Teoretyczne podstawy informatyki

Repetytorium 3: automaty, wyrażenia regularne, gramatyki

Wzorce i automaty

- Problematyka wzorców stanowi bardzo rozwiniętą dziedzinę wiedzy. Nosi ona nazwę teorii automatów lub teorii języków, a jej podstawowe definicje i techniki stanowią istotną część informatyki.
- Poznamy trzy równoważne opisy wzorców:
  - oparty na teorii grafów, polegać będzie na wykorzystaniu ścieżek w grafie szczególnego rodzaju który nazwiemy automatem.
  - o charakterze algebraicznym, wykorzystujący notacje wyrażeń regularnych.
  - oparty o wykorzystanie definicji rekurencyjnych, nazwany gramatyką bezkontekstową.

Wzorzec

- **Wzorzec** to zbiór obiektów o pewnej rozpoznawalnej właściwości.
- Jednym z typów wzorców jest zbiór ciągów znaków, taki jak zbiór poprawnych identyfikatorów języka C, z których każdy stanowi ciąg znakowy, składający się z liter, cyfr i znaków podkreślenia oraz rozpoczyna się od litery lub znaku podkreślenia.
- Inny przykład to zbiór tablic zer i jedynek o danym rozmiarze, które czytnik znaków może interpretować jako reprezentację tego samego symbolu. Poniżej trzy tablice które można interpretować jako literę **A**.
- Zbiór wszystkich takich tablic stanowiłby wzorzec o nazwie „**A**”.

0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	0	1	0	1	0	0
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	1	0	0	0	1	1
1	0	0	0	0	0	1

0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	1	0	0	0
0	1	1	1	0	0	0
1	0	0	1	1	0	0
1	0	0	0	1	0	0

0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	1	0	1	0	0
0	1	1	1	1	1	0
1	1	0	0	0	1	1
1	0	0	0	0	0	1
0	0	0	0	0	0	0

Maszyny stanów i automaty

- **Programy służące do wyszukiwania wzorców** często charakteryzują się szczególną strukturą. W kodzie można identyfikować określone pozycje, w których można wyróżnić pewne charakterystyczne elementy związane z postępem działania programu w zakresie osiągnięcia stawianego mu celu, jakim jest znalezienie wystąpienia wzorca.
- Takie pozycje określa się mianem **stanów** (ang. *states*). Ogólne zachowanie programu można postrzegać jako **przechodzenie od jednego stanu do drugiego** w miarę odczytywania danych wejściowych.

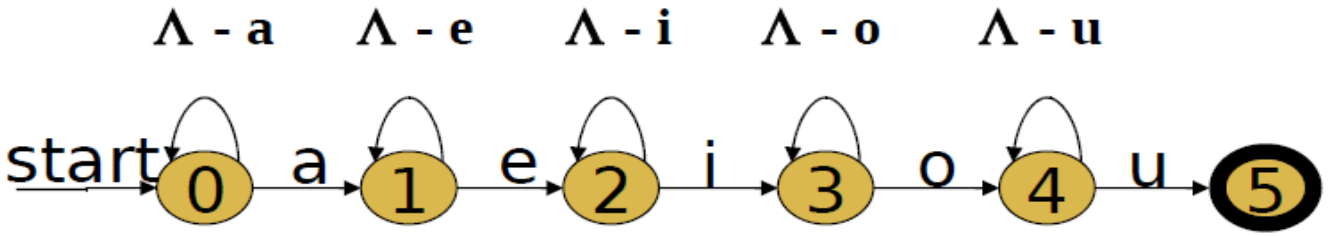
Przykład

Przykład:

- Prosty program w języku **C**, sprawdzający ciąg znaków w celu określenia czy zawiera on wszystkie pięć samogłosek w kolejności alfabetycznej. Rozpoczynając analizę od początku ciągu znaków, program najpierw wyszukuje znak **a**.
- Można powiedzieć że pozostaje w stanie **0** do czasu, aż znajdzie wystąpienie **a**, a wówczas przechodzi do stanu **1**. W stanie **1**, szuka wystąpienia znaku **e**, a kiedy je znajdzie, przechodzi do stanu **2**... Stan **i** można interpretować jako sytuacje, w której program napotkał już po kolei **i** pierwszych samogłosek, gdzie **i=0,1, ..., 5**.
- Owe sześć stanów stanowi całość wymaganych przez program informacji podczas przeglądania ciągu wejściowego od lewej do prawej strony.

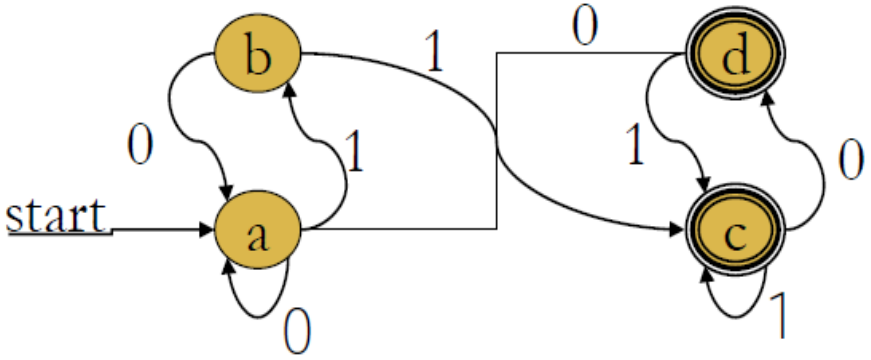
Automat rozpoznający ciągi liter

- **Stany programu są reprezentowane za pomocą grafu skierowanego**, którego krawędzie etykietuje się zbiorami znaków. Krawędzie takie nazywa się **przejściami** (ang. *transition*).
- Niektóre wierzchołki są zaznaczone jako **stany końcowe** (ang. *accepting states*)**.** Osiągnięcie takiego stanu oznacza znalezienie poszukiwanego wzorca i jego akceptację.
- Jeden z wierzchołków jest zawsze określany jako **stan początkowy** (ang. *start state*) – stan w którym następuje rozpoczęcie procesu rozpoznawania wzorca. Wierzchołek taki oznacza się przez umieszczenie prowadzącej do niego strzałki która nie pochodzi od innego wierzchołka. Graf posiadający opisywaną postać nosi nazwę **automatu skończonego** (ang. *finite automaton*) lub po prostu automatu.



Filtr odbijający

- Automat pobiera ciąg zer i jedynek.
- Celem jego jest „wygładzenie” ciągu przez traktowanie pojedynczego symbolu **0**, który jest otoczony dwoma symbolami **1** jako „szumu” i zastąpienie go symbolem **1**.
- W podobny sposób jest traktowany symbol **1** gdy jest otoczony dwoma symbolami **0** – zastępujemy go symbolem **0**.
- Stanem początkowym jest a. Stanami końcowymi (akceptującymi) są c i d.



Wejście	Stan	Wyjście
	a	0
0		
	a	0
1		
	b	0
0		
	a	0
1		
	b	0
1		
	c	1
0		
	d	1
1		
	c	1

Automaty a ich programamy

- **Automaty stanowią abstrakcje**. Podejmują decyzje o akceptacji lub odrzuceniu danego ciągu znaków wejściowych przez sprawdzenie, czy istnieje ścieżka od stanu początkowego do pewnego stanu końcowego, której krawędzie będą zaetykietowane wartościami ciągu.
- Działanie filtru można interpretować jako fakt że automat odrzuca podciągi: **0,01,010,0101,** natomiast akceptuje podciągi **01011, 010110, 0101101**.
- Działanie automatu rozpoznającego litery jest takie że akceptuje ciągi znaków, takich jak „abstemiou”, ale odrzuca ciąg „abstemious”, ponieważ nie da się przejść nigdzie ze stanu 5 w przypadku końcowego znaku s.

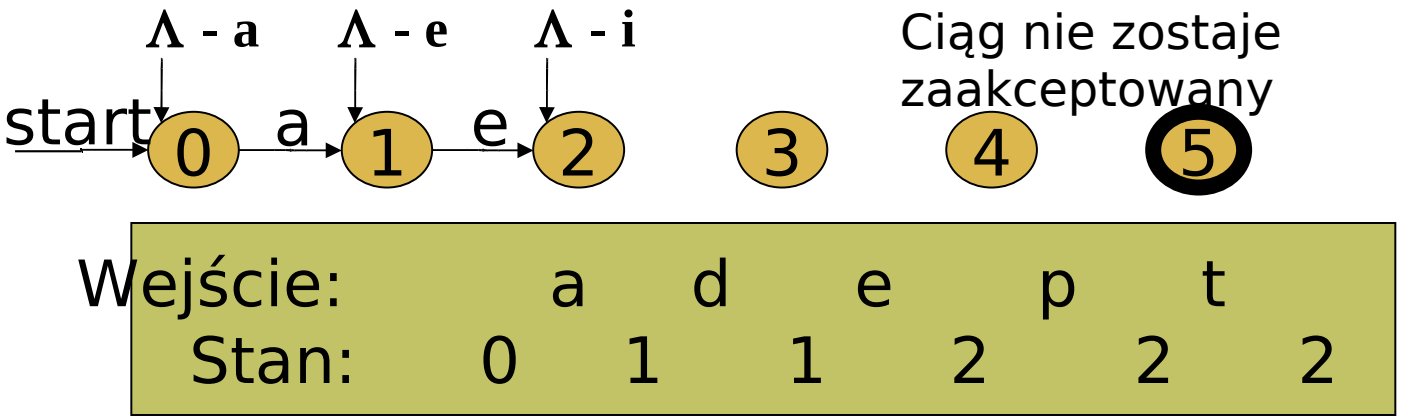
Automaty a ich programy

- **Programy tworzone dla automatów mogą podejmować decyzje o akceptacji lub odrzuceniu na różne sposoby.**
- Przykładowo program akceptujący ciągi znaków i wykorzystujący automat ze slajdu 6. powinien zaakceptować zarówno słowo „abstemious” jak i „abstemiou”.
- W momencie przetworzenia litery **u** program wypisywałby całe słowo bez dalszego badania go.
- Program wykorzystujący automat ze slajdu 8. powinien interpretować każdy stan akceptacji jako akcję polegającą na wypisaniu znaku **1**, zaś każdy stan odrzucenia jako akcję polegającą na wypisaniu znaku **0**.

Automaty deterministyczne i niedeterministyczne

- Jedna z podstawowych operacji wykonywanych za pomocą automatu jest pobranie ciągu symboli $\mathbf{a_1,a_2, \ldots, a_k}$ i przejście od stanu początkowego ścieżką, której krawędzie posiadają etykiety zawierające te symbole po kolei.
- Tzn. dla **i = 1, 2, …, k** symbol $\mathbf{a_i}$ należy do zbioru $\mathbf{S_i}$, który etykietuje **i**-ta krawędź ścieżki.
- Konstruowanie takiej ścieżki oraz sekwencji jej kolejnych stanów określa się mianem **symulacji** (ang. *simulation*) automatu dla ciągu wejściowego $\mathbf{a_1,a_2, \ldots, a_k}$.
- O takiej ścieżce mówi się że posiada etykietę $\mathbf{a_1,a_2, \ldots, a_k}$.

Symulacja działania automatu ze str. 6 dla słowa ‘’ adept ‘’.

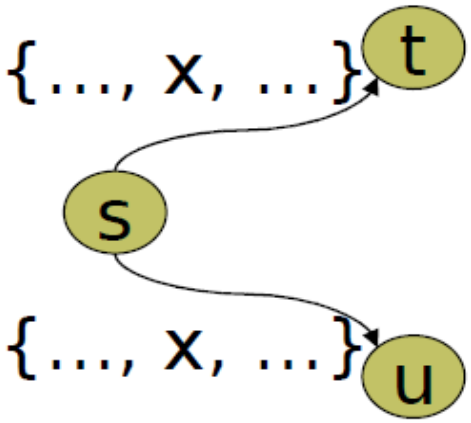


Automaty deterministyczne

- Automaty które dotychczas omówiliśmy posiadają pewną istotną własność.
- Dla każdego stanu **s** oraz dowolnego znaku wejściowego **x** istnieje co najwyżej jedno przejście ze stanu **s**, którego etykieta zawiera znak **x**.
- O tego rodzaju automatach mówimy że są **deterministyczne** (ang. *deterministic*).
- Symulacja automatu deterministycznego dla danej sekwencji wejściowej jest bardzo prosta. W każdym stanie **s** dla kolejnego znaku wejściowego **x** należy rozpatrzyć każdą etykietę przejść ze stanu **s**. Jeżeli uda się znaleźć przejście, którego etykieta zawiera znak **x**, to przejście to określa właściwy stan następny. Jeżeli żadna nie zawiera znaku **x**, to automat „zamiera” i nie może przetwarzać dalszych znaków wejściowych.

Automaty niedeterministyczne

- **Automaty niedeterministyczne** (ang. *nondeterministic*) mogą (ale nie muszą) posiadać dwa (lub więcej) przejścia z danego stanu zawierające ten sam symbol.
- Warto zauważyć, że automat deterministyczny jest równocześnie automatem niedeterministycznym, który nie posiada wielu przejść dla jednego symbolu.
- Automaty niedeterministyczne nie mogą być bezpośrednio implementowane za pomocą programów, ale stanowią przydatne pojęcie abstrakcyjne.



W momencie podjęcia próby symulacji automatu niedeterministycznego w przypadku ciągu wejściowego składającego się ze znaków $\mathbf{a_1, a_2, \ldots, a_k}$ może okazać się, że ten sam ciąg etykietuje wiele ścieżek. Wygodnie jest przyjąć, że automat niedeterministyczny akceptuje taki ciąg wejściowy, jeżeli co najmniej jedna z etykietowanych ścieżek prowadzi do stanu akceptującego.

niedeterminizm = „zgadywanie''

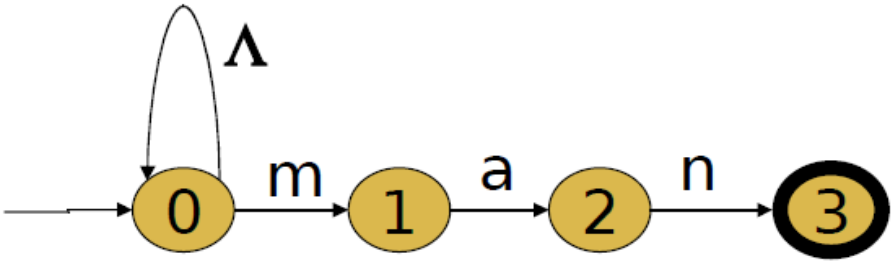
Równoważność automatów

- Mówimy, że automat **A** jest **równoważny** (ang. *equivalent*) automatowi **B**, jeżeli akceptują ten sam zbiór ciągów wejściowych.
Innymi słowy, jeżeli $\mathbf{a_1a_2...a_k}$ jest dowolnym ciągiem symboli, to spełnione są dwa następujące warunki:
- Jeżeli istnieje ścieżka zaetykietowana jako $\mathbf{a_1a_2...a_k}$ wiodąca od stanu początkowego automatu **A** do pewnego stanu akceptującego automatu **A**, to istnieje również ścieżka zaetykietowana $\mathbf{a_1a_2...a_k}$ wiodąca od stanu początkowego automatu **B** do stanu końcowego tego automatu.
- Jeżeli istnieje ścieżka zaetykietowana jako $\mathbf{a_1a_2...a_k}$ wiodąca od stanu początkowego automatu **B** do pewnego stanu akceptującego automatu **B**, to istnieje również ścieżka zaetykietowana $\mathbf{a_1a_2...a_k}$ wiodąca od stanu początkowego automatu **A** do stanu końcowego tego automatu.

Konstrukcja automatu niedeterministycznego

- **Podstawa:**
  - Jeżeli stanem początkowym automatu niedeterministycznego **N** jest $\mathbf{s_0}$, to stanem początkowym automatu deterministycznego **D** jest $\mathbf{\{s_0\}}$, czyli zbiór zawierający tylko element $\mathbf{s_0}$.
- **Indukcja:**
  - Załóżmy, że ustalono, iż **S**, zbiór stanów automatu **N**, jest stanem automatu **D**.
  - Rozpatrujemy po kolei każdy możliwy znak wejściowy **x**. Dla danego **x** określamy, że **T** jest zbiorem stanów **t** automatu **N**, takich, że dla pewnego stanu **s** należącego do zbioru **S** istnieje przejście z **s** do **t** etykietą zawierającą znak **x**.
  - Wówczas zbiór **T** jest stanem automatu **D** i istnieje przejście z **S** do **T** względem znaku wejściowego **x**.

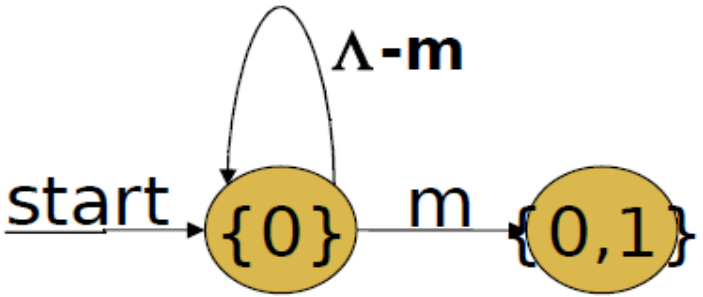
Konstrukcja automatu deterministycznego D



Niedeterministyczny automat rozpoznający ciąg znaków kończący się sekwencja ‘’ man ‘’.

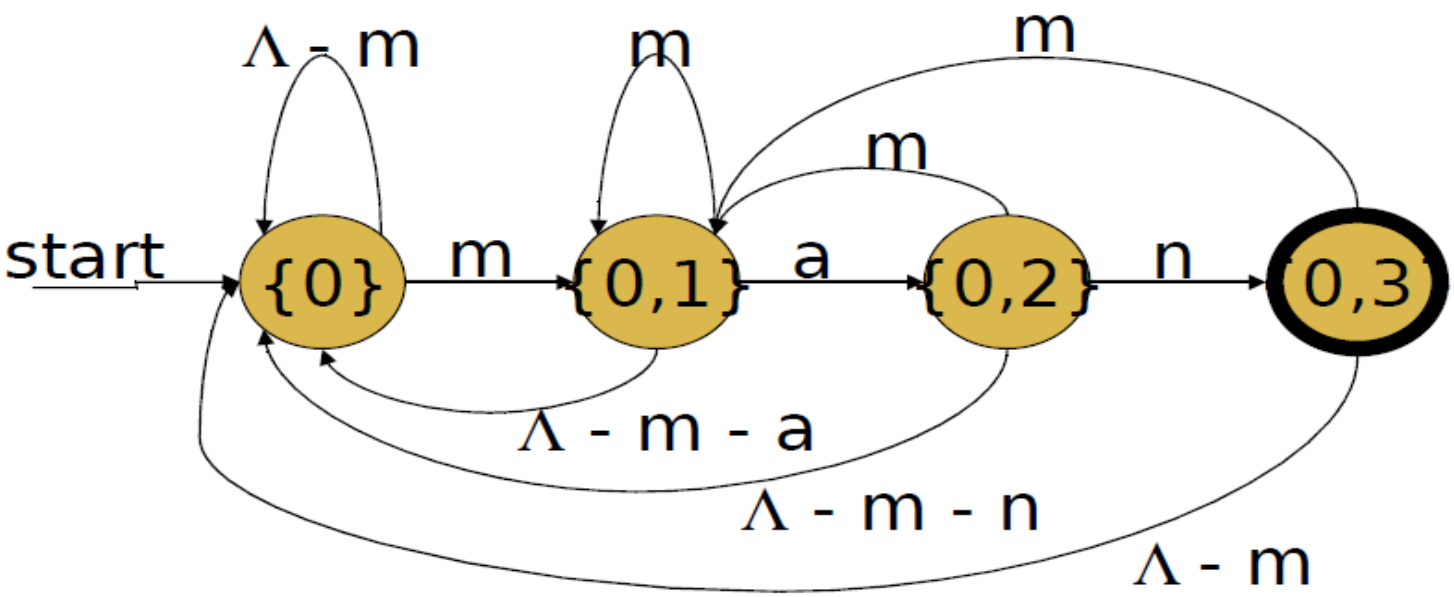
Rozpoczynamy od zbioru **{0}**,
który jest stanem początkowym automatu **D**.

Stan {0} i jego przejścia



Dla dowolnej litery oprócz **m** ze stanu 0 następuje przejście do stanu 0, w przypadku **m** następuje przejście do stanu 0 lub 1. Automat D potrzebuje stanu {0}, który już posiada oraz stanu {0,1} który należy dodać.

Automat deterministyczny D



Konstrukcje automatu można uznać za skończoną. Przejścia ze stanu **{0,3}** nie prowadza do żadnego stanu automatu **D** którego jeszcze nie sprawdziliśmy.
Stan **{0}** oznacza że odczytany ciąg nie kończy się żadnym przedrostkiem wyrazu **man**, stan **{0,1}** że kończy się sekwencją **m**, stan **{0,2}** że kończy się sekwencją **ma**, stan **{0,3}** że kończy się sekwencja **man**.

Minimalizacja automatów

- Jednym z zagadnień dotyczących automatów jest kwestia określenia **minimalnej liczby stanów** wymaganych do wykonania danego zadania.
- Posiadając pewien automat możemy zadać pytanie czy istnieje równoważny automat posiadający mniejszą liczbę stanów, a jeśli tak, jaka jest najmniejsza liczba stanów dowolnego automatu równoważnego.
- **Dla automatów deterministycznych zawsze istnieje pewien unikatowy automat deterministyczny o minimalnej liczbie stanów, równoważny z danym automatem.**
- Dowodzimy przez pokazanie że nie ma stanów nierównoważnych.
- Nie istnieje podobna teoria w sytuacji automatów niedeterministycznych.

Wyrażenia regularne

- **Wyrażenia regularne** (ang. *regular expressions*) stanowią algebraiczny sposób definiowania wzorców.
- Wyrażenia regularne stanowią analogię do algebry wyrażeń arytmetycznych oraz do algebry relacyjnej.
- Zbiór wzorców które można wyrazić w ramach algebry wyrażeń regularnych odpowiada dokładnie zbiorowi wzorców, które można opisać za pomocą automatów.

Operandy wyrażeń regularnych

- Wyrażenia regularne posiadają pewne rodzaje operandów niepodzielnych (ang. *atomic operands*). Poniżej lista:
  - Znak
  - Symbol ε
  - Symbol ∅
  - Zmienna która może być dowolnym wzorcem zdefiniowanym za pomocą wyrażenia regularnego.
- **Wartość wyrażenia regularnego jest wzorcem** składającym się ze zbioru ciągów znaków, który często **określa się mianem języka** (ang. *language*).
- Język określony przez wyrażenia regularne **E** oznaczony będzie jako **L(E)** lub określany jako **język wyrażenia E**.

Operatory wyrażeń regularnych

- **Suma:**
  - Symbol sumy (ang. *union*) oznacza się za pomocą symbolu **|** . Jeżeli **R** i **S** są dwoma wyrażeniami regularnymi, to **R | S** oznacza sumę języków określanych przez **R** i **S**. To znaczy **L(R| S) = L(R) ∪ L(S)**.
  - **L(R) i L(S) są zbiorami ciągów znakowych, notacja sumowania jest uzasadniona.**
- **Złożenie:**
  - Operator złożenia (ang. *concatenation*) nie jest reprezentowany przez żaden odrębny symbol.
  - Jeżeli **R** i **S** są wyrażeniami regularnymi to **RS** oznacza ich złożenie. **L(RS)**, czyli język określony przez **RS**, jest tworzony z języków **L(R)** i **L(S)** w sposób następujący:
    - Dla każdego ciągu znakowego **r** należącego do **L(R)** oraz każdego ciągu znakowego **s** należącego do **L(S)**, ciąg **rs**, czyli złożenie ciągów **r** i **s**, należy do **L(RS)**.
  - **Złożenie dwóch list takich jak ciągi znaków, jest wykonywane przez pobranie po kolei elementów pierwszej z nich i uzupełnienie ich po kolei elementami drugiej listy.**

Operatory wyrażeń regularnych

Domknięcie:

- Operator domknięcia (ang. *closure*), jest to operator jednoargumentowy przyrostkowy. Domknięcie oznacza się za pomocą symbolu *, tzn. **R*** oznacza domkniecie wyrażenia regularnego **R**. Operator domknięcia ma najwyższy priorytet.
- Efekt działania operatora domknięcia można zdefiniować jako „określenie występowania zera lub większej liczby wystąpień ciągów znaków w R”.
- Oznacza to że **L(R*)** składa się z:
  - Ciągu pustego ε, który można interpretować jako brak wystąpień ciągów znaków w **L(R)**.
  - Wszystkich ciągów znaków języka **L(R)**. Reprezentują one jedno wystąpienie ciągów znaków w **L(R)**.
  - Wszystkich ciągów znaków języka **L(RR)**, czyli złożenia języka **L(R)** z samym sobą. Reprezentują one dwa wystąpienia ciągów znaków z **L(R)**.
  - Wszystkich ciągów znaków języka **L(RRR), L(RRRR)** i tak dalej, które reprezentują trzy, cztery i więcej wystąpień ciągów znaków z **L(R)**.
- Nieformalnie można napisać: **R* = e | R | RR | RRR | …**

  Wyrażenie po prawej stronie to nie jest wyrażeniem regularnym ponieważ zawiera nieskończoną liczbę wystąpień operatora sumy. Wszystkie wyrażenia regularne są tworzone ze skończonej liczby wystąpień operatorów.

Kolejność operatorów wyrażeń regularnych

- Istnieje określona kolejność wykonywania trzech działań wyrażeń regularnych: sumy, złożenia oraz domknięcia. Kolejność ta jest następująca:
  - Domknięcie (najwyższy priorytet)
  - Złożenie
  - Suma (najniższy priorytet)
- **Przykład:**

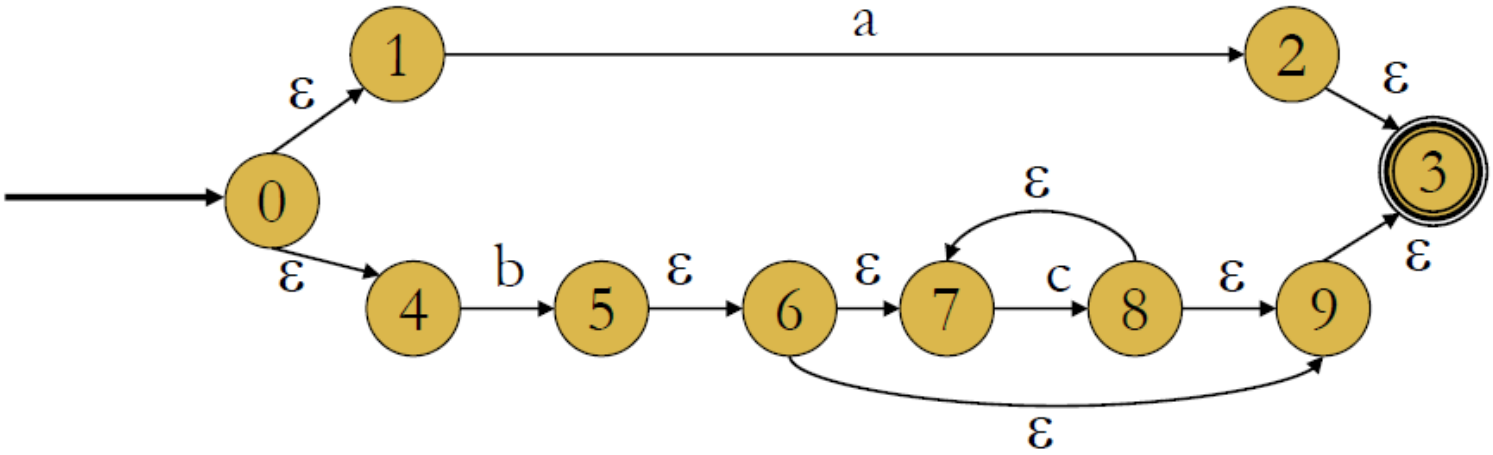
a | bc*d = (a | (b (c*)) d)

Od wyrażeń regularnych do automatów

- Istnieje **sposób na zamianę** dowolnego wyrażenia regularnego na automat niedeterministyczny, a następnie przez użycie **konstrukcji podzbiorów** – zamiany takiego automatu na automat deterministyczny.
- Istnieje także możliwość zamiany dowolnego automatu na wyrażenie regularne, którego język dokładnie odpowiada zbiorowi ciągów znaków akceptowanych przez automat. Stąd automaty i wyrażenia regularne dają te same możliwości opisywania języków.

Automaty z epsilon przejściami

- Należy rozszerzyć notacje używaną w przypadku automatów w celu umożliwienia opisu krawędzi **posiadających etykietę ε**. Takie automaty wciąż akceptują ciąg znaków **s** wtedy i tylko wtedy, gdy ścieżka zaetykietowana ciągiem **s** wiedzie od stanu początkowego do stanu akceptującego. **Symbol ε**, ciąg pusty, **jest „niewidoczny”** w ciągach znaków, stąd w czasie konstruowania etykiety danej ścieżki w efekcie usuwa się wszystkie symbole **ε** i używa tylko rzeczywistych znaków.



Automat z ε-przejściami dla wyrażenia a | bc*

Od wyrażeń regularnych do automatów z epsilon przejściami

- Wyrażenie regularne zamienia się na automat przy użyciu algorytmu opracowanego na podstawie **indukcji zupełnej** względem liczby wystąpień operatorów w wyrażeniu regularnym.
- **Twierdzenie S(n):**

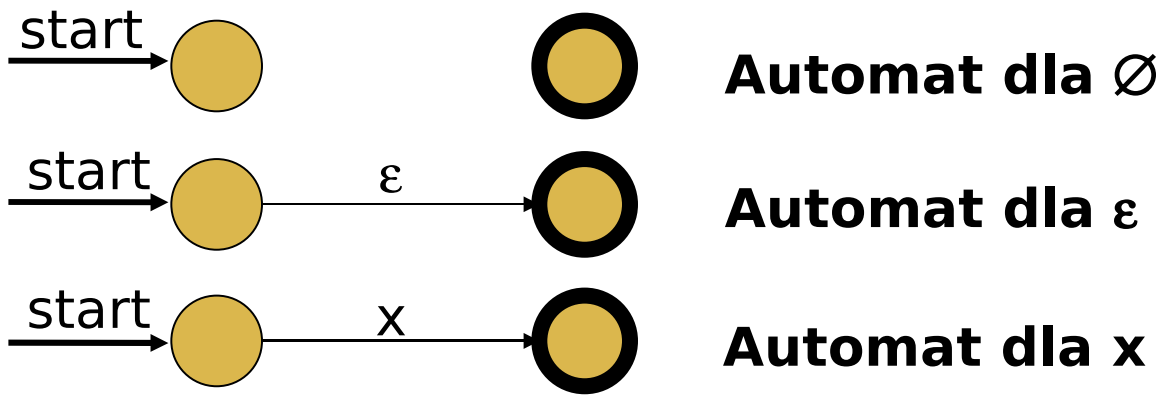
Jeżeli **R** jest wyrażeniem regularnym o **n** wystąpieniach operatorów i braku zmiennych jako operatorów niepodzielnych, to istnieje automat **A** z ε-przejściami, który akceptuje ciągi znaków należące do języka **L(R)** i żadne inne.

Ponadto automat **A**:

- posiada tylko jeden stan akceptujący,
- nie posiada krawędzi wiodących do jego stanu początkowego,
- nie posiada krawędzi wychodzących z jego stanu akceptującego.

Podstawa

- Jeżeli **n=0**, to **R** musi być operandem niepodzielnym, którym jest ∅, **ε** lub **x** dla pewnego symbolu **x**.
- Dla owych trzech przypadków można zaprojektować 2-stanowy automat, spełniający wymagania twierdzenia **S(0)**.



- **Automaty dla przypadków bazowych. Każdy spełnia warunki 1, 2, 3 (patrz poprzednia strona).**

Indukcja

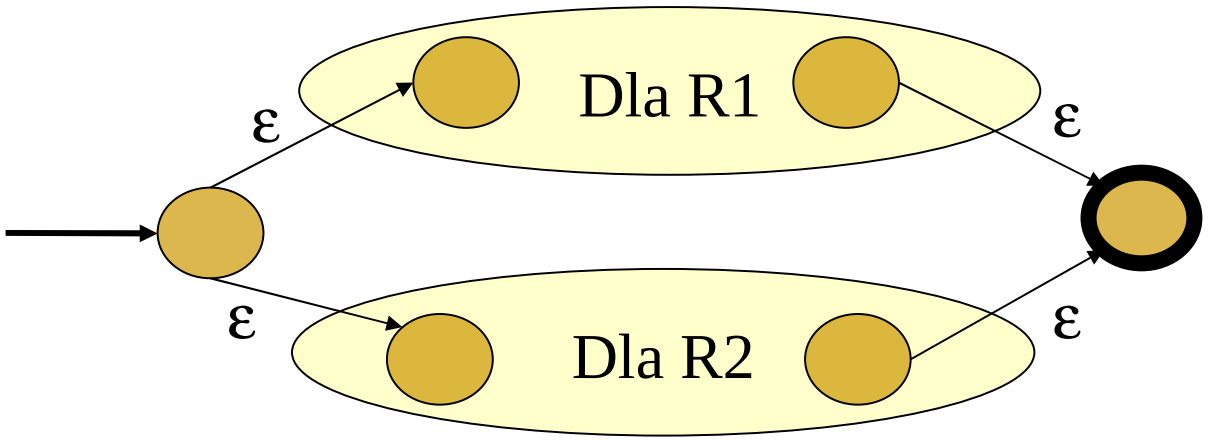
- Zakładamy teraz, ze **S(i)** jest prawdziwe dla wszystkich **i ≤ n**.
- To znaczy, że dla każdego wyrażenia regularnego **R** o maksymalnie **n** wystąpieniach istnieje automat spełniający warunek hipotezy indukcyjnej i akceptujący wszystkie ciągi znaków języka **L(R)** i żadnych innych.
- Zajmiemy się tylko najbardziej zewnętrznym operatorem w **R**, co oznacza, że wyrażenie **R** może mieć tylko formę

  R1 | R2, R1 R2, R1*

  w zależności od tego czy ostatni użyty operator był operatorem sumy, złożenia lub domknięcia.
- Wyrażenie **R1**, **R2** nie mogą posiadać więcej niż **n** operatorów.

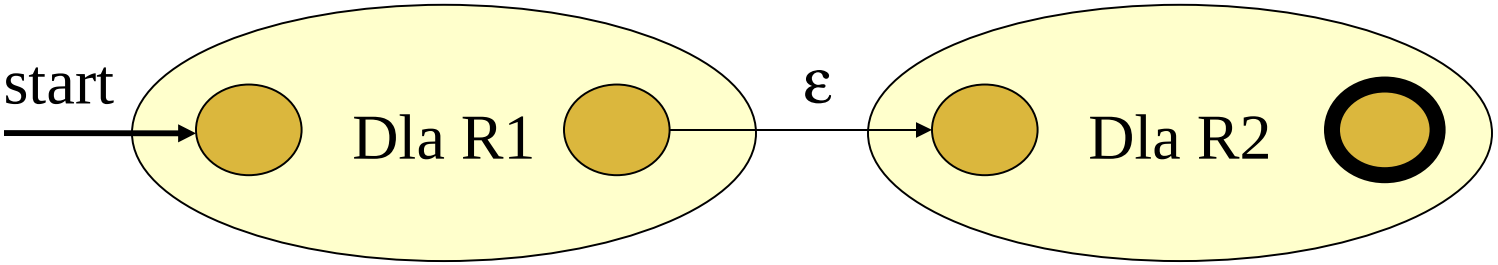
Przypadek 1: R = R1 | R2

- Przechodzimy krawędzią zaetykietowaną symbolem ε do stanu początkowego automatu dla **R1** lub automatu dla **R2**.
- Następnie przechodzimy do stanu akceptującego tego automatu, a później przejściem ε do stanu akceptującego automatu **R**.



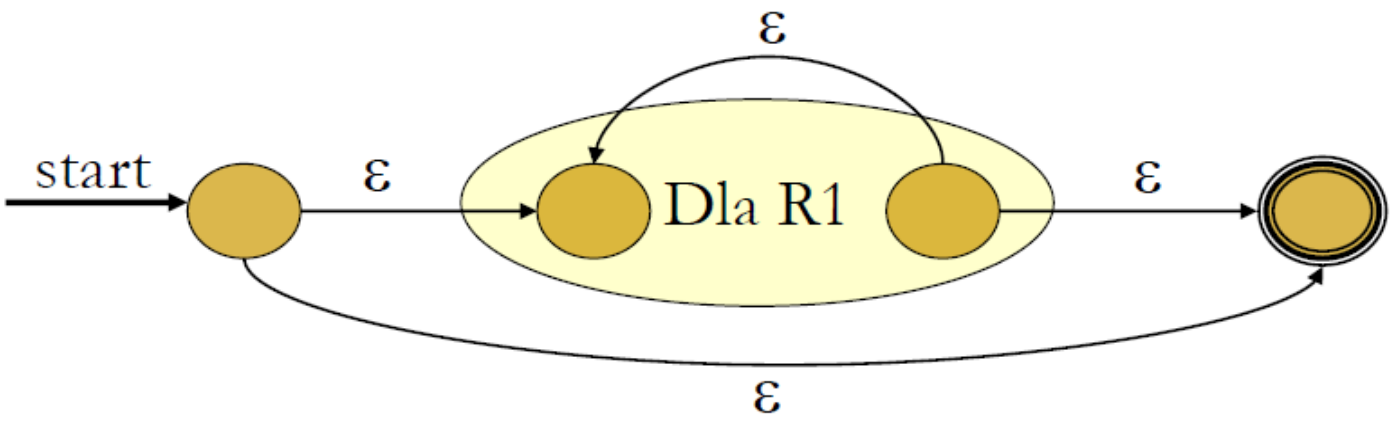
Przypadek 2: R = R1 R2

- Automat posiada jako swój stan początkowy stan początkowy automatu dla wyrażenia **R1**, a jako swój stan akceptujący – stan akceptujący dla wyrażenia **R2**.
- Dodajemy także **ε** - przejście ze stanu akceptującego automatu dla wyrażenia **R1** do stanu początkowego automatu dla wyrażenia **R2**.
- Stan akceptujący pierwszego automatu przestaje być stanem akceptującym, a stan początkowy drugiego automatu przestaje być stanem początkowym w skonstruowanym automacie.



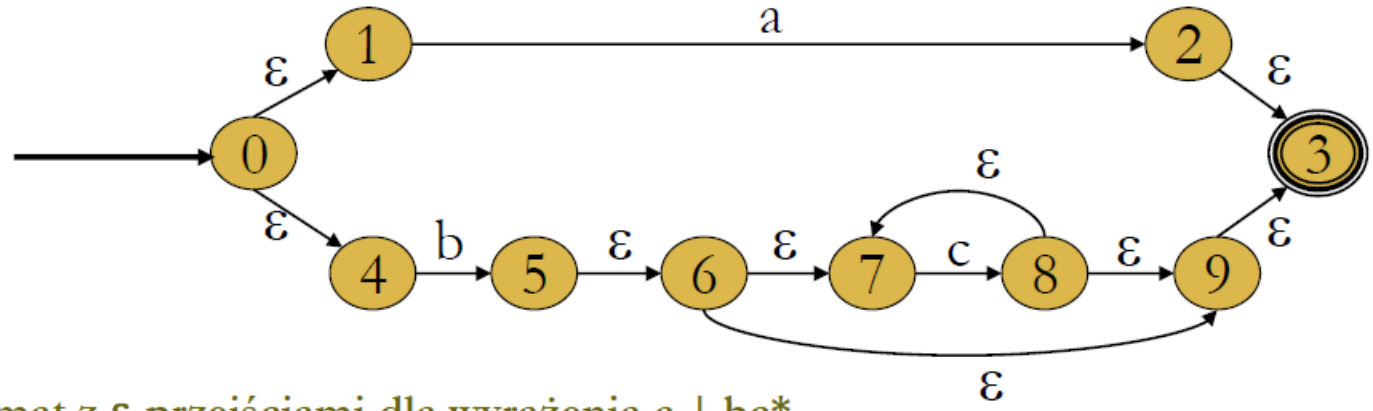
Przypadek 3: R = R1*

- Do automatu dla wyrażenia **R1** dodajemy nowy stan początkowy i akceptujący.
- Stan początkowy posiada **ε** przejście do stanu akceptującego (a więc akceptowany jest ciąg **ε**) oraz do stanu początkowego automatu dla wyrażenia **R1**.
- Stan akceptujący automatu dla wyrażenia **R1** otrzymuje **ε**-przejście z powrotem do swojego stanu początkowego oraz do stanu akceptującego automatu dla wyrażenia **R**.
- Stan początkowy i akceptujący automatu dla wyrażenia **R1** nie są stanami początkowym i akceptującym konstruowanego automatu. Etykiety ścieżek odpowiadają ciągom należącym do języka **L(R1*)** czyli **L(R)**.



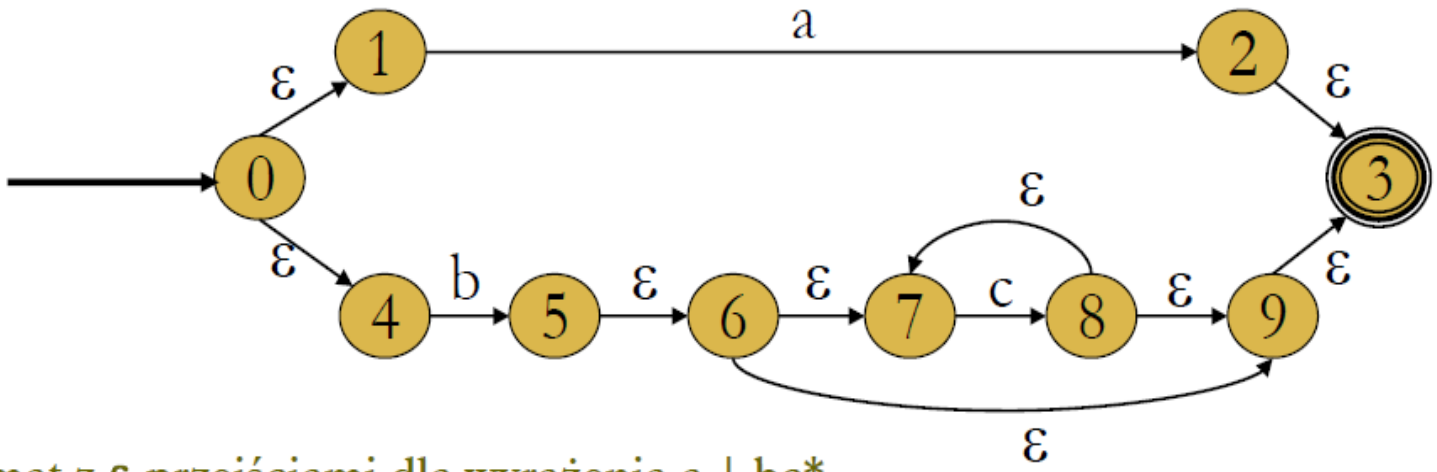
Eliminacja epsilon-przejść

- Jeżeli stanem bieżącym jest dowolny stan **s** automatu z **ε**-przejściami, oznacza to że jednocześnie stanem bieżącym jest dowolny stan, do którego można się dostać z **s** w wyniku przejścia ścieżki zawierającej krawędzie zaetykietowane symbolem **ε**.
- Wynika to z faktu, że bez względu na to, jaki ciąg etykietuje wybraną ścieżkę prowadzącą do **s**, ten sam ciąg będzie także stanowił etykietę ścieżki rozszerzonej o **ε**-przejścia.

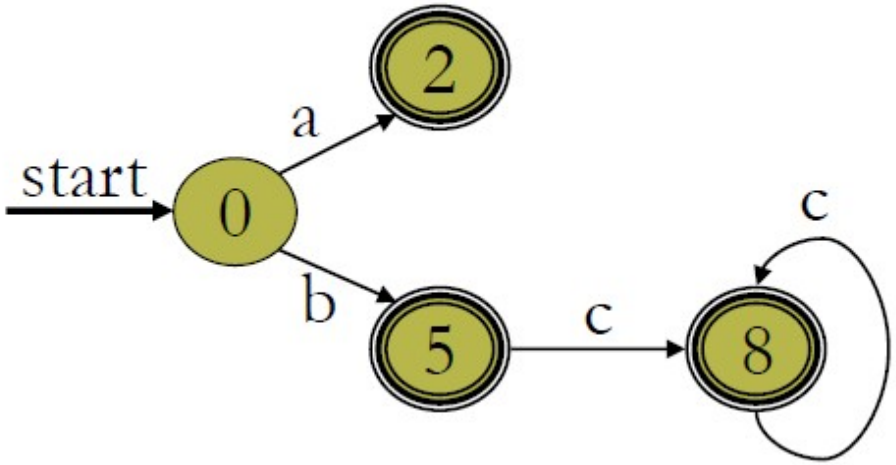


Automat z ε-przejściami dla wyrażenia a | bc*

Eliminacja epsilon-przejść



Automat z ε-przejściami dla wyrażenia a | bc*

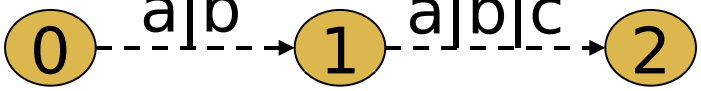


Automat skonstruowany na podstawie eliminacji ε - przejść.

Automat akceptuje wszystkie ciągi języka L (a | bc*).

Od automatów do wyrażeń regularnych.

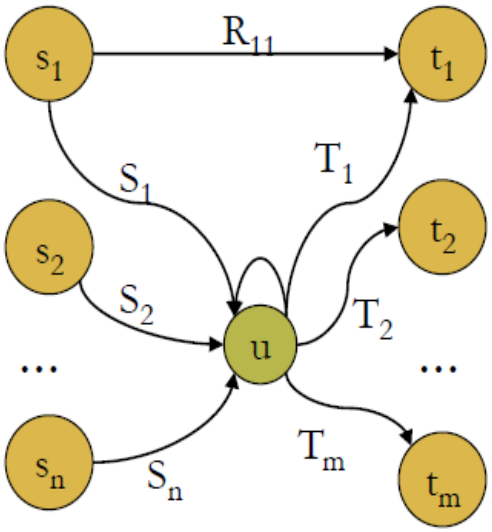
- Dla każdego automatu istnieje **A** wyrażenie regularne, którego język dokładnie odpowiada zbiorowi ciągu znaków akceptowanych przez automat **A**.
- Konstrukcja polega na eliminacji stanów automatów. Etykiety krawędzi, które są zbiorami znaków, zastępuje się bardziej skomplikowanymi wyrażeniami regularnymi.
- Jeżeli dla pewnej krawędzi istnieje etykieta **{x1,x2,...xn}**, zastępuje się ją wyrażeniem regularnym **x1 | x2 | … | xn**, które reprezentuje ten sam zbiór symboli.
- Etykietę ścieżki można postrzegać jako złożenie wyrażeń regularnych opisujących krawędzie tej ścieżki, lub jako język zdefiniowany przez złożenie tych wyrażeń.
- **Przykład:**
  - Wyrażenia regularne etykietujące krawędzie to **a | b** i **a | b | c**. Zbiór znaków etykietujących tę ścieżkę składa się z tych, które występują w języku zdefiniowanym przez wyrażenia regularne: **(a | b)(a | b | c)** czyli **{aa, ab, ac, ba, bb, bc}**.



Ścieżka z wyrażeniami regularnymi jako etykietami. Etykieta ścieżki należy do wyrażeń regularnych utworzonych w wyniku złożeń.

Konstrukcja eliminacji stanów.

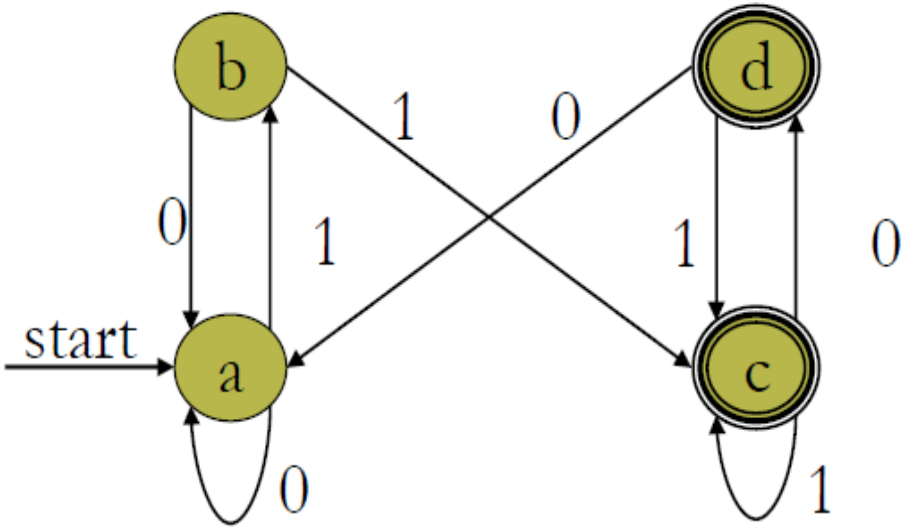
- Kluczowym etapem konwersji z postaci automatu na wyrażenie regularne jest eliminacja stanów. Chcemy wyeliminować stan **u**, ale chcemy zachować etykiety krawędzi występujące w postaci wyrażeń regularnych, tak aby zbiór etykiet ścieżek miedzy dowolnymi pozostałymi stanami nie uległ zmianie.
- Poprzedniki stanu **u** to $\mathbf{s_1, s_2, ..., s_n}$ zaś następniki stanu **u** to $\mathbf{t_1, t_2, ..., t_m}$ (mogą też istnieć stany wspólne).



Zbiór ciągów znaków etykietujących ścieżki wiodące z wierzchołków $\mathbf{s_i}$ do wierzchołka **u**, włącznie z ścieżkami biegnącymi kilkakrotnie wokół pętli $\mathbf{u \rightarrow u}$, oraz z wierzchołka **u** do wierzchołka $\mathbf{t_j}$, jest opisany za pomocą wyrażenia regularnego $\mathbf{S_i\ U^*\ T_j}$.

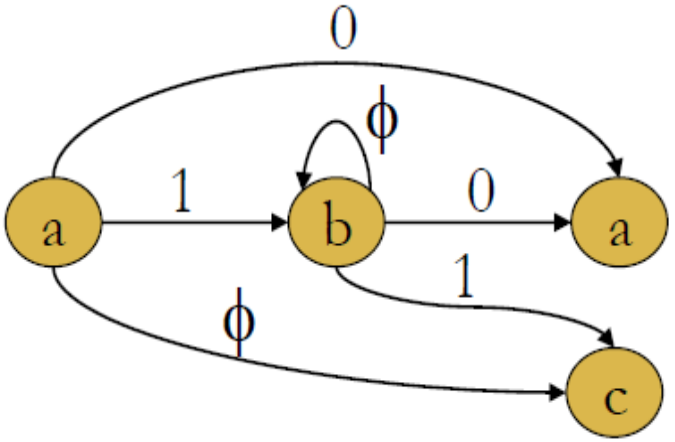
Po eliminacji wierzchołka **u** należy zastąpić etykietę $\mathbf{R_{ij}}$, czyli etykietę krawędzi $\mathbf{s_i \rightarrow t_j}$ przez etykietę $\mathbf{R_{ij} \mid S_i\ U^*\ T_j}$.

Redukcja filtra odbijającego

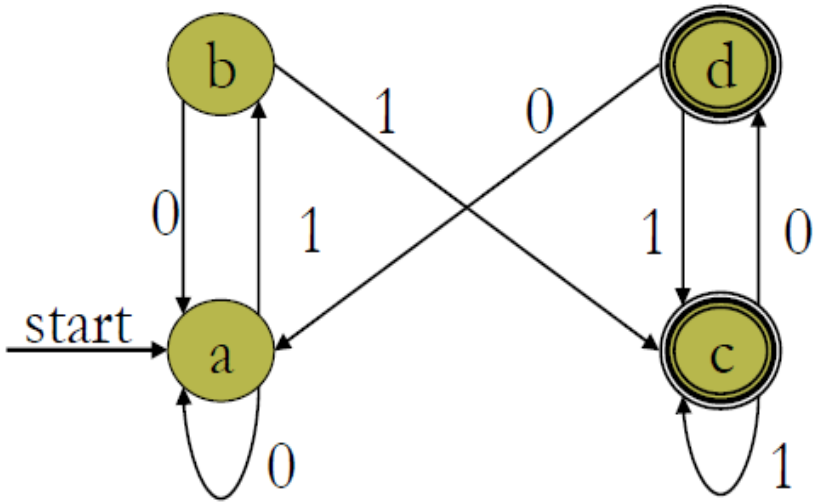


Automat skończony filtra odbijającego.

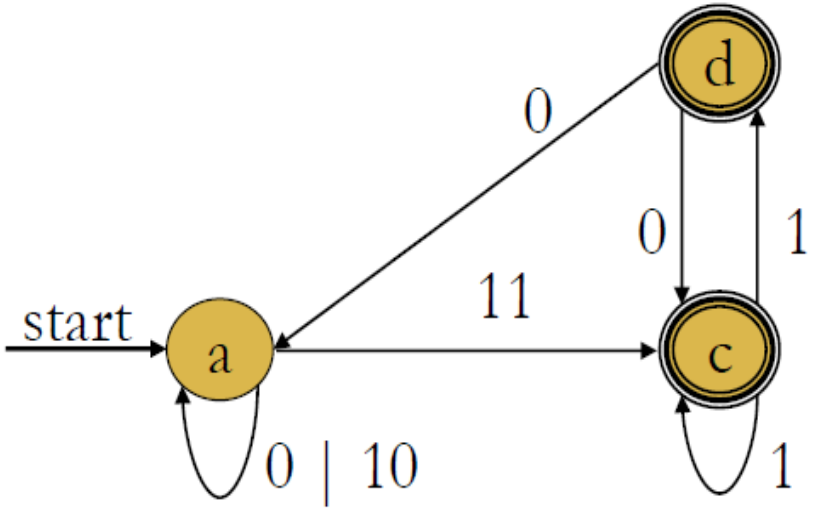
Stan b, jego poprzedniki oraz następniki.



Redukcja filtra odbijającego



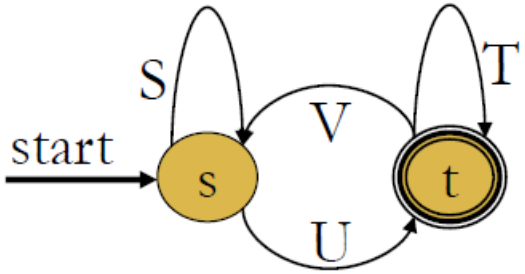
Automat skończony filtra odbijającego.



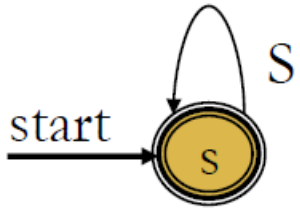
Automat skończony filtra odbijającego po wyeliminowaniu stanu b.

Redukcja zupełna automatu

- W celu otrzymania wyrażenia regularnego określającego wszystkie ciągi znaków akceptowane przez automat **A** i żadne inne, należy rozpatrzyć po kolei każdy stan akceptujący **t** automatu **A**.
- Każdy ciąg znaków akceptowany przez automat **A** jest akceptowany dlatego, że etykietuje on ścieżkę wiodąca ze stanu początkowego **s** do pewnego stanu akceptującego **t**.



Automat zredukowany do dwóch stanów.
Wyrażenie regularne: S*U(T | VS*U)*



Automat posiadający jedynie stan początkowy.
Wyrażenie regularne: S*

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 że dwa wyrażenia połączone znakiem plus stanowią wyrażenie.
(3) **E-E**.
(4) **E*E**.
(5) **E/E**.

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 kategorią syntaktyczną umieszczoną 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.

Produkcja

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 X_i będącego kategorią syntaktyczną wybieramy ciągi znaków ze zbioru $L(<X_i>)$ na wszystkie możliwe sposoby.

Produkcja

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 ; {}

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>.

Drzewa rozbioru

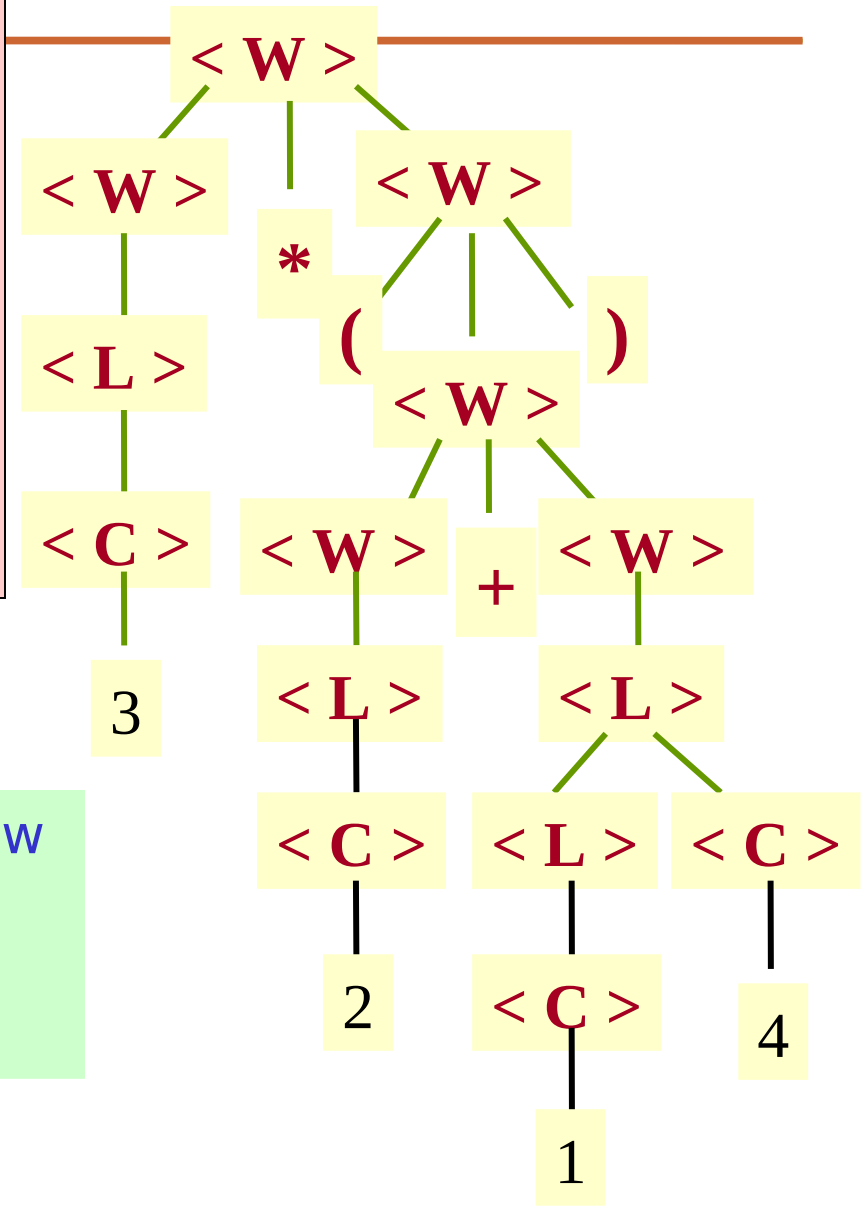
Możemy ilustrować przynależność s do L(<S>) w formie drzewa, zwanego ***drzewem rozbioru*** lub ***drzewem analizy składniowej*** (ang. parse tree).

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 reprezentuje **zastosowanie produkcji**.

Tzn. **kategoria syntaktyczna** etykietująca wierzchołek stanowi cześć nagłówkową produkcji. Etykiety potomków wierzchołka, 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.

Drzewa rozbioru i drzewa wyrażeń

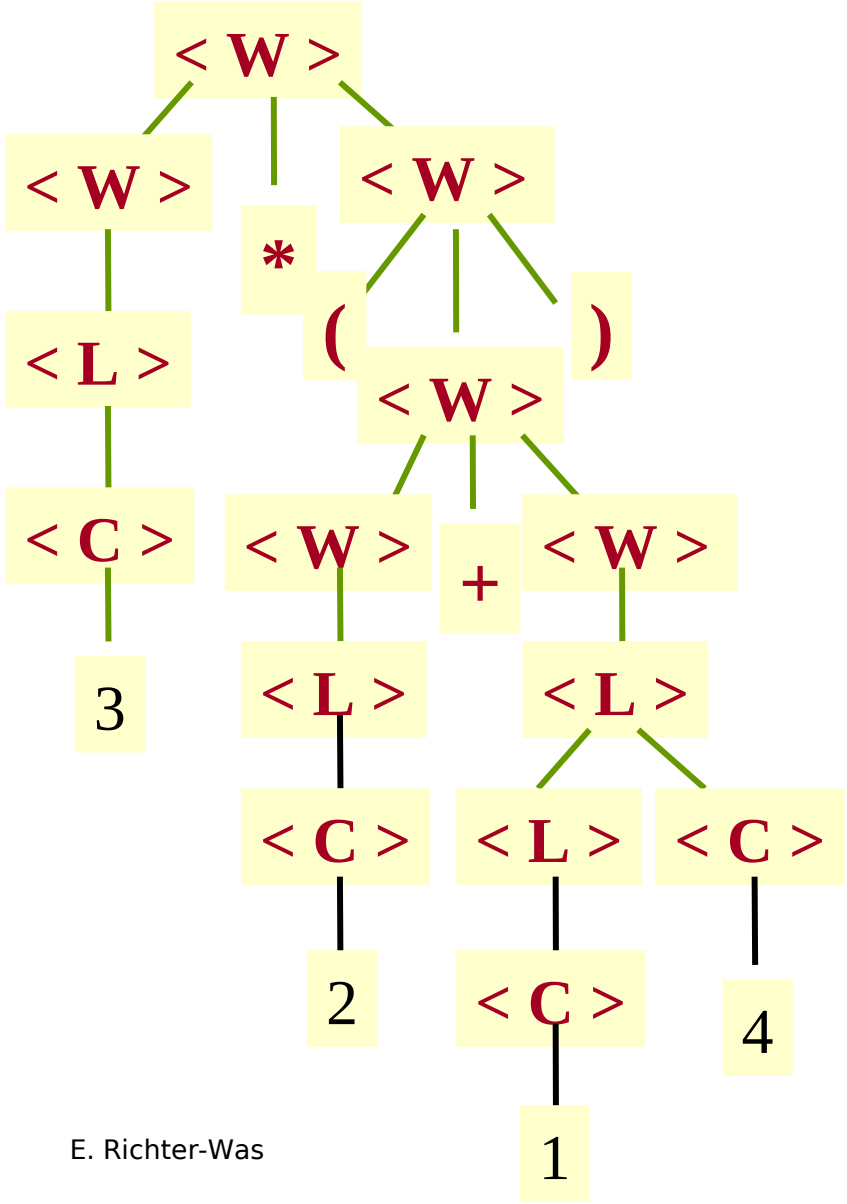
Mając sformułowaną gramatykę wyrażeń możemy drzewa rozbioru przekonwertować na drzewa wyrażeń, dokonując trzech transformacji:

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

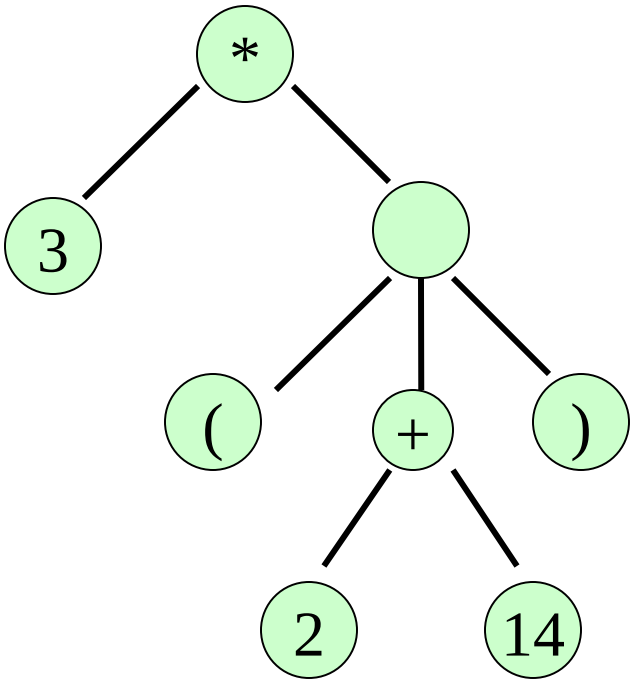
(2) Operatory zostają przesunięte z liści do ich wierzchołków nadrzędnych. To znaczy symbol operatora, taki jak +, staje się etykietą wierzchołka umieszczonego nad nim, który wcześniej był zaetykietowany kategoria syntaktyczna „wyrażenia”.

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

- drzewo rozbioru



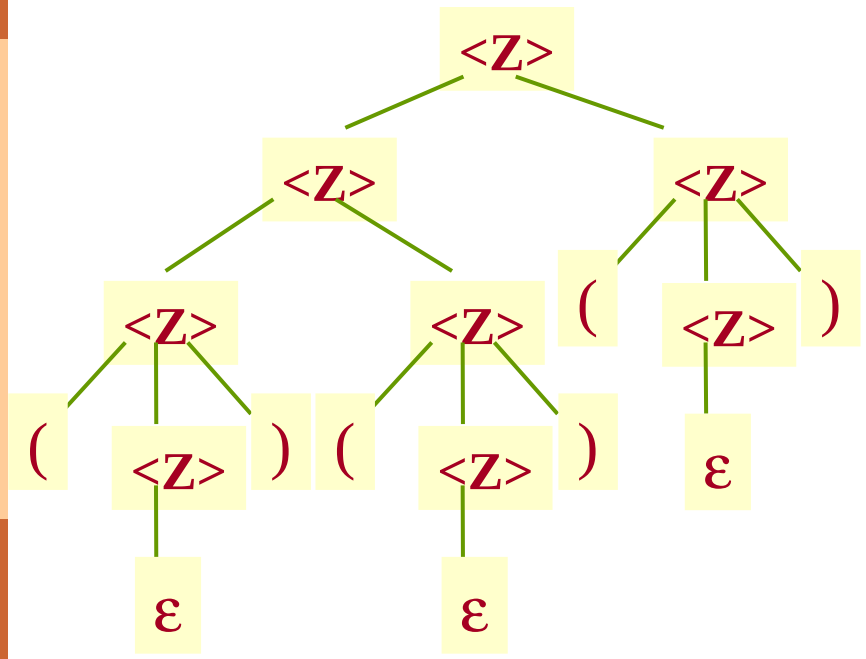
drzewo wyrażeń

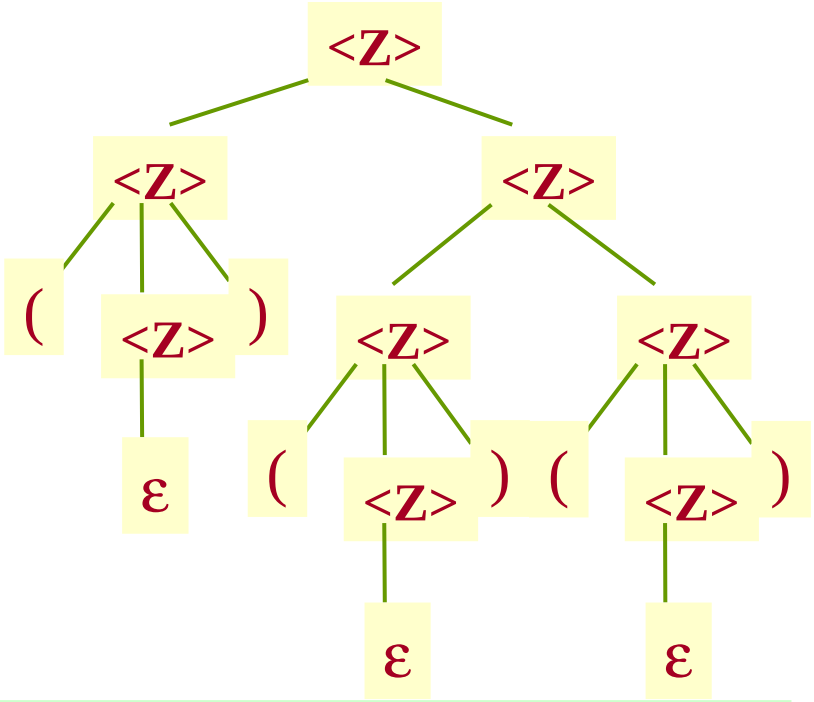


Niejednoznaczność i projektowanie gramatyk

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

<Z> → ε
<Z> → (<Z>)
<Z> → <Z> <Z>

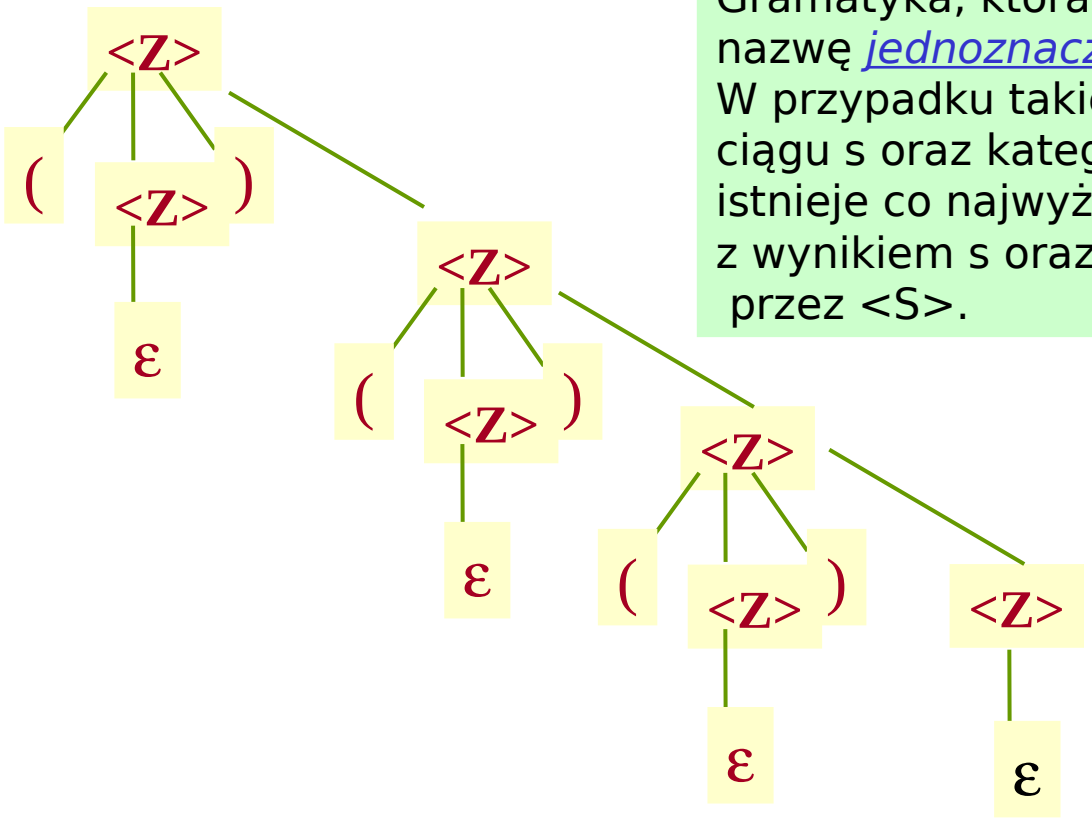




Gramatyka w której istnieją dwa lub więcej drzewa rozbioru o tym samym wyniku oraz tej samej kategorii syntaktycznej etykietującej korzeń jest nazywana *gramatyka niejednoznaczna. (ang. ambigous).* Wystarczy żeby istniał choć jeden taki ciąg który jest niejednoznaczny.

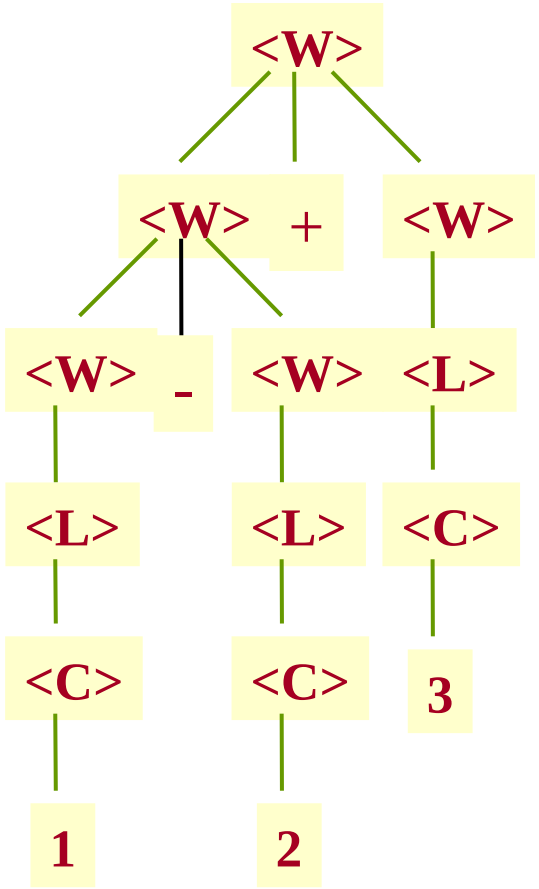
Rozpatrzmy inną gramatykę zbilansowanych nawiasów. Chcemy utworzyć drzewo rozbioru dla ciągu znaków ( ) ( ) ( ). Można utworzyć tylko jedno takie drzewo.

<Z> → ε
<Z> → (<Z>) <Z>

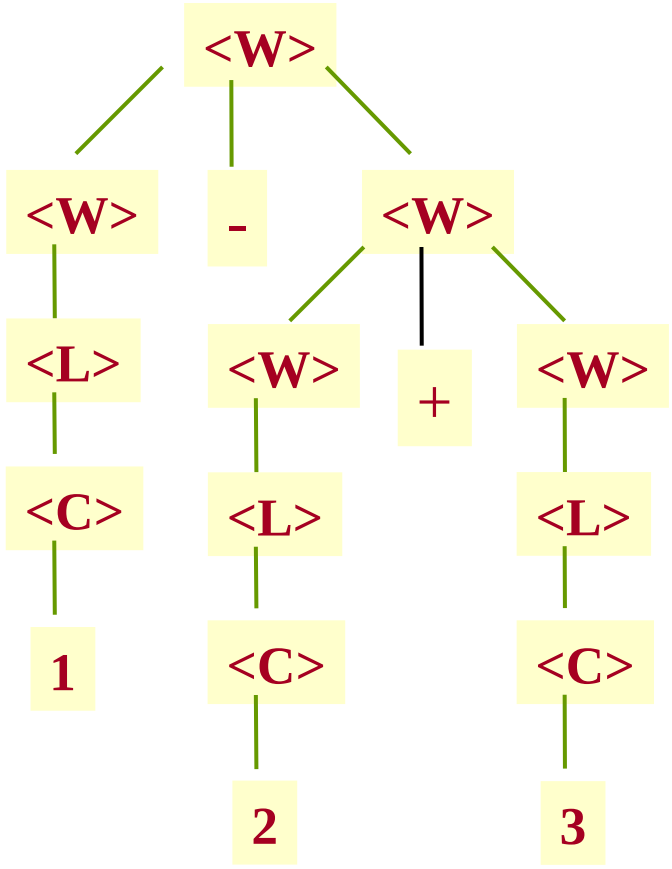


Gramatyka, która nie jest niejednoznaczna nosi nazwę *jednoznacznej (ang. unambigous)* .
W przypadku takiej gramatyki dla każdego ciągu s oraz kategorii syntaktycznej <S> istnieje co najwyżej jedno drzewo rozbioru z wynikiem s oraz korzeniem zaetykietowanym przez <S>.

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

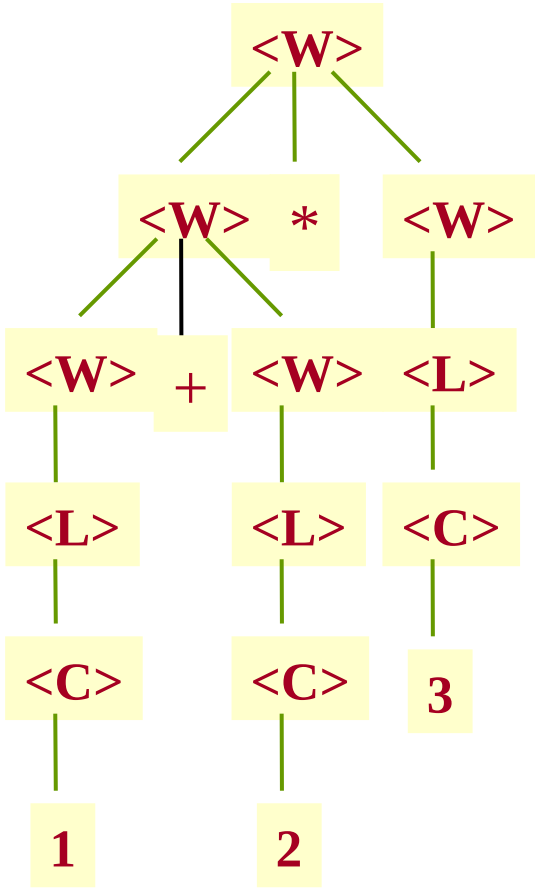


Poprawne drzewo rozbioru
1-2+3 = 2

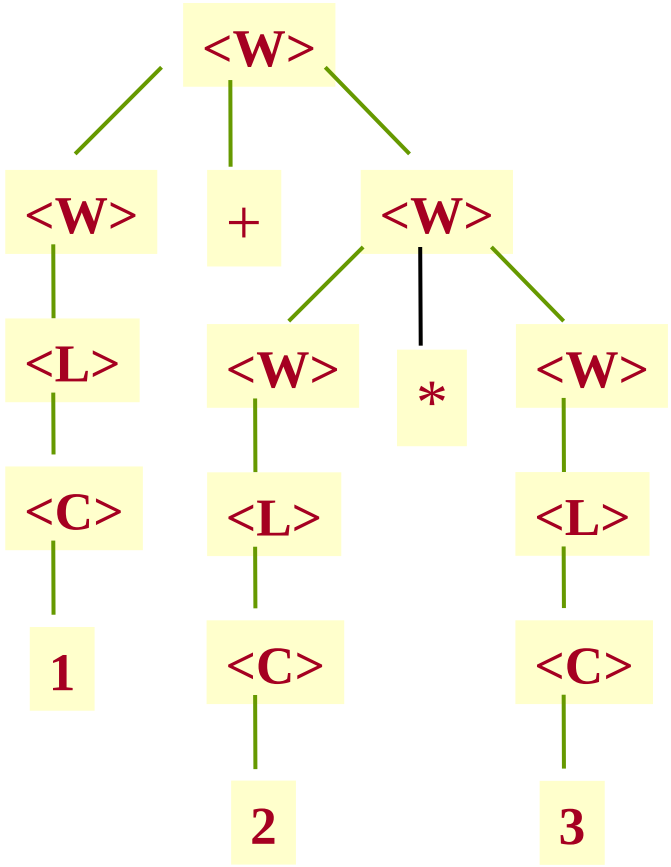


Niepoprawne drzewo rozbioru
1-2+3=-4

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

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.

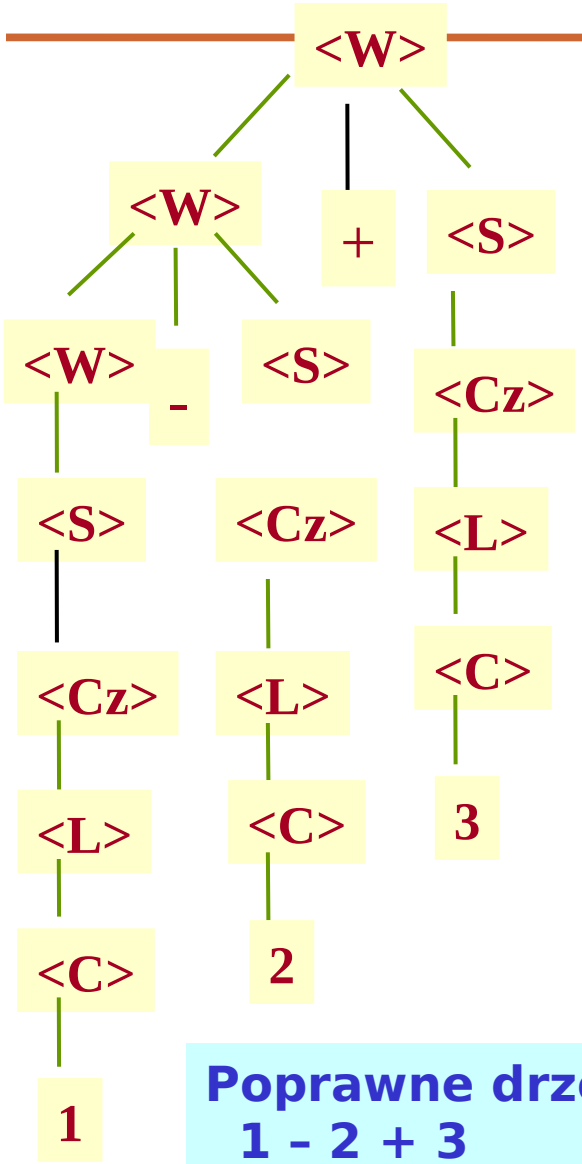
< Składnik> - generuje iloczyn lub iloraz czynników. Pojedynczy czynnik jest składnikiem, a więc stanowi ciąg czynników rozdzielonych operatorami * lub /. Przykładami składników są 12 lub 12/3*45.

<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.

Jednoznaczne gramatyki wyrażeń

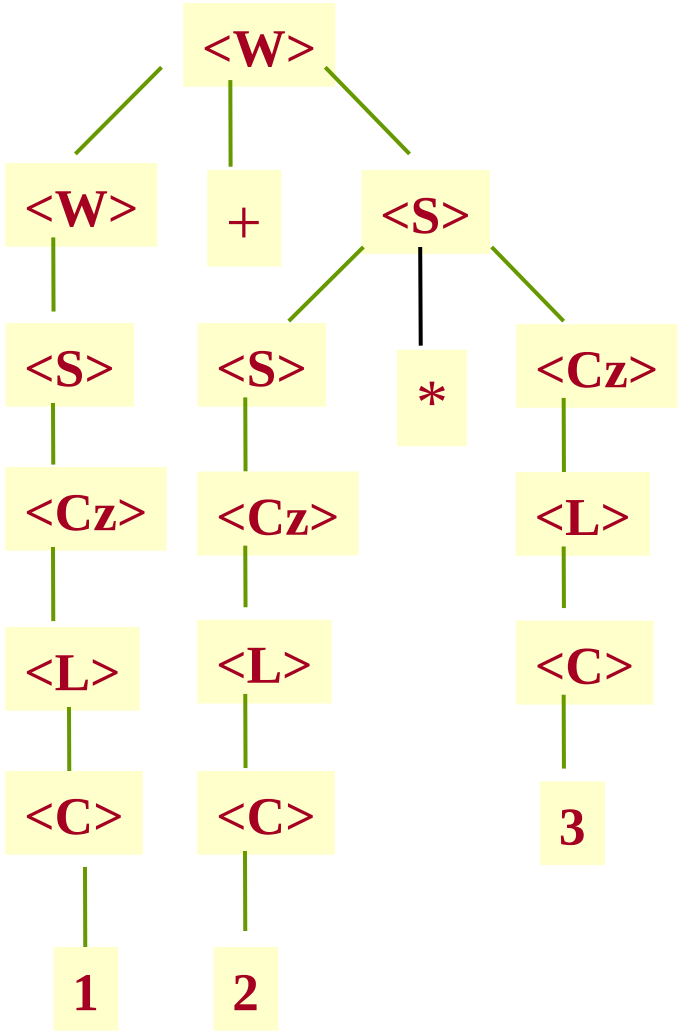
(1) <W> → <W> + <S> | <W> - <S> | <S>
(2) <S> → <S> * <Cz> | <S> / <Cz> | <Cz>
(3) <Cz> → (<W>) | <L>
(4) <L > → <L><C> | <C>
(5) <C> → 0 | 1 | ..... | 9

gramatyka jednoznaczna wyrażeń arytmetycznych



Poprawne drzewo rozbioru
1 – 2 + 3

Poprawne drzewo rozbioru
1 + 2 * 3



Analiza składniowa

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.

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”.

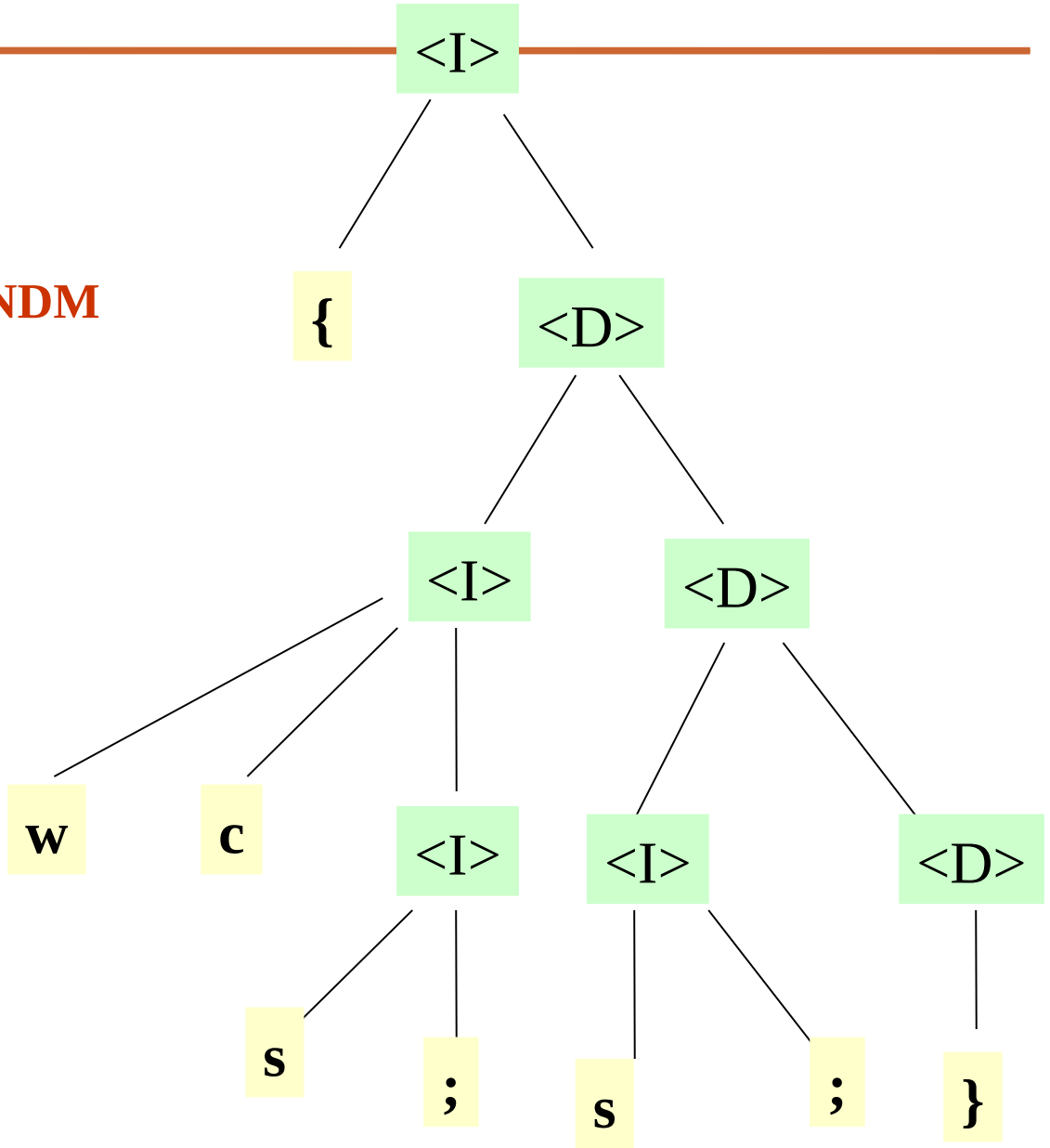
Konstruowanie drzewa rozbioru

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 ***wstepujacej (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.

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



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 {ε}, kiedy R jest ε, oraz L(<S>) jest $\varnothing$, kiedy R jest $\varnothing$.

Indukcja:

Załóżmy, że hipoteza indukcyjna jest spełniona w przypadku wyrażeń regularnycho 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>.

Gramatyka dla wyrażenia regularnego: a | bc*

1. Tworzymy kategorie syntaktyczne dla trzech symboli, które pojawiają się w tym wyrażeniu:
   < A > → a < B > → b < C > → c

2. Tworzymy gramatykę dla c*: **< D > → < C > < D > | ε**
   Wówczas L(<D>) = L(<C>))* = c*

3. Tworzymy gramatykę dla bc*: **< E > → < B > < D >**

4. Tworzymy gramatykę dla całego wyrażenia regularnego a | bc*:
   < F > → < A > | < E >

końcowa postać gramatyki

<F> → <A> | <E>
<E> → <B> <D>
<D> → <C> <D> | ε
<A> → a
<B > → b
<C> → 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:

$< S > \rightarrow 0 < S > 1$
$< S > \rightarrow 0\ 1$

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

Tylko definicje.....

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

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

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

Gramatyka nieograniczona to taka, która dopuszcza produkcje o postaci $\alpha \rightarrow \beta$, 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 pewnego języka.

⇒ 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.