietowany przez Xi. Jeżeli Xi jest kategorią syntaktyczną, wykorzystujemy hipotezę indukcyjną w celu stwierdzenia, że Ti istnieje, zaś jeżeli Xi jest symbolem terminalnym, nie ma potrzeby wykorzystywania hipotezy indukcyjnej w celu stwierdzenia, że istnieje jednowierzchołkowe drzewo zaetykietowane przez Xi.

A zatem drzewo skonstruowane jak to po lewej stronie, posiada wynik s oraz korzeń zaetykietowany przez ‹S›.



Drzewa rozbioru i drzewa wyrażeń

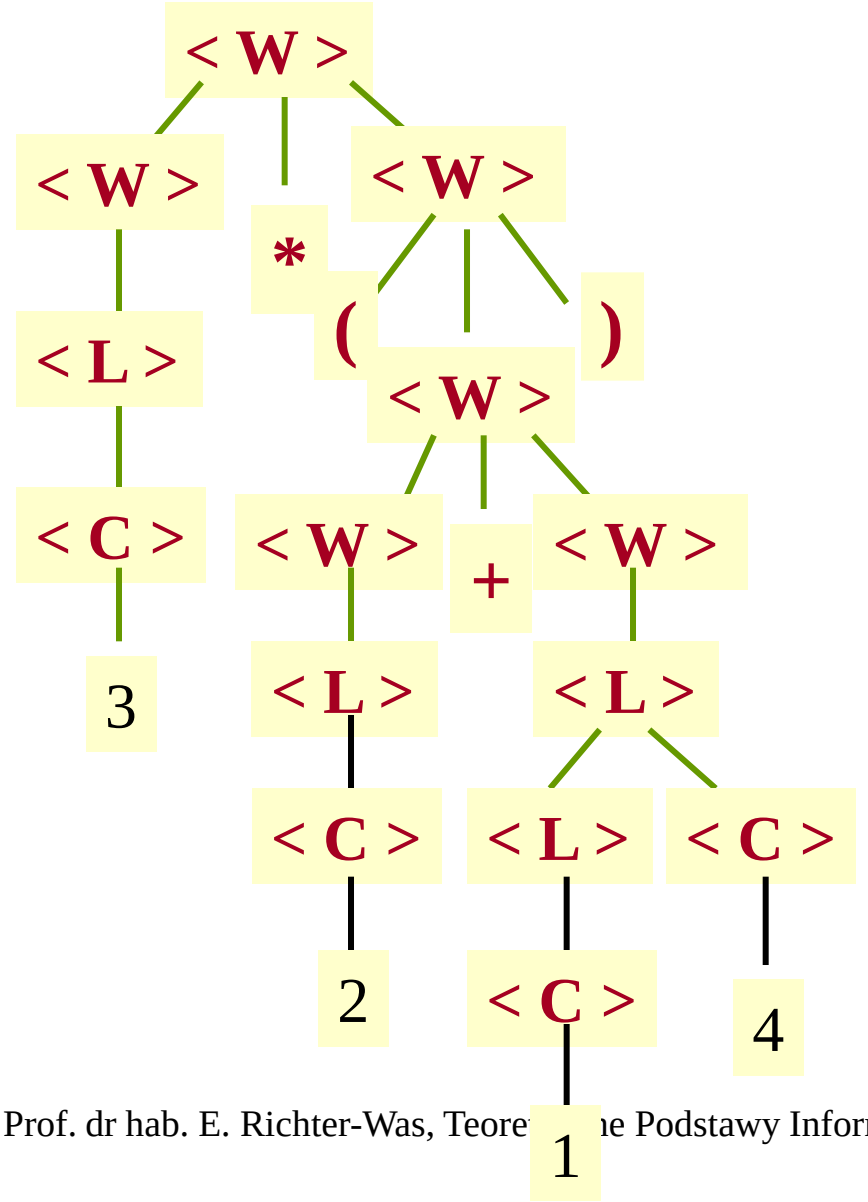
Mając sformułowaną gramatykę wyrażeń możemy drzewa rozbioru przekonwertować na drzewa wyrażeń, dokonując trzech transformacji:

(1) Wierzchołki związane z poszczególnymi operandami niepodzielnymi są łączone w jeden wierzchołek zaetykietowany danym operandem

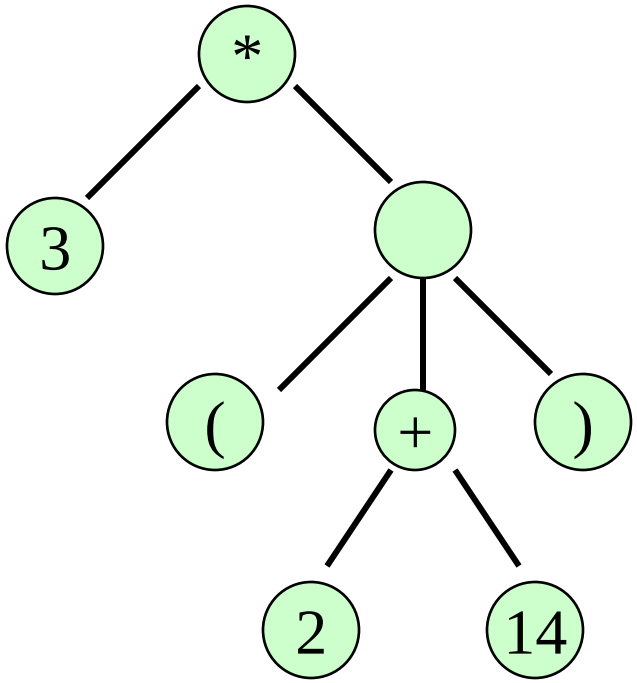
(2) Operatory zostają przesunięte z liści do ich wierzchołków nadrzędnych. To znaczy symbol operatora, taki jak +, staje się etykietą wierzchołka umieszczonego nad nim, który wcześniej był zaetykietowany kategoria syntaktyczna „wyrażenia".

(3) Wierzchołki wewnętrzne, których etykietami wciąż są „wyrażenia" zostają usunięte.

drzewo rozbioru



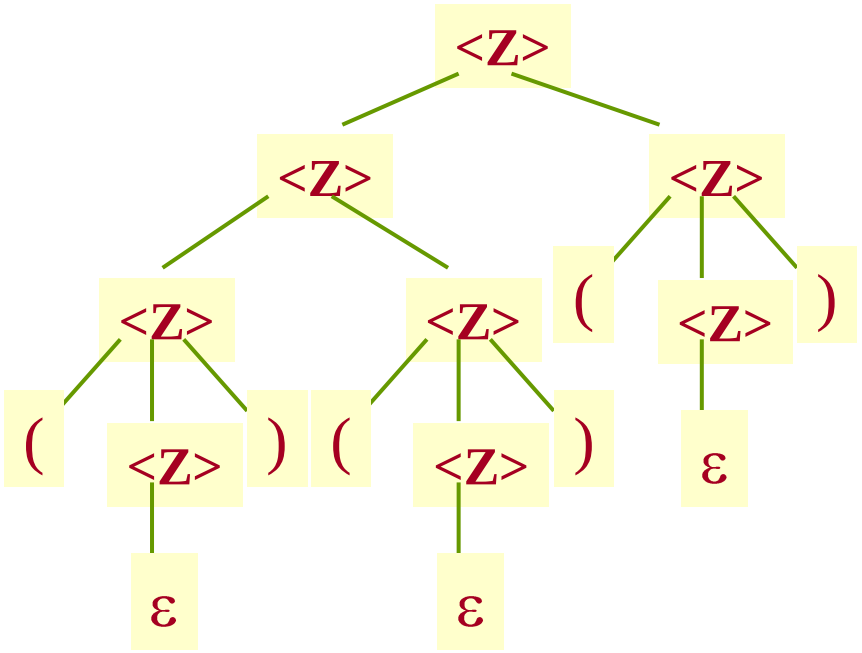
drzewo wyrażeń

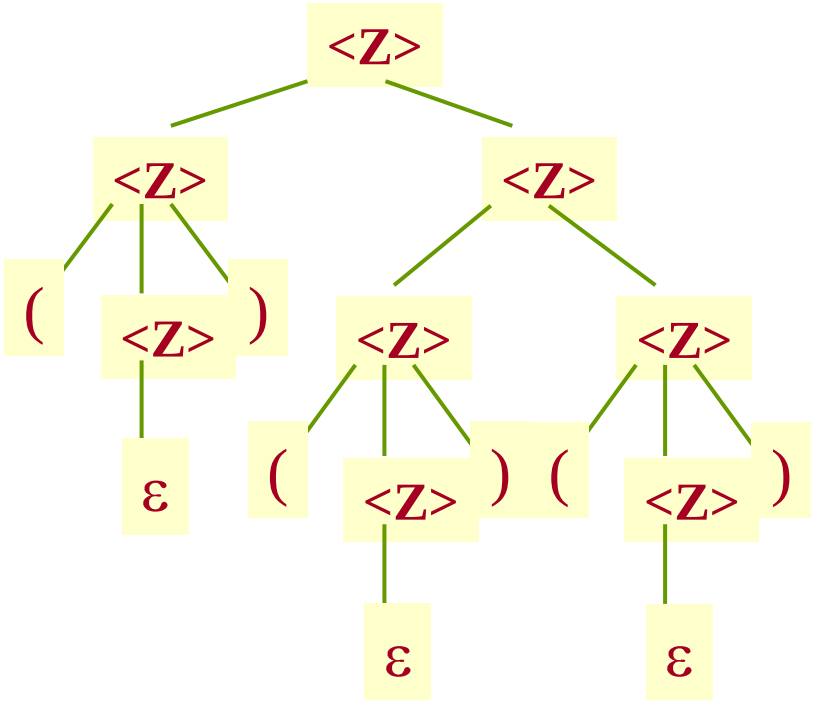


Niejednoznaczność i projektowanie gramatyk

Rozpatrzmy gramatykę zbilansowanych nawiasów. Chcemy utworzyć drzewo rozbioru dla ciągu znaków ( ) ( ) ( ). Można utworzyć dwa takie drzewa.

<Z> → ε
<Z> → (<Z>)
<Z> → <Z> <Z>





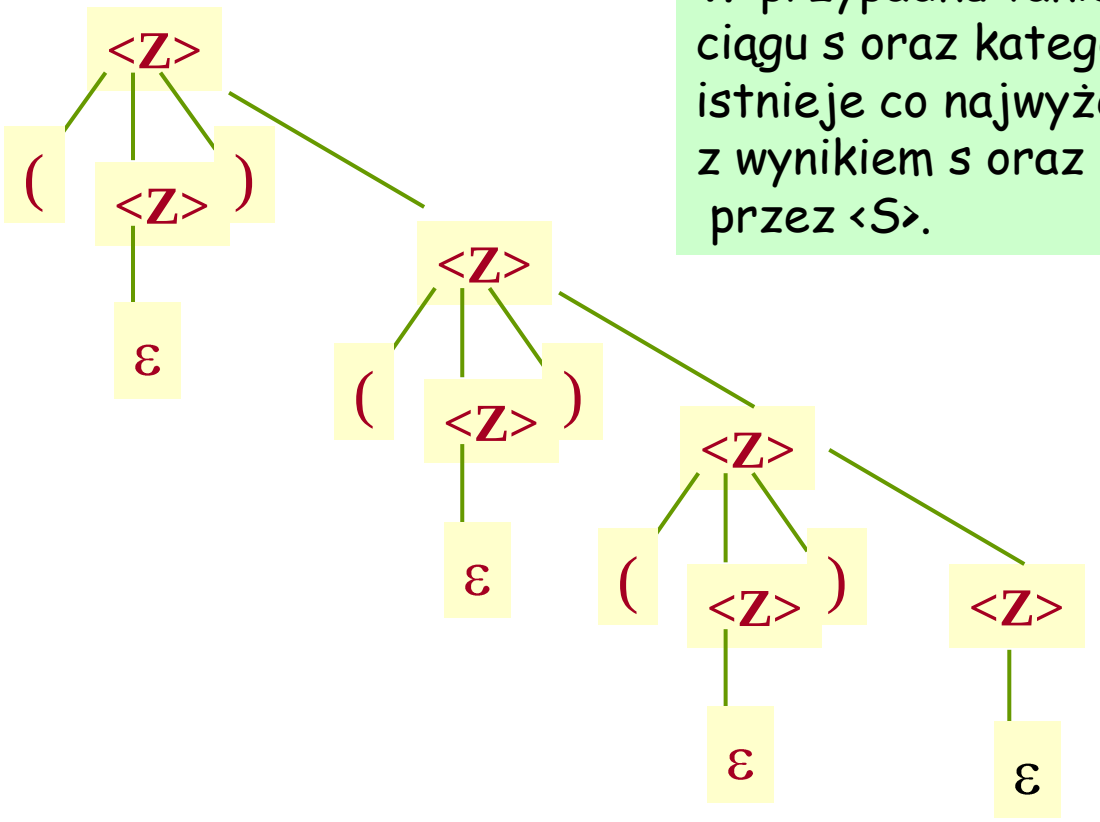
Gramatyka w której istnieją dwa lub więcej drzewa rozbioru o tym samym wyniku oraz tej samej kategorii syntaktycznej etykietującej korzeń jest nazywana *gramatyka niejednoznaczna. (ang. ambigous).* Wystarczy żeby istniał choć jeden taki ciąg który jest niejednoznaczny.

Rozpatrzmy inną gramatykę zbilansowanych nawiasów. Chcemy utworzyć drzewo rozbioru dla ciągu znaków ( ) ( ) ( ). Można utworzyć tylko jedno takie drzewo.

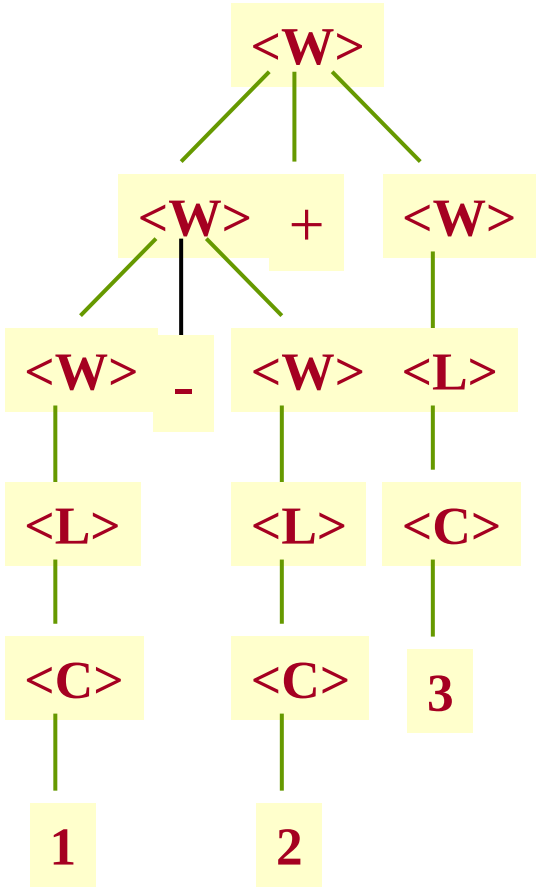
$\langle Z \rangle \rightarrow \varepsilon$

$\langle Z \rangle \rightarrow (\langle Z \rangle) \langle Z \rangle$

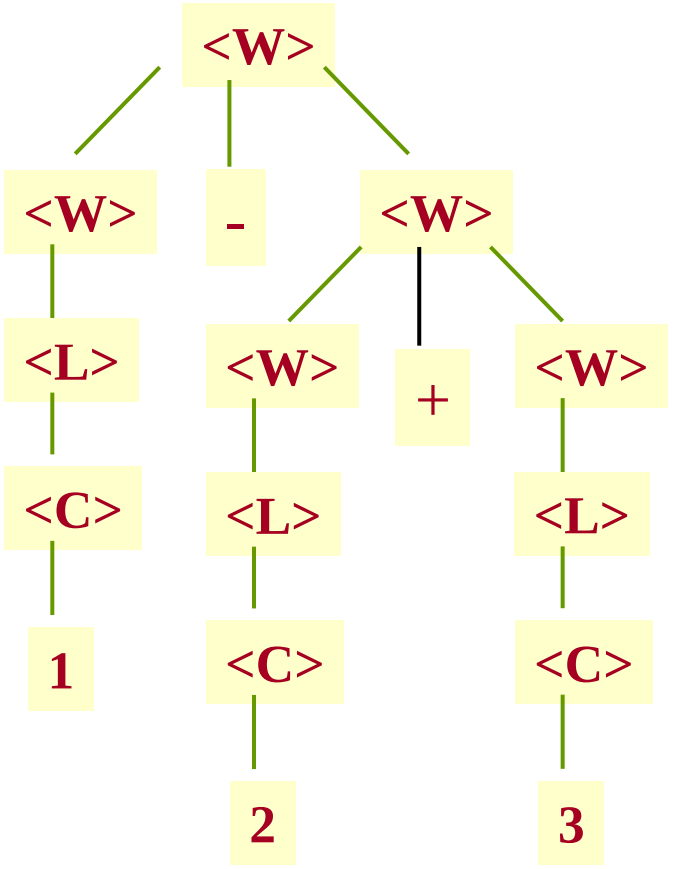
Gramatyka, która nie jest niejednoznaczna nosi nazwę *jednoznacznej (ang. unambigous)* .
W przypadku takiej gramatyki dla każdego ciągu s oraz kategorii syntaktycznej <S> istnieje co najwyżej jedno drzewo rozbioru z wynikiem s oraz korzeniem zaetykietowanym przez <S>.



Niejednoznaczność gramatyk wyrażeń może być poważnym problemem.
Niektóre drzewa rozbioru mogą dawać złe wartości dla wyrażeń.
Dwa drzewa rozbioru dla wyrażenia: 1-2+3

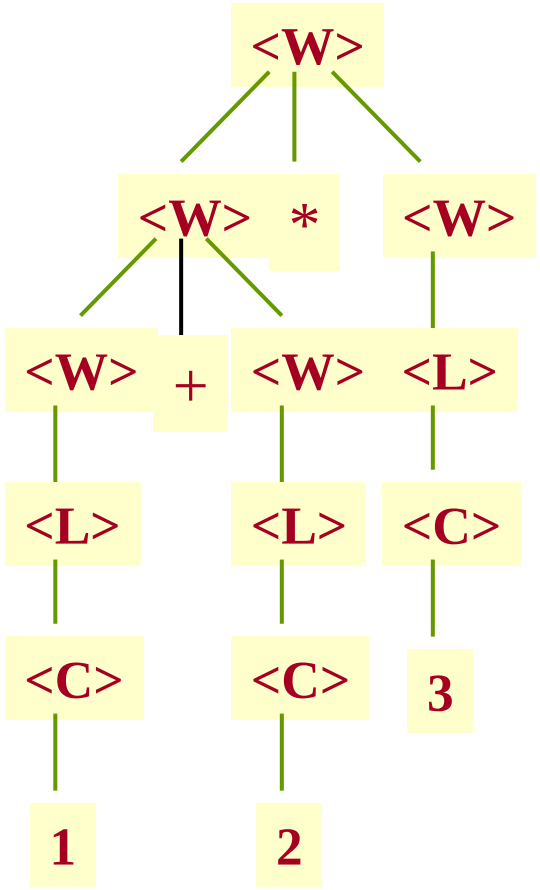


Poprawne drzewo rozbioru
1-2+3 = 2

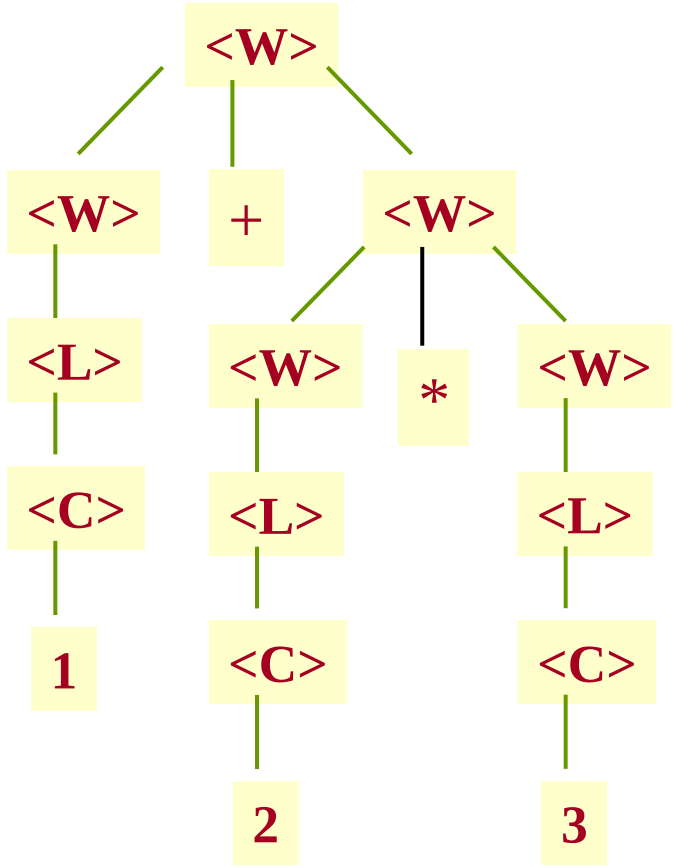


Niepoprawne drzewo rozbioru
1-2+3=-4

Niejednoznaczność gramatyk wyrażeń może być poważnym problemem.
Niektóre drzewa rozbioru mogą dawać złe wartości dla wyrażeń.
Dwa drzewa rozbioru dla wyrażenia: 1+2*3



Niepoprawne drzewo rozbioru
1 + 2 * 3 = 9



Poprawne drzewo rozbioru
1 + 2 * 3 = 7

Istota niejednoznaczności

Analizator składniowy który konstruuje drzewa rozbioru dla programów stanowi podstawowy element kompilatora. Jeżeli gramatyka opisująca język programowania jest niejednoznaczna, oraz jeżeli jej niejednoznaczności są wyprowadzone jednostronnie, to w przypadku przynajmniej części programów istnieje więcej niż jedno drzewo rozbioru. Jeżeli gramatyka programu jest niejednoznaczna, kompilator nie może podjąć prawidłowej decyzji odnośnie do drzewa rozbioru dla pewnych programów, a w związku z tym nie może zdecydować, jakie działania powinien wykonać program w języku maszynowym.
Kompilatory muszą korzystać ze specyfikacji które są jednoznaczne.

Jednoznaczne gramatyki wyrażeń

Konstrukcja jednoznacznej gramatyki polega na zdefiniowaniu trzech kategorii syntaktycznych o następującym znaczeniu:

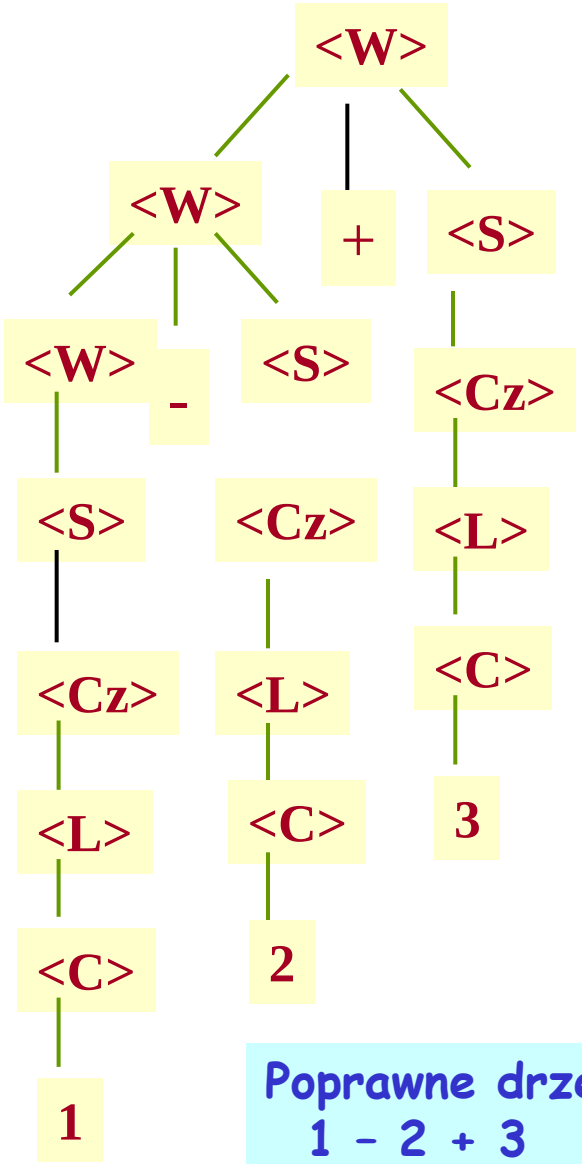
<Czynnik> - generuje wyrażenia, które nie mogą zostać rozdzielone, to znaczy czynnik jest albo pojedynczym operandem, albo dowolnym wyrażeniem umieszczonym w nawiasie.

< Składnik> - generuje iloczyn lub iloraz czynników. Pojedynczy czynnik jest składnikiem, a więc stanowi ciąg czynników rozdzielonych operatorami * lub /. Przykładami składników są 12 lub 12/3*45.

<Wyrażenie> - generuje różnicę lub sumę jednego lub większej liczby składników. Pojedynczy składnik jest wyrażeniem, a więc stanowi sekwencję składników rozdzielonych operatorami + lub -.
Przykładami wyrażeń są 12, 12/3*45 lub 12+3*45-6.

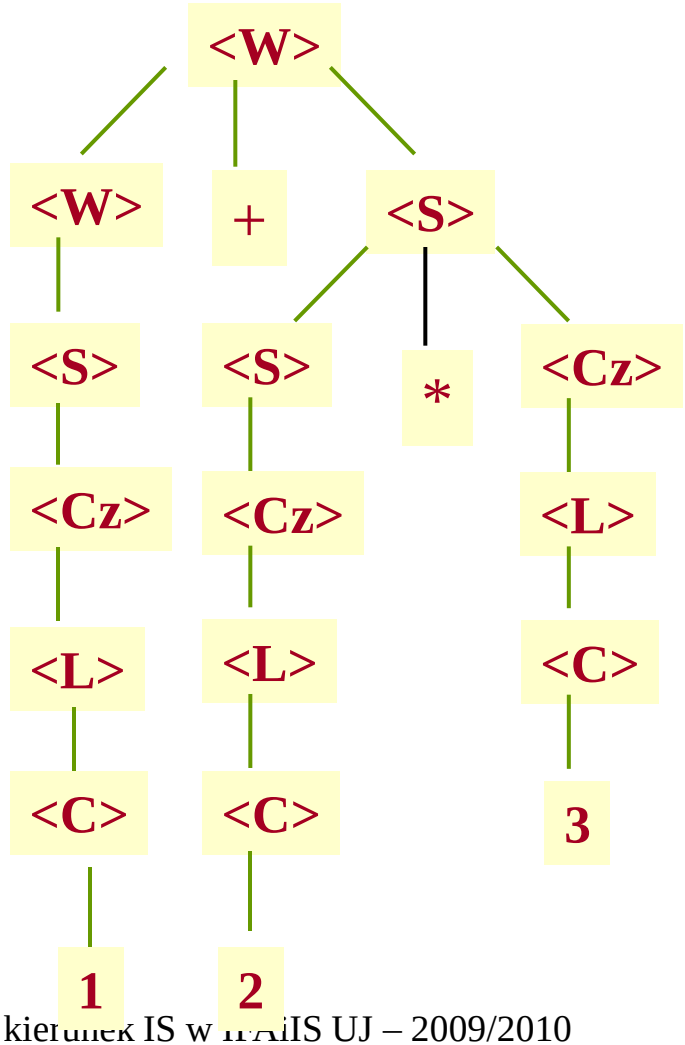
(1) <W> → <W> + <S> | <W> - <S> | <S>
(2) <S> → <S> * <Cz> | <S> / <Cz> | <Cz>
(3) <Cz> → (<W>) | <L>
(4) <L > → <L><C> | <C>
(5) <C> → 0 | 1 | ..... | 9

gramatyka jednoznaczna wyrażeń arytmetycznych



Poprawne drzewo rozbioru
1 - 2 + 3

Poprawne drzewo rozbioru
1 + 2 * 3



Rozróżnienie miedzy wyrażeniami, składnikami i czynnikami wymusza poprawne grupowanie wyrażeń na różnych poziomach pierwszeństwa działań.

Gramatyki przypominają wyrażenia regularne tym, że obie notacje opisują języki, ale nie definiują bezpośrednio algorytmu określania, czy dany ciąg znaków należy do definiowanego języka. W przypadku wyrażeń regularnych pokazaliśmy jak można konwertować wyrażenia regularne najpierw na automat niedeterministyczny, a potem na automat deterministyczny. Ten drugi można bezpośrednio implementować jako program.

Analogiczny proces można opisać w przypadku gramatyk. Konwersja gramatyki na automat deterministyczny jest niemożliwa. Istnieje możliwość przekonwertowania gramatyki na program, który podobnie jak automat odczytuje dane wejściowe i określa czy dany ciąg wejściowy należy do języka gramatyki. Najważniejsza z takich technik nosi nazwę *rozbioru lewostronnego (ang. LR parsing).*

Analiza składniowa i konstrukcja drzew rozbioru

Technika zwana *schodzeniem rekurencyjnym (ang. recursive descent),* w przypadku której gramatyka jest zastępowana kolekcją wzajemnie rekurencyjnych funkcji, z których każda odpowiada jednej kategorii syntaktycznej gramatyki. Celem działania funkcji S, która odpowiada kategorii syntaktycznej <S>, jest odczytanie ciągu znaków wejściowych, które tworzą ciąg należący do języka L(<S>) oraz zwrócenie wskaźnika do korzenia drzewa rozbioru tego ciągu.

Część zasadniczą produkcji można traktować jako sekwencję warunków – symboli terminalnych i kategorii syntaktycznych – które muszą zostać spełnione, aby móc określić ciąg znaków występujących w części nagłówkowej produkcji.

Przykładowo można rozpatrzyć gramatykę jednoznaczna wyrażeń o zbilansowanej liczbie nawiasów.

(1) $\langle Z \rangle \rightarrow \varepsilon$
(2) $\langle Z \rangle \rightarrow (\langle Z \rangle)\ \langle Z \rangle$

Produkcja (2) określa, że jednym ze sposobów znalezienia ciągu o zbilansowanej liczbie nawiasów jest:
(1) znalezienie znaku (
(2) znalezienie ciągu z bilansowanej liczbie nawiasów
(3) znalezienie ciągu)
(4) znalezienie kolejnego ciągu o zbilansowanej liczbie nawiasów

Warunek dotyczący symbolu terminalnego jest spełniony wówczas, gdy okaże się, że dany symbol terminalny jest kolejnym symbolem wejściowym, ale warunek nie może zostać spełniony, jeżeli symbolem wejściowym jest pewien inny znak. Aby określić, czy kategoria syntaktyczna określona w części zasadniczej produkcji jest spełniona, należy wywołać funkcję dla tej kategorii syntaktycznej.

Chcemy określić czy sekwencja symboli terminalnych $X_1X_2...X_n$ jest ciągiem należącym do kategorii syntaktycznej <S>, a jeżeli tak, to znaleźć jego drzewo rozbioru. Do pliku wejściowego wstawiane są symbole $X_1X_2...X_n$ ENDM gdzie ENDM to tzw. *znacznik końcowy*, (ang. *end marker*). *Kursor wejściowy* (ang. *imput kursor*) oznacza symbol terminalny który ma zostać poddany przetworzeniu, czyli bieżący symbol terminalny. Jeżeli dane wejściowe stanowią ciąg znaków, to kursor może być wskaźnikiem znaku.
Analizę składniową programu rozpoczyna się od wywołania funkcji S dla początkowej kategorii syntaktycznej <S>, przy kursorze wejściowym wskazującym na początek ciągu wejściowego.

Inicjalizacja programu wykrywającego <S> na wyjściu.

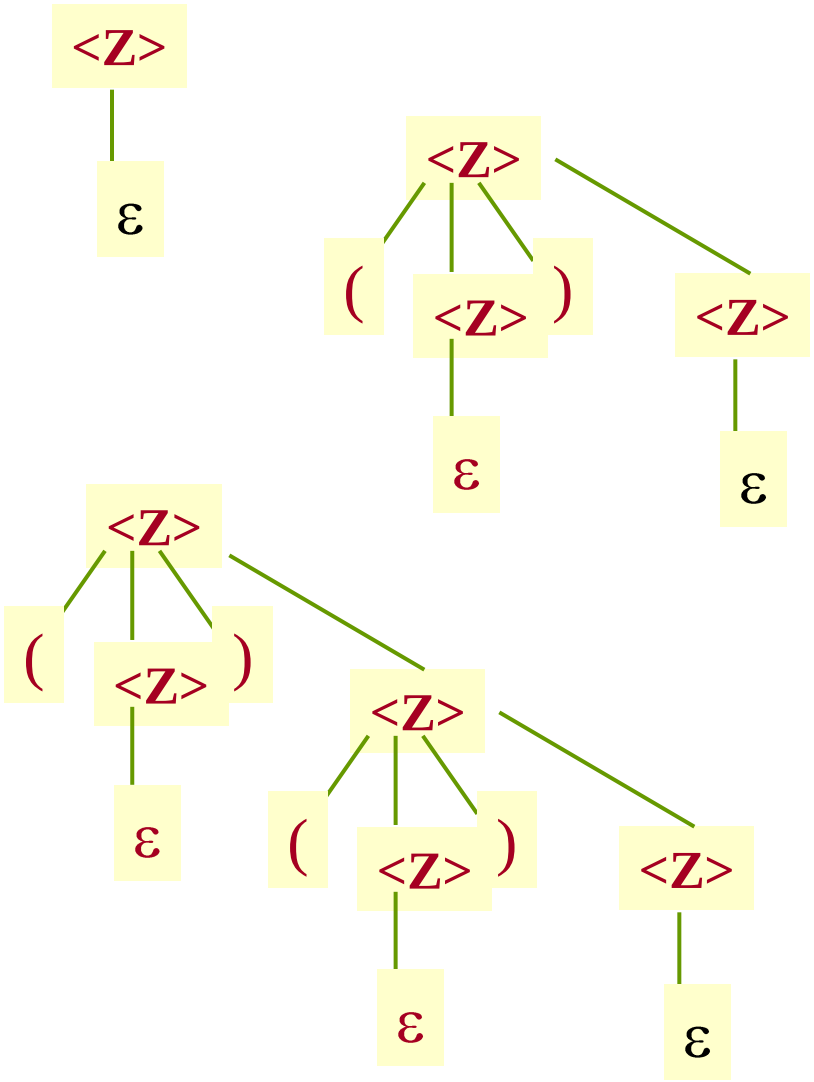
=> Gdy rozpatrywana jest część zasadnicza produkcji i dochodzi się w produkcji do symbolu terminalnego **a**, należy wyszukać odpowiadający mu symbol terminalny **a** na pozycji wskazywanej przez kursor wejściowy. Jeżeli zostanie znaleziony symbol **a**, kursor przechodzi do kolejnego symbolu terminalnego danych wejściowych. Jeżeli bieżący symbol wejściowy różni się od **a**, to dopasowanie kończy się niepowodzeniem i nie można znaleźć drzewa rozbioru dla ciągu wejściowego.

=> Gdy rozpatrywana jest część zasadnicza produkcji i dochodzi się w produkcji do kategorii syntaktycznej **<T>**, należy wywołać funkcje **T** dla **<T>**. Jeżeli wywołanie kończy się niepowodzeniem, cały proces analizy składniowej kończy się niepowodzeniem, a ciąg wejściowy można określić jako nie należący do analizowanego języka. Jeżeli wywołanie funkcji T kończy się powodzeniem, to następuje przesuniecie kursora wejściowego o pewną ilość znaków (odpowia dających funkcji T) oraz funkcja T zwraca drzewo, które jest drzewem rozbioru przetwarzanej części ciągu wejściowego.

Kiedy opisywane działania zakończą się sukcesem w przypadku każdego symbolu określonego w części zasadniczej produkcji, należy zestawić drzewo rozbioru dla fragmentu danych wejściowych reprezentowanych przez te produkcję.

Funkcja konstruująca drzewa rozbioru dla ciągów zawierających zbilansowana liczbę nawiasów.

```
TREE Z()
{
TREE fusrtZ, secondZ;
if ( *nextTerminal == ‘(‘) /*zgodnie z produkcja 2*/{
        nextTerminal++;
        firstZ = Z();
        if(secondZ == FAILED)
            return FAILED;
        else
            return makeNode4(‘Z’,makeNode0(‘(‘),firstZ,makeNode0(‘)’),secondZ);
        }
        else / *pierwsze wołanie Z zakończone niepowodzeniem */
              return FAILED;
}
else / * zgodnie z produkcja 1 */
    return makeNode1(‘Z’, makeNode0(‘e’));
}
```



Drzewa skonstruowane w wyniku rekurencyjnego wywoływania funkcji.

Jeżeli kategoria syntaktyczna <S> reprezentuje język, którego ciągi znaków należy rozpoznawać i analizować, proces analizy rozpoczyna się od ustawienia kursora wejściowego na pierwszym wejściowym symbolu terminalnym. Wywołanie funkcji S powoduje skonstruowanie drzewa rozbioru (o ile takie istnieje) dla danych wejściowych i kończy się niepowodzeniem, jeżeli ciąg wejściowy nie należy do języka L(<S>).

Konstrukcja analizatorów składniowych schodzenia rekurencyjnego

Kluczowe jest zapewnienie, aby dla każdej kategorii syntaktycznej <S>, jeżeli istnieje więcej niż jedna produkcja posiadająca jako część nagłówkową <S>, można było tylko przez zbadanie bieżącego symbolu terminalnego (często określanego mianem *symbolu antycypowanego (ang. lookahead)* określić jedną produkcję <S>, którą należy wypróbować.
Nie jest możliwe stwierdzenie czy dla danej gramatyki istnieje algorytm, który zawsze będzie podejmował poprawne decyzje. Jeżeli dysponuje się strategią którą uważa się za poprawną, to dla każdej kategorii syntaktycznej <S> można zaprojektować funkcje S, której działanie będzie polegało na:

(1) Zbadaniu symbolu antycypowanego i zdecydowaniu która produkcję wypróbować. Zakładamy, że zasadnicza cześć wybranej produkcji ma postać $X_1X_2....X_n$.

(2) Dla i=1,2,....,n wykonaniu następujących działań w przypadku elementu X_i:

(a) jeżeli Xi jest symbolem terminalnym, należy sprawdzić, czy jest symbolem antycypowanym. Jeżeli tak, należy przesunąć kursor wejściowy. Jeżeli nie, dane wywołanie funkcji S kończy się niepowodzeniem.

(b) jeżeli Xi jest kategorią syntaktyczną, na przykład <T>, należy wywołać funkcje T odpowiadającą danej kategorii syntaktycznej. Jeżeli T kończy się niepowodzeniem, to niepowodzeniem kończy się również wywołanie funkcji S Jeżeli T kończy się sukcesem należy zachować zwrócone drzewo.

Algorytm analizy składniowej oparty na tabeli

Rekurencyjne wywołania funkcji implementuje się zwykle za pomocą stosu zapisów aktywacji. Funkcje analizatora składniowego schodzenia rekurencyjnego wykonują bardzo specyficzne działania. Istnieje możliwość zastąpienia ich pojedynczą funkcją, która bada tabele i samodzielnie manipuluje stosem.

Funkcja S kategorii syntaktycznej <S> w pierwszej kolejności decyduje o tym, jakiej produkcji należy użyć, a następnie przechodzi sekwencje kolejnych etapów -jeden etap dla każdego symbolu należącego do części zasadniczej wybranej produkcji. Zatem można utworzyć stos symboli gramatycznych który z grubsza będzie przypominał stos zapisów aktywacji. Na stos zostają odkładane zarówno symbole terminalne jak i kategorie syntaktyczne. Kiedy kategoria syntaktyczna <S> znajduje się na szczycie stosu najpierw określa się odpowiednią produkcję. Następnie należy zastąpić <S> częścią zasadniczą produkcji (lewy koniec na szczycie stosu). Kiedy na szczycie stosu mamy symbol terminalny i odpowiada on bieżącemu symbolowi wejściowemu należy zdjąć element ze stosu i przesunąć kursor wejściowy.

Tabele analizy składniowej

Alternatywą dla pisania zbioru funkcji rekurencyjnych jest skonstruowanie *tabeli analizy składniowej* (ang. *parsing table*), której wiersze odpowiadają kategoriom syntaktycznym, zaś kolumny odpowiadają możliwym symbolom antycypowanym. Wartość umieszczona w polu określonym przez wiersz kategorii syntaktycznej <S> oraz kolumnę symbolu antycypowanego X jest numerem produkcji, której częścią nagłówkową jest <S>, i która musi zostać wykorzystana w celu rozszerzenia <S> w przypadku, gdy symbolem antycypowanym jest X.

Przykład:

Gramatyka

(1) $\langle Z \rangle \rightarrow \varepsilon$
(2) $\langle Z \rangle \rightarrow (\langle Z \rangle)\ \langle Z \rangle$

Tabela analizy składniowej

	(	)	ENDM
<Z>	2	1	1

Gramatyka

(1) <I> → w c <I>
(2) <I> → { <D>
(3) <I> → s ;
(4) <D> → <I> <D>
(5) <D> → }

Tabela analizy składniowej

	w	c	{	}	s	;	ENDM
<I>	1		2		3		
<D>	4		4	5	4		

Postać gramatyki przedstawionej powyżej umożliwia jej analizę składniową za pomocą schodzenia rekurencyjnego lub za pomocą analizy składniowej opartej na tabeli.
<D> -kategoria syntaktyczna „dokończenie".

Etapy działania analizatora składniowego dla ciągu: {w c s ; s ; } ENDM

STOS	SYMBOL ANTYCYP.	RESZTA
<I>	{	wcs;s;}ENDM
{<D>	{	wcs;s;}ENDM
<D>	w	cs;s;}ENDM
<I><D>	w	cs;s;}ENDM
wc<I><D>	w	cs;s;}ENDM
c<I><D>	c	s;s;}ENDM
<I><D>	s	;s;} ENDM
s;<D>	s	;s;} ENDM
;<D>	;	s;} ENDM
<D>	s	;} ENDM
<I><D>	s	;} ENDM
s; <D>	s	;} ENDM
; <D>	;	} ENDM
<D>	}	ENDM
}	}	ENDM
ε	ENDM	ε

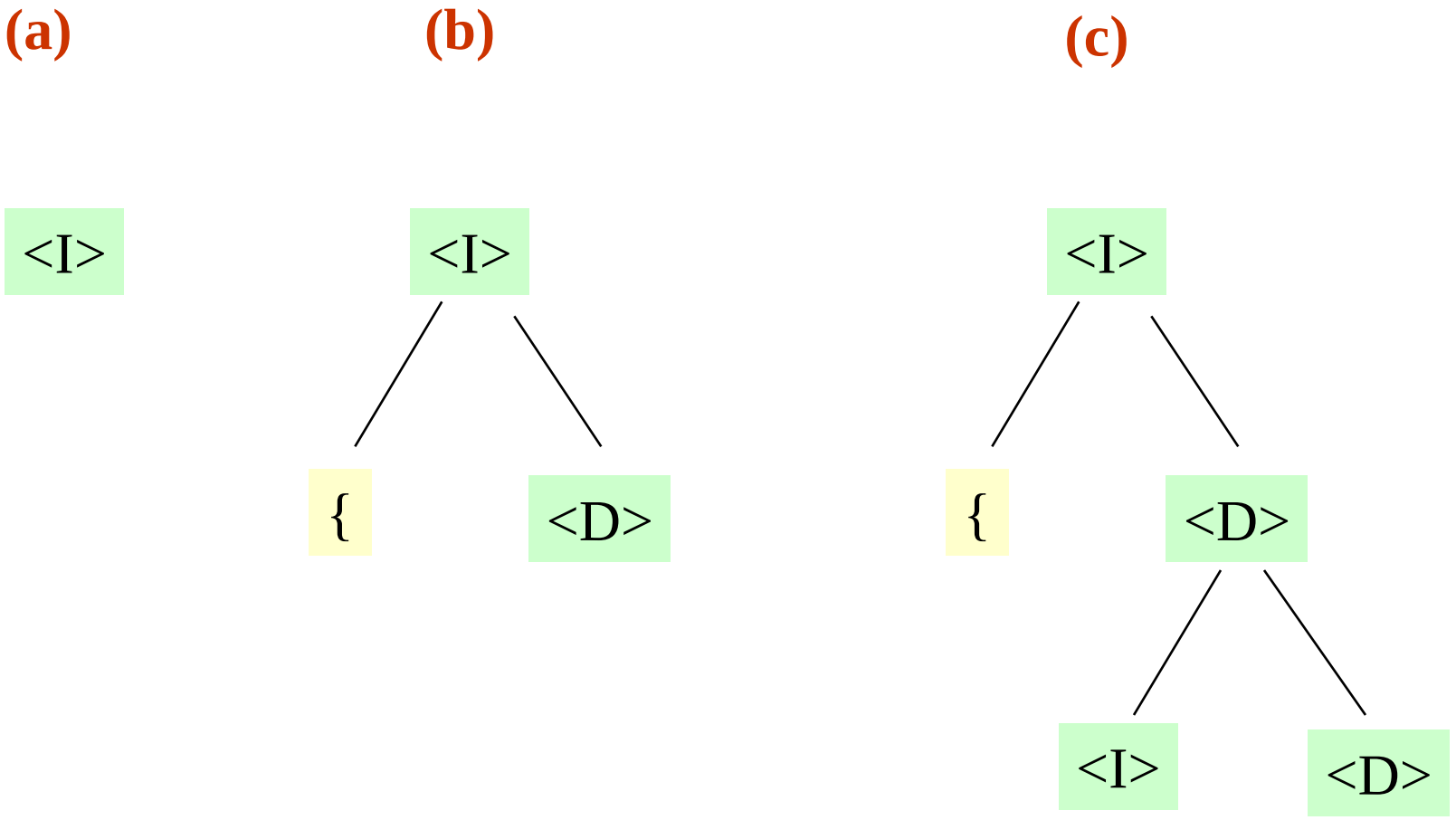
Konstruowanie drzewa rozbioru

Opisany algorytm określa czy dany ciąg znaków należy do danej kategorii syntaktycznej, ale nie tworzy drzewa rozbioru. Istnieje jednak możliwość wprowadzenia prostej modyfikacji algorytmu, pozwalającej również na utworzenie drzewa rozbioru, kiedy okaże się że ciąg wejściowy należy do kategorii syntaktycznej za pomocą której zainicjalizowano stos.

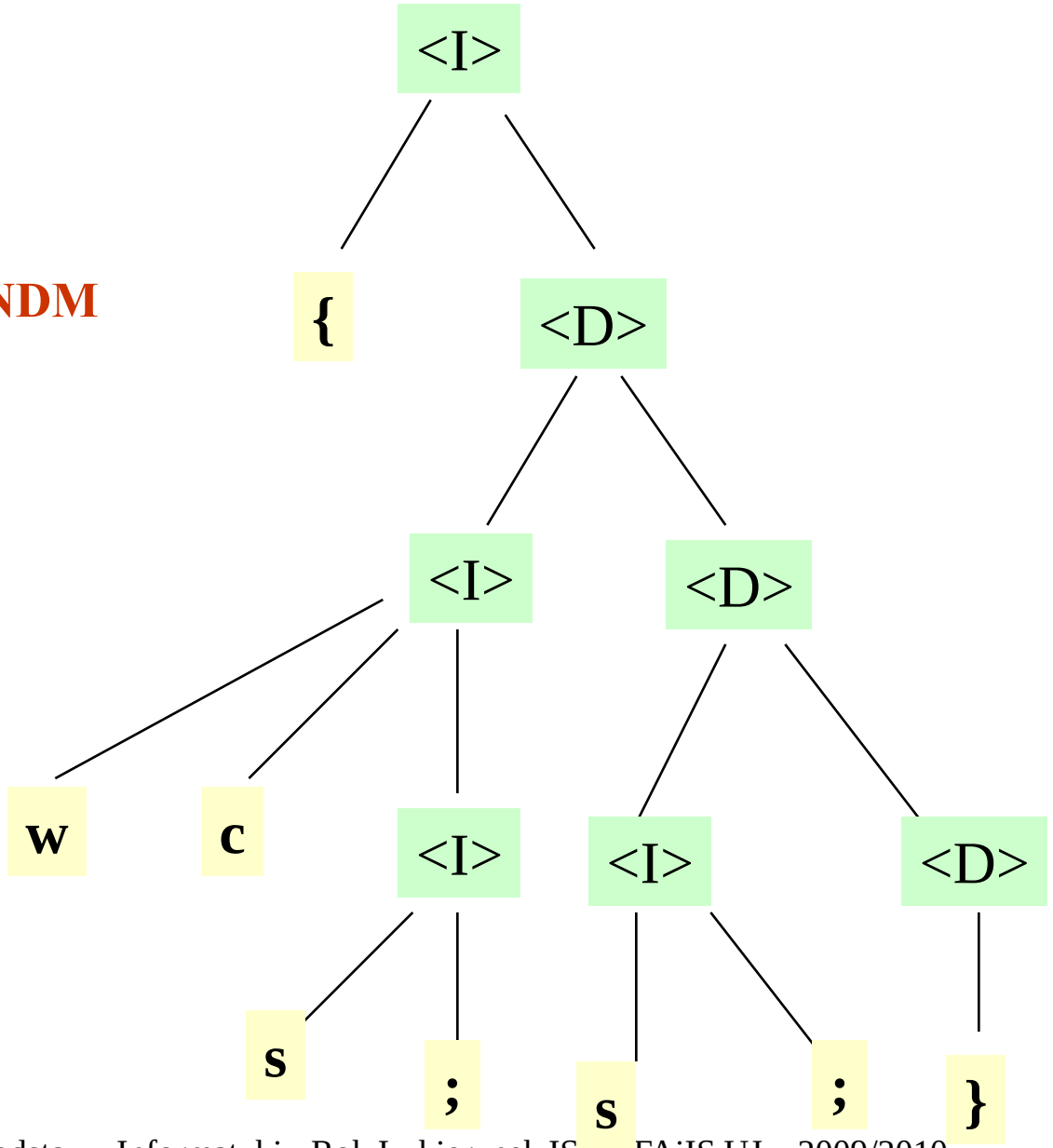
Analizator składniowy schodzenia rekurencyjnego, tworzy drzewo rozbioru wg. konwencji *wstępującej (ang. bottom-up),* tzn. rozpoczynając od liści i łącząc je w coraz większe poddrzewa w miarę kolejnych powrotów z wywołań funkcji.

W przypadku analizatora składniowego opartego na tabeli odpowiedniejszym sposobem jest budowanie drzewa według konwencji *zstępującej (ang. top-down).* Oznacza to rozpoczęcie konstrukcji od korzenia i w miarę wybierania kolejnych produkcji, za pomocą których maja być rozszerzane kategorie syntaktyczne na szczycie stosu, jednocześnie tworzy się potomków pewnego wierzchołka należącego do konstruowanego drzewa. Potomkowie ci odpowiadają symbolom należącym do części zasadniczej wybranej produkcji.

Etapy konstruowania drzewa rozbioru dla ciągu: {w c s ; s ; } ENDM



Pełne drzewo rozbioru
dla analizy składniowej
dla ciągu: {w c s ; s ; } ENDM



Konwertowanie gramatyk na możliwe do analizy składniowej.

Wiele gramatyk wymaga wprowadzenia modyfikacji aby można było przeprowadzając analizę składniową za pomocą metody schodzenia rekurencyjnego lub opartej na tabeli.

a) eliminujemy *rekurencje lewostronna* , tzn.

$\langle L \rangle \rightarrow \langle L \rangle \langle I \rangle \mid \varepsilon$

zamieniamy na

$\langle L \rangle \rightarrow \langle I \rangle \langle L \rangle \mid \varepsilon$

b) stosujemy *lewostronny rozkład na czynniki*, tzn. tworzymy nowe kategorie syntaktyczne $\langle X \rangle \rightarrow \langle C \rangle \langle D \rangle$ jeżeli dwie produkcje dla $\langle X \rangle$ zaczynają się wspólnym symbolem C.

Gramatyki a wyrażenia regularne

Zarówno gramatyki jak i wyrażenia regularne są notacjami służącymi do opisywania języków.

=> Dotychczas pokazaliśmy że notacja wyrażeń regularnych jest równoważna z dwiema innymi notacjami – automatami deterministycznymi oraz niedeterministycznymi.

=> Gramatyki dają większą możliwość opisu od notacji wyrażeń regularnych. Każdy język możliwy do opisu przez wyrażenia regularne można też opisać przy pomocy gramatyk. Istnieją natomiast języki które można opisać za pomocą gramatyk, ale nie można za pomocą wyrażeń regularnych.

Symulowanie wyrażeń regularnych za pomocą gramatyk

Twierdzenie:

Dla każdego wyrażenia regularnego R istnieje gramatyka, taka, że dla jednej z należących do niej kategorii syntaktycznych <S> zachodzi związek

$$L(<S>) = L(R).$$

Podstawa:

Przypadek podstawowy to n=0, gdzie wyrażenie regularne R posiada zero wystąpień operatorów. Wówczas R jest albo pojedynczym symbolem, np. x, albo jest ε lub $\varnothing$. Tworzymy nową kategorie syntaktyczną <S>. Gdy R=x, tworzymy również produkcje $<S> \rightarrow x$. Zatem $L(<S>) = \{x\}$, zaś L(R) jest tym samym językiem zawierającym jeden ciąg znaków. Jeżeli R jest równe ε, w podobny sposób tworzymy produkcję $<S> \rightarrow \varepsilon$ dla <S>, a jeśli $R = \varnothing$, nie tworzymy dla <S> w ogóle żadnej produkcji. Wówczas $L(<S>)$ to $\{\varepsilon\}$, kiedy R jest ε, oraz $L(<S>)$ jest $\varnothing$, kiedy R jest $\varnothing$.

Indukcja:

Załóżmy, że hipoteza indukcyjna jest spełniona w przypadku wyrażeń regularnych o n lub mniejszej liczbie wystąpień operatorów. Niech R będzie wyrażeniem regularnym o n+1 wystąpieniach operatorów. Istnieją trzy przypadki, w zależności od tego, czy ostatnim operatorem użytym do skonstruowania wyrażenia regularnego R jest operator sumy, złożenia czy domknięcia.
Zakładamy, że mamy gramatykę G1 z kategorią syntaktyczną <S1> oraz gramatykę G2 z kategorią syntaktyczną <S2>, takie, ze L(<S1>) = L(R1) oraz L(<S2>) = L(R2).

(1) R = R1 | R2. Tworzymy nową kategorię syntaktyczną <S> oraz do produkcji dodajemy <S> $\rightarrow$ <S1> | <S2>. Wówczas L(<S>) = L(R1) $\cup$ L(R2) = L(R).

(2) R = R1 R2. Tworzymy nową kategorię syntaktyczną <S> oraz do produkcji dodajemy <S> $\rightarrow$ <S1> <S2>. Wówczas L(<S>) = L(R1) $\cup$ L(R2) = L(R).

(3) R = R1*. Tworzymy nową kategorię syntaktyczną <S> oraz do produkcji dodajemy <S> $\rightarrow$ <S1> <S> | ε. Wówczas L(<S>) = L(<S1>)*, ponieważ <S> generuje ciągi znaków zawierające zero lub więcej kategorii <S1>.

Konstrukcja gramatyki dla wyrażenia regularnego: a | bc*

1. Tworzymy kategorie syntaktyczne dla trzech symboli, które pojawiają się w tym wyrażeniu:
   $\langle A \rangle \rightarrow a$ $\langle B \rangle \rightarrow b$ $\langle C \rangle \rightarrow c$

2. Tworzymy gramatykę dla c*: $\langle D \rangle \rightarrow \langle C \rangle \langle D \rangle \mid \varepsilon$
   Wówczas $L(\langle D \rangle) = L(\langle C \rangle))^* = c^*$

3. Tworzymy gramatykę dla bc*: $\langle E \rangle \rightarrow \langle B \rangle \langle D \rangle$

4. Tworzymy gramatykę dla całego wyrażenia regularnego a | bc*:
   $\langle F \rangle \rightarrow \langle A \rangle \mid \langle E \rangle$

końcowa postać gramatyki

$\langle F \rangle \rightarrow \langle A \rangle \mid \langle E \rangle$
$\langle E \rangle \rightarrow \langle B \rangle \langle D \rangle$
$\langle D \rangle \rightarrow \langle C \rangle \langle D \rangle \mid \varepsilon$
$\langle A \rangle \rightarrow a$
$\langle B \rangle \rightarrow b$
$\langle C \rangle \rightarrow c$

Język posiadający gramatykę ale nie posiadający wyrażenia regularnego

Język E będzie zbiorem znaków składających się z jednego lub większej liczby symboli 0, po których występuje ta sama liczba symboli 1, to znaczy:

E = { 01, 0011, 000111, ...}

W celu opisania ciągów znaków języka E można użyć przydatnej notacji opartej na wykładnikach. Niech s^n, gdzie s jest ciągiem znaków, zaś n liczba całkowitą, oznacza ss....s (n razy), to znaczy s złożone ze sobą n razy. Wówczas:

$E = \{0^1 1^1, 0^2 1^2, 0^3 1^3, ...\}$ lub $E = \{0^n 1^n \mid n \geq 1\}$

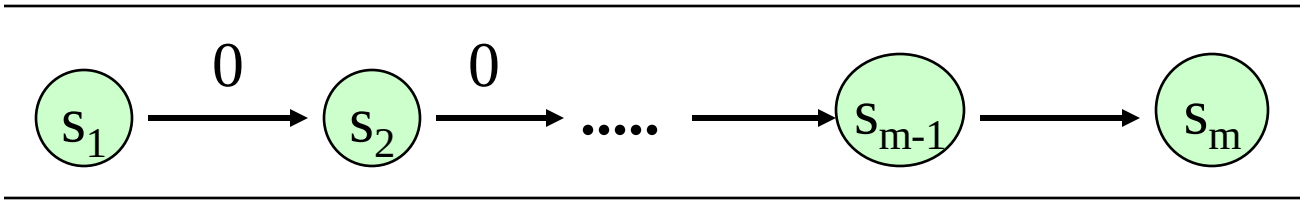
Język E można zapisać za pomocą gramatyki:

$\langle S \rangle \rightarrow 0 \langle S \rangle 1$
$\langle S \rangle \rightarrow 0\ 1$

Ponieważ nie istnieją żadne inne ciągi znaków możliwe do utworzenia na podstawie tych dwóch produkcji, E = L(<S>).

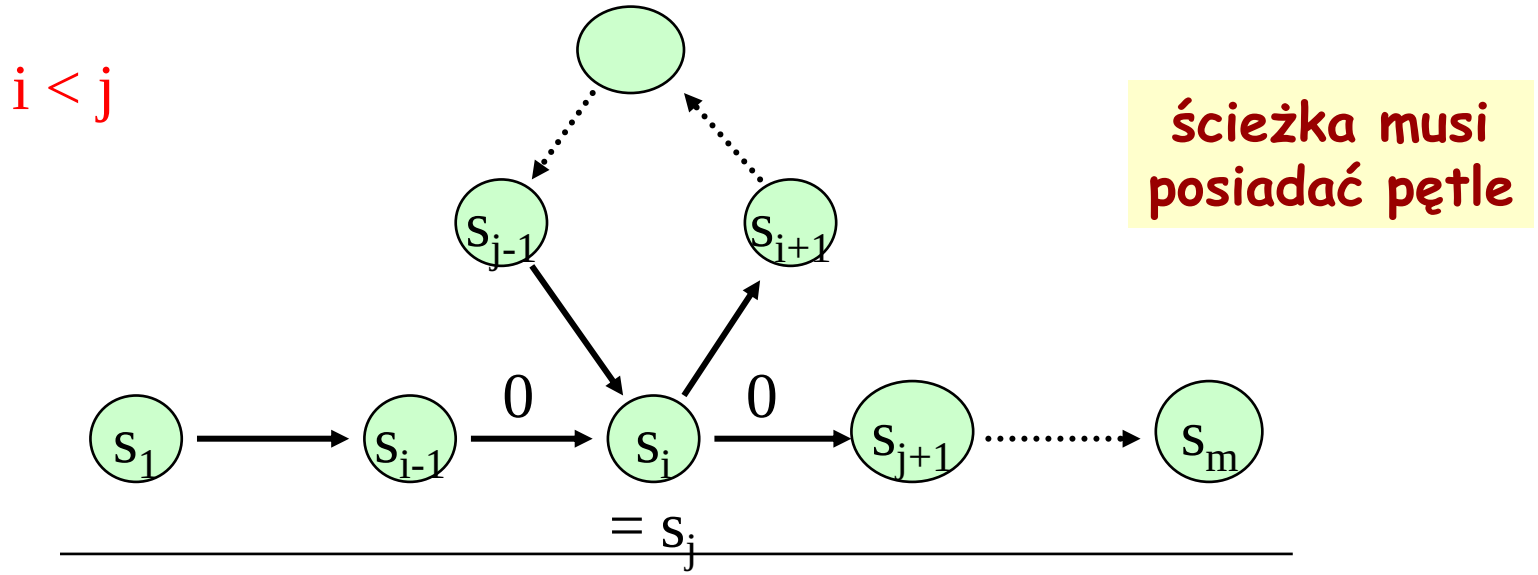
Dowód niedefiniowalnosci języka E za pomocą dowolnego wyrażenia regularnego.

Dowód niedefiniowalnosci języka E poprzez wyrażenie regularne przeprowadza się wykazując brak możliwości zbudowania deterministycznego automatu skończonego (a wiec brak wyrażenia regularnego który można by zamienić na równoważny automat deterministyczny).



wprowadzanie symboli 0 do automatu *A*

Załóżmy, że język E jest językiem pewnego deterministycznego automatu skończonego A. Wówczas A posiada pewną liczbę stanów, np. m. Jeżeli automat A otrzymuje na wejściu ciąg 000..... Stan początkowy nieznanego automatu nosi nazwę s0. Automat musi posiadać przejście z s0 do s1 itd.itd. Odczytanie stanu i powoduje przejście do si. Automat jednak nie może pamiętać ile symboli 0 odczytano, niektóre stany mogą być zdegenerowane (pętle).



Automat nie może „pamiętać" ile symboli 0 odczytano. Jeżeli znajduje się w stanie sm, odczytanych mogło zostać dokładnie m symboli 0. Wówczas musi być prawda, że jeżeli rozpocznie się w stanie m i wprowadzi do automatu *A* dokładnie m symboli 1, to automat *A* dochodzi do stanu akceptującego.

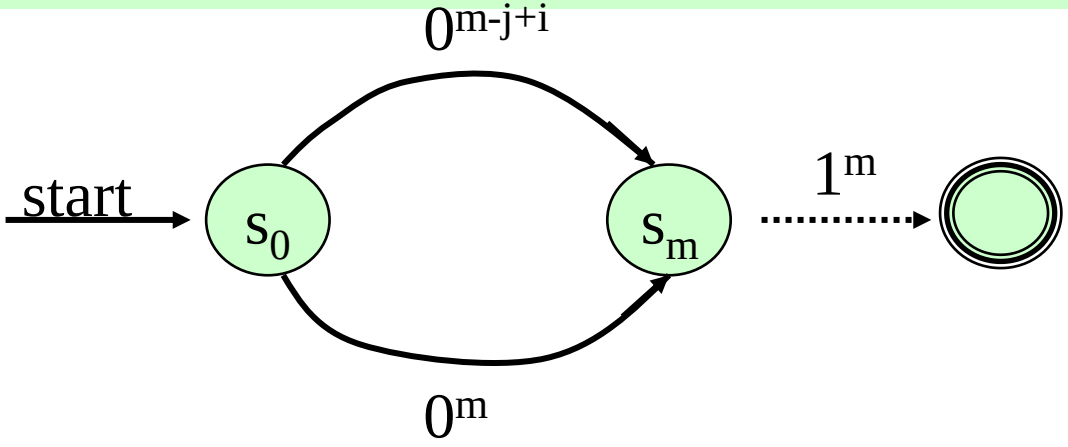
Załóżmy, że do automatu wprowadzono m-j+i symboli 0. i symboli 0 powoduje przejście automatu *A* ze stanu s0 do stanu si, który jest tym samym stanem, co sj. Widać, że m-j symboli 0 powoduje przejście automatu *A* ze stanu sj do stanu sm. Zatem m-j+i symboli 0 powoduje przejście automatu *A* ze stanu s0 do stanu sm.

Stad m-j+i symboli 0, po których następuje m symboli 1 powoduje przejście automatu A ze stanu s0 do stanu akceptującego. Innymi słowy, ciąg znaków $0^{m-j+i}\,1^m$ należy do języka automatu A. Jednak z uwagi na fakt ze j >i, ciąg ten posiada więcej symboli 1 niż symboli 0 i nie należy do języka E. Można wyciągnąć wniosek, że język automatu A nie odpowiada dokładnie językowi E, co jest sprzeczne z przyjętym założeniem.

Wystartowaliśmy z założenia że język E posiada deterministyczny automat skończony i doszliśmy do sprzeczności. A więc założenie było fałszywe. Stąd wnioskujemy że język E nie posiada też wyrażenia regularnego.

Język $E = \{0^n1^n \mid n \geq 1\}$ to jeden z przykładów nieskończonej liczby języków, które można określić za pomocą gramatyki, ale nie można za pomocą wyrażenia regularnego.

Automat A nie ma możliwości stwierdzenia, czy odczytano m symboli 0 lub m-j+i symboli 0



Tylko definicje.....

Gramatyka jest *prawostronnie liniowa,* jeżeli każda produkcja ma postać: $A \to w\,B$ lub $A \to w$.

Gramatyka jest *lewostronnie liniowa,* jeżeli każda produkcja ma postać: $A \to Bw$ lub $A \to w$.

Gramatyka która jest *lewostronnie liniowa,* lub *prawostronnie liniowa* to *gramatyka regularna*.

Gramatyka nieograniczona to taka, która dopuszcza produkcje o postaci $\alpha \to \beta$, gdzie α, β są dowolnymi łańcuchami symboli tej gramatyki, przy czym $\alpha \neq \beta$.

Gramatyka kontekstowa to taka gramatyka nieograniczona, dla której β jest co najmniej tak długie jak α.

Posumowanie

⇒ Gramatyka bezkontekstowa wykorzystuje model funkcji rekurencyjnych.
⇒ Jednym z ważnych zastosowań gramatyk są specyfikacje języków programowania. Gramatyki stanowią zwięzłą notacje opisu ich składni.
⇒ Drzewa analizy składniowej (drzewa rozbioru) stanowią formę reprezentacji, która przedstawia strukturę ciągu znaków zgodną z daną gramatyką.
⇒ Niejednoznaczność – to problem, który pojawia się w sytuacji gdy ciąg znaków posiada dwa lub więcej odrębnych drzew analizy składniowej, przez co nie posiada unikatowej struktury zgodnie z daną gramatyką

Posumowanie

⇒Metoda zamiany gramatyki na analizator składniowy to algorytm pozwalający stwierdzić, czy dany ciąg znaków należy do pewnegojęzyka.

⇒ Gramatyki posiadają większe możliwości w zakresie opisu języków niż wyrażenia regularne. Gramatyki oferują co najmniej tak samo duże możliwości opisu języków, jak wyrażenia regularne przez przedstawienie sposobu symulowania wyrażeń regularnych za pomocą gramatyk. Istnieją jednakże języki które można wyrazić za pomocą gramatyk, ale nie można za pomocą wyrażeń regularnych.