TEORETYCZNE PODSTAWY INFORMATYKI:

POWTÓRKA CZ. II

25/01/2016

WFAiS UJ, Informatyka Stosowana
I rok studiów, I stopień

Wykład 8

Modele danych: grafy

- Podstawowe pojęcia
- Grafy wywołań
- Grafy skierowane i nieskierowane
- Grafy planarne, kolorowanie grafów
- Inplementacja grafów
  - Listy sąsiedztwa
  - Macierze sąsiedztwa
- Składowe spójne, algorytm Kruskala
- Sortowanie topologiczne
- Najkrótsze drogi: algorytm Dikstry i Floyda-Warshalla

Graf

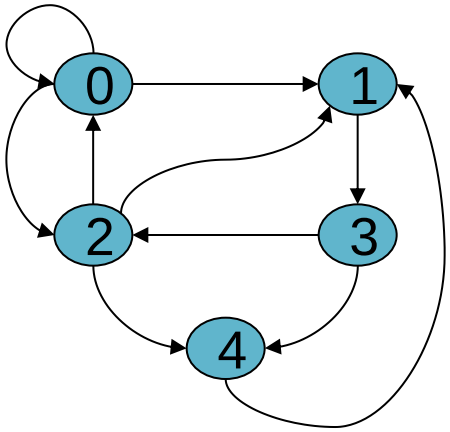
- **Graf to jest relacja binarna**.
- Dla grafów mamy ogromne możliwości wizualizacji jako **zbiór punktów** (zwanych **wierzchołkami**) **połączonych liniami lub strzałkami** (nazwanych **krawędziami**). Pod tym względem graf stanowi **uogólnienie drzewiastego modelu danych**.
- Podobnie jak drzewa, grafy występują w różnych postaciach: grafów **skierowanych** i **nieskierowanych** lub **etykietowanych** i **niezaetykietowanych**.
- Grafy są przydatne do analizy szerokiego zakresu problemów: obliczanie odległości, znajdowanie cykliczności w relacjach, reprezentacji struktury programów, reprezentacji relacji binarnych, reprezentacji automatów i układów elektronicznych.
- **Teoria grafów** jest dziedziną matematyki zajmującą się właściwościami grafów.

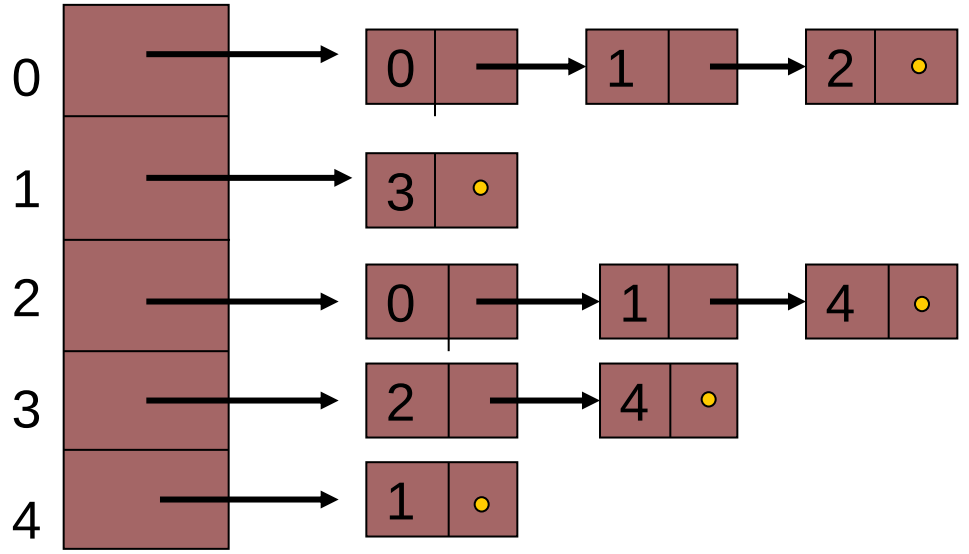
Implementacje grafów

- Istnieją dwie standardowe metody reprezentacji grafów.
  - Pierwsza z nich, **listy sąsiedztwa** (ang. *adjacency lists*), jest, ogólnie rzecz biorąc, podobna do implementacji relacji binarnych.
  - Druga, **macierze sąsiedztwa** (ang. adjacency matrices), to nowy sposób reprezentowania relacji binarnych, który jest bardziej odpowiedni dla relacji, w przypadku którym liczba istniejących par stanowi znaczącą część całkowitej liczby par, jakie mogłyby teoretycznie istnieć w danej dziedzinie.

Listy sąsiedztwa

- Listy sąsiedztwa zostały posortowane wg. kolejności, ale następniki mogą występować w **dowolnej kolejności** na odpowiedniej liście sąsiedztwa.



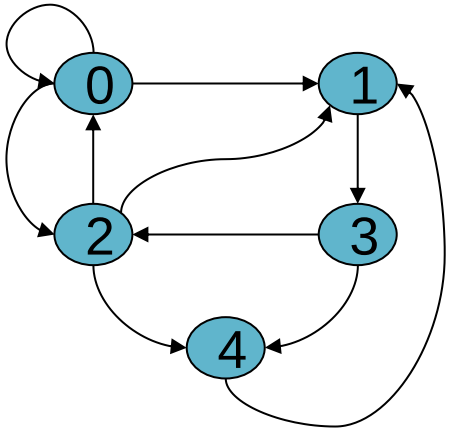


Macierz sąsiedztwa

Tworzymy dwuwymiarową tablicę;

BOOLEAN vertices[MAX][MAX];

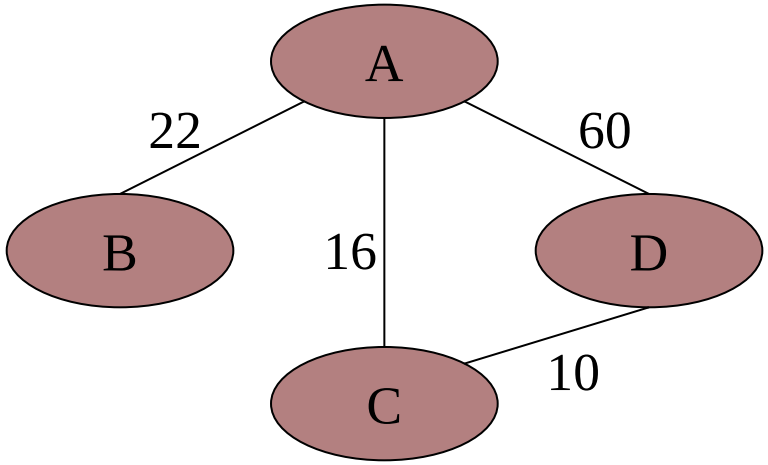
w której element **vertices[u][v]** ma wartość **TRUE** wówczas, gdy istnieje krawędź (u, v), zaś FALSE, w przeciwnym przypadku.



	0	1	2	3	4
0	1	1	1	0	0
1	0	0	0	1	0
2	1	1	0	0	1
3	0	0	1	0	1
4	0	1	0	0	0

Składowa spójna grafu nieskierowanego

- Każdy graf nieskierowany można podzielić na jedna lub większą liczbę **spójnych składowych** (ang. connected components).
- Każda spójna składowa to taki zbiór wierzchołków, że dla każdych dwóch z tych wierzchołków istnieje łącząca je ścieżka. Jeżeli graf składa się z jednej spójnej składowej to mówimy że jest **spójny** (ang. *connected*).

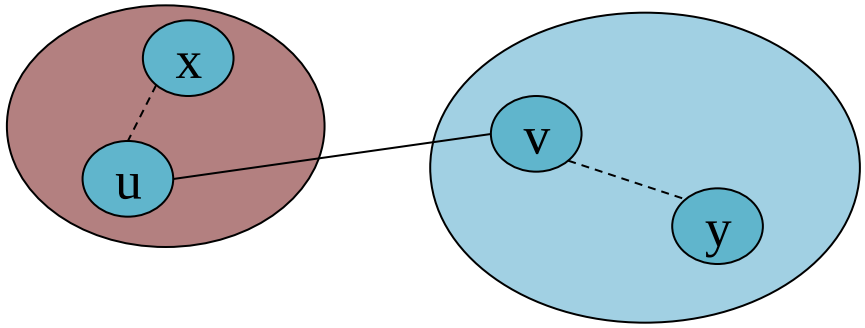


To jest graf spójny

Algorytm wyznaczania spójnych składowych

Przeprowadzamy rozumowanie indukcyjne.

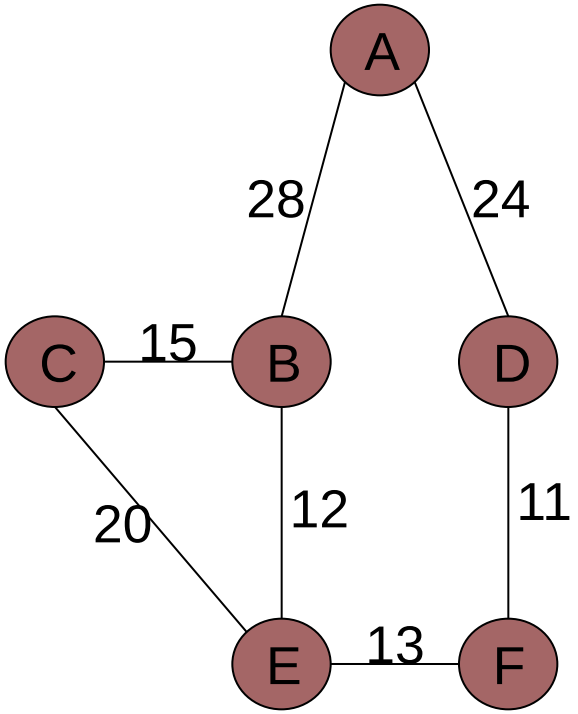
- **Podstawa:**
  - Graf G_0 zawiera jedynie wierzchołki grafu G i żadnej jego krawędzi. Każdy wierzchołek stanowi odrębną spójną składową .
- **Indukcja:**
  - Zakładamy, że znamy już spójne składowe grafu G_i po rozpatrzeniu pierwszych i krawędzi, a obecnie rozpatrujemy (i+1) krawędź {u, v}.
    - jeżeli wierzchołki u, v należą do jednej spójnej składowej to nic się nie zmienia
    - jeżeli do dwóch różnych, to łączymy te dwie spójne składowe w jedną.



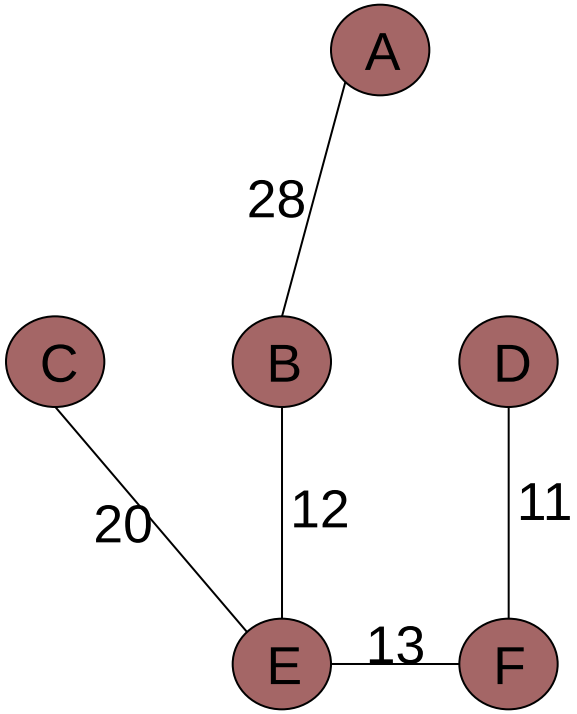
Minimalne drzewa rozpinające

- **Drzewo rozpinające** (ang. *spanning tree*) grafu nieskierowanego G stanowi zbiór wierzchołków tego grafu wraz z podzbiorem jego krawędzi, takich że:
  - łączą one wszystkie wierzchołki, czyli istnieje droga miedzy dwoma dowolnymi wierzchołkami która składa się tylko z krawędzi drzewa rozpinającego.
  - tworzą one drzewo nie posiadające korzenia, nieuporządkowane. Oznacza to że nie istnieją żadne (proste) cykle.
- Jeśli graf **G** stanowi pojedynczą spójną składową to drzewo rozpinające zawsze istnieje. **Minimalne drzewo rozpinające** (ang. *minimal spanning tree*) to drzewo rozpinające, w którym suma etykiet jego krawędzi jest najmniejsza ze wszystkich możliwych do utworzenia drzew rozpinających tego grafu.

Minimalne drzewa rozpinające



Graf nieskierowany

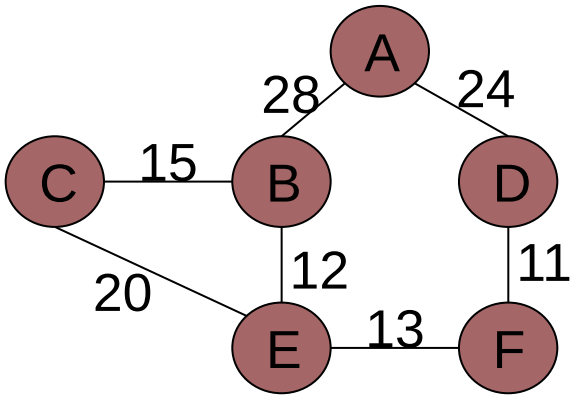


Drzewo rozpinające

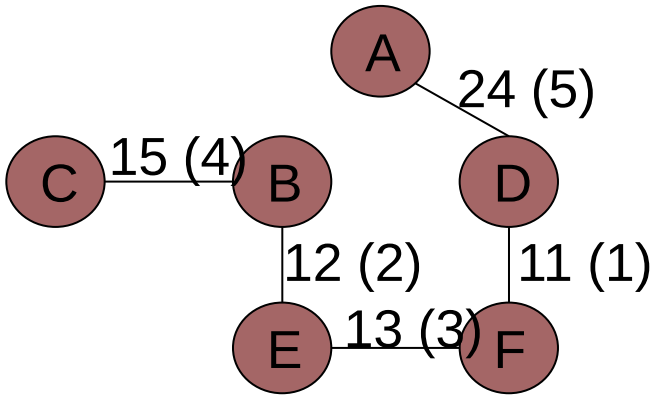
Algorytm Kruskala

- Istnieje wiele algorytmów do znajdowania minimalnego drzewa rozpinającego.
- Jeden z nich to algorytm Kruskala, który stanowi proste rozszerzenie algorytmu znajdowania spójnych składowych. Wymagane zmiany to:
  - należy rozpatrywać krawędzie w kolejności zgodnej z rosnącą wartością ich etykiet,
  - należy dołączyć krawędź do drzewa rozpinającego tylko w takim wypadku gdy jej końce należą do dwóch różnych spójnych składowych.

Algorytm Kruskala



Graf nieskierowany



Minimalne drzewo rozpinające
(w nawiasach podano kolejność dodawanych krawędzi)

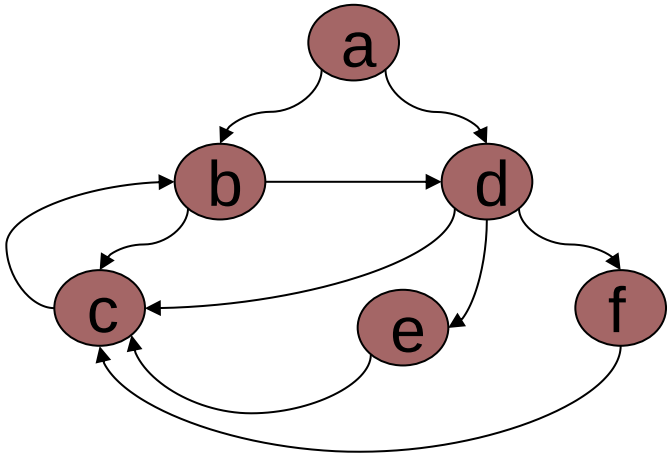
Algorytm Kruskala

- Algorytm Kruskala jest dobrym przykładem **algorytmu zachłannego** (ang. greedy algorithm), w przypadku którego podejmowany jest szereg decyzji, z których każdą stanowi wybranie opcji najlepszej w danym momencie.
  - Lokalnie podejmowane decyzje polegają w tym przypadku na wyborze krawędzi dodawanej do formowanego drzewa rozpinającego.
  - Za każdym razem wybierana jest krawędź o najmniejsze wartości etykiety, która nie narusza definicji drzewa rozpinającego, zabraniającej utworzenia cyklu.
- Dla algorytmu Kruskala można wykazać, ze jego rezultat jest optymalny globalnie, to znaczy że daje on w wyniku drzewo rozpinające o minimalnej wadze.
- Czas wykonania algorytmu jest **O(m log m)** gdzie **m** to jest większa z wartości liczby wierzchołków i liczby krawędzi.

Algorytm przeszukiwania w głąb

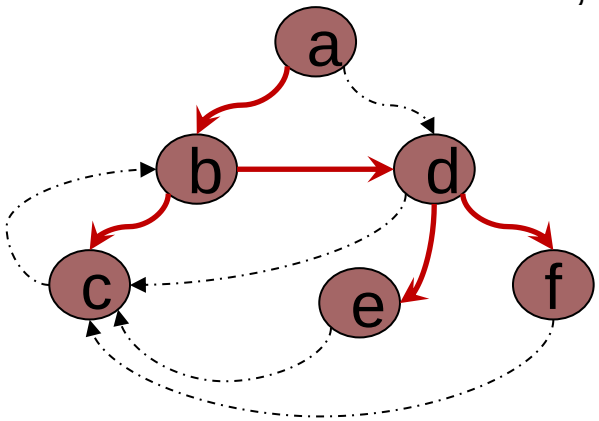
- Jest to podstawowa metoda badania grafów skierowanych.
- Bardzo podobna do stosowanych dla drzew, w których startuje się od korzenia i rekurencyjnie bada wierzchołki potomne każdego odwiedzonego wierzchołka.
- Trudność polega na tym ze w grafie mogą pojawiać się cykle… Należy wobec tego znaczyć wierzchołki już odwiedzone i nie wracać więcej do takich wierzchołków.
  - Z uwagi na fakt, że w celu uniknięcia dwukrotnego odwiedzenia tego samego wierzchołka jest on odpowiednio oznaczany, graf w trakcie jego badania zachowuje się podobnie do drzewa.
- W rzeczywistości można narysować drzewo, którego krawędzie rodzic-potomek będą niektórymi krawędziami przeszukiwanego grafu G.
  - Takie drzewo nosi nazwę **drzewa przeszukiwania w głąb** (ang. *depth-first-search*) dla danego grafu.

Algorytm przeszukiwania w głąb

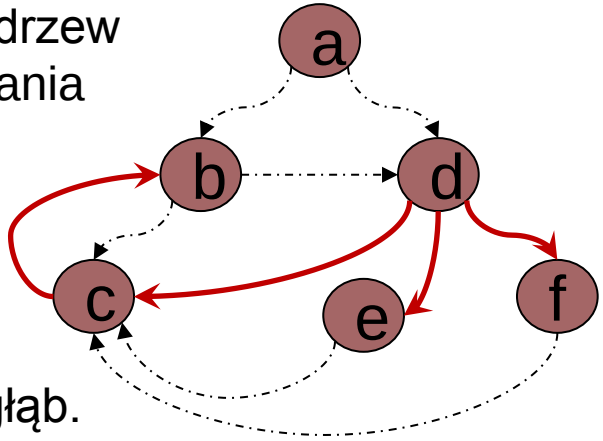


Graf skierowany

Las przeszukiwania:
dwa drzewa o korzeniach a, d

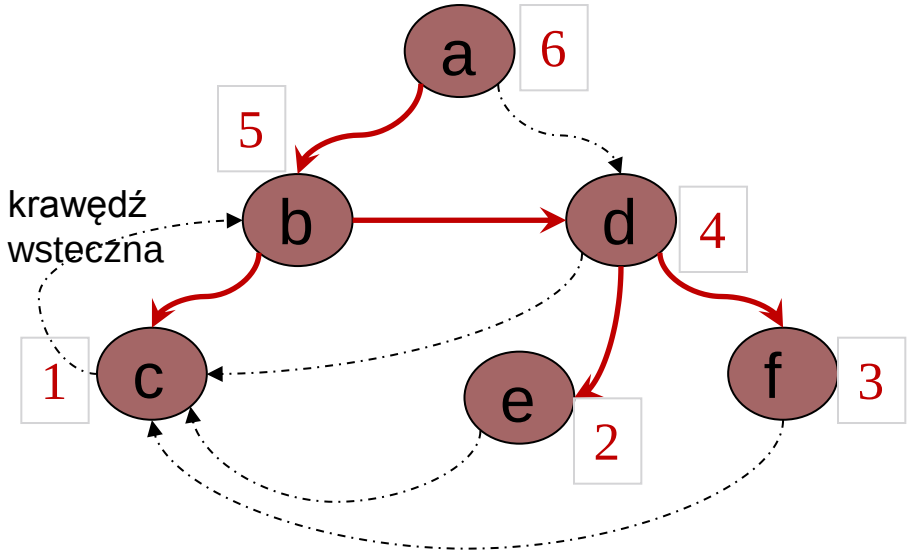


Jedno z
możliwych drzew
przeszukiwania



Las przeszukiwania w głąb.

Drzewo przeszukiwania w głąb



- Po (podczas) konstruowaniu drzewa przeszukiwania w głąb można ponumerować jego wierzchołki w **kolejności wstecznej** (ang. *post-order*).

Znajdowanie spójnych składowych

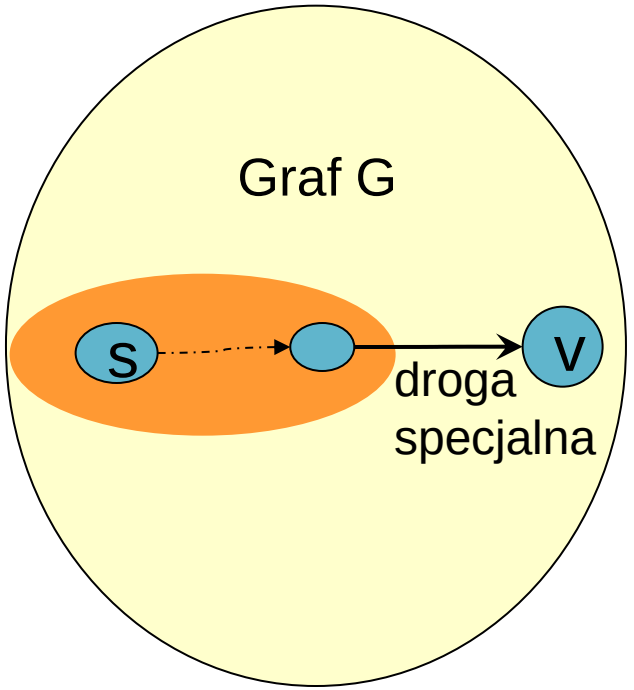
- Do znajdowania spójnych składowych możemy użyć algorytmu poszukiwania w głąb.
- Traktujemy graf nieskierowany jako graf skierowany, w którym każda krawędź nieskierowana została zastąpiona dwiema krawędziami skierowanymi wiodącymi w obu kierunkach.
- Do reprezentacji grafu używamy list sąsiedztwa.
- Tworzymy las przeszukiwania w głąb grafu skierowanego. Każde drzewo w tym lesie odpowiada jednej składowej spójności grafu nieskierowanego.
- Czas wykonania algorytmu **O(m)**
  - przy użyciu struktury drzewiastej czas wykonania wynosi **O(m log n)**.

Algorytm Dikstry

- Szukamy najkrótszej drogi pomiędzy dwoma wierzchołkami
  - Rozpatrujemy graf **G** (skierowany lub nieskierowany), w którym wszystkie krawędzie zaetykietowano wartościami reprezentującymi ich długości.
  - **Długość** (ang. distance) danej drogi stanowi wartość sumy etykiet związanych z nią krawędzi. Minimalna odległość z wierzchołka **u** do wierzchołka **v** to minimalna długość którejś z dróg od **u** do **v**.

Algorytm Dikstry

- Traktujemy wierzchołek **s** jako **wierzchołek źródłowy**. Na etapie pośrednim wykonywania algorytmu w grafie G istnieją tzw. **wierzchołki ustalone** (ang. settled), tzn. takie dla których znane są odległości minimalne. W szczególności zbiór takich wierzchołków zawiera również wierzchołek s.
- Dla **nieustalonego wierzchołka v** należy zapamiętać długość najkrótszej **drogi specjalnej** (ang. *special path*) czyli takiej która rozpoczyna się w wierzchołku źródłowym, wiedzie przez ustalone wierzchołki, i na ostatnim etapie przechodzi z obszaru ustalonego do wierzchołka v.

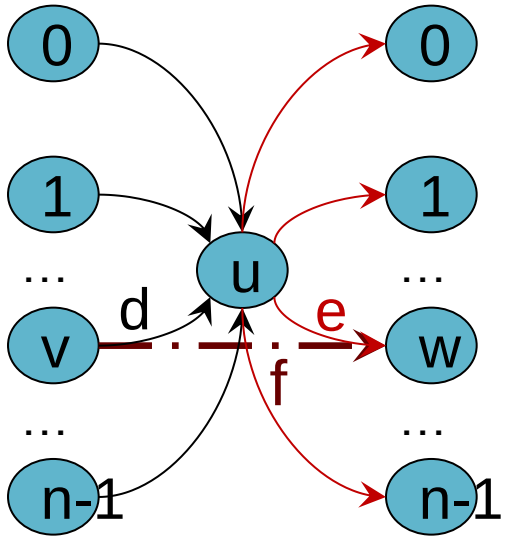


Algorytmy znajdowania najkrótszych dróg

- Jeśli potrzebne jest **poznanie minimalnych odległości miedzy wszystkimi parami wierzchołków w grafie** o n wierzchołkach, które posiadają etykiety o wartościach nieujemnych, można uruchomić algorytm Dijkstry dla każdego z n wierzchołków jako wierzchołka źródłowego.
- Czas wykonania **algorytmu Dijkstry wynosi O(m log n)**, gdzie m oznacza większą wartość z liczby wierzchołków i liczby krawędzi. Znalezienie w ten sposób minimalnych odległości miedzy wszystkimi parami wierzchołków zajmuje czas rzędu **O(m n log n)**.
- Jeśli m jest bliskie swojej maksymalnej wartości $m \approx n^2$ to można skorzystać z implementacji algorytmu Dijkstry który działa w czasie **$O(n^2)$**. Wykonanie go n razy daje czas rzędu **$O(n^3)$**.

Algorytm Floyda-Warshalla

- Podstawa algorytmu jest działanie polegające na rozpatrywaniu po kolei **każdego wierzchołka** grafu jako **elementu centralnego** (ang. *pivot*).
- Kiedy wierzchołek **u** jest elementem centralnym, staramy się wykorzystać fakt, że **u** jest wierzchołkiem pośrednim między wszystkimi parami wierzchołków.
- Dla każdej pary wierzchołków, na przykład **v** i **w**, jeśli suma etykiet krawędzi **(v, u)** oraz **(u, w)** (na rysunku d+e) , jest mniejsza od bieżąco rozpatrywanej etykiety f krawędzi wiodącej od **v** do **w**, to wartość **f** jest zastępowana wartością **d+e**.



Podsumowanie

PROBLEM	ALGORYTM(Y)	CZAS WYKONANIA
Minimalne drzewo rozpinające	Algorytm Kruskala	O(m log n)
Znajdowanie cykli	Przeszukiwanie w głąb	O(m)
Uporządkowanie topologiczne	Przeszukiwanie w głąb	O(m)
Osiągalność w przypadku pojedynczego źródła	Przeszukiwanie w głąb	O(m)
Spójne składowe	Przeszukiwanie w głąb	O(m)
Najkrótsza droga dla pojedyncz. źródła	Algorytm Dijskry	O(m log n)
Najkrótsza droga dla wszystkich par	Algorytm Dijskry	O(m n log n)
	Algorytm Floyda	$O(n^3)$

Wykład 10, part I

Wzorce, automaty

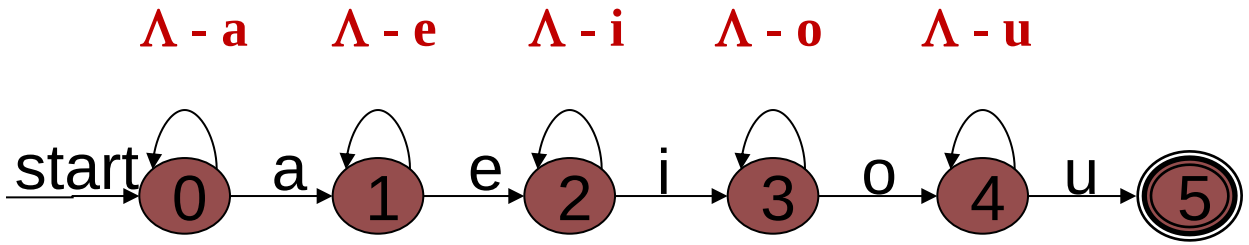
- Definicja
- Grafy reprezentujące maszyny stanów
- Symulacje automatów
- Automaty deterministyczne
- Automaty niedeterministyczne
  - Przejście do automatu deterministycznego
- Minimalizacja automatów

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.

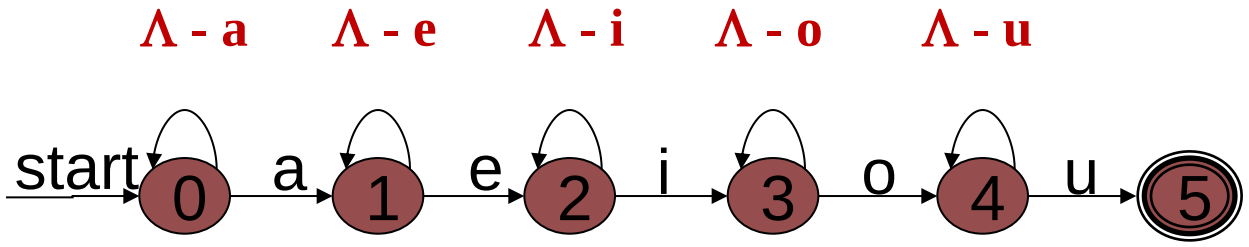
Automat rozpoznający ciągi liter

- 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.



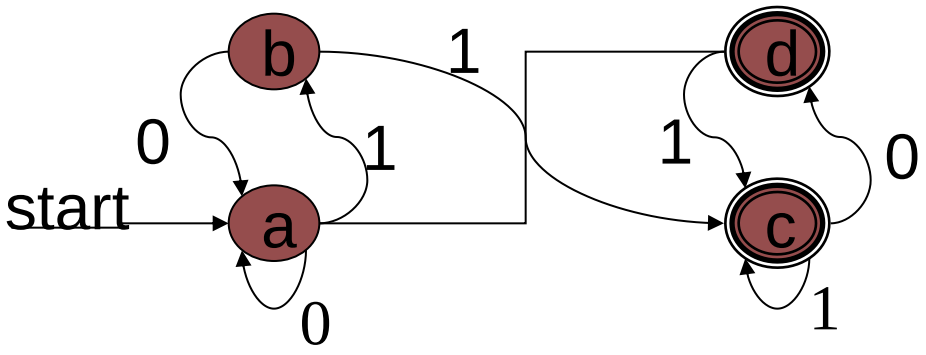
Automat rozpoznający ciągi liter

- 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

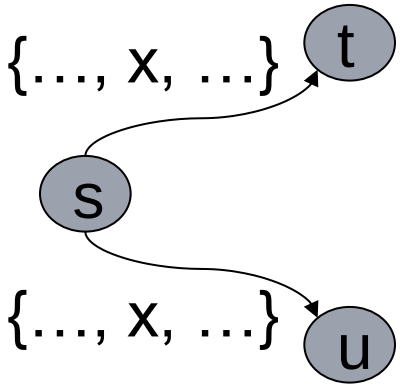
Automaty deterministyczne

- Można z łatwością przerobić automat deterministyczny na program.
- Dla każdego stanu tworzy się fragment kodu. Kod **stanu s** bada znak wejściowy i określa, które (o ile jakiekolwiek) przejście ze **stanu s** powinno zostać wybrane.
- Jeżeli zostanie wybrane przejście ze **stanu s** do **stanu t**, to kod napisany dla stanu s musi uwzględniać przekazanie sterowania do kodu stanu t, np. przy użyciu instrukcji go to.

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.**

Automaty niedeterministyczne



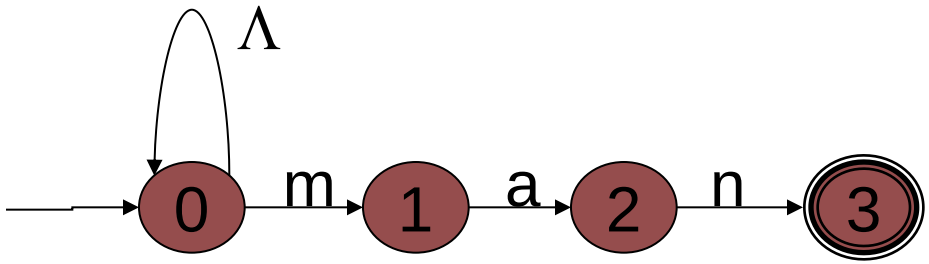
W momencie podjęcia próby symulacji automatu niedeterministycznego w przypadku ciągu wejściowego składającego się ze znaków a_1,a_2, …, a_k może okazać się, że ten sam ciąg etykietuje wiele ścieżek. Wygodnie jest przyjąć, że automat niedeterministyczny akceptuje taki ciąg wejściowy, jeżeli co najmniej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

niedeterminizm = „przypadkowość”

Jak zastąpić automat niedeterministyczny deterministycznym?

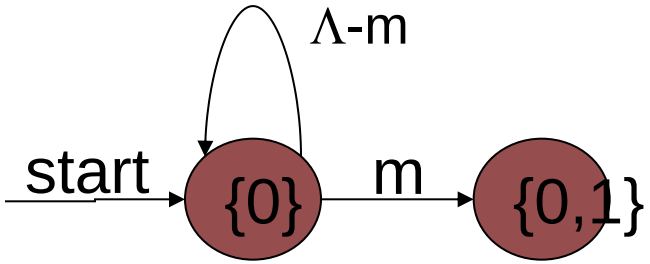
- Automat niedeterministyczny zastępujemy deterministycznym **przez skonstruowanie równoważnego** automatu deterministycznego. Technika ta nosi nazwę konstrukcji podzbiorów (ang. *subset construction*).

Konstrukcja automatu deterministycznego D



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

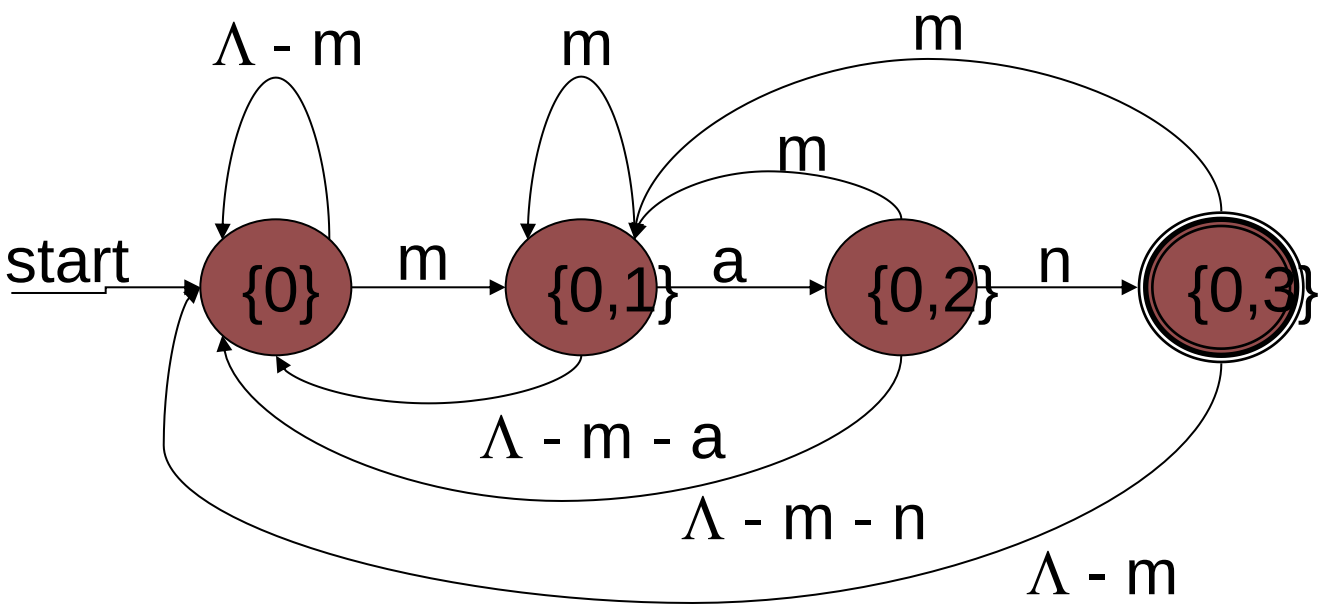
Rozpoczynamy od zbioru **{0}**, który jest stanem początkowym automatu D.



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.
- **Nierównoważność dwóch stanów:**
  - Podstawa: Jeżeli stan s jest stanem akceptującym, a stan t nie jest stanem akceptującym (lub odwrotnie), to s i t nie są równoważne.
  - Indukcja: Jeżeli istnieje pewien symbol wejściowy x, taki, że ze stanów s i t następują przejścia względem tego symbolu do dwóch różnych stanów, o których wiadomo że nie są równoważne, to stany s i t nie są równoważne.

Wykład 10, part II

Wyrażenia regularne

- Definicja
- Operatory
- Prawa algebraiczne
- Od wyrażeń do automatów z ε-przejściami
  - Eliminacja ε-przejść
- Od automatów do wyrażeń
  - Eliminacja stanów
- Redukcja zupełna automatów

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.

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”.

Kolejność operatorów wyrażeń regularnych

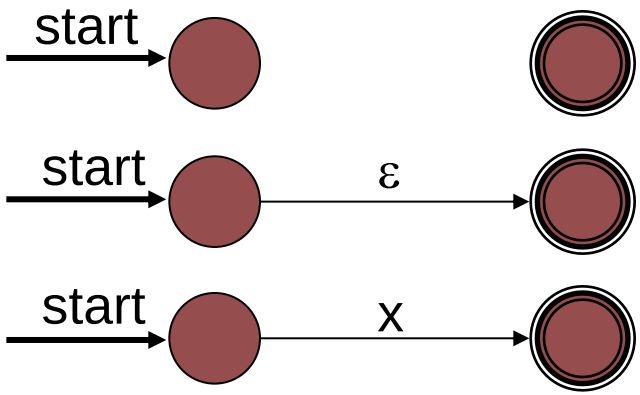
- Istnieje określona kolejność wykonywania **trzech działań wyrażeń regularnych**: sumy, złożenia oraz domknięcia. Kolejność ta jest następująca:
  1. Domknięcie (najwyższy priorytet)
  2. Złożenie
  3. Suma (najniższy priorytet)
- **Przykład:**
  a | bc*d = (a | (b (c*)) d)

Od wyrażeń regularnych do automatów z epsilon przejściami

- **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).



Automat dla ∅

Automat dla ε

Automat dla x

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

Indukcja

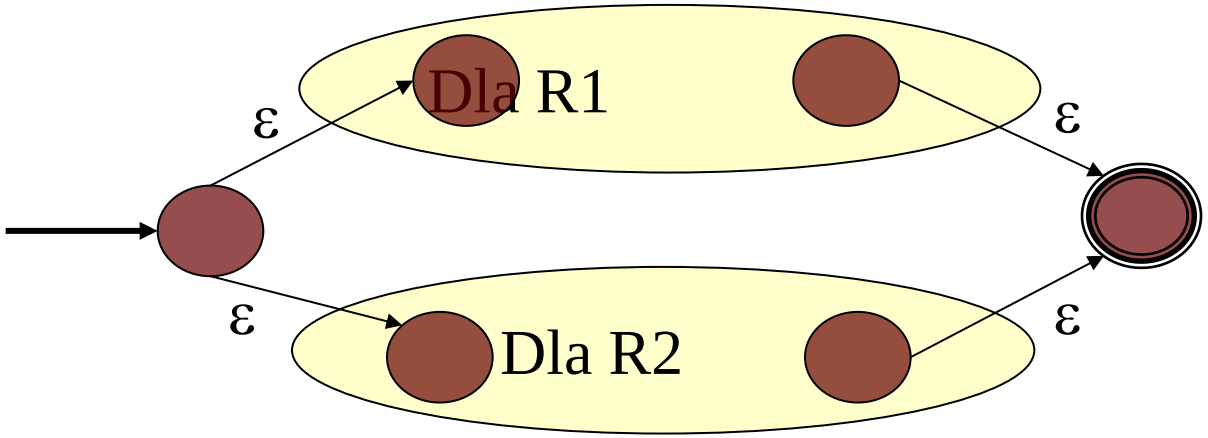
- Zakładamy teraz, ze S(i) jest prawdziwe dla wszystkich i ≤ n.
- To znaczy, że dla każdego wyrażenia regularnego R o maksymalnie n wystąpieniach istnieje automat spełniający warunek hipotezy indukcyjnej i akceptujący wszystkie ciągi znaków języka L(R) i żadnych innych.
- Zajmiemy się tylko najbardziej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

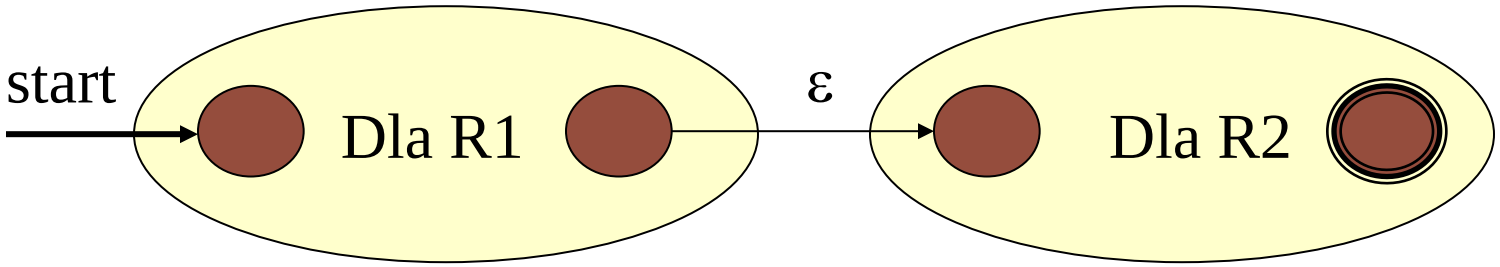
Przypadek 1: R = R1 | R2

- 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.



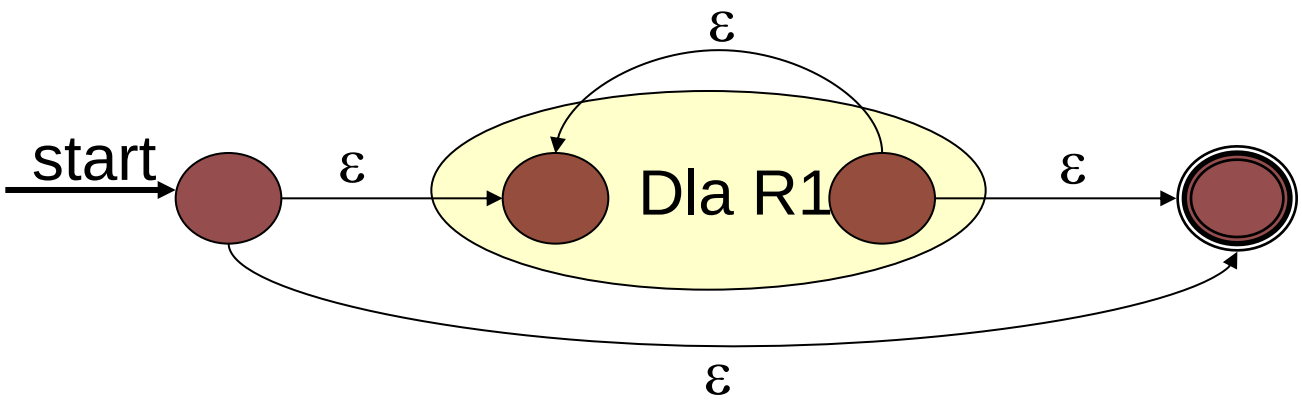
Przypadek 2: R = R1 R2

- 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.



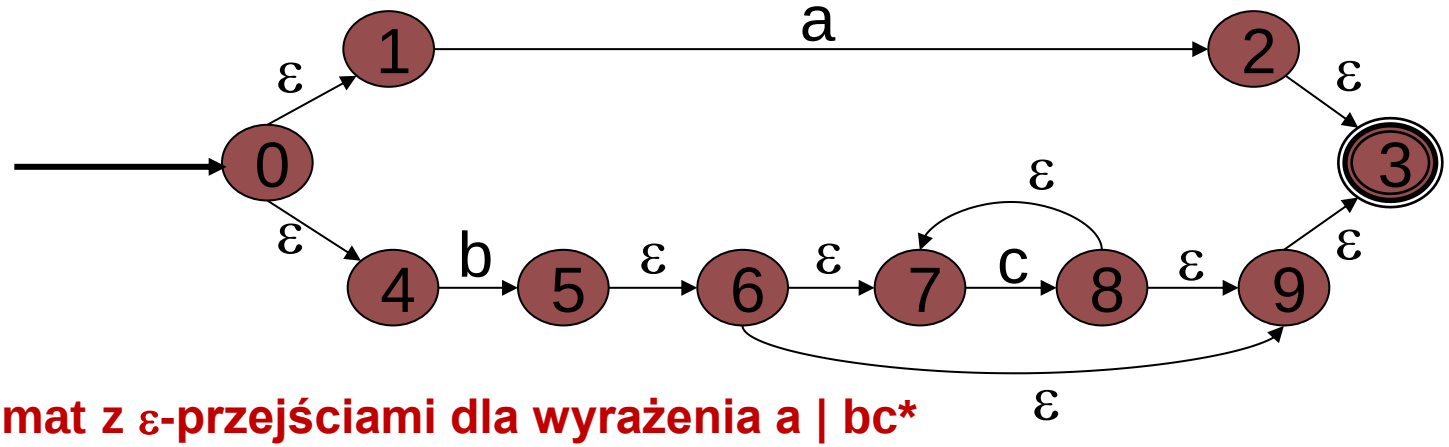
Przypadek 3: R = R1*

- 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).



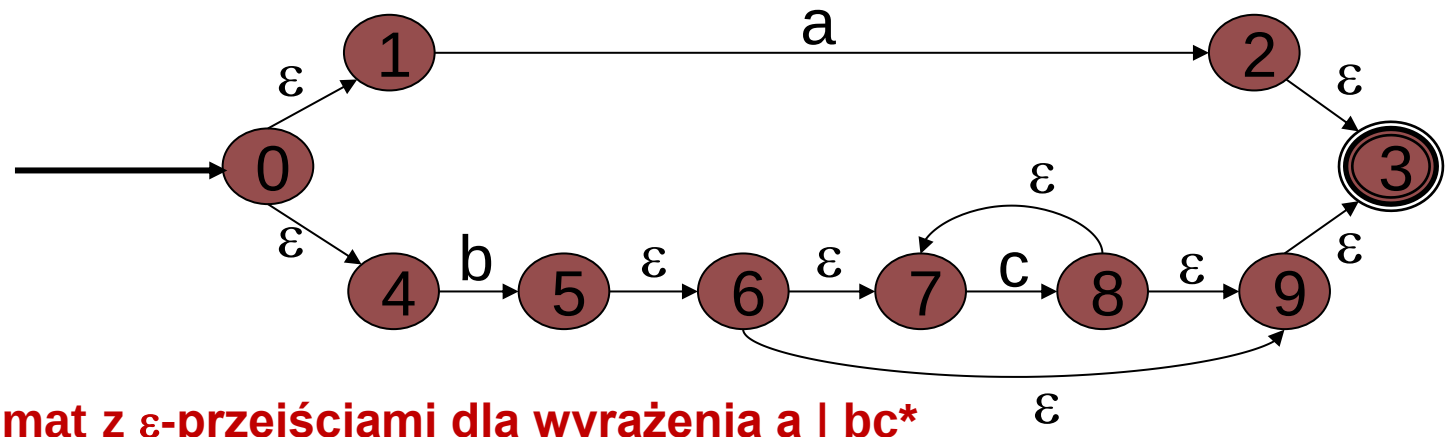
Eliminacja epsilon-przejść

- 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.

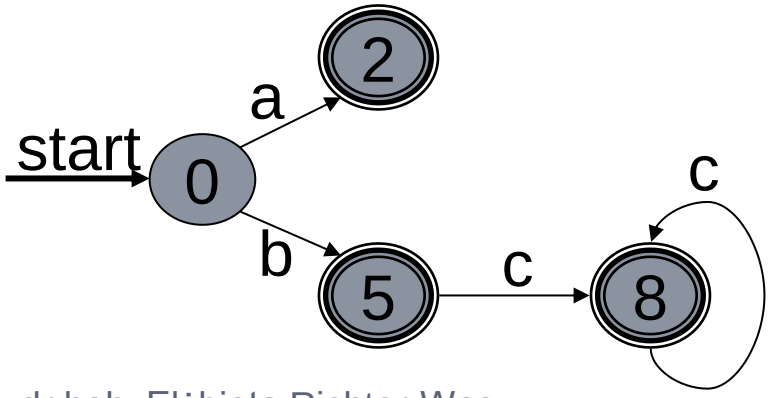


Automat z ε-przejściami dla wyrażenia a | bc*

Eliminacja epsilon-przejść



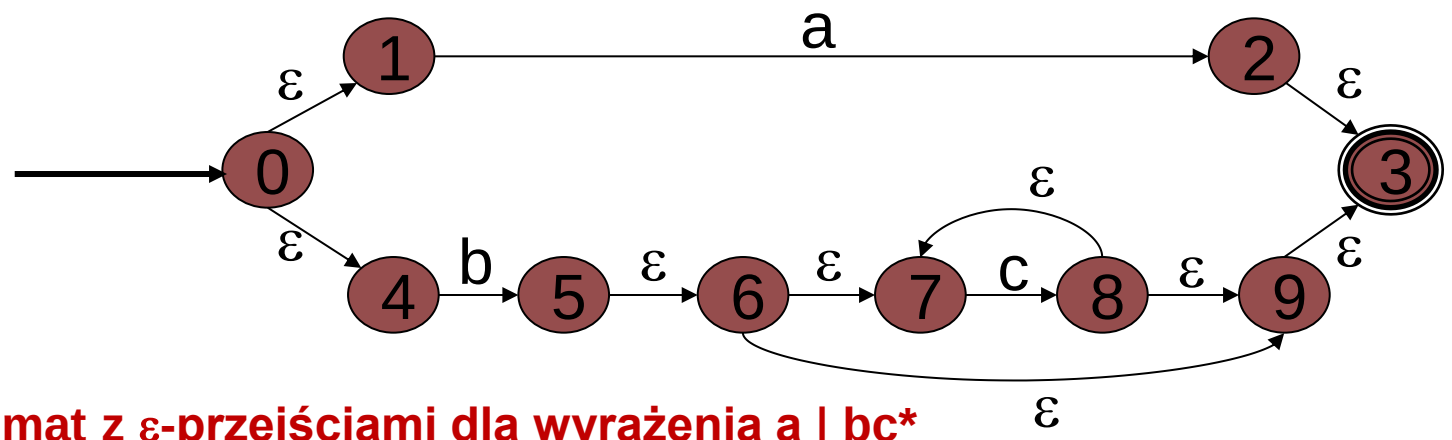
Automat z ε-przejściami dla wyrażenia a | bc*



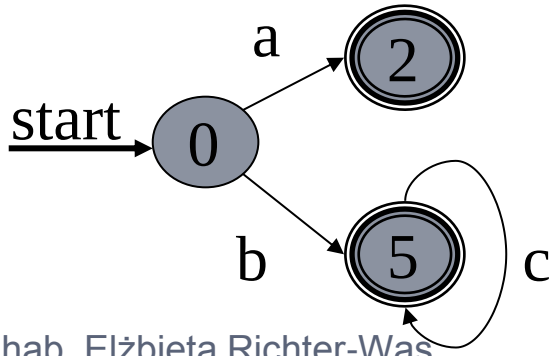
Automat skonstruowany na podstawie eliminacji ε - przejść.

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

Eliminajca epsilon-przejść

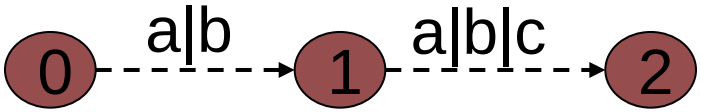


Automat z ε-przejściami dla wyrażenia a | bc*



- **Prostsza wersja automatu dla języka L (a | bc*).**
- **Wykorzystano obserwacje ze stany (5) i (8) są równoważne i mogą zostać połączone.**

Od automatów do wyrażeń regularnych



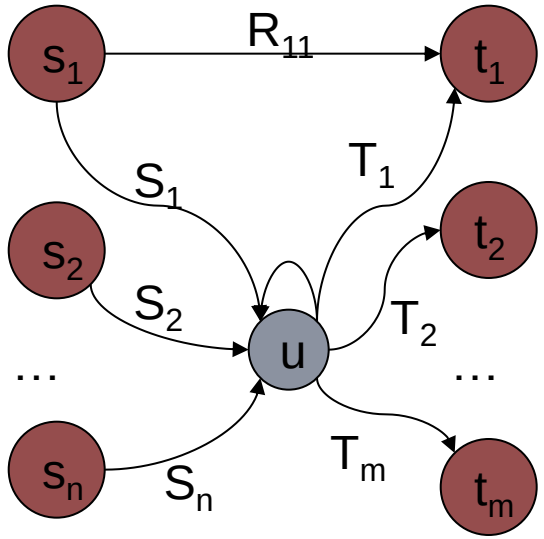
Ścieżka z wyrażeniami regularnymi jako etykietami. Etykieta ścieżki należy do wyrażeń regularnych utworzonych w wyniku złożeń.

Konstrukcja eliminacji stanów

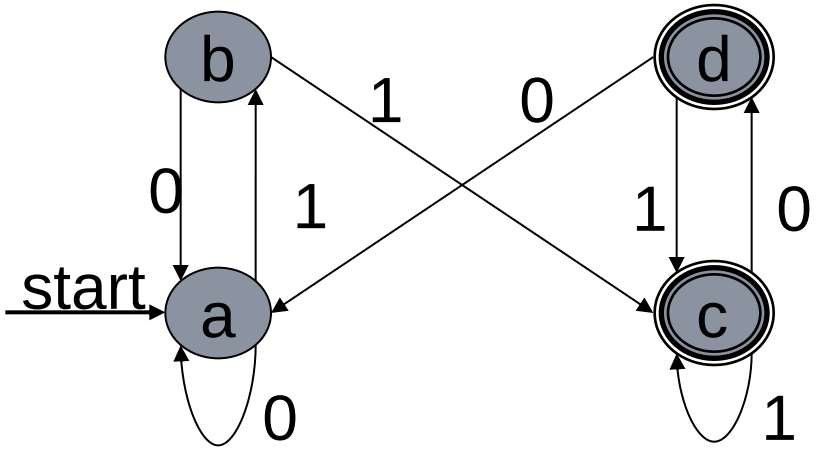
- Kluczowym etapem konwersji z postaci automatu na wyrażenie regularne jest **eliminacja stanów**. Chcemy **wyeliminować stan u**, ale chcemy zachować etykiety krawędzi występujące w postaci wyrażeń regularnych, tak aby zbiór etykiet ścieżek miedzy dowolnymi pozostałymi stanami nie uległ zmianie.
- **Poprzedniki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

Konstrukcja eliminacji stanów

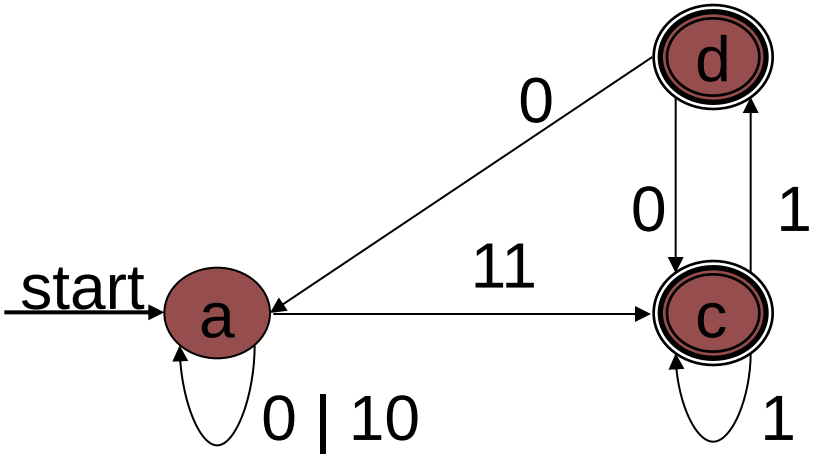
- Zbiór ciągów znaków etykietujących ścieżki wiodące **z wierzchołków s_i do wierzchołka u**, włącznie z ścieżkami biegnącymi kilkakrotnie wokół pętli u → u, oraz **z wierzchołka u do wierzchołka t_j**, jest opisany za pomocą wyrażenia regularnego **S_i U* T_j**.
- Po eliminacji wierzchołka u należy **zastąpić etykietę R_{ij}**, czyli etykietę krawędzi $s_i \rightarrow t_j$ **przez etykietę R_{ij} | Si U* T_j**.



Konstrukcja eliminacji stanów

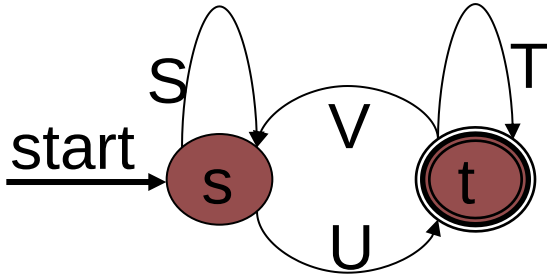


Automat skończony filtra odbijającego.

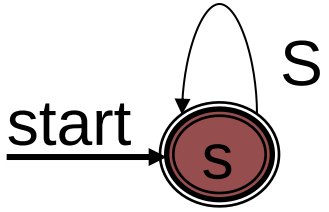


Automat skończony filtra odbijającego po wyeliminowaniu stanu b.

Redukcja zupełna automatu



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



Automat posiadający jedynie stan początkowy.
Wyrażenie regularne: S*

Wykład 11

Gramatyki

- Gramatyki bezkontekstowe
- Język gramatyk
- Drzewa rozbioru
- Niejednoznaczność gramatyk
- Analiza składniowa i konstrukcja drzew rozbioru
- Gramatyki a wyrażenia regularne

Opis wzorców

Opis wzorców polegający 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**:

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>

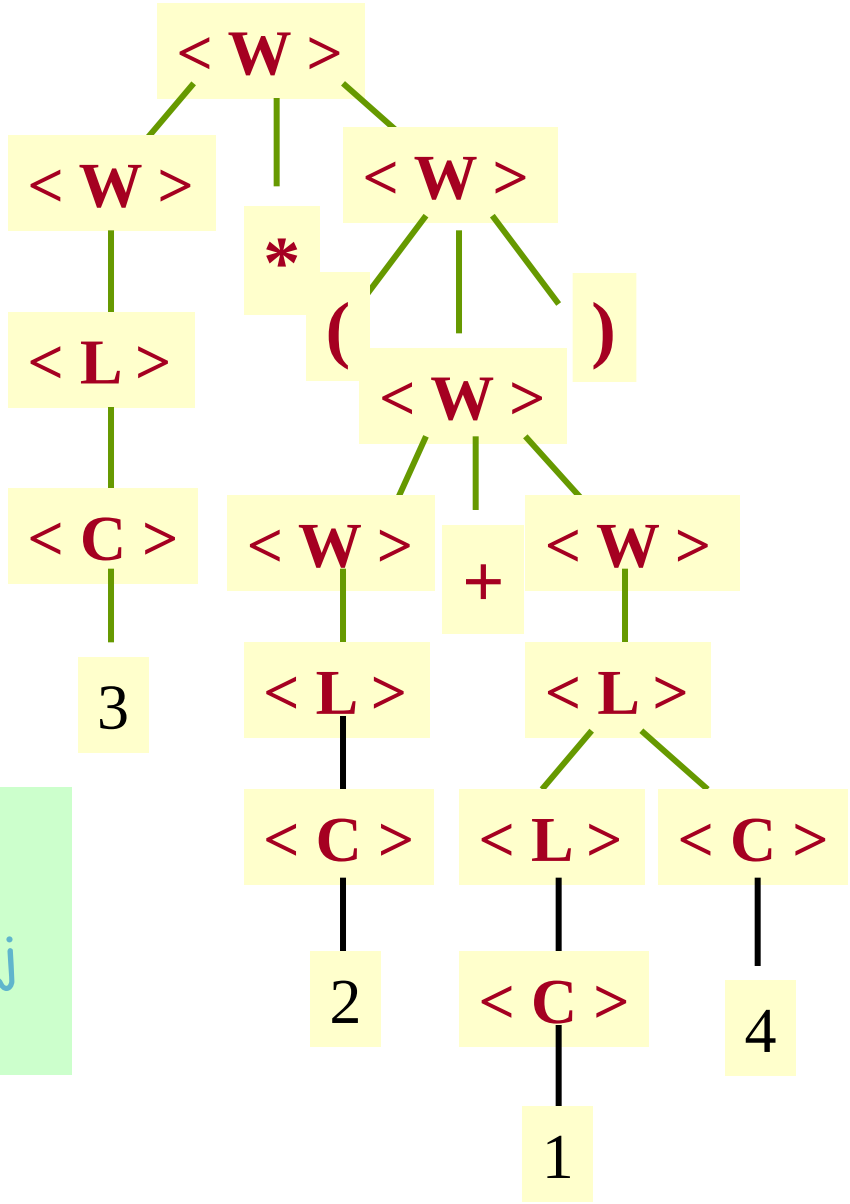
<Wyrażenie> → <Wyrażenie> - <Wyrażenie>

<Wyrażenie> → <Wyrażenie> * <Wyrażenie>

<Wyrażenie> → <Wyrażenie> / <Wyrażenie>

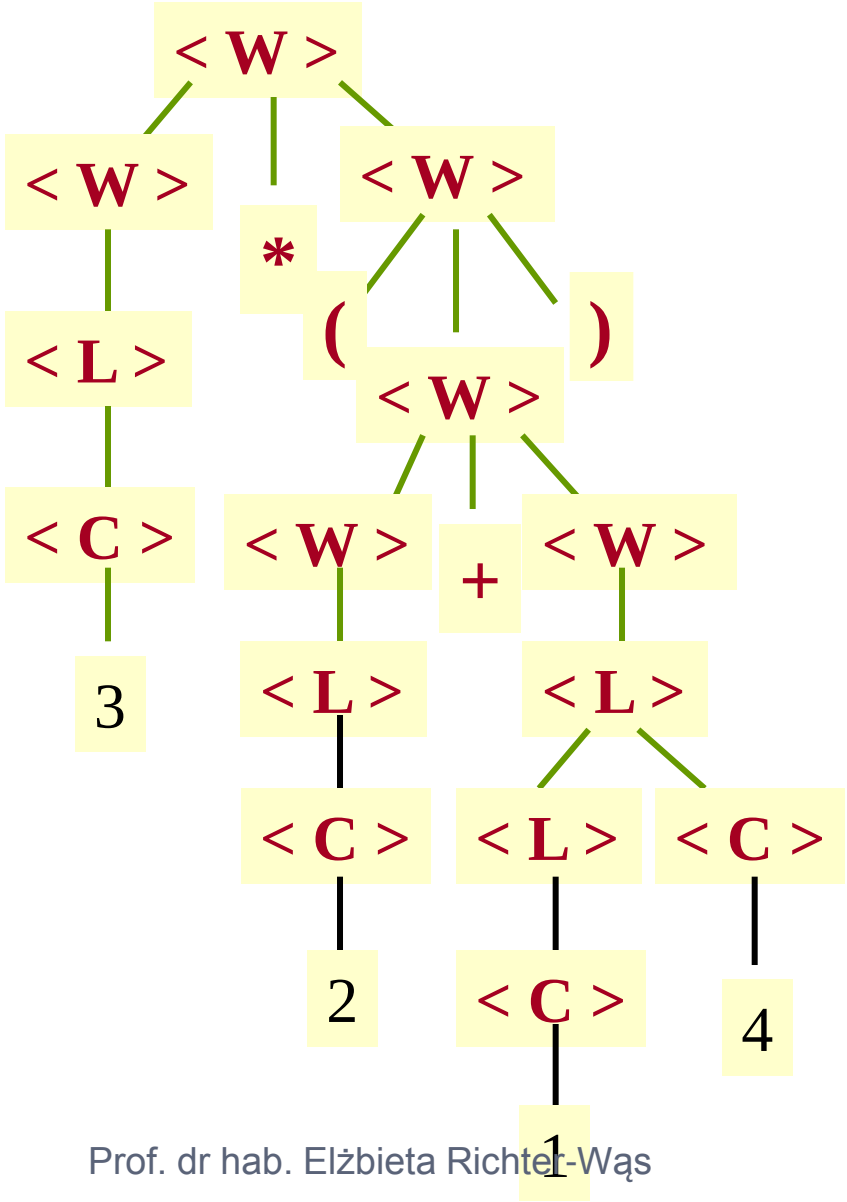
Gramatyka wyrażeń w której liczby zdefiniowano przy pomocy konstrukcji gramatycznych.

<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>

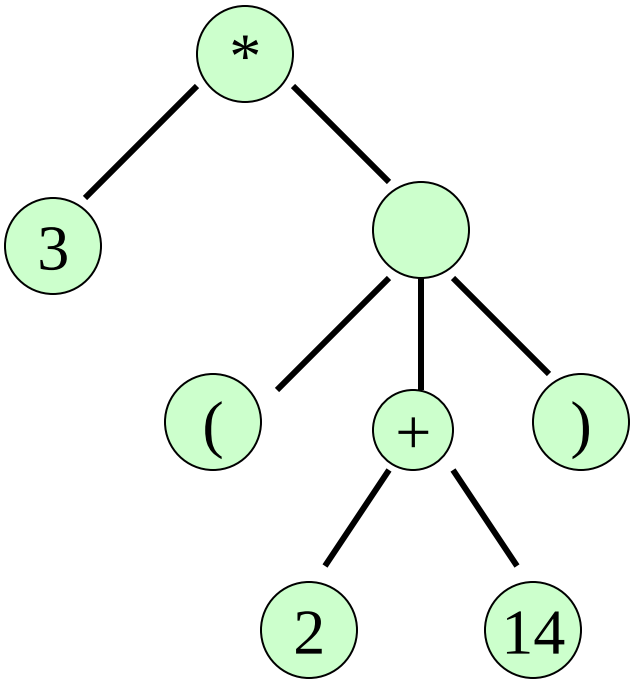


Drzewo rozbioru dla ciągu znaków
3 * (2 + 14)
przy użyciu gramatyki zdefiniowanej powyżej.

drzewo rozbioru



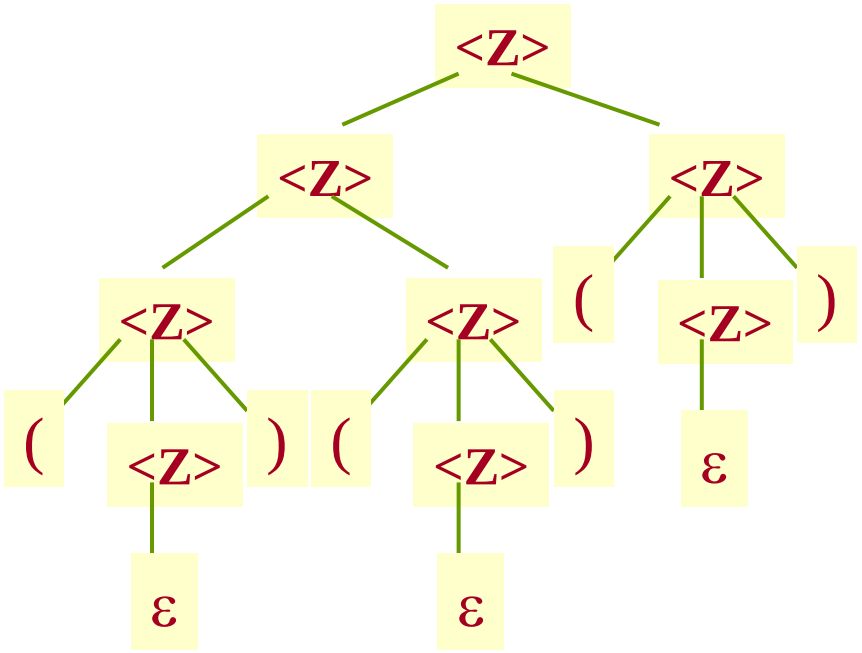
drzewo wyrażeń

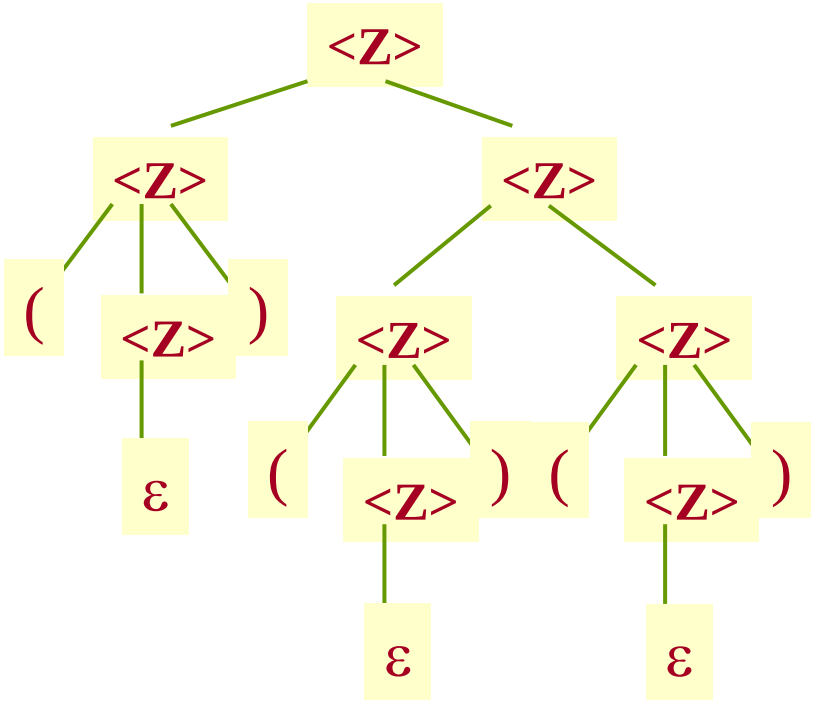


Niejednoznaczność i projektowanie gramatyk

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

<Z> → ε
<Z> → (<Z>)
<Z> → <Z> <Z>





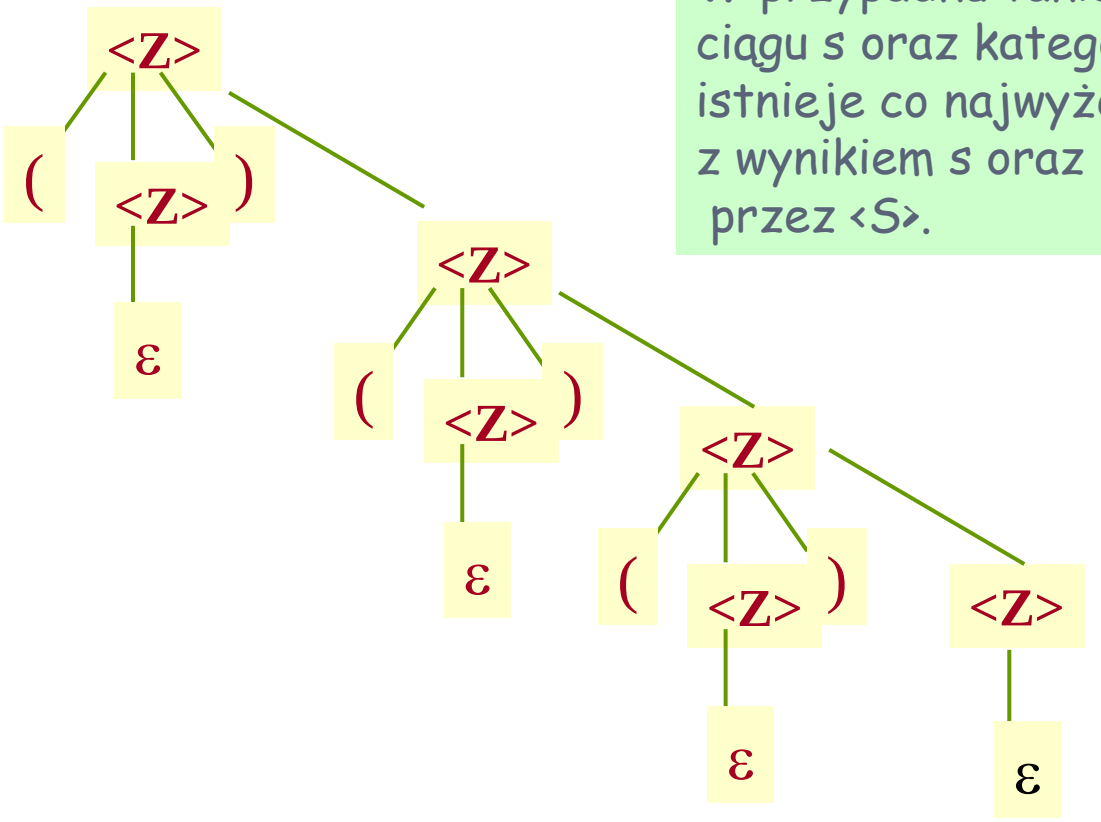
Gramatyka w której istnieją dwa lub więcej drzewa rozbioru o tym samym wyniku oraz tej samej kategorii syntaktycznej etykietującej korzeń jest nazywana *gramatyką niejednoznaczna. (ang. ambigous).* Wystarczy żeby istniał choć jeden taki ciąg który jest niejednoznaczny.

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

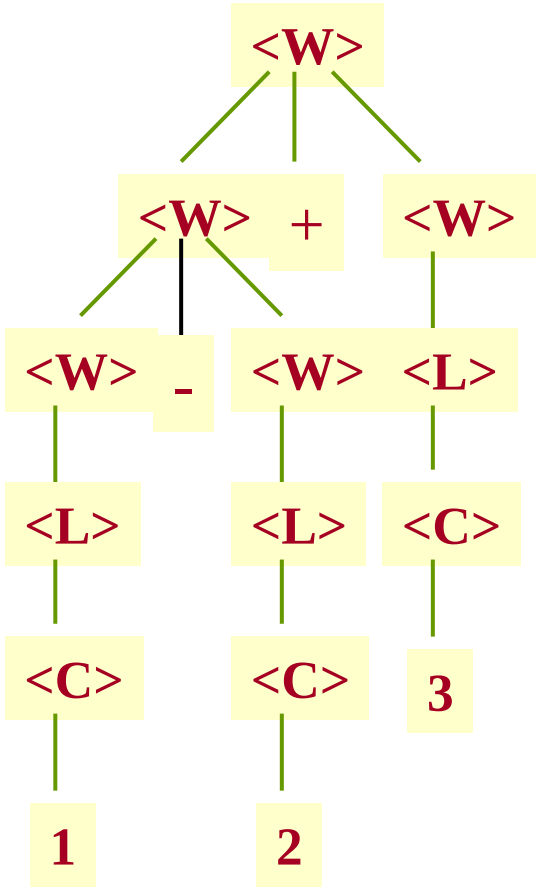
$\langle Z \rangle \rightarrow \varepsilon$

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

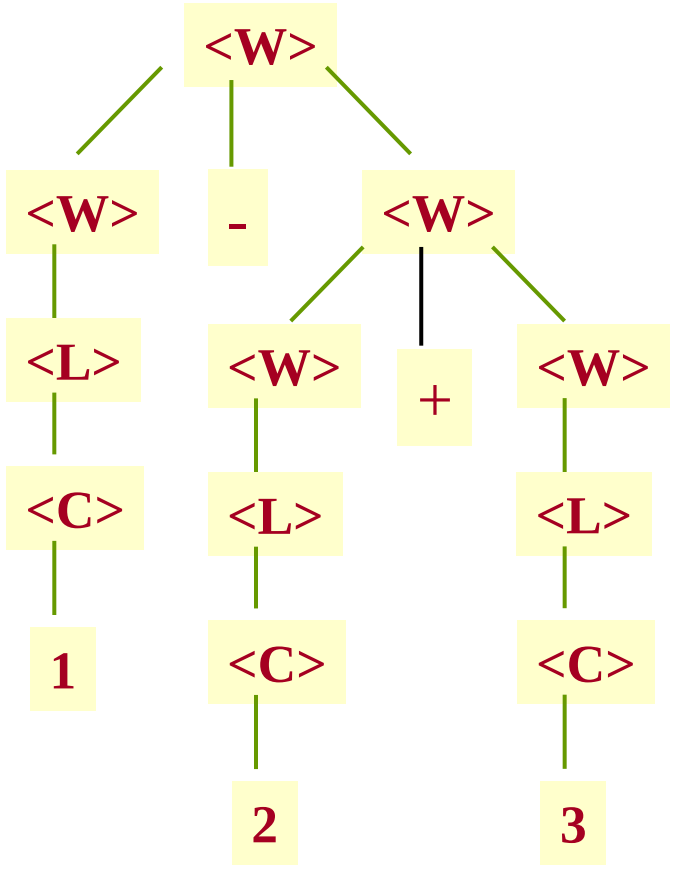
Gramatyka, która nie jest niejednoznaczna nosi nazwę *jednoznacznej (ang. unambigous)* .
W przypadku takiej gramatyki dla każdego ciągu s oraz kategorii syntaktycznej <S> istnieje co najwyżej jedno drzewo rozbioru z wynikiem s oraz korzeniem zaetykietowanym przez <S>.



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

Jednoznaczne gramatyki wyrażeń

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

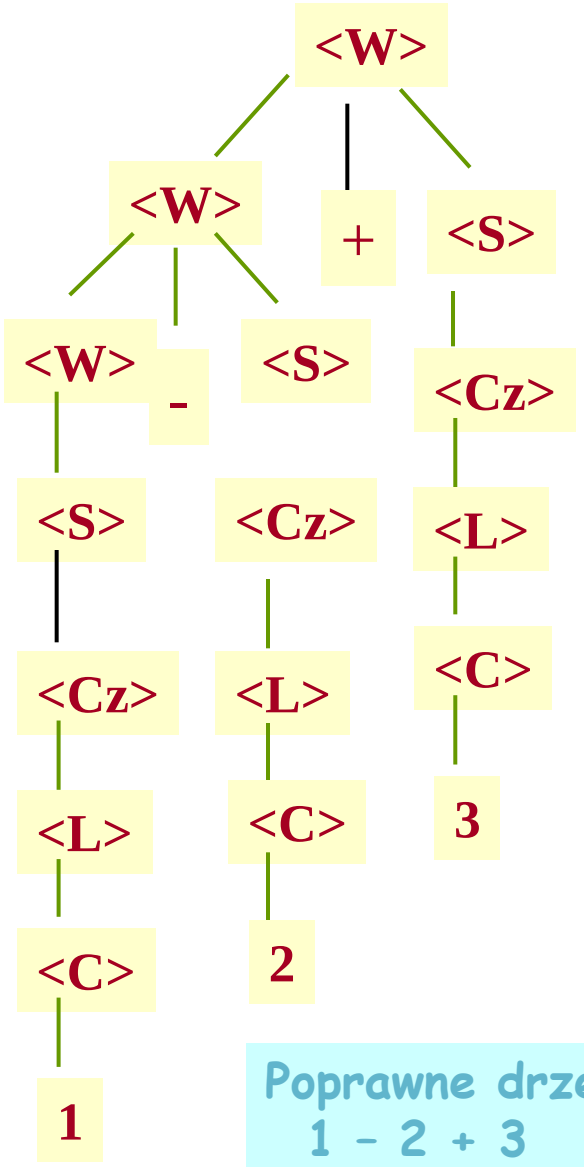
<Czynnik> - generuje wyrażenia, które nie mogą zostać rozdzielone, to znaczy czynnik jest albo pojedynczym operandem, albo dowolnym wyrażeniem umieszczonym w nawiasie.

< 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.

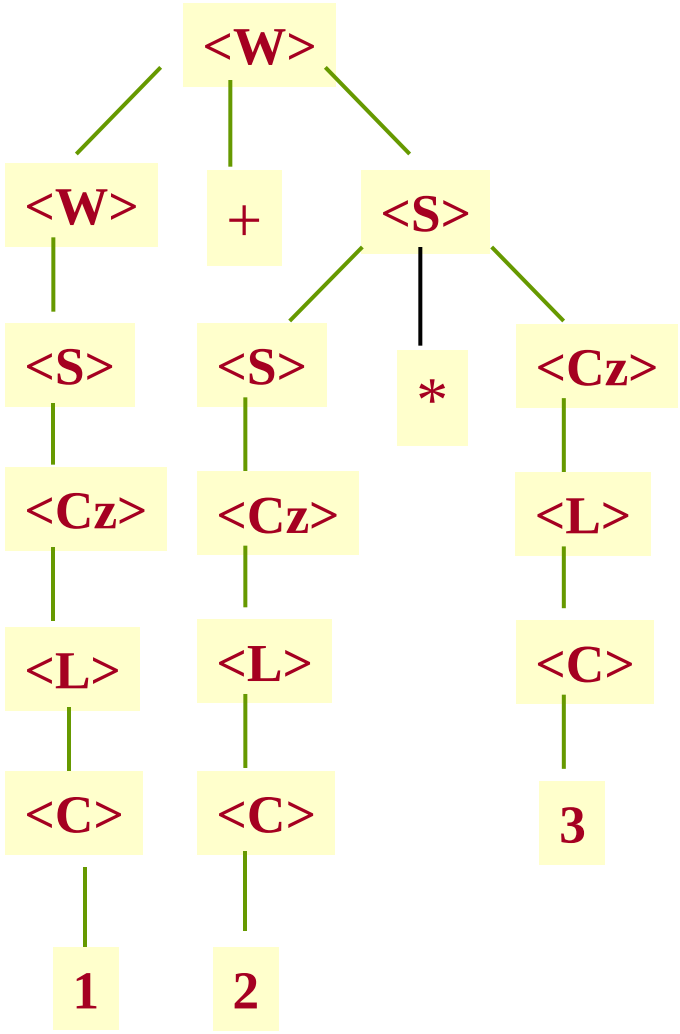
(1) <W> → <W> + <S> | <W> - <S> | <S>
(2) <S> → <S> * <Cz> | <S> / <Cz> | <Cz>
(3) <Cz> → (<W>) | <L>
(4) <L > → <L><C> | <C>
(5) <C> → 0 | 1 | ..... | 9

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).*

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>.

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

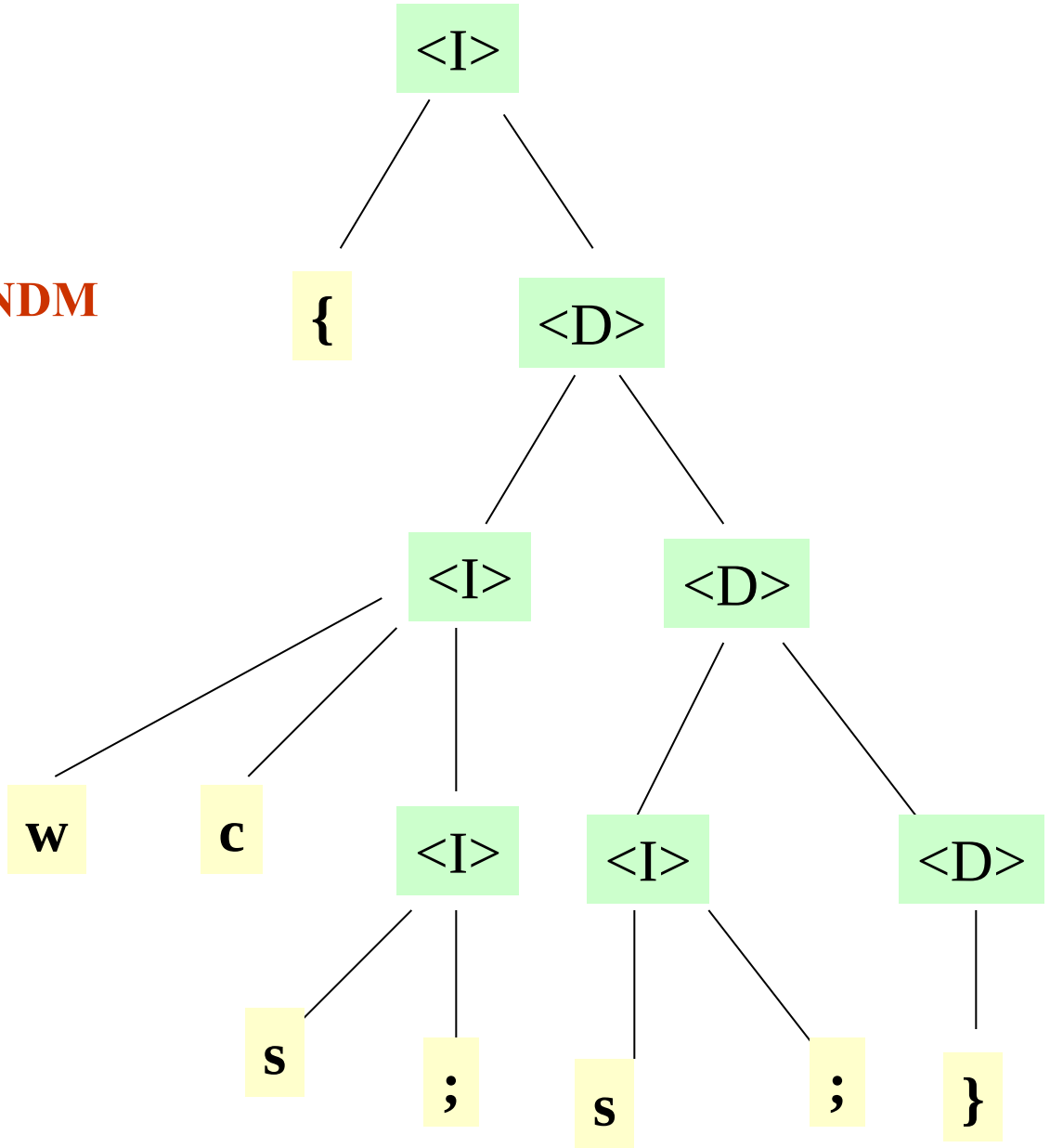
Uproszczona gramatyka instrukcji

<I> → w c <I>
<I> → {<L>}
<I> → s;

<L> → <L> <I>
<L> → ε

Uproszczona notacja

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



Konstrukcja gramatyki 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 > → 0 < S > 1
< S > → 0 1

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

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