

# Konstrukcja kompilatorów

Jan Daciuk

Katedra Inteligentnych Systemów Interaktywnych, Wydział ETI, Politechnika Gdańsk

semestr zimowy 2024-2025



# Organizacja przedmiotu

- sem. 5
- studia dzienne
- wymiar tygodniowy:
  - ▶ 1W ( $7 \times 2h + 1h$ )
  - ▶ 1P (2h co dwa tygodnie)
- prowadzący:
  - ▶ wykład:
    - ★ Jan Daciuk ([Jan.Daciuk@pg.edu.pl](mailto:Jan.Daciuk@pg.edu.pl))
  - ▶ projekt:
    - ★ Jan Daciuk ([Jan.Daciuk@pg.edu.pl](mailto:Jan.Daciuk@pg.edu.pl))
    - ★ konsultacje czw. 11:15-12:00 EA 423



# Zasady zaliczania

## wykład

obecność	10 pkt
zaliczenie	40 pkt
razem	50 pkt

## projekt

2 projekty po	15 pkt
2 projekty po	10 pkt
razem	50 pkt

- wspólna ocena dla obu części
- ocena zależy od sumy punktów za obie części
- trzeba zaliczyć **obie** części
  - ▶ co najmniej 25 pkt za wykład
  - ▶ co najmniej 25 pkt za projekt
- nie można zaliczać awansem

## skala ocen

punkty	ocena
(90 - 100]	bdb
(80 - 90]	db+
(70 - 80]	db
(60 - 70]	dst+
(50 - 60]	dst
[0 - 50]	ndst



# Wykład



- kolokwium za 40 pkt w drugim tygodniu po zakończeniu wykładu
- zalicza co najmniej 25 pkt
- sposób komunikacji z prowadzącymi:
  - ① na zajęciach
  - ② na konsultacjach (terminy podane na stronach prowadzących)
  - ③ przez pocztę elektroniczną na adres uczelniany:
    - ★ tytuł: [KK] temat
    - ★ treść
    - ★ podpis! (**nie** pseudonim)
- obecność
  - ▶ obowiązkowa, punktowana



# Projekt

- 5 projektów I, P1, ..., P4, z czego 4 oceniane
  - ▶ projekt I nie jest oceniany, realizowany na zajęciach
  - ▶ projekty P1 i P2 po 15 pkt
    - ★ realizowane w domu przed zajęciami
    - ★ oddawane na zajęciach — student musi umieć wytłumaczyć każdy wiersz kodu
  - ▶ projekt P3
    - ★ realizowany w całości na zajęciach (10 pkt)
  - ▶ projekt P4
    - ★ realizowany w całości w domu przed zajęciami (10 pkt)
- do zaliczenia potrzeba co najmniej 25 pkt
- oddanie projektów po terminie -3 pkt za każdy tydzień
- zaliczenie odbywa się wyłącznie w trakcie semestru
- zadania **muszą** być wykonywane samodzielnie
  - ▶ niesamodzielne wykonanie zadania  $\equiv 0$  pkt



Tematy:

I Wprowadzenie (bez oceny)

P1 Wyrażenia regularne i analiza leksykalna języka programowania

P2 Analiza składniowa języka programowania

P3 Analiza semantyczna

P4 Analizator XML

Języki programowania:

- C
- Turbo Pascal
- Modula
- Ada
- Simula 67



## Harmonogram zajęć

Termin	Salá	Gr															
Wykład wt 14-16	EA 32		1.10	8.10	15.10	22.10	29.10	5.11	12.11	19.11	26.11	3.12	10.12	17.12	24.12	31.12	7.01
			1-2	3-4	5-6	7-8	9-10		11-12	12-14	15						14.01
C wt 16-18	NE 239	C	1.10	8.10	15.10	22.10	29.10	5.11	12.11	19.11	26.11	3.12	10.12	17.12	24.12	31.12	7.01
		C <sub>1</sub>	I		P1		P2			P2		P3		P3		P4	14.01
Moduła wt 16-18	NE 239	M	1.10	8.10	15.10	22.10	29.10	5.11	12.11	19.11	26.11	3.12	10.12	17.12	24.12	31.12	7.01
		M <sub>1</sub>	I		P1		P2			P2		P3			P3	P4	14.01
Pascal cz 12-14	NE 239	P	3.10	10.10	17.10	24.10	31.10	7.11	14.11	21.11	28.11	5.12	12.12	19.12	29.21	2.01	9.01
		P <sub>1</sub>	I		P1	P2			P2		P3		P3	P4		16.01	
Ada cz 12-14	NE 239	A	3.10	10.10	17.10	24.10	31.10	7.11	14.11	21.11	28.11	5.12	12.12	19.12	29.21	2.01	9.01
		A <sub>1</sub>	I		P1	P2			P2		P3			P4			
Simula pt 13-15	NE 239	S	4.10	11.10	18.10	25.10	1.11	8.11	15.11	22.11	29.11	6.12	13.12	20.12	27.21	3.01	10.01
		S <sub>1</sub>	I		P1	P2			P2		P3		P3	P4		17.01	
		S <sub>2</sub>	I		P1										P4	P4	24.01

W połowie semestru możliwa redukcja liczby grup projektowych.



Podstawowa:

- ① Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman: *Kompilatory. Reguły, metody i narzędzia*, Wydanie II, WNT, Warszawa 2019. Niestety tłumaczenie na polski kawy firmy WITCOM Witold Sikorski.

Uzupełniająca:

- ① John E. Hopcroft, Rajeev Motwani, Jeffrey Ullman, *Wprowadzenie do teorii automatów, języków i obliczeń*, PWN, Warszawa 2005.
- ② Niklaus Wirth, *Algorytmy + struktury danych = programy*, WNT, Warszawa, 1980.
- ③ Mariusz Szwoch, *Automaty, języki formalne i translatory*, PWNT, Gdańsk 2008.

Narzędzia `flex` i `bison` mają swoje podręczniki w systemie Linux. Wywołuje się je jako: `man flex` i `man bison` lub (lepiej) jako `info <nazwa narzędzia>`.



## Cel przedmiotu

Celem przedmiotu jest opanowanie przez studentów umiejętności samodzielnego tworzenia translatorów, w szczególności kompilatorów języków programowania wysokiego poziomu, a także zrozumienie zasad ich działania, co jest pomocne przy radzeniu sobie w przypadku występowania błędów kompilacji. Po udanym zaliczeniu przedmiotu studenci powinni być w stanie napisać analizator szerokiej klasy kodów źródłowych z wykorzystaniem generatorów analizatorów leksykalnych i generatora analizatorów składniowych.



# Biurokratyczne absurdy

## Efekt kierunkowy

[1673] [K\_W11] zna metody

archiwizacji danych w sieci, rozumie koncepcje modelowania dokumentów cyfrowych i najważniejsze standardy reprezentacji dokumentu w postaci parsownalnej

[1677] [K\_W15] zna architektury

komputerów, procesy systemu operacyjnego, systemy plików, programy do przetwarzania tekstu, zasady zarządzania dyskami i pamięcią RAM, budowę kompilatorów, skanerów, parserów i analizatorów semantycznych

[1683] [K\_U01] tworzy i testuje

oprogramowanie z użyciem nowoczesnych metod i platform technologicznych, dostosowuje proces twórczy do potrzeb projektu, analizuje zakres i pracochłonność określonych zmian oprogramowania

## Efekt z przedmiotu

Student rozumie koncepcje składni języków programowania modelowanej z użyciem gramatyk bezkontekstowych oraz użycia systemu typów danych do modelowania znaczenia.

Student podaje definicje i klasyfikacje gramatyk i automatów formalnych. Przedstawia działanie analizatorów leksykalnych, składniowych i semantycznych. Student wyjaśnia budowę kompilatorów i rolę ich komponentów. Przedstawia metody optymalizacji kodu. Wykorzystuje wyrażenia regularne do tworzenia analizatorów leksykalnych. Tworzy parsery danych oraz programów komputerowych

Student potrafi wykorzystać generatory kompilatorów do tworzenia kompilatorów oraz analizatorów składniowych do różnych zastosowań

## Sposób weryfikacji i oceny efektu

[SW1] Ocena wiedzy faktograficznej

[SW1] Ocena wiedzy faktograficznej

[SU4] Ocena umiejętności korzystania z metod i narzędzi

[SU1] Ocena realizacji zadania

- ① Ogólne wiadomości o translatorach
- ② Narzędzia do tworzenia kompilatora
  - ① generator analizatorów leksykalnych **flex**
  - ② generator analizatorów składniowych **bison**
- ③ Algorytmy i struktury danych do tworzenia kompilatorów
  - ① wyrażenia regularne i automaty skończone
  - ② języki bezkontekstowe
  - ③ analiza składniowa
  - ④ analiza semantyczna
  - ⑤ generowanie i optymalizacja kodu



## program

```
#include <stdio.h>
int
main(const int argc,
     const char *argv[])
{
    printf(''Hello world!'');
}
```

## komputer

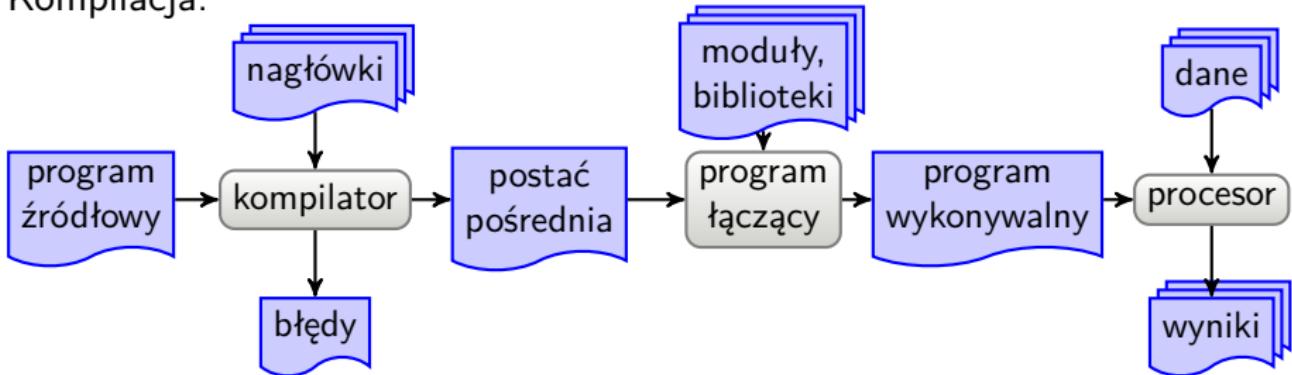
Słowo maszynowe (Intel 64 i IA-32):

1	2	3	4	5	6
---	---	---	---	---	---

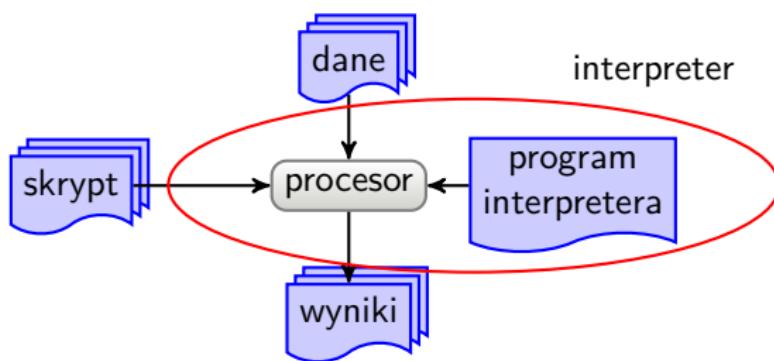
- 1 przedrostek (do 4 1B)
- 2 kod rozkazu (1B ÷ 3B)
- 3 tryb (0 ÷ 1B)
- 4 SIB (0 ÷ 1B)
- 5 przesunięcie (1, 2 lub 4B)
- 6 natychmiastowy (1, 2 lub 4B)

- program jest tekstem (ciągiem znaków)
- komputer interpretuje zawartość pamięci (ciąg bitów/bajtów)
- rozkazy procesora są różne od poleceń języków wysokiego poziomu
- różne procesory mają różne zestawy rozkazów

## Kompilacja:



## Interpretacja:



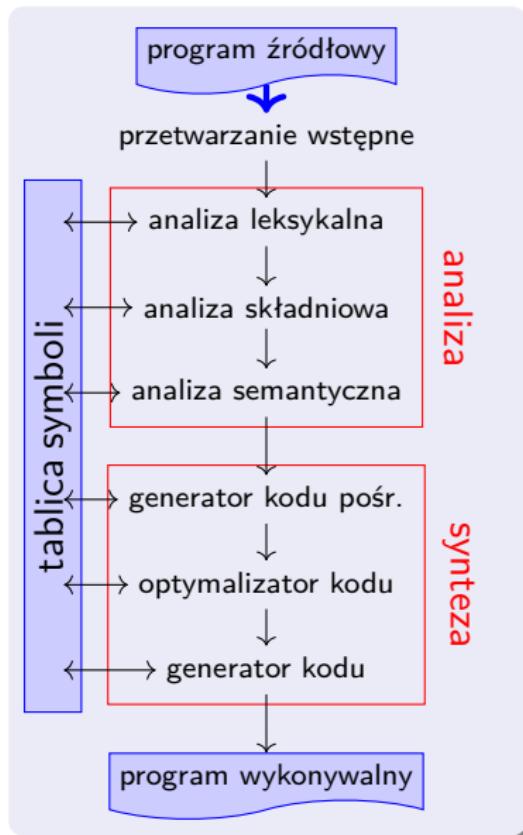


Inne schematy tłumaczenia:

- tłumaczenie na kod maszyny wirtualnej
  - ▶ kod źródłowy jest tłumaczony na kod pośredni
  - ▶ kod pośredni jest wykonywany przez maszynę wirtualną
- kompilacja dokładnie o czasie (ang. *just in time*)
  - ▶ tłumaczenie jest dokonywane w trakcie wykonania programu
  - ▶ tłumaczony jest tylko aktualnie wykonywany fragment programu
  - ▶ raz przetłumaczony fragment może być wielokrotnie wykorzystywany
  - ▶ często tłumaczyc się na najpierw na kod maszyny wirtualnej
- makroprzetwarzanie
  - ▶ definiowane są reguły przekształcania dopasowanego tekstu na inny tekst
  - ▶ obecnie stosowane głównie jako etap przetwarzania wstępne w komplikacji i asemblacji
  - ▶ obecnie bardzo rzadko stosowane samodzielnie do tłumaczenia programów

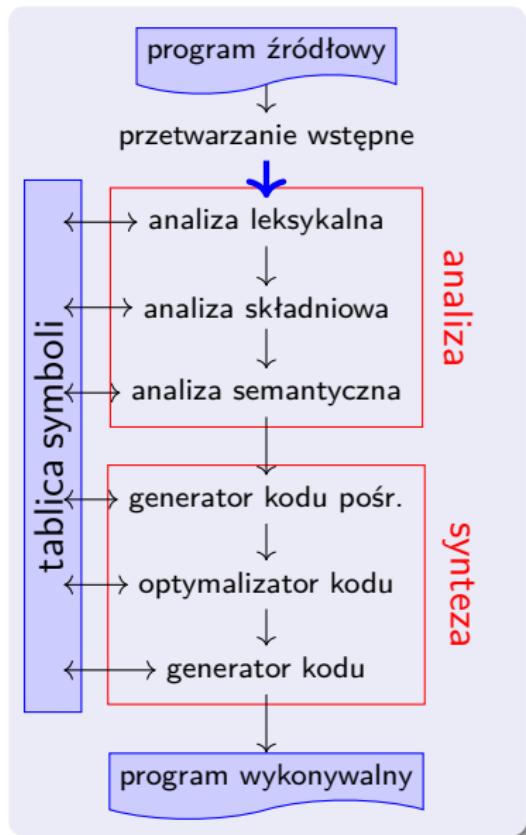


## Budowa kompilatora



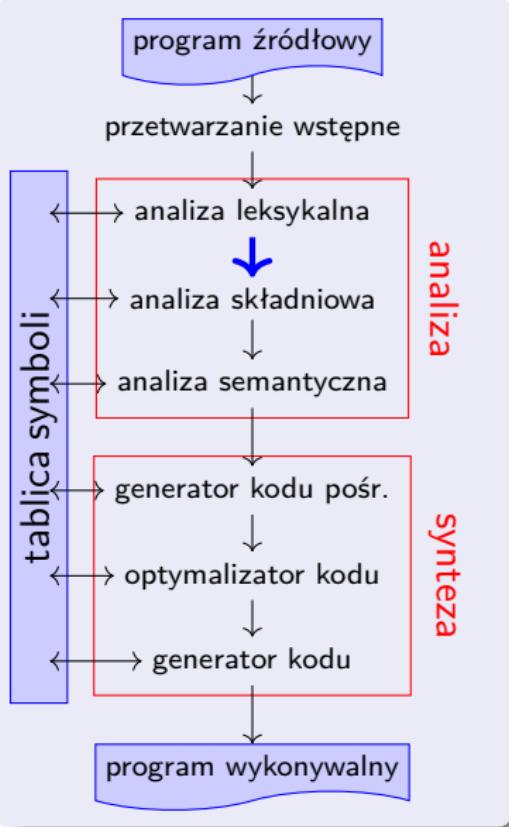
dane

```
#define MAX 50
const int C = 5;
int a = 7;
double b;
...
b = MAX + a * C;
b = b + C;
```



dane

```
const int C = 5;  
int a = 7;  
double b;  
...;  
b = 50 + a * C  
b = b + C;
```

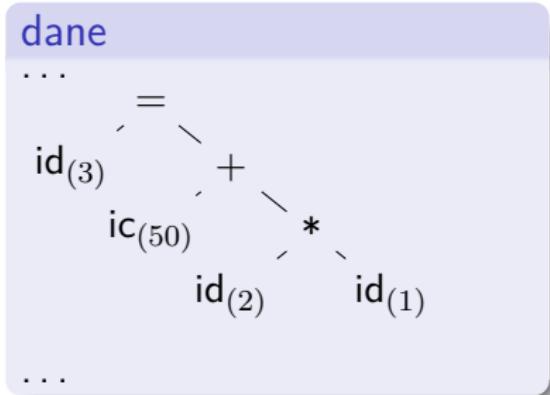
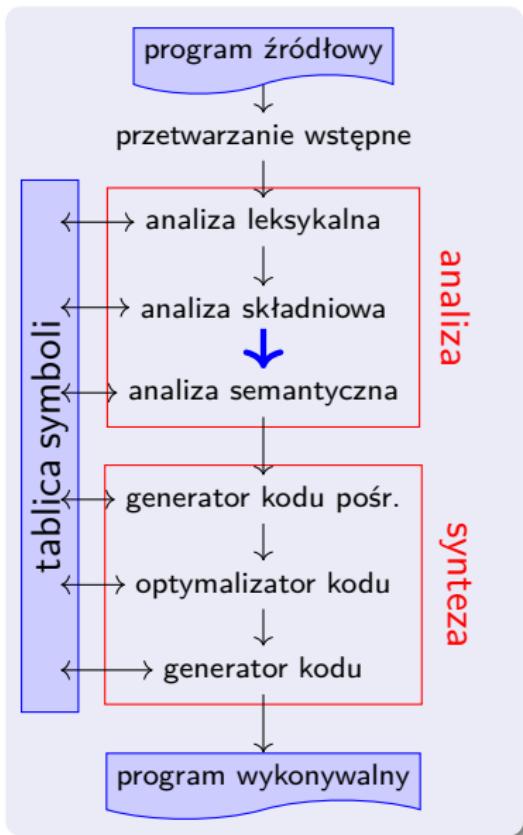


## dane

$\text{kw}_{(7)} \text{ kw}_{(12)} \text{id}_{(1)} = \text{ic}_{(5)} ;$   
 $\text{kw}_{(12)} \text{id}_{(2)} = \text{ic}_{(7)} ;$   
 $\text{kw}_{(8)} \text{id}_{(3)} ;$   
 ...  
 $\text{id}_{(3)} = \text{ic}_{(50)} + \text{id}_{(2)} * \text{id}_{(1)} ;$   
 $\text{id}_{(3)} = \text{id}_{(3)} + \text{id}_{(1)} ;$

## tablica symboli

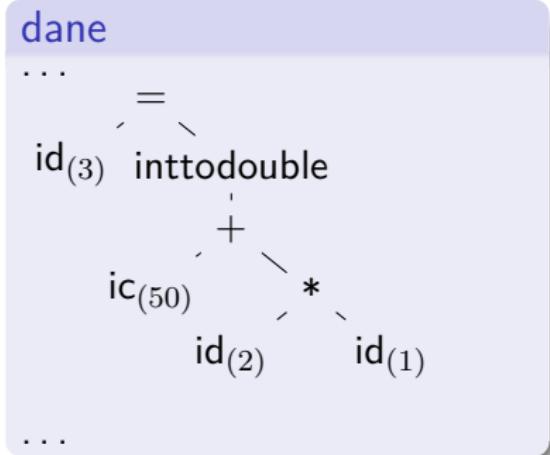
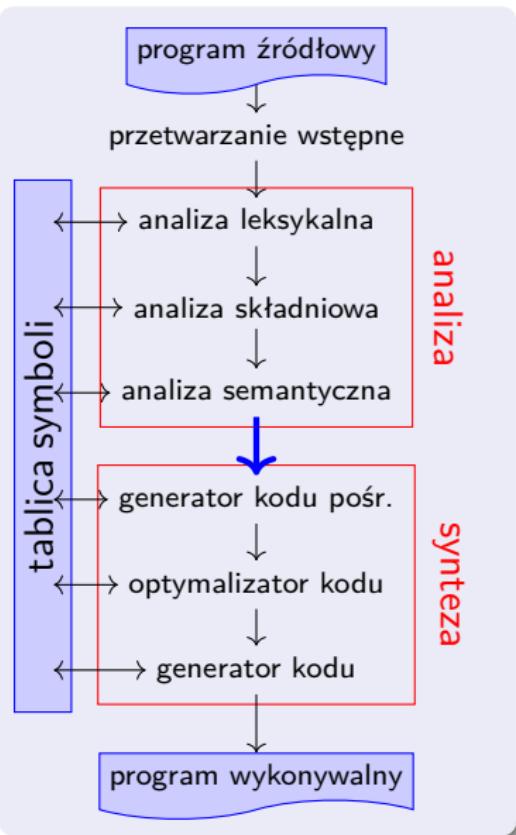
lp	nazwa	typ	wartość
1	C	1	...
2	a	2	...
3	b	8	...



This diagram shows the full symbol table with the following entries:

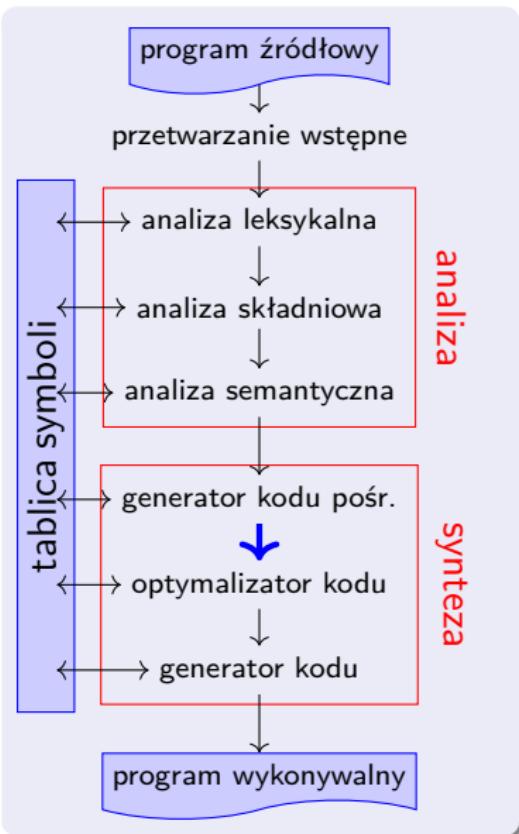
lp	nazwa	typ	wartość
1	C	1	...
2	a	2	...
3	b	8	...

The table is labeled **tablica symboli** at the top. Ellipses above and below the table indicate it contains more entries.



**tablica symboli**

lp	nazwa	typ	wartość
1	C	1	...
2	a	2	...
3	b	8	...

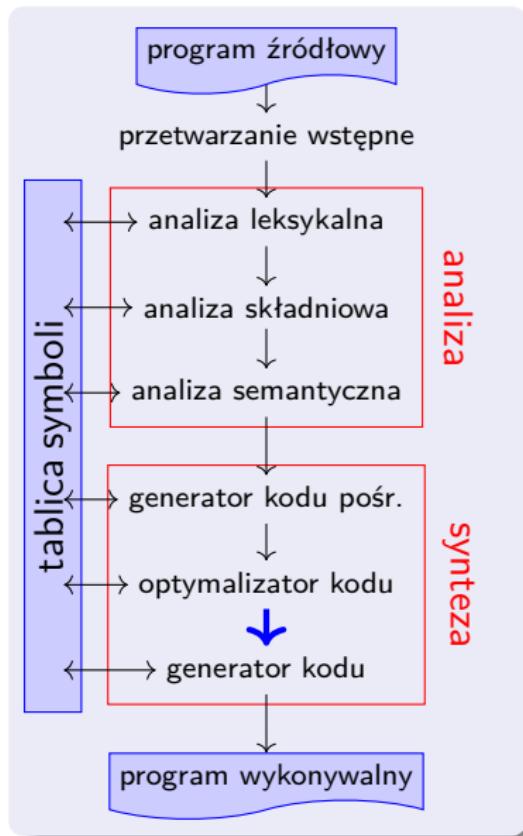


## dane

```
...  
temp1 ← id(2)  
temp2 ← temp1 * id(1)  
temp3 ← temp2 + 50  
temp4 ← inttодouble(temp3)  
temp5 ← inttодouble(50)  
temp4 ← temp4 + temp5  
...
```

## tablica symboli

lp	nazwa	typ	wartość
1	C	1	...
2	a	2	...
3	b	8	...

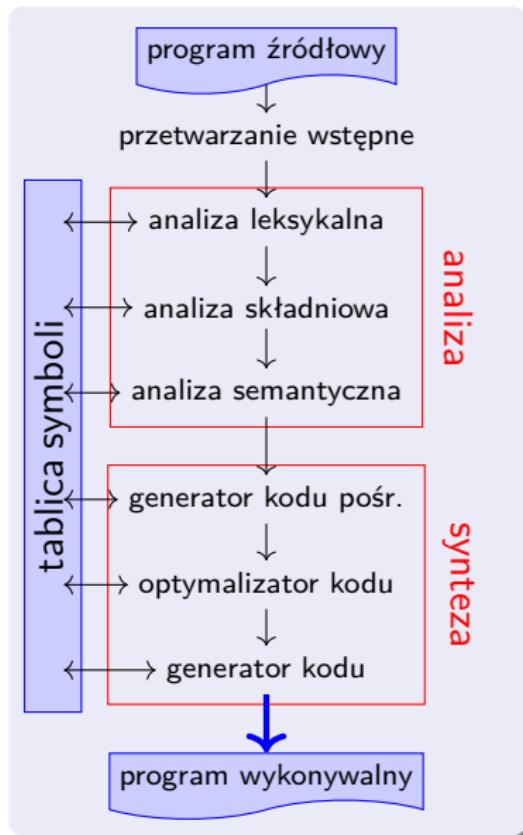


dane

...  
temp<sub>1</sub> ← 6  
temp<sub>1</sub> ← temp<sub>1</sub> \* id<sub>(2)</sub>  
temp<sub>1</sub> ← temp<sub>1</sub> + 50  
temp<sub>2</sub> ← inttodouble(temp<sub>1</sub>)  
...

tablica symboli

lp	nazwa	typ	wartość
1	C	1	...
2	a	2	...
3	b	8	...



## dane

...  
MOV R1,6  
MUL R1,6  
ADD R1,#50  
CAL inttодouble  
...

## tablica symboli

lp	nazwa	typ	wartość
1	C	1	...
2	a	2	...
3	b	8	...



Występuje w nielicznych językach programowania wysokiego poziomu i w wielu asemblerach (makroassemblerach). Zadania:

- makroprzetwarzanie
- dołączanie plików
- tłumaczenie warunkowe
- ustawianie parametrów tłumaczenia

Ponieważ najbardziej znane jest z C/C++, zajmiemy się tym językiem.



Makroprzetwarzanie jest zastępowaniem jednego tekstu innym.

Makrodefinicja:

- definiuje, co i gdzie ma być zastępowane
- przykłady:
  - ① `#define MAX_SIZE 128`
  - ② `#define MAX(a,b) ((a) > (b) ? (a) : (b))`

Makrowywołanie

- wystąpienie wcześniej zdefiniowanego tekstu (także z parametrami)
- przykłady:
  - ① `int Symbols[MAX_SIZE];`
  - ② z różnymi typami
    - ① `if (a < MAX(i, MAX_SIZE))`
    - ② `ocena = MAX(suma, 2.0);`



Makroprzetwarzanie jest zastępowaniem jednego tekstu innym.

Makrodefinicja:

- definiuje, co i gdzie ma być zastępowane
- przykłady:
  - ① `#define MAX_SIZE 128`
  - ② `#define MAX(a,b) ((a) > (b) ? (a) : (b))`

Makrowywołanie

- wystąpienie wcześniej zdefiniowanego tekstu (także z parametrami)
- przykłady:
  - ① `int Symbols[128];`
  - ② z różnymi typami
    - ① `if (a < ((i) > (128) ? (i) : (128)))`
    - ② `ocena = ((suma) > (2.0) ? (suma) : (2.0));`



## automaton.c

```
/**> automaton.c ***/
#include <automaton.h>

int create_state(void) {
#ifndef DEBUG
    fprintf(stderr, "create_state,
              states=%d\n", states);
#endif
    if (states + 1 < MAX) {
        return states++;
    }
}
/**> EOF automaton.c ***/
```

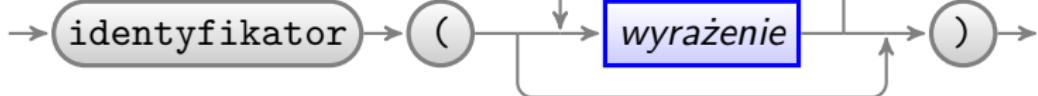
## automaton.h

```
/**> automaton.h ***/
#ifndef __automaton_h__
#define __automaton_h__
...
#endif
/**> EOF automaton.h ***/
```

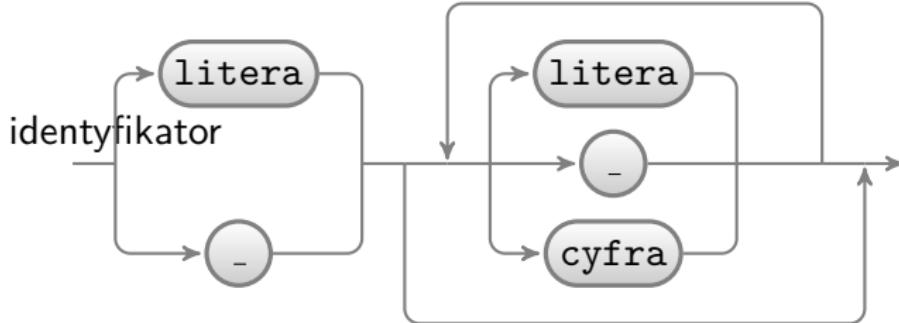
instrukcja warunkowa



wywołanie funkcji



ale



Jak odróżnić słowa kluczowe od identyfikatorów? Czy warto robić to na poziomie analizy składniowej?



Zadaniem analizy leksykalnej jest:

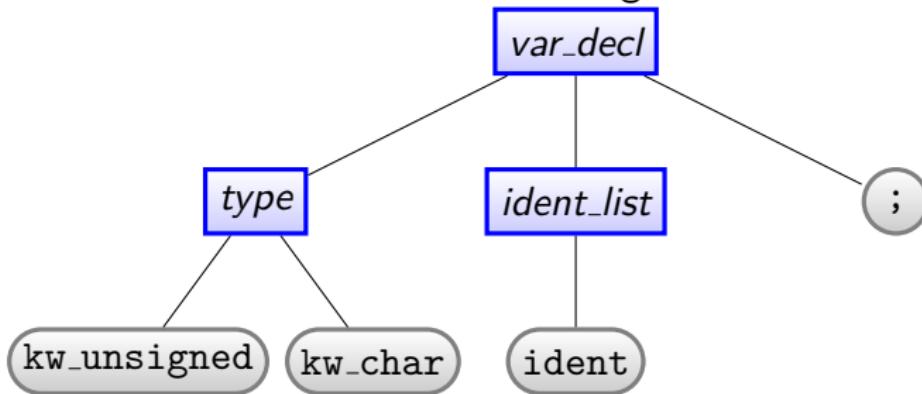
- rozpoznawanie elementów takich jak identyfikatory, stałe liczbowe różnych typów, napisy (na ogół w cudzysłowach lub apostrofach), operatory, interpunkcja itp.
- poszczególnym rodzajom elementów przypisane są odpowiednie numery
- ustalanie wartości tych elementów, dla których ma to sens, np. postaci identyfikatora, wartości liczbowych dla liczb, zawartości cudzysłówów dla napisów itp.
- pomijanie nieistotnych znaków, np. odstępów, tabulacji, przejścia do nowego wiersza itp., pomijanie komentarzy
- wykrywanie błędów takich jak brak znaków końca komentarza, znaki końca komentarza bez wcześniejszego otwarcia komentarza, napis nie kończący się w jednym wierszu itp.



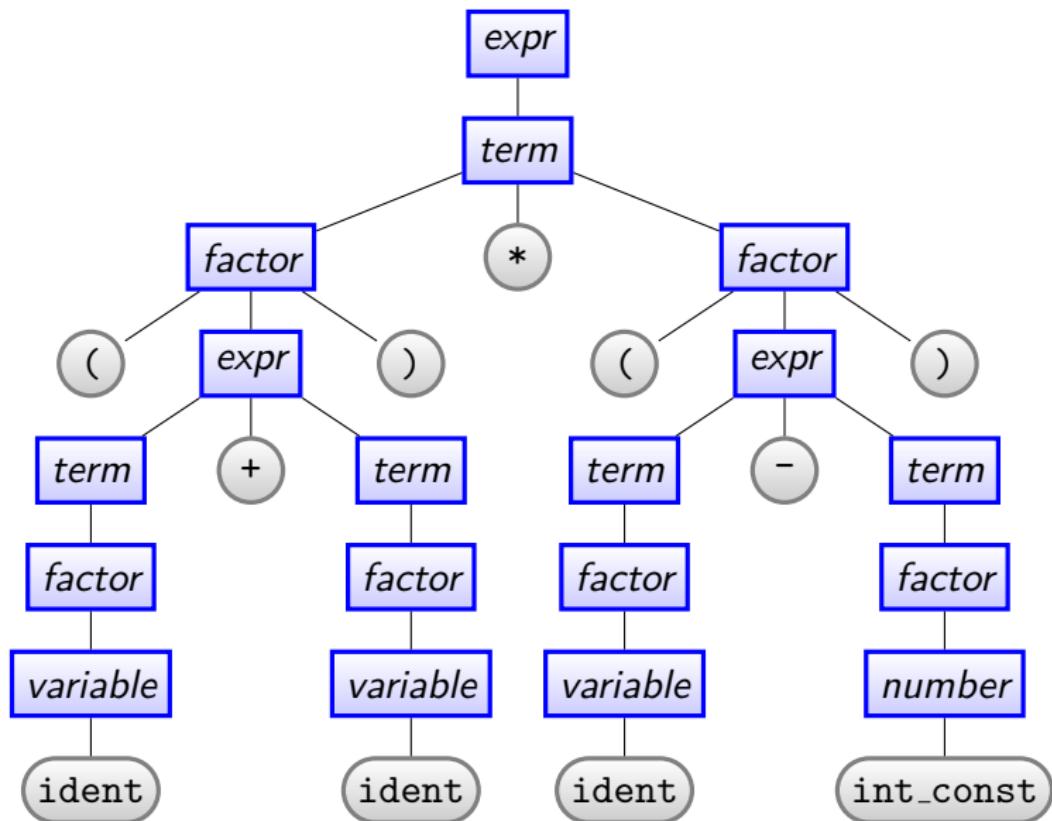
Zadaniem analizy składniowej jest:

- rozpoznanie struktury programu
- utworzenie drzewa rozbioru składniowego (w sposób jawnym lub niejawnym)
- sterowanie przebiegiem kompilacji — pobieranie danych z analizatora leksykalnego, rozpoczęcie analizy semantycznej i generowanie kodu dla rozpoznanych konstrukcji składniowych
- sprawdzanie poprawności składniowej programu i informowanie użytkownika o błędach składniowych

Drzewo składniowe dla zdania `unsigned char c;`



Uproszczone drzewo składniowe dla zdania:  $(a + b) * (c - 5)$





Zadaniem analizy semantycznej jest:

- ustalanie typów stałych, zmiennych i wyrażeń
- sprawdzanie zgodności typów
- ustalanie przekształceń typów
- ustalanie powiązań między deklaracjami i różnymi wystąpieniami stałych, zmiennych, procedur i funkcji
- wykrywanie użycia zmiennych, procedur i funkcji, których nie zdefiniowano
- wykrywanie użycia zmiennych, którym nie nadano wartości początkowych



# Generowanie kodu pośredniego



Celem fazy generowania kodu jest przekształcenie drzewa składniowego (danego w jawnej postaci lub pośrednio przez analizę sterowaną składnią) do postaci pośredniej. Postać pośrednia jest niezależna od docelowej maszyny. Jako postać pośrednia wykorzystywane są:

- drzewa składniowe
- kod trójadresowy
- zapis przyrostkowy



# Optymalizacja kodu

Celem optymalizacji kodu jest takie jego przekształcenie, by wynikowy program wykonywał się szybciej lub używał mniejszej ilości pamięci operacyjnej. Używa się różnych technik, takich jak:

- usuwanie wspólnych podwyrażeń
- przemieszczanie stałego kodu poza pętle
- propagacja kopii
- usuwanie zmiennych indukcyjnych
- optymalizacja przydziału rejestrów

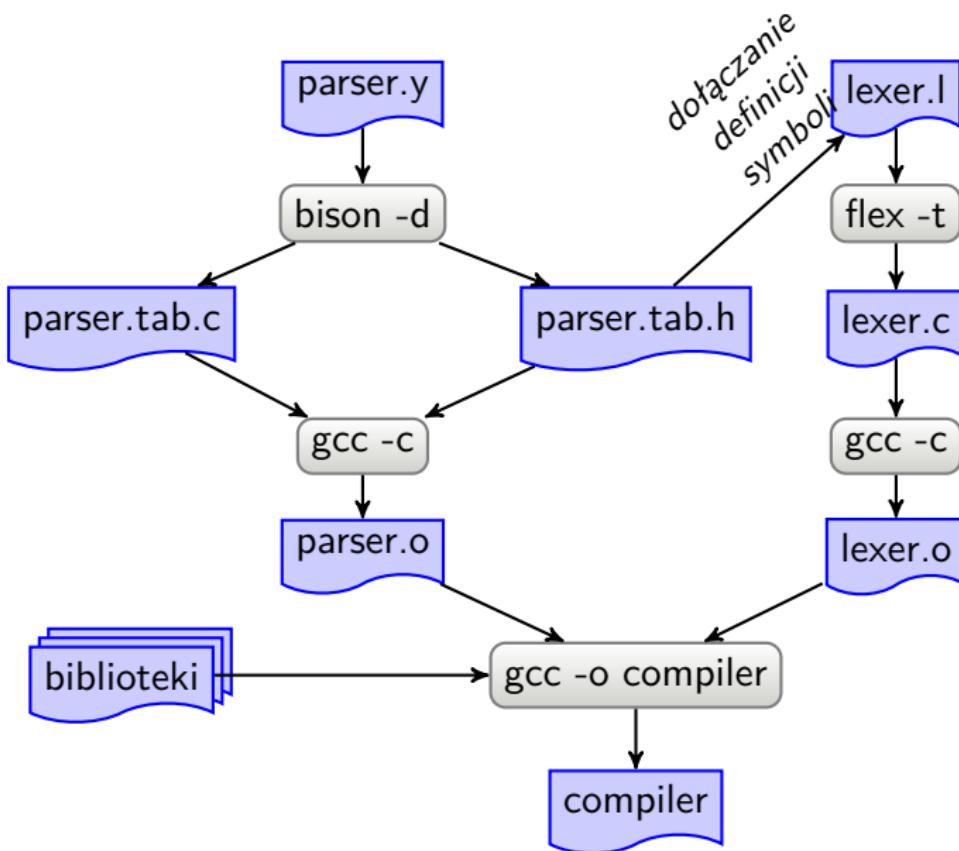


# Generowanie kodu wynikowego



Generowanie kodu jest fazą przekształcającą kod pośredni (na ogólny zoptymalizowany) na kod wynikowy będący bądź to kodem maszynowym, bądź to programem w języku asemblera. W kodzie wynikowym na ogólny używa się tylko adresów względnych. Umożliwia to łączenie różnych modułów programu i bibliotek. W trakcie generowania kodu rozwiązuje się takie problemy jak:

- zarządzanie pamięcią
- wybór rozkazów
- przydział rejestrów
- wybór kolejności obliczeń





Program **flex** jest generatorem analizatorów leksykalnych. Na wejściu dostaje plik we własnym języku. Plik ten standardowo ma rozszerzenie „.l” (małe L). Na wyjściu tworzy analizator leksykalny w języku C, który należy następnie skompilować. Możliwe są dwa tryby pracy:

- ① samodzielny: analizator samodzielnie przetwarza dane wejściowe; w tym trybie konieczne jest dołączenie biblioteki fleksa za pomocą opcji `LDFLAGS=-l` programu łączącego
- ② razem z analizatorem składniowym: analizator leksykalny dostarcza danych do analizatora składniowego; w tym trybie konieczne jest dołączenie tekstowe pliku nagłówkowego tworzonego przez generator analizatorów składniowych na podstawie programu analizatora składniowego



Program źródłowy dla analizatora leksykalnego składa się z trzech części, oddzielonych od siebie pojedynczymi wierszami zawierającymi wyłącznie dwa znaki procenta:

*deklaracje*

`%%` ← znaki kończące część deklaracji

*reguły*

`%%` ← znaki kończące część reguł

*definicje funkcji*

Poszczególne części mogą być puste. Jeśli brak jest definicji funkcji, to można też opuścić drugi wiersz z procentami.



Część deklaracji zawiera (w dowolnej kolejności):

- kod w języku C, w szczególności
  - ▶ dyrektywy dołączania plików
  - ▶ makrodefinicje
  - ▶ deklaracje zmiennych
  - ▶ prototypy funkcji
  - ▶ komentarze
- deklaracje w języku fleksa
  - ▶ nazwane definicje wzorców, które można wykorzystać w części regułowej
  - ▶ deklaracje warunków początkowych reguł (omówione później)

Deklaracje są interpretowane jako deklaracje fleksa lub języka C w zależności od ich umiejscowienia:

- kod w języku C umieszczany jest:
  - ▶ pomiędzy wierszem zawierającym wyłącznie znaki „%{”, a wierszem zawierającym wyłącznie znaki „%}” (**nie „}%”**)
  - ▶ w wierszach zaczynających się znakiem odstępu lub tabulacji
- kod dla fleksa umieszczany jest:
  - ▶ od pierwszej kolumny w pozostałych wierszach części deklaracji



Deklaracje języka to nazwane definicje wzorców. Mają postać wierszy z dwoma częściami rozdzielonymi białymi znakami:

nazwa              wzorzec

**Nazwa** jest niepustym ciągiem liter, znaków podkreślenia, myślników i cyfr zaczynająca się literą lub znakiem podkreślenia.

**Wzorzec** jest podawany jako rozszerzone wyrażenie regularne. Deklaracja przypisuje określonemu wzorcowi podaną nazwę. Można się później do niego odwołać w części regułowej (drugiej części) przez nazwę podaną w nawiasach klamrowych. Nazywanie wzorców nie jest konieczne. Można ich użyć bezpośrednio w części regułowej bez ich nazywania. Nazywanie wzorców może poprawiać czytelność kodu.



# Wyrażenia regularne

Wyrażeniem regularnym nad alfabetem  $\Sigma$  jest:

- ① zbiór pusty  $\emptyset$
- ② pusty ciąg symboli  $\varepsilon$  (ciąg o długości zero)
- ③ symbol  $\sigma$  z alfabetu  $\Sigma$
- ④ sklejenie  $R_1 R_2$  dwóch wyrażeń regularnych  $R_1$  i  $R_2$ 
  - po słowie z  $R_1$  następuje słowo z  $R_2$
- ⑤ alternatywa  $R_1 | R_2$  dwóch wyrażeń regularnych  $R_1$  i  $R_2$ 
  - występuje słowo albo z  $R_1$ , albo z  $R_2$
- ⑥ domknięcie przechodnie  $R^*$  wyrażenia regularnego  $R$ 
  - występuje dowolna liczba słów (w tym 0) z  $R$  jedno po drugim
- ⑦ wyrażenie regularne ujęte w nawiasy
- ⑧ nic ponadto

Najczęściej mamy do czynienia z rozszerzonymi wyrażeniami regularnymi.

Większość rozszerzeń nie zwiększa mocy rozpoznawanego języka.

Wyrażenie regularne opisuje język regularny i jest rozpoznawane przez automat skończony. Taki sam język może być opisany gramatyką regularną.

## Podstawowe cegiełki

- **x** — każdy zwykły znak oznacza samego siebie
- **\+** — znaki specjalne można dopasować poprzedzając je znakiem odwrotnego ukośnika; sam odwrotny ukośnik też należy do tej kategorii. Uwaga:
  - ▶ **\n** – znak nowego wiersza
  - ▶ **\r** – znak powrotu karetki (z Windows)
  - ▶ **\t** – znak tabulacji poziomej
  - ▶ **\f** – znak nowej strony
  - ▶ **\b** – znak cofnięcia karetki
  - ▶ **\xnn** – znak o kodzie szesnastkowym *nn*
- **"+"** — cudzysłowy także pozwalają cytować znaki specjalne; w cudzysłowach można umieścić dłuższy ciąg znaków, np. **"++"**
- **«EOF»** — pasuje do końca pliku

## Klasy znaków

- **[aeiouy]** — pojedynczy znak spośród tych podanych w nawiasach; wewnątrz klasy znaki specjalne poza „\”, „-”, „]” i „^”, które nie są na początku, tracą specjalne znaczenie; warianty:
  - ▶ **[0-9]** — ciągły zakres można podać stosując myślnik (podany przykład oznacza pojedynczą cyfrę)
  - ▶ **[^0-9]** dopełnienie klasy (tutaj: wszystkie znaki poza cyframi, **włączając znak nowego wiersza**)
  - ▶ zdefiniowane klasy znaków **[:alphanum:]**, **[:alpha:]**, **[:blank:]**, **[:cntrl:]**, **[:digit:]**, **[:graph:]**, **[:lower:]**, **[:print:]**, **[:punct:]**, **[:space:]**, **[:upper:]**, **[:xdigit:]** oznaczają to, co odp. funkcje w C
- **.** — kropka oznacza dowolny znak oprócz znaku nowego wiersza

## Typowe błędy:

- **[^0-9^a-z]** oznacza każdy znak oprócz cyfry, małej litery i znaku „^”, włączając nowy wiersz — raczej chodziło o: **[^0-9a-z\n]**
- **[0-9 | "++"]** nie oznacza cyfry lub dwóch znaków plus (oznacza cyfrę, kreskę pionową, cudzysłów lub plus) — prawidłowo: **[0-9]|"++"**

**Złożone wyrażenia.** Dla wyrażeń  $r_1$  i  $r_2$ :

- $r_1r_2$  — wyrażenia zapisane bezpośrednio jedno po drugim dopasowują się do sklejenia elementów rozpoznawanych przez składowe wyrażenia, np. **.[A-Za-z]** rozpoznaje dwa znaki, z których pierwszy nie jest znakiem nowego wiersza, a drugi jest literą łacińską
- $r_1|r_2$  — dopasowuje się do jednego wyrażenia spośród tych rozdzielonych kreską pionową, np. **[0-9]|\+|-** dopasuje się do cyfry, znaku plus lub znaku minus
- **( $r_1$ )** — nawiasy grupują wyrażenia, pozwalając zmienić domyślną kolejność wartościowania wyrażenia, np. **[0-9](.|e)** to cyfra i dowolny znak oprócz nowego wiersza, natomiast **[0-9].|e** to cyfra i dowolny znak albo pojedyncza litera „e”.

**Powtórzenia.** Dla wyrażenia  $r$ :

- $r\{n,m\}$  — oznacza powtórzenie wyrażenia  $r$  od  $n$  do  $m$  razy,  
np.  $\{2,3\}$  oznacza od dwóch do trzech gwiazdek; warianty:
  - ▶  $r\{n,\}$  —  $r$  co najmniej  $n$  razy
  - ▶  $r\{,m\}$  —  $r$  co najwyższej  $m$  razy
  - ▶  $r\{n\}$  —  $r$  dokładnie  $n$  razy
- $r?$  — to samo, co  $r\{0,1\}$  lub  $r\{,1\}$ , czyli nieobowiązkowe wystąpienie wyrażenia  $r$
- $r^*$  — to samo, co  $r\{0,\}$ , czyli dowolna liczba wystąpień wyrażenia  $r$ , włączając w to brak wystąpienia
- $r^+$  — to samo, co  $r\{1,\}$ , czyli co najmniej jedno wystąpienie  $r$ , równoważne  $rr^*$

**Kontekst.** Dla wyrażeń  $r_1$  i  $r_2$ :

- $\hat{r}_1$  — oznacza wystąpienie wyrażenia  $r_1$  na początku wiersza, np. `^#define` wykrywa dyrektywę `#define` zapisaną na początku wiersza; **uwaga:** różnica pomiędzy  $\hat{r}_1$  a  $\backslash nr_1$  polega na tym, że w pierwszym przypadku znak nowego wiersza nie należy do dopasowanego tekstu
- $r_1\$$  — oznacza wystąpienie wyrażenia  $r_1$  na końcu wiersza, np. `/.*/$` dopasowuje się do komentarza w języku C++; **uwaga:** różnica pomiędzy  $r_1\$$  a  $r_1\backslash n$  polega na tym, że w drugim przypadku znak nowego wiersza wchodzi w skład dopasowanego tekstu
- $r_1/r_2$  — oznacza wystąpienie wyrażenia  $r_1$ , po którym bezpośrednio występuje wyrażenia  $r_2$ , np. `[A-Za-z][A-Za-z0-9]*`: oznacza identyfikator przed dwukropkiem (prawdopodobnie etykietę); **uwaga:** wyrażenie  $r_2$  nie wchodzi w skład dopasowanego tekstu

## Opcje dla wyrażeń.

W wyrażeniu: (?r-s:wzorzec):

- **r** oznacza ciąg liter oznaczających opcje, które mają być włączone:
  - ▶ i — pomija wielkość liter
  - ▶ s — kropka („.”) oznacza także nowy wiersz
  - ▶ x — pomija białe znaki i komentarze (w C) w wyrażeniach
- **s** oznacza ciąg liter oznaczających opcje, które mają być wyłączone (opcje jak wyżej).

## Przykłady:

- (?i:begin) jest równoważne [Bb] [Ee] [Gg] [Ii] [Nn]
- (?-i:begin) jest równoważne begin
- (?s:.) jest równoważne [\x00-\xFF]
- (?-s:.) jest równoważne [^\n]
- (?x:a /\* b \*/ c) jest równoważne ac

## Odwołania do nazwanych definicji

Kiedy nazwiemy jakieś wyrażenie regularne, np.:

d [0-9]

to możemy je wykorzystać w innym wyrażeniu regularnym, np.:

```
float (({d}+\.{d}*)|({d}*\.{d}+))([eE](\+|-)?{d}{1,2})?
```

Na ogół czynimy tak dla poprawy czytelności. Nadużywanie nazw wyrażeń prowadzi do znaczącego pogorszenia czytelności.



## Przykłady

*Identyfikator* — niepusty ciąg liter łacińskich, cyfr i znaków podkreślenia zaczynający się literą lub znakiem podkreślenia:

[A-Za-z\_] [A-Za-z0-9\_]\*

*Liczba całkowita* — niepusty ciąg cyfr:

[0-9]+

*Liczba rzeczywista (w C)* — ciąg cyfr z obowiązkową kropką dziesiętną, przy czym albo część całkowita, albo część ułamkowa może być pominięta, z nieobowiązkową częścią, składającą się z litery *E*, nieobowiązkowego znaku i ciągu cyfr:

(([0-9]+\. [0-9]\*)([0-9]\*\. [0-9]+))([eE](\+|-)?[0-9]{1,2})?)

*Napis* — ciąg znaków nie zawierający cudzysłowów poprzedzony i zakończony cudzysłowem:

\\"[^\\\"\\n]\*\\"



## Flex — reguły

Część regułowa programu dla fleksa składa się z wierszy postaci:

wzorzec      działanie

lub postaci:

<określenie warunków>wzorzec      działanie

Wzorzec jest wyrażeniem regularnym (lub jego nazwą w nawiasach klamrowych). Działanie:

- jest kodem w języku C
- musi być poprzedzone co najmniej jednym odstępem lub znakiem tabulacji
- może być puste — oznacza wtedy pominięcie wzorca (brak działania dla niego)
- jeśli zawiera lewy nawias klamrowy, może być kontynuowane w następnych wierszach



## Wykonywanie reguł:

- jeżeli do tekstu wejściowego pasuje wzorzec, wykonywane jest skojarzone z nim działanie
- jeżeli do tekstu pasuje więcej niż jeden wzorzec, wybierany jest wzorzec, który dopasowuje się do dłuższego tekstu
- jeśli więcej niż jeden wzorzec dopasowuje się do najdłuższego tekstu, wybierane jest działanie skojarzone z pierwszym z tych wzorców według kolejności reguł w programie
- jeżeli do tekstu wejściowego nie pasuje żaden wzorzec, wykonywana jest reguła domyślna
- reguła domyślna dopasowuje się do pojedynczego znaku i kopiuje go na wyjście (wypisuje go)
- dopasowany tekst jest usuwany z wejścia



Działania we fleksie są kodem w języku C. Mogą zawierać instrukcje `return`. Mogą też używać następujących zmiennych:

- `yytext` — dopasowany tekst
- `yyleng` — długość dopasowanego tekstu
- `YY_START`, `YYSTATE` — bieżący warunek początkowy (omówiony później)
- `yylval` — wartość elementu przekazywanego do analizatora składniowego
- `yylineno` — numer bieżącego wiersza pliku wejściowego, jeśli sekcja deklaracji zawiera wiersz „`%option yylineno`”.
- `yyout` — plik (typu `FILE *`), który związany jest ze standardowym wyjściem
- `yyin` — plik związany ze standardowym wejściem



Mogą też zawierać specjalne dyrektywy:

- | — działanie takie same, jak dla następnej reguły
- ECHO — wypisuje dopasowany tekst, tj. `printf("%s", yytext);`
- BEGIN(*w*) — ustawienie warunku początkowego *w*
- REJECT — (na ogół po wykonaniu czegoś) próba dopasowania następnej reguły do tego samego wejścia
- yymore() — następna reguła dopisze dopasowany tekst na końcu `yytext`
- yyless(*n*) (na ogół po wykonaniu czegoś) zwraca dopasowany tekst na wejście poza pierwszymi *n* znakami
- unput(*c*) — umieszcza znak *c* na wejściu i niszczy `yytext`
- input() — czyta i zwraca następny znak na wejściu
- yy\_terminate() — kończy działanie analizatora leksykalnego (wywoływane automatycznie po napotkaniu końca pliku)



## Warunki początkowe

Dopasowanie niektórych konstrukcji za pomocą wyrażenia regularnego może okazać się kłopotliwe. Np. komentarz w języku C zaczyna się znakami „/\*” i kończy znakami „\*/”. Zapiszmy to jako wyrażenie regularne:

```
/\*.*\*/
```

Zgadza się? Nie do końca. Dopasowanie następuje do najdłuższego tekstu:

```
/* komentarz */ printf("Ha!\n"); /* komentarz */
```

Wyrażenie „pożre” także instrukcję printf umieszczoną pomiędzy komentarzami. Poprawny zapis:

```
("/*((\*+[^/])|[^*])*/*+\/)|(/*\*\*\*\/")
```



# Warunki początkowe



Deklaracja (w części deklaracji — pierwszej — programu dla fleksa):

*%s warunki*

lub:

*%x warunki*

Warunki określone są jako ciąg identyfikatorów warunków oddzielonych białymi znakami.



## Warunki początkowe

W regułach warunki podajemy w nawiasach kątowych przed wzorcem, np.:

```
<STR,CHAR>.    yymore();  
<INITIAL>.    ECHO;
```

Zapis <\*> oznacza „dla wszystkich warunków początkowych”, np. reguła domyślna mogłaby być zapisana jako:

```
<*>. | \n    ECHO;
```

Reguły bez podanych jawnie warunków początkowych są aktywne we wszystkich warunkach początkowych przy deklaracji warunków zaczynającej się „%s”, a nieaktywne przy deklaracji zaczynającej się „%x”. Ustawienie właściwego warunku początkowego odbywa się w działaniach dla reguł i polega na wywołaniu makra BEGIN z argumentem będącym nazwą warunku. Analizator leksykalny zaczyna pracę w warunku początkowym INITIAL.



## usuwanie komentarzy w języku C

```
%x KOMENTARZ /* deklaracja warunku KOMENTARZ */  
%%  
<INITIAL>"/*" BEGIN(KOMENTARZ); /* wejście w warunek */  
<INITIAL>"*/" fprintf(stderr, "Koniec komentarza bez początku\n");  
<KOMENTARZ>"*/" BEGIN(INITIAL); /* powrót */  
<KOMENTARZ>.|\n /* pomiń wnętrze komentarza */  
%%  
int yywrap(void) {  
    if (YY_START == KOMENTARZ) {  
        fprintf(stderr, "Brak zamknięcia komentarza\n");  
    }  
    return 1;  
}
```



## Rozpoznawanie napisów (uproszczone)

```
%x NAPIS /* deklaracja warunku NAPIS */
```

```
%%
```

```
<INITIAL>\\" { BEGIN(NAPIS); }
```

```
<NAPIS>\\" { BEGIN(INITIAL); strncpy(yyval, yytext, yyleng-1); return  
STRING; }
```

```
<NAPIS>. { yymore(); }
```

```
<NAPIS>\n { fprintf(stderr, "Brak końca napisu");  
BEGIN(INITIAL); }
```

Uwaga: nie musimy pisać [^"\n"], ponieważ .." wyklucza nowy wiersz, a cudzysłów rozpoznajemy wcześniej. Funkcja yymore() powoduje, że następny dopasowany tekst zostanie dopisany na koniec zmiennej yytext.

Uwaga: yytext nie zawiera cudzysłówów.

Uwaga: pełna obsługa (z np. \" w środku napisu) wymaga użycia tablicy znaków.



## Flex — definicje funkcji



W ostatniej części programu dla fleksa umieszcza się definicje funkcji. Ich prototypy powinny być umieszczone w części deklaracji. Można używać ich w części regułowej.

Szczególne znaczenie ma funkcja `yywrap`. Jest wywoływana po napotkaniu końca strumienia wejściowego. Funkcja ta powinna zwrócić 1. Nie trzeba umieszczać jej prototypu.



Program `bison` jest generatorem analizatorów składniowych. Na wejściu dostaje plik we własnym języku. Plik ten standardowo ma rozszerzenie „.y”. Na wyjściu tworzy analizator składniowy, a wraz z tworzonym przez programistę kodem i programem we fleksie cały kompilator, który należy następnie skompilować. Plik źródłowy bisona ma trzy części:

*deklaracje*

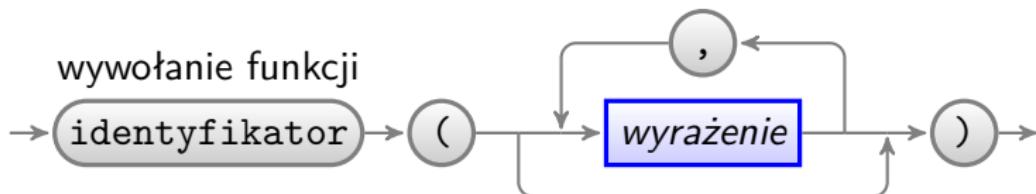
`%%` ← znaki kończące część deklaracji

*reguły*

`%%` ← znaki kończące część reguł

*definicje funkcji*

Część regułowa zawiera ciąg reguł składniowych i towarzyszących im reguł semantycznych. Reguły składniowe określają, z jakich części składa się wejście (program w języku programowania), z jakich części składają się te części itd. Np. wywołanie funkcji składa się z następujących części:



Ponieważ kompilator na wejściu ma program w postaci tekstu, a nie obrazka, reguły składniowe zapisujemy używając BNF:

```
func_call: IDENT '(' opt_act_params ')' ;
opt_act_params: %empty | act_params ;
act_params: expr | act_params ',' expr ;
```

Znaki w apostrofach dostarczane są przez analizator leksykalny. Nazwane elementy dostarczane przez analizator leksykalny wyróżniliśmy wielkimi literami. Pozostałe elementy są zmiennymi gramatyki.



## Bison — reguły

Część regułowa zawiera ciąg reguł. Reguły składniowe mają postać:

lewa\_strona: prawa\_strona;

Lewa strona jest symbolem niekońcowym (zmienną gramatyki). Prawa strona jest ciągiem symboli końcowych (dostarczanych przez analizator leksykalny) i niekońcowych. Ten sam symbol niekończowy może wystąpić po lewej stronie więcej niż raz, np.:

```
instrukcja: KW WHILE '(' warunek ')' instrukcja ;
instrukcja: IDENT '=' wyrazenie ;
```

Mogliśmy wówczas napisać:

```
instrukcja: KW WHILE '(' warunek ')' instrukcja
           | IDENT '=' wyrazenie
;
```

Jedna z alternatyw rozdzielonych kreską pionową może być pusta:

```
ciag: %empty | ciag element ;
```



Aby gramatyka bisona dała się skompilować, musi istnieć ciąg zastąpień symboli niekońcowych ciągami symboli podanymi po prawej stronie reguły prowadzący od symbolu początkowego gramatyki do ciągu symboli końcowych. Przyczyny błędów:

- ① Brak ciągu rozwinięć od symbolu początkowego dających symbol niekończowy z lewej strony reguły.
- ② Brak ciągu rozwinięć symbolu niekończowego pozwalającego zastąpić wszystkie symbole niekończowe, np. z powodu nieskończonej rekurencji.



## Typowe reguły:

- Możliwie pusty ciąg elementów ( $\text{element}^*$ ):

```
ciag: %empty
      | ciag element
      ;
;
```

- Niepusty ciąg elementów ( $\text{element}^+$ ):

```
ciag: element
      | ciag element
      ;
;
```

- Lista elementów rozdzielonych przecinkami ( $\text{element}(, \text{element})^*$ ):

```
lista: element
      | lista ',' element
      ;
;
```

- Możliwie pusta lista j.w. ( $\varepsilon | \text{element}(, \text{element})^*$ ):

```
pusta_lista: %empty | lista
      ;
;
```



Do reguł składniowych lub ich wariantów można dołączyć reguły semantyczne:

- kod w języku C w nawiasach klamrowych
- $\$1, \$2, \dots$  — wartości pierwszego, drugiego itd. elementu po prawej stronie reguły
- $\$\$$  — obliczana wartość rozwijanego elementu po lewej stronie reguły

Przykład:

```
expr: NUMBER { $$ = $1; }
      | '(' expr ')' { $$ = $2; }
      | expr '+' expr { $$ = $1 + $3; }
      | expr '*' expr { $$ = $1 * $3; }
      ...
}
```

Brak reguły semantycznej oznacza regułę domyślną:  
 $\{ \quad \$\$ = \$1; \quad \}$



**Obsługa błędów** umożliwia wykrywanie błędów, sygnalizowanie ich i dalszą analizę wejścia od punktu, w którym wpływ błędu na dalszy tekst powinien być znacznie mniejszy. Odbywa się to przez standardowo zdefiniowany wirtualny (nie jest dostarczany przez analizator leksykalny) symbol końcowy `error`. Przykład:

```
wiersze: %empty
| wiersze '\n'
| wiersze error '\n'
| wiersze wyr '\n'
```

Kiedy nastąpi błąd składniowy, analizator składniowy otrzyma na wejściu symbol `error`. Będzie próbował go dopasować, usuwając ze stosu część związaną z przeanalizowaną częściowo zmienną `wyr` i pomijając na wejściu wszystko do następnego symbolu po symbolu `error`.



Część poświęcona deklaracjom w pliku źródłowym dla programu bison zawiera:

- deklaracje, makrodefinicje, dołączanie plików w języku C
- deklaracje bisona

Instrukcje języka C umieszczane są w taki sam sposób, jak w programie źródłowym dla fleksa, czyli:

- pomiędzy wierszami ze znakami „%{” i „%}”
- w wierszach zaczynających się białymi znakami (odstępami lub znakami tabulacji) — dotyczy to także komentarzy

Deklaracje bisona zaczynają się od pierwszej kolumny wierszy, które nie są umieszczone pomiędzy wierszami ze znakami „%{” i „%}”.



Domyślnie typem zmiennej `yyvalue` fleksa, w której jest przekazywana wartość rozpoznanych elementów, jest `int`. Można to zmienić, np. w celu ustalenia tego typu na `double` można podać w deklaracjach w języku C:

```
#define YYSTYPE double
```

Jeżeli typem jest unia, to można zadeklarować ją:

## z poziomu języka C

```
typedef union {
    int      i;
    double   d;
    char    *s;
} YYSTYPE;
```

## w deklaracjach bisona

```
%union {
    int      i;
    double   d;
    char    *s;
}
```



## Bison — deklaracje

Elementy końcowe gramatyki rozpoznawane przez fleksa deklaruje się w bisonie za pomocą dyrektywy `%token`, której argumentem jest ciąg nazw oddzielanych białymi znakami, np:

```
%token KW_FOR KW_WHILE KW_IF KW_SWITCH
```

Jeśli typ wartości elementów rozpoznawanych przez fleksa (typ YYSTYPE zmiennej `yylval`) jest unią, np:

```
%union {  
    int      i;  
    double   d;  
    char    *s;  
}
```

to nazwy typów elementów należy podać po dyrektywie `%token` w nawiasach kątowych:

```
%token<i> KW_IF KW_WHILE KW_SWITCH INT_CONST  
%token<d> REAL_CONST  
%token<s> IDENT STRING_CONST
```



## Bison — deklaracje

Symbole niekońcowe deklaruje się za pomocą dyrektywy `%type`. Deklaracja jest potrzebna tylko wtedy, gdy symbole mają różne typy, np. deklarowane za pomocą dyrektywy `%union`. Przykład:

```
%type<i> expr
%type<s> proc_head
```

Można też zamiast tych deklaracji i używania typów w `%token` podawać jawnie typy w regułach semantycznych umieszczając identyfikator typu w nawiasach kątowych po symbolu dolara, np:

```
proc_head: KW PROCEDURE IDENT '(' formal_params ')' {
    strcpy($<s>$, $<s>2);
}
```

Domyślnie typ `$$` jest taki sam jak `$1`.



**Priorytet operatorów** rozstrzyga sytuacje, w których istniejące reguły prowadzą do niejednoznaczności, np.:

## wyrażenia arytmetyczne

```
expr: expr '+' expr
      { $$ = $1 + $3; }
    | expr '*' expr
      { $$ = $1 * $3; }
...
;
2 + 2 * 2
(2 + 2) * 2 czy 2 + (2 * 2)?
```

## wyrażenia regularne

```
re: re re
   { $$ = caten($1,$2); }
 | re '|'|' re
   { $$ = altern($1,$3); }
...
a|bb
(a|b)b czy a|(bb)?
```

Priorytet operatorów jest rozstrzygany przez kolejność dyrektyw `%left`, `%right`, `%nonassoc` i `%precedence`.



# Bison — deklaracje

**Priorytet operatorów** rośnie dla kolejnych tekstowo deklaracji %left, %right i %nonassoc. Priorytet jest taki sam dla operatorów podanych w tym samym wierszu. Dodatkowo dla ciągów operatorów o równym priorytecie deklarowanych za pomocą:

- ① %left — operatory wiążą lewostronnie, tzn.:  
 $x \ op \ y \ op \ z = (x \ op \ y) \ op \ z$
- ② %right — operatory wiążą prawostronnie, tzn.:  
 $x \ op \ y \ op \ z = x \ op (y \ op \ z)$
- ③ %nonassoc — operatory nie mogą wystąpić w łańcuchu, tzn.:  
 ~~$x \ op \ y \ op \ z$~~  jest zabronione.
- ④ %precedence — brak sympatii w łączeniu

## Przykład

```
%left '<' '>' '=' LE NE GE      ← najniższy priorytet  
%left '+' '-'                   ← średni priorytet  
%left '*' '/'                  ← najwyższy priorytet
```



**Priorytet** konstrukcji składniowej można zmienić nadając jej priorytet innego operatora za pomocą frazy %prec.

## Przykład (z podręcznika bisona)

```
%token NUM
%left '-' '+'
%left '*' '/'
%precedence NEG /* minus jednoargumentowy */
.
.
.
exp: NUM { $$ = $1; }
| exp '+' exp { $$ = $1 + $3; }
| exp '-' exp { $$ = $1 - $3; }
| exp '*' exp { $$ = $1 * $3; }
| exp '/' exp { $$ = $1 / $3; }
| '-' exp %prec NEG { $$ = -$2; }
| '(' exp ')' { $$ = $2; }
;
```



Domyślnie symbolem początkowym gramatyki jest symbol niekońcowy występujący po lewej stronie pierwszej reguły programu. Można to zmienić podając jawnie symbol początkowy, np.:

```
%start PROGRAM
```



W nowszych wersjach programu bison miejsce w pliku wejściowym przechowywane jest w zmiennej typu YYLTYPE zdefiniowanej następująco:

```
typedef struct YYLTYPE
{
    int first_line;
    int first_column;
    int last_line;
    int last_column;
} YYLTYPE;
```

Użytkownik może zmienić tę definicję. Pola typu YYLTYPE są dostępne jako pola wartości elementów reguł, np. `$1.first_line`.



## Łączenie bisona z fleksem

- dyrektywy %token powodują nadanie ich argumentom (symbolom końcowym) wartości liczbowych (numerów, kodów) za pomocą dyrektyw #define w pliku z rozszerzeniem tab.h
- dyrektywa %union powoduje umieszczenie dyrektywy #define dla symbolu YYSTYPE w pliku z rozszerzeniem tab.h; symbol YYSTYPE określa typy wartości symboli końcowych zwracanych przez flex
- plik z rozszerzeniem tab.h jest włączany w programie fleksa (za pomocą dyrektywy #include)
- flex informuje o rozpoznaniu symbolu przez zwrócenie jego numeru
- pojedyncze znaki funkcjonujące jako symbole końcowe nie muszą być nazywane; flex zwraca sam znak (jego kod), a w bisonie znak w jako symbol końcowy podawany jest w apostrofach
- jeśli symbole mają wartości, przypisuje się je we fleksie zmiennej yyval (lub jej polom w przypadku użycia dyrektywy %union)
- wartości symboli końcowych dostępne są jako \$n w regułach semantycznych, gdzie n to numer symbolu po prawej stronie reguły składniowej



# Łączenie bisona z fleksem

## Przykładowy plik program.y

```
...
%token NUMBER
...
expr: NUMBER { $$ = $1; }
| expr '+' expr { $$ = $1 + $3; }
;
;
```

## program.tab.h

```
#define NUMBER 131
```

## Przykładowy plik program.l

```
%{
#include "program.tab.h"
%}
%%
[ \n\t]    ;
[0-9]+   { yyval = atoi(yytext); return NUMBER; }
.     return yytext[0];
```



# Łączenie bisona z fleksem

## Przykładowy plik program.y

```
...
%union { int i; double d; char *s; }
%token<i> NUMBER
%type<i> expr
...
expr: NUMBER { $$ = $1; }
| expr '+' expr { $$ = $1 + $3; }
;
```

## Przykładowy plik program.l

```
%{
#include "program.tab.h"
%}
%%
[0-9]+ { yyval.i = atoi(yytext); return NUMBER; }
```



**Gramatyką** nazywamy czwórkę uporządkowaną  $(V, \Sigma, P, S)$ , gdzie:

- $V$  to zbiór **zmiennych** (inaczej — symboli niekońcowych)
- $\Sigma$  (oznaczany też jako  $T$ ) to zbiór **symboli końcowych** (ang. *terminals*)
- $P$  to zbiór **reguł produkcji** postaci  $\alpha \rightarrow \beta$
- $S$  to symbol wyróżniony — **symbol początkowy** gramatyki

Konkretna postać reguł produkcji zależy od typu gramatyki.

**Język**  $L$  nad alfabetem  $\Sigma$  jest to zbiór ciągów symboli z tego alfabetu, tzn.  $L \subseteq \Sigma^*$ . Na przykład zbiór wszystkich nieujemnych liczb dwójkowych jest językiem nad alfabetem  $\{0, 1\}$ . Każdy jego podzbiór też jest takim językiem.



# Hierarchia Chomsky'ego



typ gramatyki	typ automatu	pamięć
$G^0$ – swobodna	maszyna Turinga	dostęp swobodny
$G^1$ – kontekstowo zależna	liniowo ograniczony	liniowo ograniczona
$G^2$ – bezkontekstowa	stosowy	stosowa
$G^3$ – regularna	skończony	brak



**Gramatyką regularną** nazywamy gramatykę  $(V, \Sigma, P, Z)$ , w której reguły produkcji mają jedną z dwóch postaci:

① dla  $A, B \in V, a \in \Sigma$

- ▶  $A \rightarrow a$
- ▶  $A \rightarrow Ba$
- ▶  $A \rightarrow \varepsilon$

② dla  $A, B \in V, a \in \Sigma$

- ▶  $A \rightarrow a$
- ▶  $A \rightarrow aB$
- ▶  $A \rightarrow \varepsilon$

W pierwszym przypadku mówimy o **gramatyce regularnej lewostronnej**, w drugim — o **gramatyce regularnej prawostronnej**.



**Automaty skończone** nazywamy skończonymi, ponieważ mają skońzoną liczbę stanów. Istnieją również automaty nieskończone.

Automaty dzielimy na:

- deterministyczne (DFA)
- niedeterministyczne (NFA)
- niedeterministyczne z przejściami etykietowanymi  $\varepsilon$  ( $\varepsilon$ -NFA)

Ze względu na obecność wyjścia automaty dzielimy na:

- bez wyjścia (zwykłe automaty rozpoznające język)
- z wyjściem:
  - ▶ automaty Moore'a (wyjście w stanach)
  - ▶ automaty Mealy'ego (wyjście na przejściach)

Domyślnie mówiąc/pisząc automat mamy na myśli skończony automat deterministyczny bez wyjścia.



**Deterministycznym automatem skończonym** nazywamy piątkę uporządkowaną  $M = (Q, \Sigma, \delta, q_0, F)$ , gdzie

- $Q$  to skończony zbiór **stanów**
- $\Sigma$  to skończony zbiór symboli zwany **alfabetem**
- $\delta : Q \times \Sigma \rightarrow Q$  to **funkcja przejścia**
- $q_0 \in Q$  to stan **początkowy**
- $F \subseteq Q$  to zbiór stanów **końcowych**

Rozszerzoną funkcję przejścia  $\delta^* : Q \times \Sigma^* \rightarrow Q$  definiujemy następująco:

$$\delta^*(q, \varepsilon) = q, \quad q \in Q$$

$$\delta^*(q, \sigma w) = \delta^*(\delta(q, \sigma), w), \quad q \in Q, w \in \Sigma^*, \sigma \in \Sigma$$

Językiem  $\mathcal{L}(M)$  automatu  $M$  nazywamy:

$$\mathcal{L}(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}$$

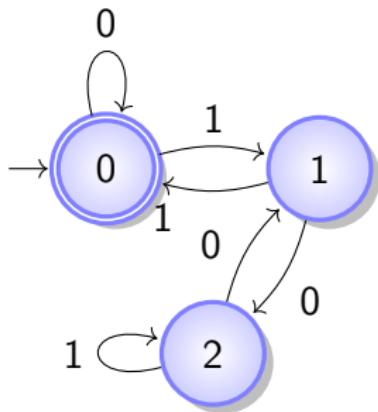
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

$$0 \bmod 3 = 0$$

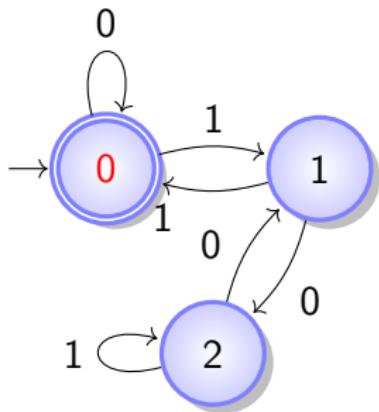
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

$$0 \bmod 3 = 0$$

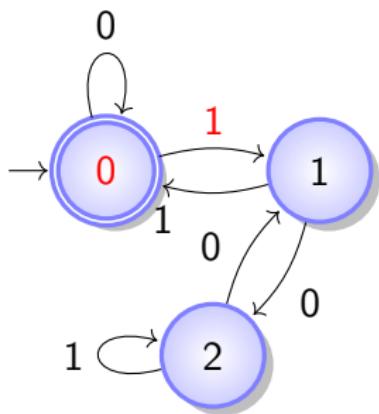
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

$$1 \bmod 3 = 1$$

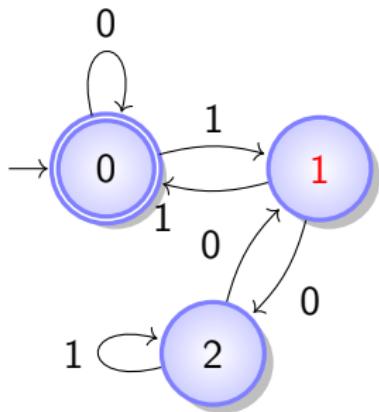
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

$$1 \bmod 3 = 1$$

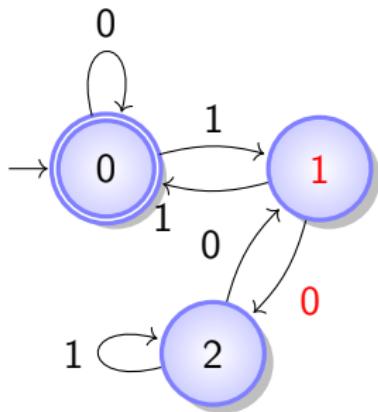
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$\begin{array}{ccccccccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 2 \text{ mod } 3 = 2 & & & & & & & & & \end{array}$$

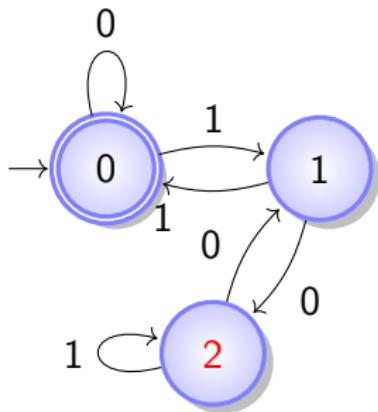
## Przykładowy automat

$$M = (\{0, 1, \textcolor{red}{2}\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

$$2 \bmod 3 = 2$$

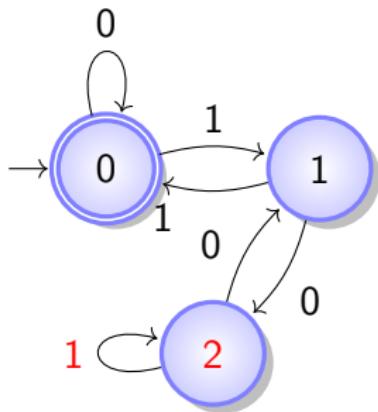
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

$$5 \bmod 3 = 2$$

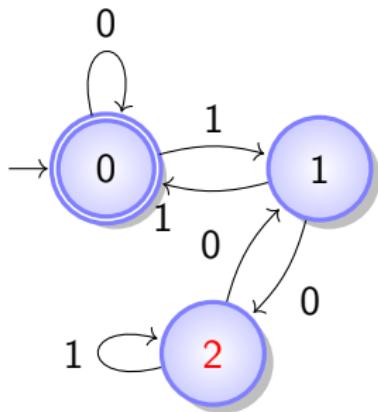
## Przykładowy automat

$$M = (\{0, 1, \textcolor{red}{2}\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

$$5 \bmod 3 = 2$$

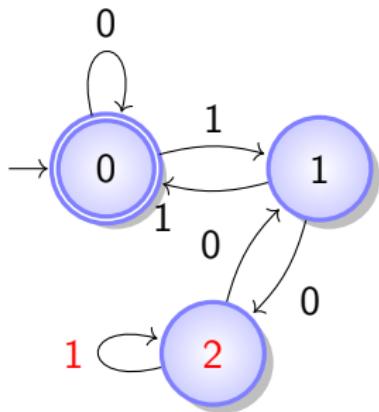
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$\begin{array}{ccccccccc} 1 & 0 & 1 & \textcolor{red}{1} & 1 & 0 & 0 & 1 & 0 & 1 \\ 11 \text{ mod } 3 = 2 \end{array}$$

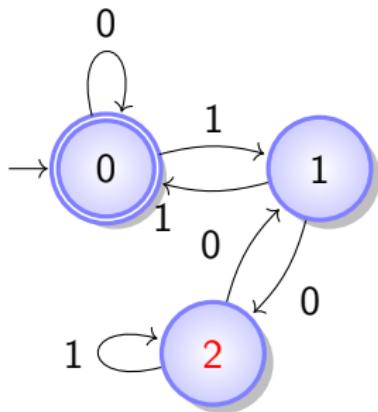
## Przykładowy automat

$$M = (\{0, 1, \textcolor{red}{2}\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 11 \bmod 3 = 2$$

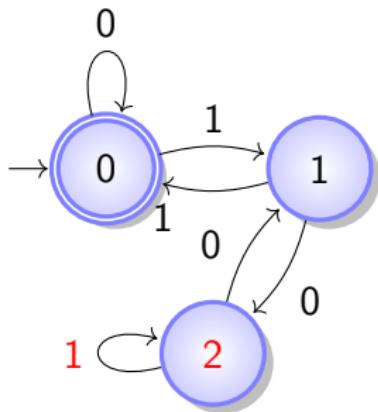
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 23 \bmod 3 = 2$$

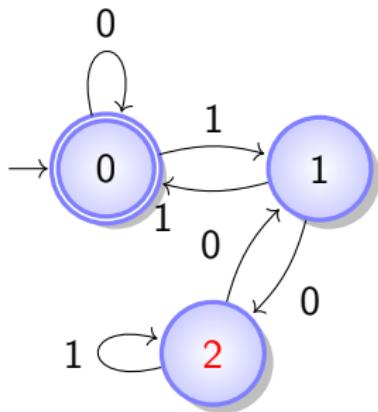
## Przykładowy automat

$$M = (\{0, 1, \textcolor{red}{2}\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 23 \ \mathbf{mod} \ 3 = 2$$

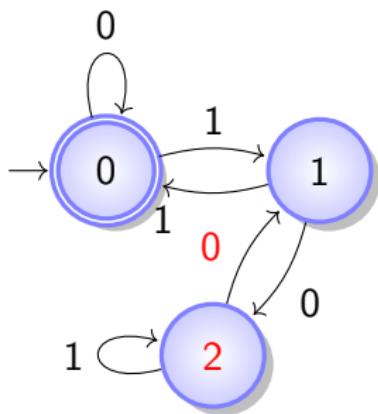
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 46 \bmod 3 = 1$$

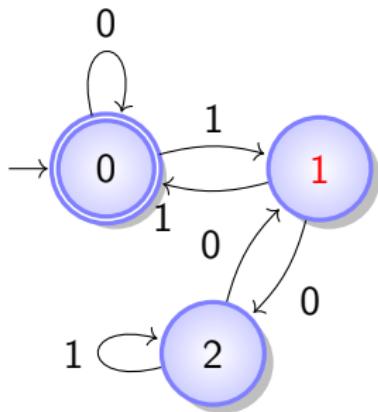
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 46 \text{ mod } 3 = 1$$

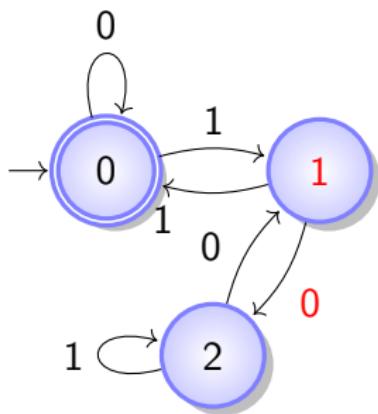
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 92 \text{ mod } 3 = 2$$

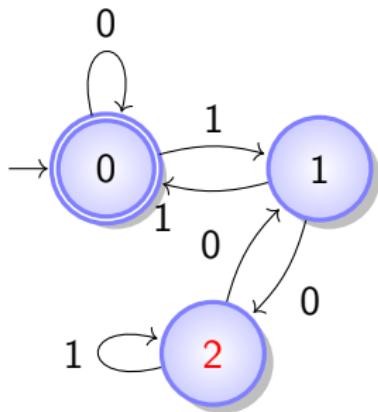
## Przykładowy automat

$$M = (\{0, 1, \textcolor{red}{2}\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 92 \bmod 3 = 2$$

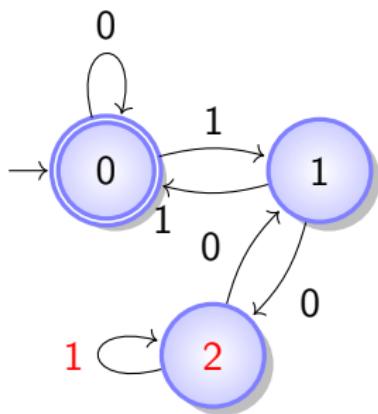
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 185 \ \text{mod} \ 3 = 2$$

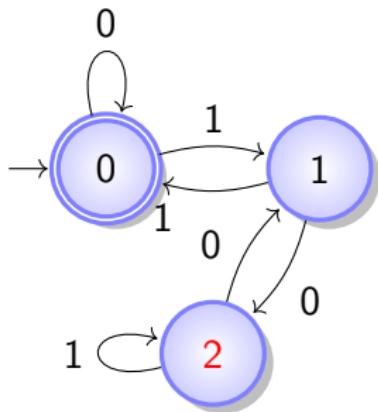
## Przykładowy automat

$$M = (\{0, 1, \textcolor{red}{2}\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 185 \ \mathbf{mod} \ 3 = 2$$

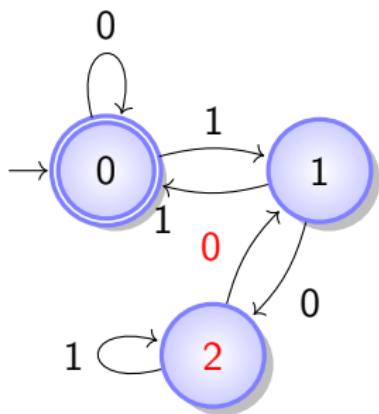
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ \textcolor{red}{0} \ 1 \\ 370 \ \mathbf{mod} \ 3 = 1$$

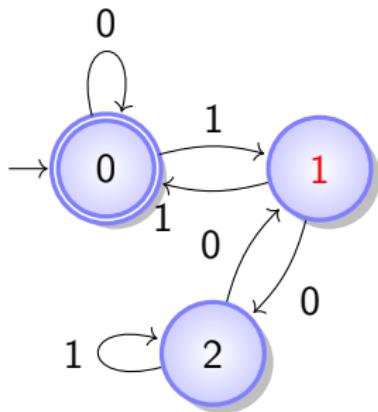
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 370 \ \mathbf{mod} \ 3 = 1$$

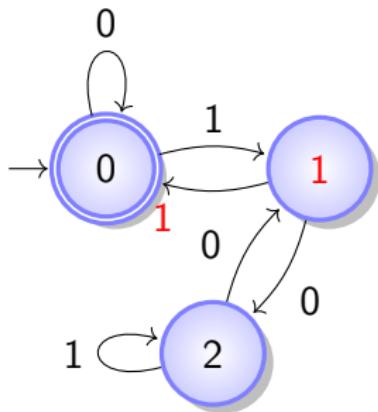
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 741 \ \text{mod} \ 3 = 0$$

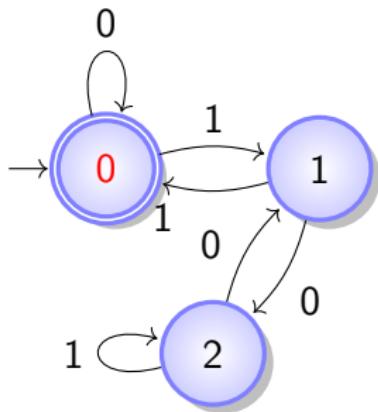
## Przykładowy automat

$$M = (\{0, 1, 2\}, \{0, 1\}, \delta, 0, \{0\})$$

$$\delta(0, 0) = 0, \delta(0, 1) = 1,$$

$$\delta(1, 0) = 2, \delta(1, 1) = 0,$$

$$\delta(2, 0) = 1, \delta(2, 1) = 2.$$



## Tabela przejść

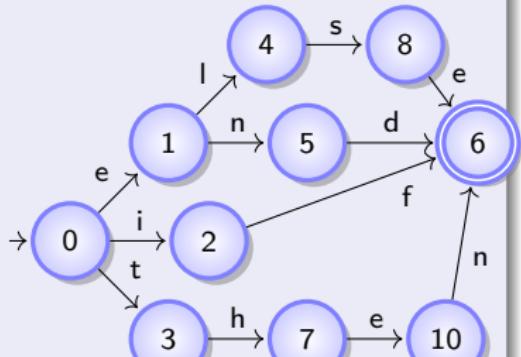
stan	wejście	
	0	1
0	0	1
1	2	0
2	1	2

## Działanie

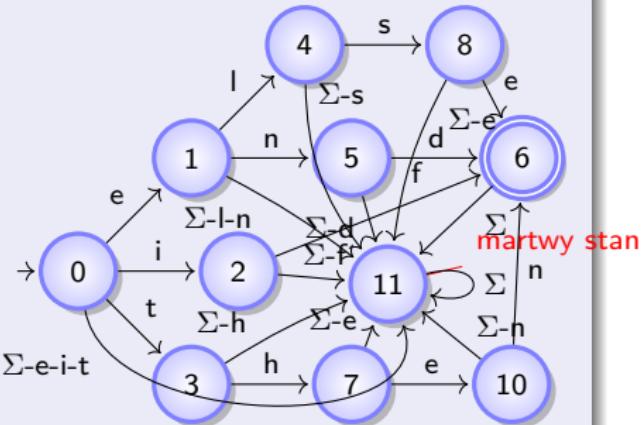
Prześledźmy działanie dla przykładowego ciągu wejściowego:

$$1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 741 \ \text{mod} \ 3 = 0$$

## automat niekompletny



## automat kompletny



Zapis  $\Sigma$ -e-i-t na przejściu oznacza wiele przejść etykietowanych wszystkimi symbolami alfabetu poza e, i oraz t.

- Automaty **kompletne** opisują sprzęt; wymagane są w niektórych algorytmach.
- Automaty **niekompletne** wykorzystywane są głównie w strukturach danych w programach.



# Automaty niedeterministyczne



**Automatem skończonym niedeterministycznym** nazywamy piątkę uporządkowaną  $M = (Q, \Sigma, \delta, q_0, F)$ , gdzie

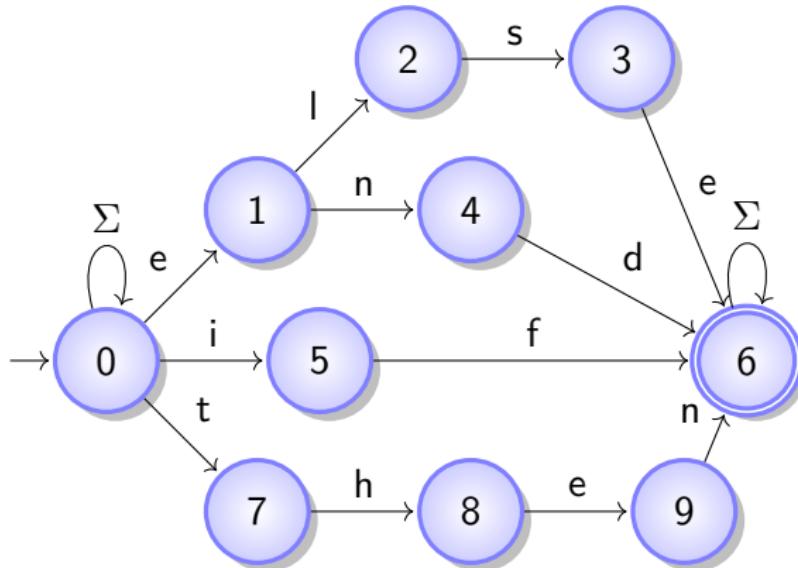
- $Q$  to skończony zbiór **stanów**
- $\Sigma$  to skończony zbiór symboli zwany **alfabetem**
- $\delta : Q \times \Sigma \rightarrow 2^Q$  to **funkcja przejścia**
- $q_0 \in Q$  to stan początkowy
- $F \subseteq Q$  to zbiór stanów końcowych

Rozszerzoną funkcję przejścia  $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$  definiujemy następująco:

$$\begin{aligned}\delta^*(q, \varepsilon) &= \{q\} & q \in Q \\ \delta^*(q, w\sigma) &= \bigcup_{p \in \delta^*(q, w)} \delta(p, \sigma) & q \in Q, \sigma \in \Sigma, w \in \Sigma^*\end{aligned}$$

Językiem  $\mathcal{L}(M)$  automatu  $M$  nazywamy:

$$\mathcal{L}(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \cap F \neq \emptyset\}$$



Automat rozpoznający teksty zawierające słowa kluczowe else, end, if i then.



## Równoważność DFA i NFA



Dla każdego skońzonego automatu deterministycznego  $M_D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D)$  istnieje równoważny automat niedeterministyczny  $M_N = (Q_N, \Sigma, \delta_N, q_{0N}, F_N)$  rozpoznający ten sam język. Otrzymujemy go zamieniając przejścia  $\delta(q, \sigma) = p$  na  $\delta(q, \sigma) = \{p\}$ .

Dla każdego skońzonego automatu niedeterministycznego istnieje równoważny automat deterministyczny rozpoznający ten sam język. Procedura otrzymywania takiego automatu deterministycznego nazywa się **determinizacją**. Automat deterministyczny może mieć wykładniczo więcej stanów niż niedeterministyczny.



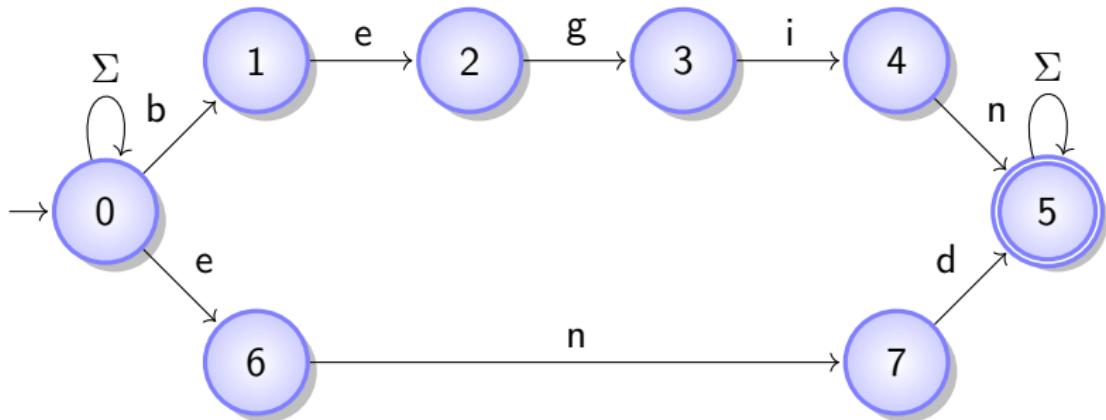
# Determinizacja

Oznaczmy:

$$\delta_D(S, \sigma) = \bigcup_{q \in S} \delta_N(q, \sigma)$$

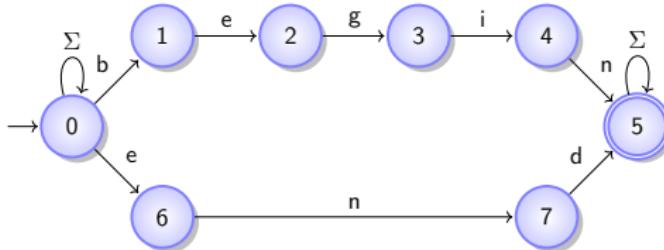
```
1: procedure DETERMINIZACJA( $M_N = (Q_N, \Sigma, \delta_N, q_{0N}, F_N)$ )
2:    $q_{0D} \leftarrow \{q_{0N}\}; Q_D \leftarrow \{q_{0D}\}$ ; wstaw  $q_{0D}$  do kolejki;
3:   while kolejka nie jest pusta do
4:      $q_D \leftarrow$  kolejny zbiór stanów z kolejki
5:     for  $\sigma \in \Sigma$  do
6:        $S \leftarrow \delta_D(q_D, \sigma)$ 
7:       if  $S \notin Q_D$  then
8:          $Q_D \leftarrow Q_D \cup \{S\}$ ; wstaw  $S$  do kolejki
9:       end if
10:      end for
11:    end while
12:     $M_D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D = \{q_d \in Q_D : \exists_{q_N \in q_D} q_N \in F_N\})$ 
13: end procedure
```

Przykład:



Automat rozpoznający teksty zawierające słowa kluczowe begin i end.

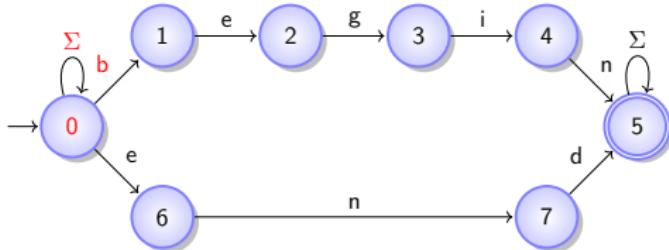
# Determinizacja



$\{0\}$

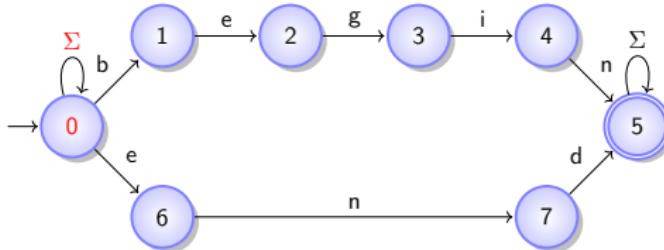
b      d      e      g      i      n       $\sigma$

# Determinizacja



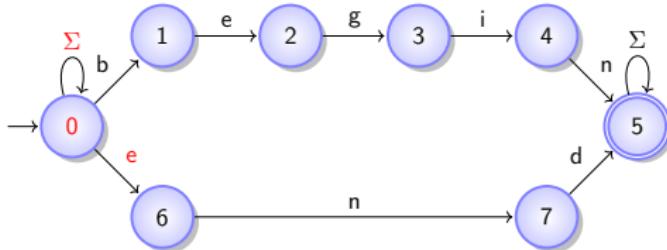
	$\{0\}$	$b$	$d$	$e$	$g$	$i$	$n$	$\sigma$
$\{0\}$								
$\{0, 1\}$								

# Determinizacja



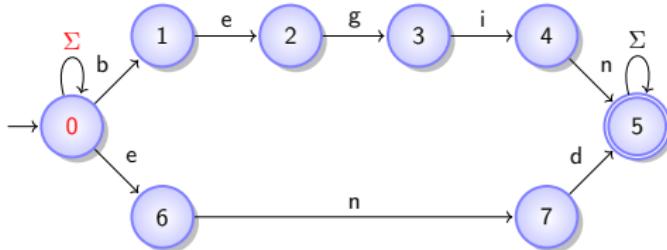
	$\{0\}$	$b$	$d$	$e$	$g$	$i$	$n$	$\sigma$
	$\{0\}$	$\{0, 1\}$	$\{0, 1\}$	$\{0\}$				

# Determinizacja

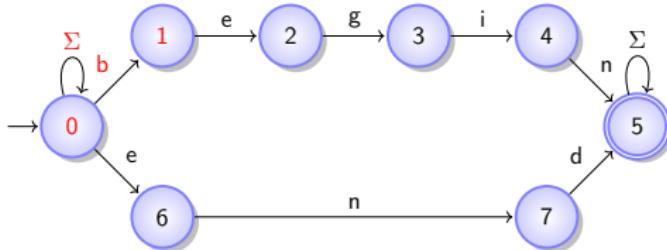


	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}						
{0, 1}							
{0, 6}							

# Determinizacja

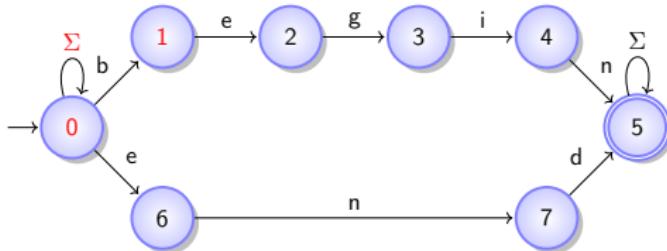


	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}						
{0, 1}							
{0, 6}							

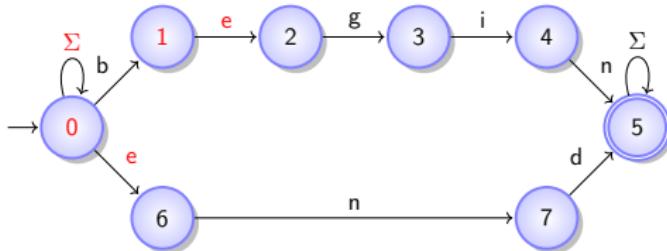


	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}						
{0, 1}		{0, 1}					
{0, 6}			{0, 6}				

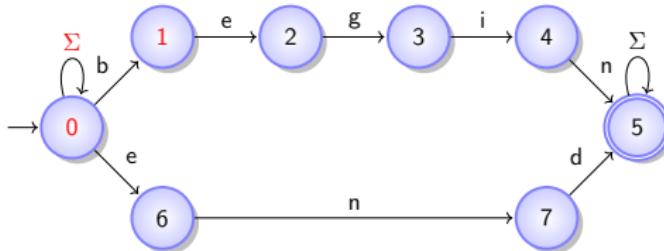
# Determinizacja



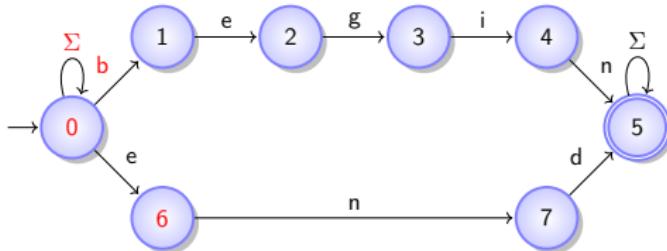
	$b$	$d$	$e$	$g$	$i$	$n$	$\sigma$
{0}	{0, 1}						
{0, 1}		{0, 1}					
{0, 6}		{0}	{0, 6}	{0}	{0}	{0}	{0}



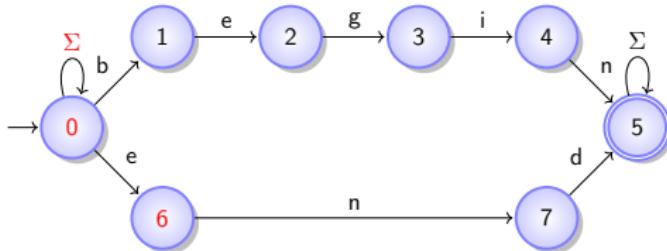
	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}						
{0, 1}	{0, 1}	{0}					
{0, 6}			{0, 6}				
{0, 2, 6}			{0, 2, 6}				



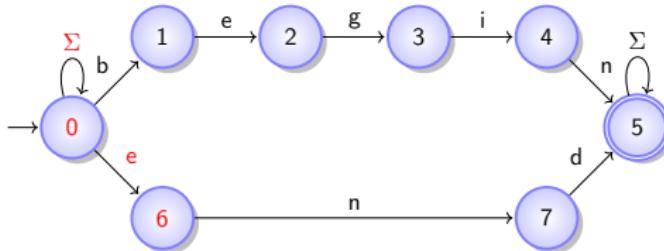
	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}						{0}
{0, 1}	{0, 1}	{0}					{0}
{0, 6}			{0, 6}				{0}
{0, 2, 6}			{0, 2, 6}				{0}



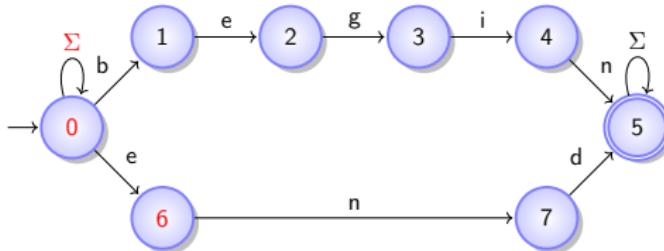
	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0}	{0}
{0, 1}	{0, 1}	{0}	{0, 2, 6}	{0}	{0}	{0}	{0}
{0, 6}	{0, 1}						
{0, 2, 6}							



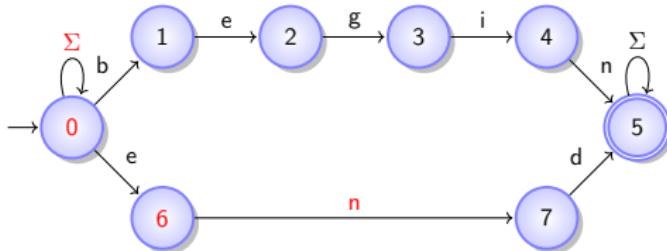
	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0}	{0}
{0, 1}	{0, 1}	{0}	{0, 2, 6}	{0}	{0}	{0}	{0}
{0, 6}	{0, 1}	{0}					
{0, 2, 6}							



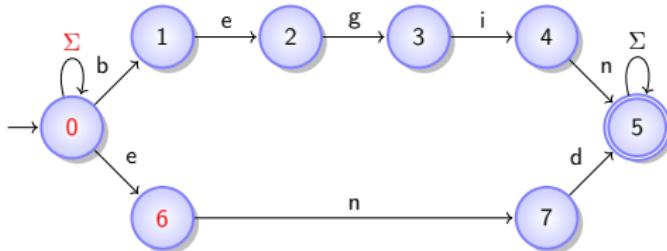
	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0}	{0}
{0, 1}	{0, 1}	{0}	{0, 2, 6}	{0}	{0}	{0}	{0}
{0, 6}	{0, 1}	{0}	{0, 6}				
{0, 2, 6}							



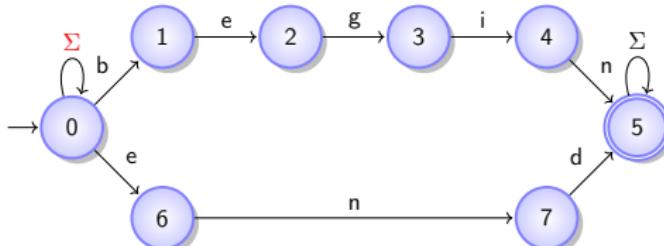
	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0}	{0}
{0, 1}	{0, 1}	{0}	{0, 2, 6}	{0}	{0}	{0}	{0}
{0, 6}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0}	{0}
{0, 2, 6}							



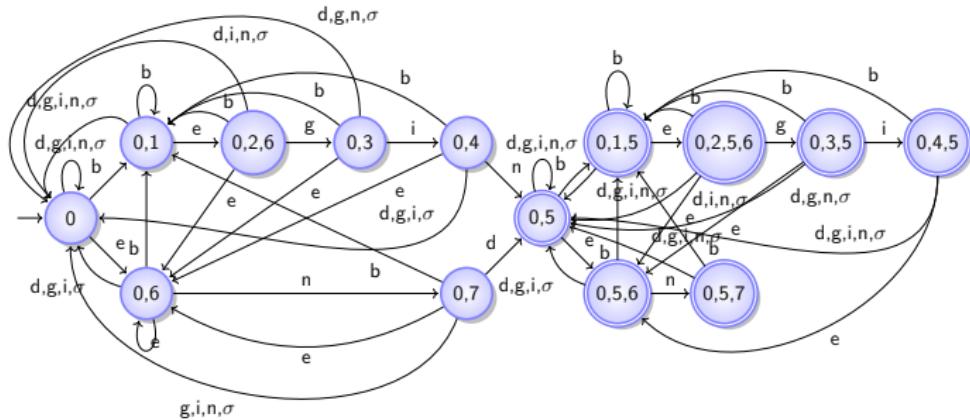
	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0}	{0}
{0, 1}	{0, 1}	{0}	{0, 2, 6}	{0}	{0}	{0}	{0}
{0, 6}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0, 7}	
{0, 2, 6}							
{0, 7}							



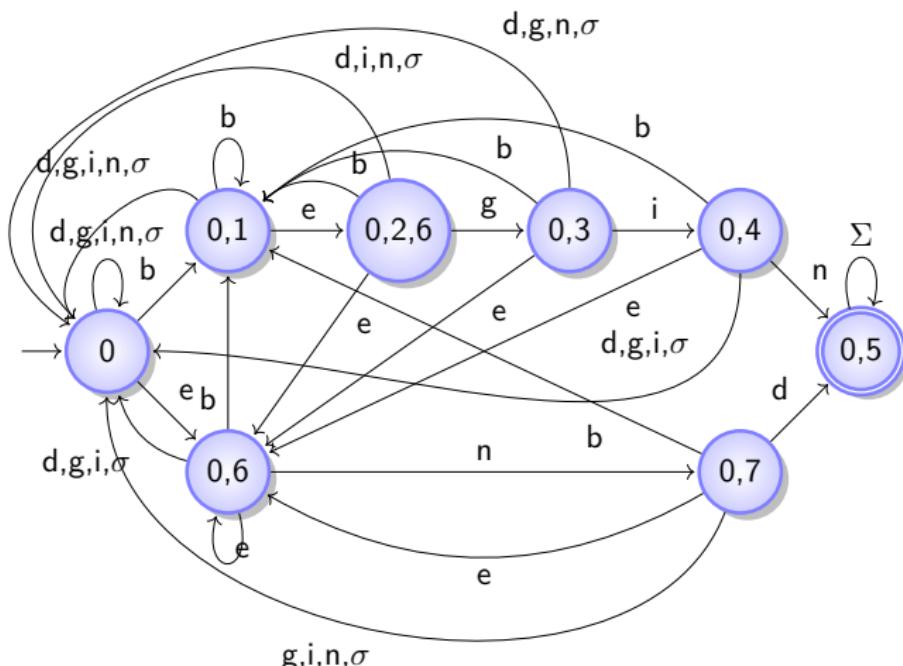
	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0}	{0}
{0, 1}	{0, 1}	{0}	{0, 2, 6}	{0}	{0}	{0}	{0}
{0, 6}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0, 7}	{0}
{0, 2, 6}							
{0, 7}							



	b	d	e	g	i	n	$\sigma$
{0}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0}	{0}
{0, 1}	{0, 1}	{0}	{0, 2, 6}	{0}	{0}	{0}	{0}
{0, 6}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0, 7}	{0}
{0, 2, 6}	{0, 1}	{0}	{0, 6}	{0, 3}	{0}	{0}	{0}
{0, 7}	{0, 1}	{0, 5}	{0, 6}	{0}	{0}	{0}	{0}
{0, 3}	{0, 1}	{0}	{0, 6}	{0}	{0, 4}	{0}	{0}
{0, 5}	{0, 1, 5}	{0, 5}	{0, 5, 6}	{0, 5}	{0, 5}	{0, 5}	{0, 5}
{0, 4}	{0, 1}	{0}	{0, 6}	{0}	{0}	{0, 5}	{0}
{0, 1, 5}	{0, 1, 5}	{0, 5}	{0, 2, 5, 6}	{0, 5}	{0, 5}	{0, 5}	{0, 5}
{0, 5, 6}	{0, 1, 5}	{0, 5}	{0, 5, 6}	{0, 5}	{0, 5}	{0, 5, 7}	{0, 5}
{0, 2, 5, 6}	{0, 1, 5}	{0, 5}	{0, 5, 6}	{0, 3, 5}	{0, 5}	{0, 5}	{0, 5}
{0, 5, 7}	{0, 1, 5}	{0, 5}	{0, 5, 6}	{0, 5}	{0, 5}	{0, 5}	{0, 5}
{0, 3, 5}	{0, 1, 5}	{0, 5}	{0, 5, 6}	{0, 5}	{0, 4, 5}	{0, 5}	{0, 5}
{0, 4, 5}	{0, 1, 5}	{0, 5}	{0, 5, 6}	{0, 5}	{0, 5}	{0, 5}	{0, 5}



Po minimalizacji:

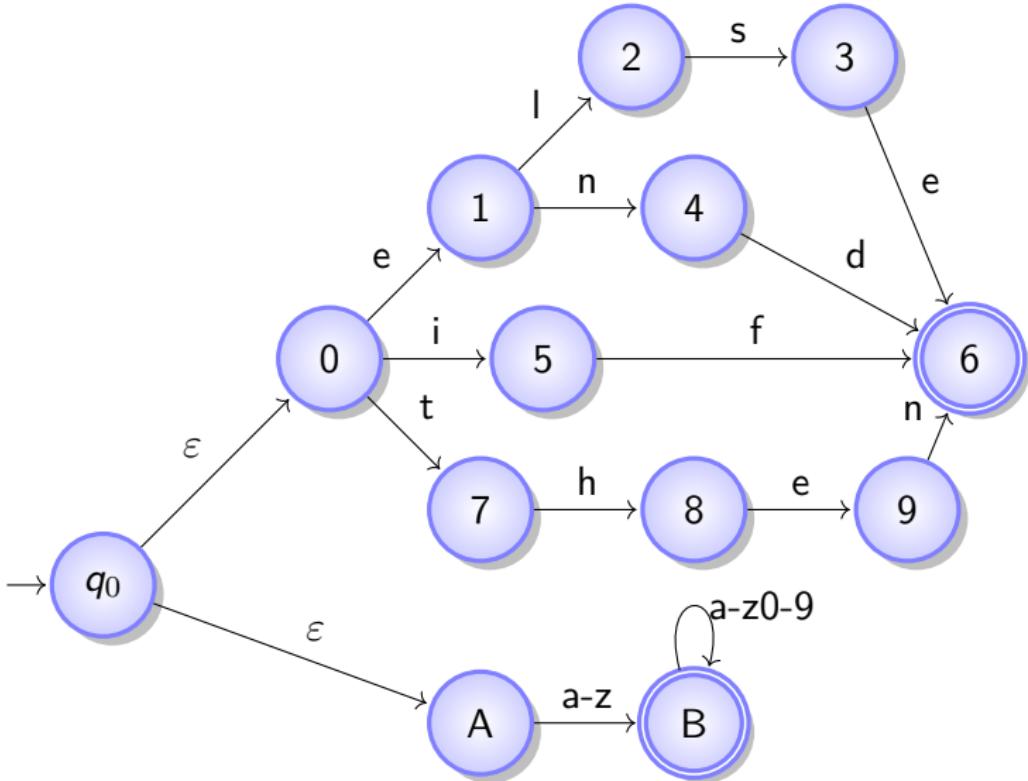




Automatem skończonym niedeterministycznym z przejściami etykietowanymi  $\varepsilon$  nazywamy piątkę uporządkowaną  $M = (Q, \Sigma, \delta, q_0, F)$ , gdzie  $Q$ ,  $\Sigma$ ,  $q_0$  i  $F$  definiowane są jak w zwykłym automacie niedeterministycznym, natomiast funkcja przejścia  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$  umożliwia zmianę stanu także bez pobrania symbolu z wejścia, jeśli między tymi stanami istnieje przejście z etykietą  $\varepsilon$ . Dla takich automatów definiujemy otoczenie- $\varepsilon$   $d_\varepsilon(q)$  stanu  $q$ :

$$\begin{aligned} q &\in d_\varepsilon(q) \\ \forall p \in d_\varepsilon(q) \delta(p, \varepsilon) &\in d_\varepsilon(q) \end{aligned}$$

Wówczas determinizacja przebiega tak jak dla NFA, ale instrukcję  $q_{0D} \leftarrow \{q_{0N}\}$  zastępujemy  $q_{0D} \leftarrow d_\varepsilon(\{q_{0N}\})$ , zaś  $S \leftarrow \delta_D(q_D, \sigma)$  instrukcją  $S \leftarrow d_\varepsilon(\delta_D(q_D, \sigma))$ .





# Minimalizacja



Jako rozmiar  $|M|$  automatu  $M$  przyjmuje się tradycyjnie liczbę jego stanów  $|Q|$ . W praktyce rozmiar automatu przechowywanego w pamięci programu zależy od liczby jego przejść.

Wśród wszystkich automatów deterministycznych rozpoznających dany język jest zawsze dokładnie jeden (z dokładnością do izomorfizmu), który ma najmniejszą liczbę stanów. Automat taki nazywamy **minimalnym**.

$$\forall M: \mathcal{L}(M) = \mathcal{L}(M_{min}) \quad |M_{min}| \leq |M|$$

Proces polegający na uzyskaniu automatu minimalnego nazywamy **minimalizacją**. Są trzy rodziny ogólnych algorytmów minimalizacji:

- ① Stany dzielimy na dwie klasy: końcowe i niekońcowe. Następnie na podstawie przejść między stanami (do których klas stanów prowadzą przejścia lub odwrotne przejścia w danej klasie stanów) dokonujemy dalszego podziału klas. Uzyskane klasy stanowią stany automatu minimalnego. Istnieją trzy takie algorytmy:
  - ① algorytm Hopcrofta o złożoności  $\mathcal{O}(|Q| \log |Q|)$
  - ② algorytm Hopcroft-Ullman o złożoności  $\mathcal{O}(|Q|^2)$
  - ③ algorytm Aho-Sethi-Ullman o złożoności  $\mathcal{O}(|Q|^2)$
- ② odwracamy kierunek przejść automatu, determinizujemy, ponownie odwracamy i ponownie determinizujemy — algorytm Brzozowskiego o złożoności wykładniczej.
- ③ porównujemy parami stany automatu, co wymaga na ogół porównywania dalszych stanów — algorytm Watsona z usprawnieniami o złożoności  $\mathcal{O}(|Q|^2 \log \log |Q|)$ .



```
1:  $\Pi \leftarrow \{Q\}$ ;  $\Pi' \leftarrow \{F, Q \setminus F\}$ 
2: while  $\Pi' \neq \Pi$  do
3:    $\Pi \leftarrow \Pi'$ 
4:   for  $S \in \Pi$  do
5:     for  $\sigma \in \Sigma$  do
6:       if przejścia z  $S$  z etykietą  $\sigma$  prowadzą do różnych zbiorów
    then
7:       Podziel  $S$  na podzbiory w podziale  $\Pi'$  wg tych zbiorów
8:       end if
9:     end for
10:   end for
11: end while
12:  $Q_{min} \leftarrow \Pi' \setminus$  stany bezużyteczne
13: Poprowadź przejścia między stanami  $Q_{min}$  jeśli istnieją takie przejścia
    między stanami w zbiorach w  $\Pi'$ 
```



```
1:  $\Pi \leftarrow \{Q\}; \Pi' \leftarrow \{F, Q \setminus F\}$            ▷ podziel zbiór stanów na zbiory
2: while  $\Pi' \neq \Pi$  do
3:    $\Pi \leftarrow \Pi'$ 
4:   for  $S \in \Pi$  do
5:     for  $\sigma \in \Sigma$  do
6:       if przejścia z  $S$  z etykietą  $\sigma$  prowadzą do różnych zbiorów
    then
7:         Podziel  $S$  na podzbiory w podziale  $\Pi'$  wg tych zbiorów
8:         end if
9:     end for
10:   end for
11: end while
12:  $Q_{min} \leftarrow \Pi' \setminus \text{stany bezużyteczne}$ 
13: Poprowadź przejścia między stanami  $Q_{min}$  jeśli istnieją takie przejścia
    między stanami w zbiorach w  $\Pi'$ 
```



```
1:  $\Pi \leftarrow \{Q\}; \Pi' \leftarrow \{F, Q \setminus F\}$ 
2: while  $\Pi' \neq \Pi$  do ▷ dopóki są nowe podziały
3:    $\Pi \leftarrow \Pi'$ 
4:   for  $S \in \Pi$  do
5:     for  $\sigma \in \Sigma$  do
6:       if przejścia z  $S$  z etykietą  $\sigma$  prowadzą do różnych zbiorów
    then
7:       Podziel  $S$  na podzbiory w podziale  $\Pi'$  wg tych zbiorów
8:       end if
9:     end for
10:   end for
11: end while
12:  $Q_{min} \leftarrow \Pi' \setminus \text{stany bezużyteczne}$ 
13: Poprowadź przejścia między stanami  $Q_{min}$  jeśli istnieją takie przejścia
    między stanami w zbiorach w  $\Pi'$ 
```



```
1:  $\Pi \leftarrow \{Q\}$ ;  $\Pi' \leftarrow \{F, Q \setminus F\}$ 
2: while  $\Pi' \neq \Pi$  do
3:    $\Pi \leftarrow \Pi'$                                  $\triangleright \Pi$  – stary podział,  $\Pi'$  – nowy podział
4:   for  $S \in \Pi$  do
5:     for  $\sigma \in \Sigma$  do
6:       if przejścia z  $S$  z etykietą  $\sigma$  prowadzą do różnych zbiorów
    then
7:       Podziel  $S$  na podzbiory w podziale  $\Pi'$  wg tych zbiorów
8:       end if
9:     end for
10:   end for
11: end while
12:  $Q_{min} \leftarrow \Pi' \setminus$  stany bezużyteczne
13: Poprowadź przejścia między stanami  $Q_{min}$  jeśli istnieją takie przejścia
    między stanami w zbiorach w  $\Pi'$ 
```

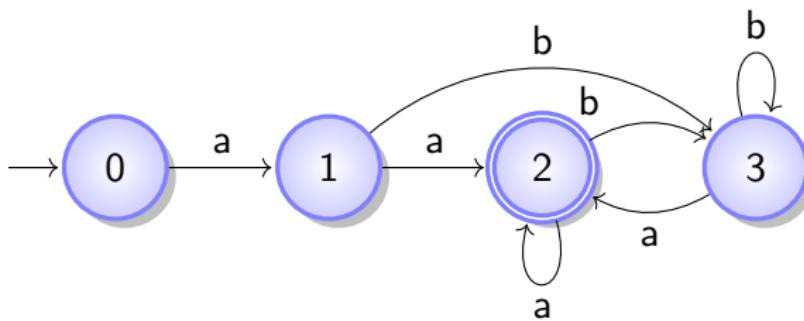


```
1:  $\Pi \leftarrow \{Q\}$ ;  $\Pi' \leftarrow \{F, Q \setminus F\}$ 
2: while  $\Pi' \neq \Pi$  do
3:    $\Pi \leftarrow \Pi'$ 
4:   for  $S \in \Pi$  do            $\triangleright$  dla wszystkich zbiorów w podziale
5:     for  $\sigma \in \Sigma$  do
6:       if przejścia z  $S$  z etykietą  $\sigma$  prowadzą do różnych zbiorów
    then
7:       Podziel  $S$  na podzbiory w podziale  $\Pi'$  wg tych zbiorów
8:       end if
9:     end for
10:   end for
11: end while
12:  $Q_{min} \leftarrow \Pi' \setminus$  stany bezużyteczne
13: Poprowadź przejścia między stanami  $Q_{min}$  jeśli istnieją takie przejścia
    między stanami w zbiorach w  $\Pi'$ 
```



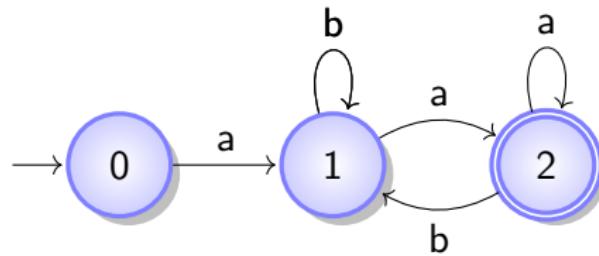
```
1:  $\Pi \leftarrow \{Q\}$ ;  $\Pi' \leftarrow \{F, Q \setminus F\}$ 
2: while  $\Pi' \neq \Pi$  do
3:    $\Pi \leftarrow \Pi'$ 
4:   for  $S \in \Pi$  do
5:     for  $\sigma \in \Sigma$  do           ▷ dla wszystkich etykiet przejść
6:       if przejścia z  $S$  z etykietą  $\sigma$  prowadzą do różnych zbiorów
7:         Podziel  $S$  na podzbiory w podziale  $\Pi'$  wg tych zbiorów
8:       end if
9:     end for
10:    end for
11:  end while
12:   $Q_{min} \leftarrow \Pi' \setminus$  stany bezużyteczne
13:  Poprowadź przejścia między stanami  $Q_{min}$  jeśli istnieją takie przejścia
    między stanami w zbiorach w  $\Pi'$ 
```

Poddajmy minimalizacji automat:

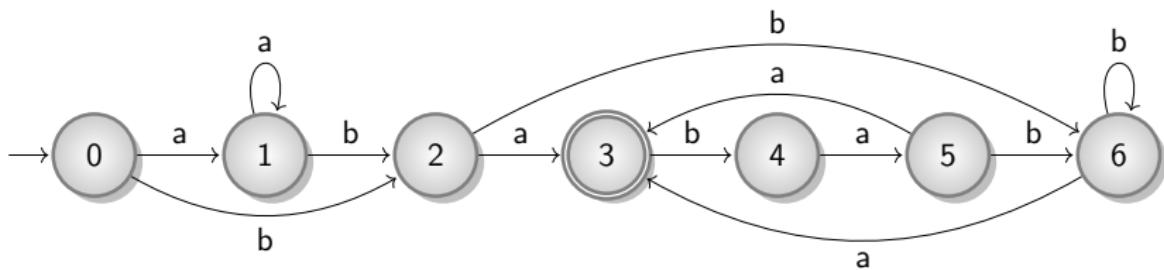


$\Pi'$	$S$	$\sigma$	podział $S$
$\{\{0, 1, 3\}, \{2\}\}$	$\{0, 1, 3\}$	a	$\{\{0\}, \{1, 3\}\}$
$\{\{0\}, \{1, 3\}, \{2\}\}$	$\{1, 3\}$	b	$\{\{1, 3\}\}$

Wynikowy minimalny automat:

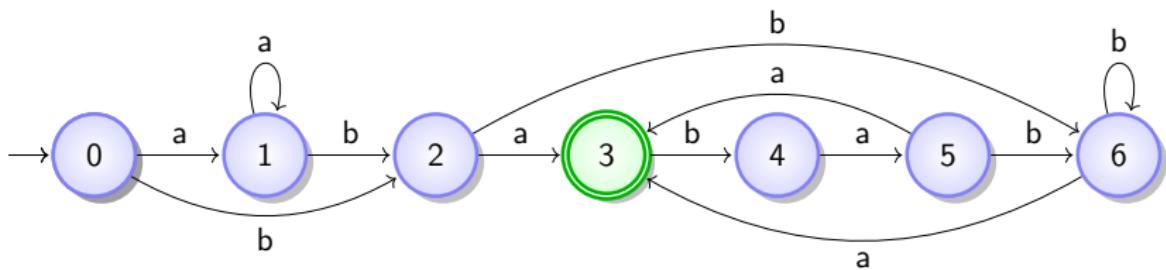


Inny przykład:



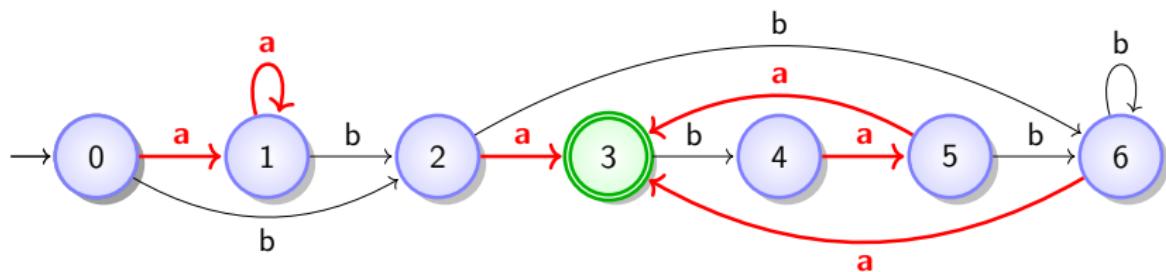
0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓

Inny przykład:



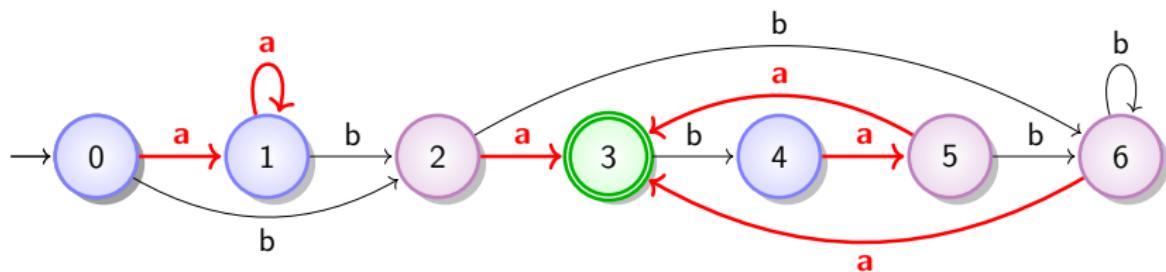
0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓

Inny przykład:



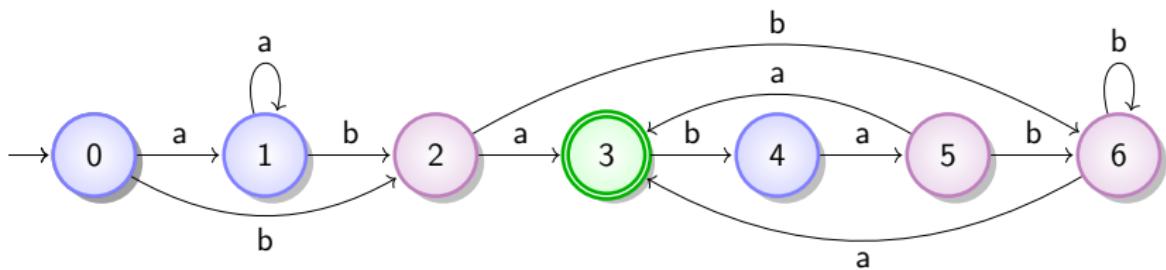
	0	1	2	3	4	5	6
a	↓	↓	↓	↓	↓	↓	↓
b	1	1	3	5	3	3	3

Inny przykład:



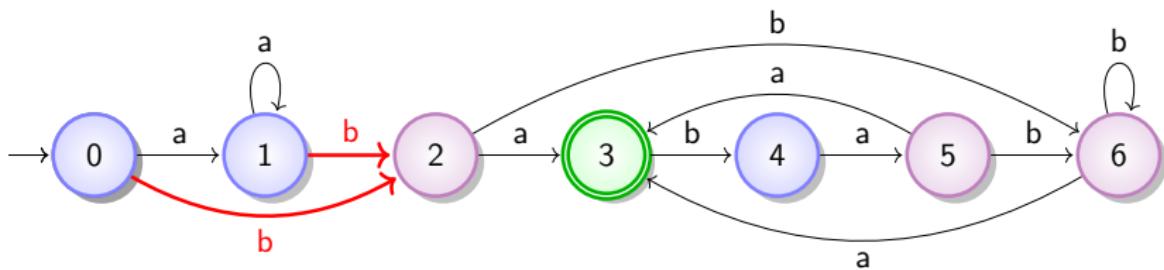
0	1	2	3	4	5	6
a	↓	↓	↓	↓	↓	↓
1	1	3		5	3	3

Inny przykład:



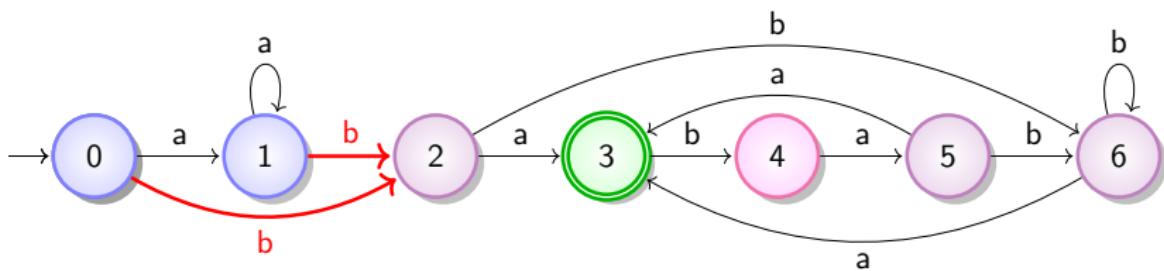
0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓

Inny przykład:



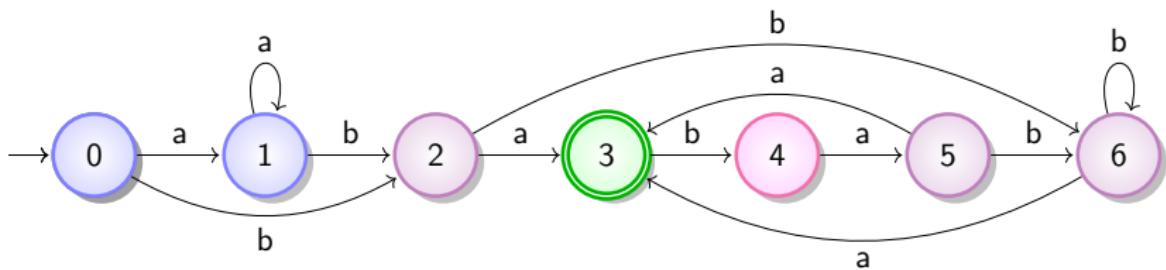
	0	1	2	3	4	5	6
b	↓	↓	↓	↓	↓	↓	↓
	2	2			∅		

Inny przykład:



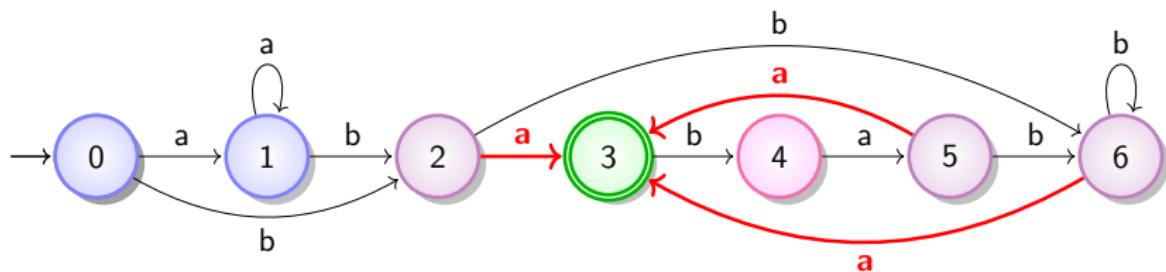
0	1	2	3	4	5	6
b	↓	↓	↓	↓	↓	↓
2	2				∅	

Inny przykład:



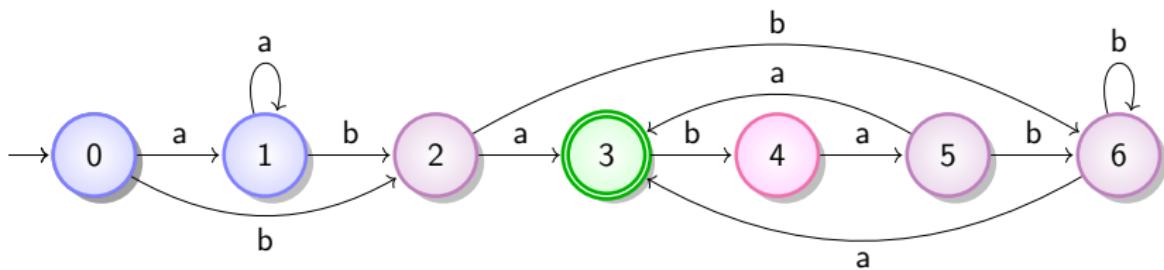
0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓

Inny przykład:



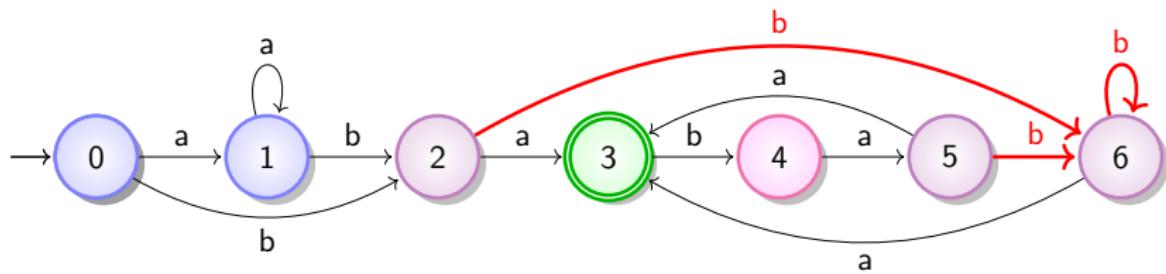
0	1	2	3	4	5	6
a	↓	↓	↓	↓	↓	↓
3	3	3	3	3	3	3

Inny przykład:



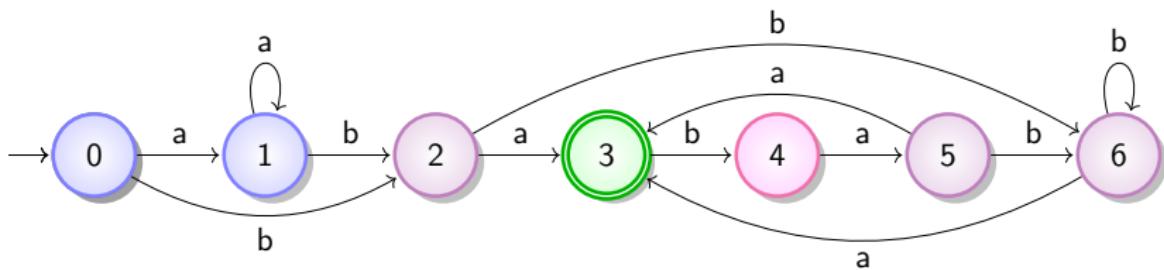
0	1	2	3	4	5	6
a	↓	↓	↓	↓	↓	↓
3	3	3	3	3	3	3

Inny przykład:



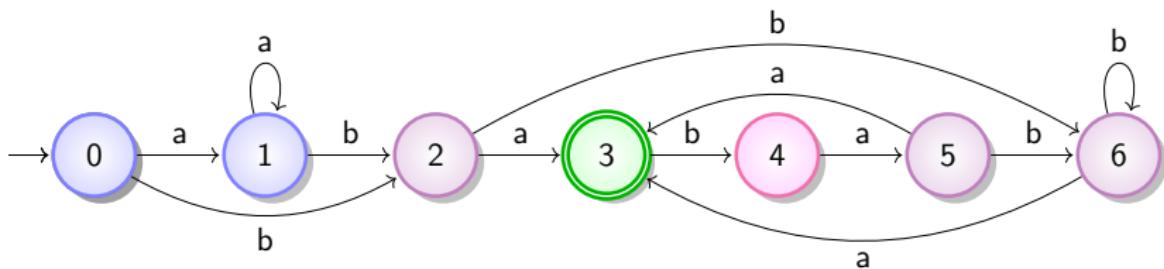
	0	1	2	3	4	5	6
b	↓	↓	↓	↓	↓	↓	↓
	6	6	6	6	6	6	6

Inny przykład:



	0	1	2	3	4	5	6
b	↓	↓	↓	↓	↓	↓	↓
	6	6	6	6	6	6	6

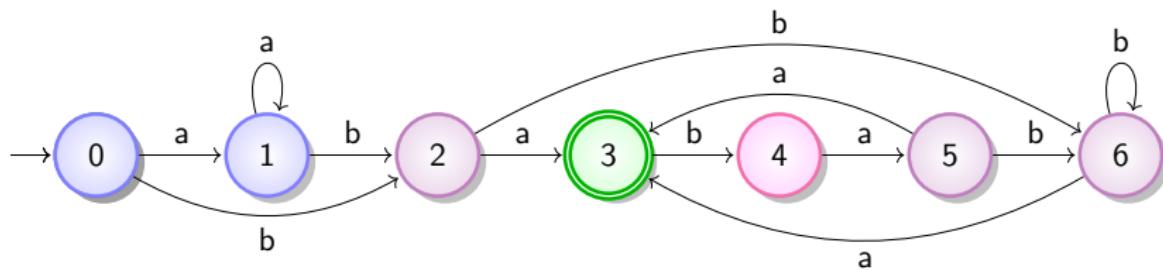
Inny przykład:



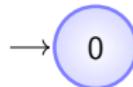
Po minimalizacji:



Inny przykład:

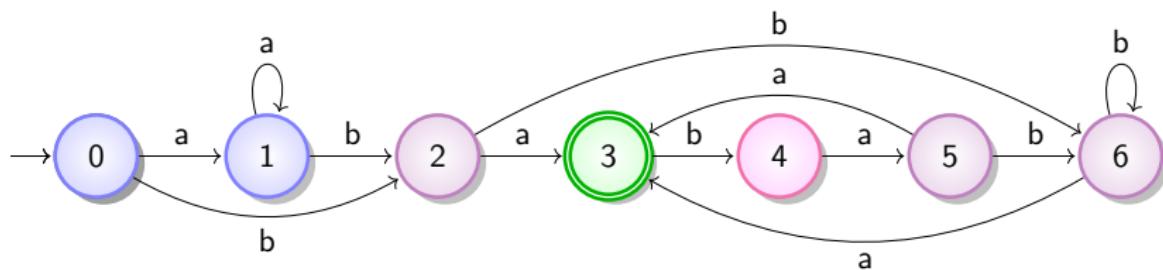


Po minimalizacji:



0    1    2    3    4    5    6  
↓    ↓    ↓    ↓    ↓    ↓    ↓

Inny przykład:

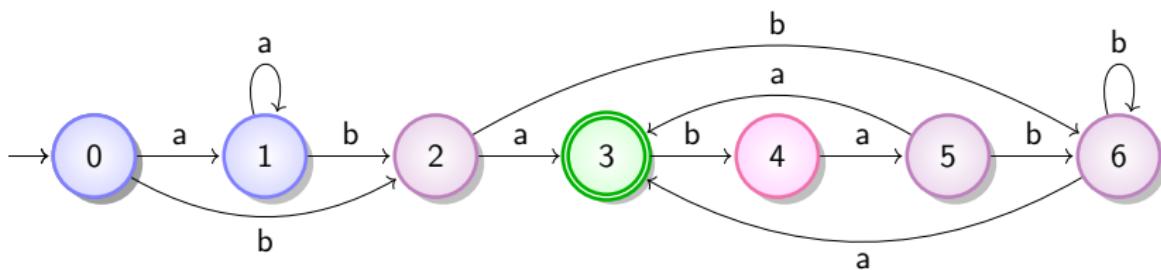


Po minimalizacji:



0    1    2    3    4    5    6  
↓    ↓    ↓    ↓    ↓    ↓    ↓

Inny przykład:

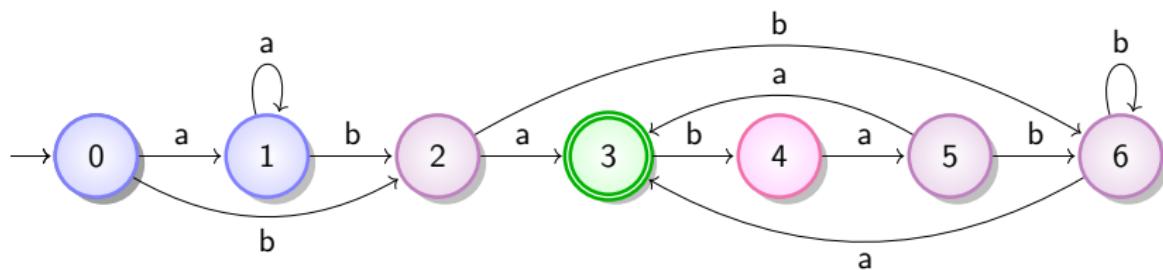


Po minimalizacji:



0    1    2    3    4    5    6  
↓    ↓    ↓    ↓    ↓    ↓    ↓

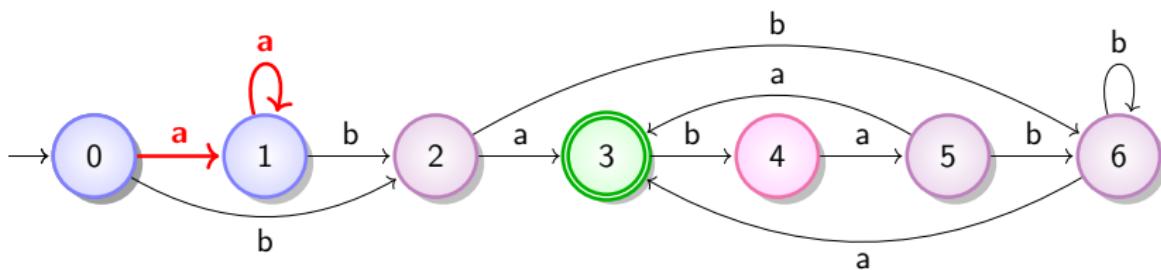
Inny przykład:



Po minimalizacji:



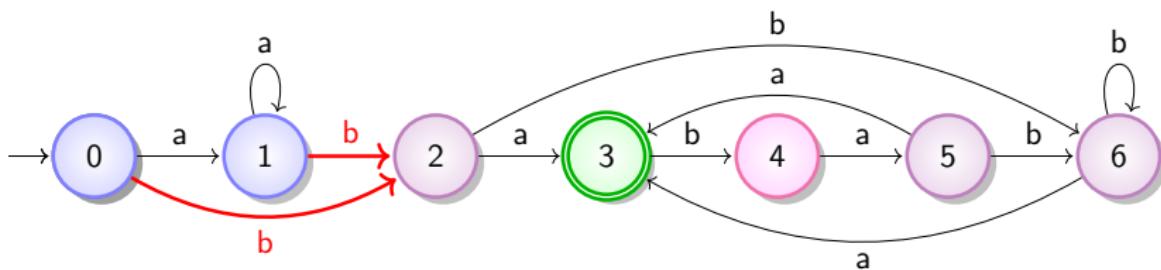
Inny przykład:



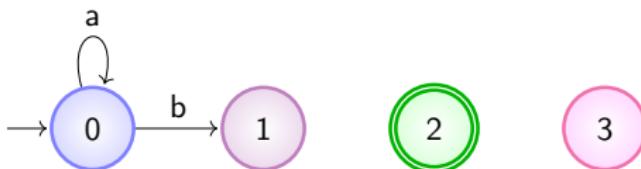
Po minimalizacji:



Inny przykład:

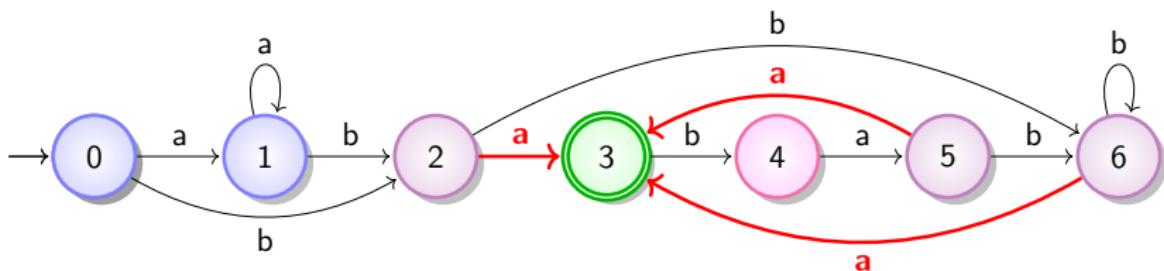


Po minimalizacji:

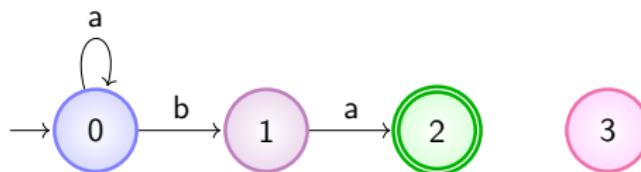


0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓

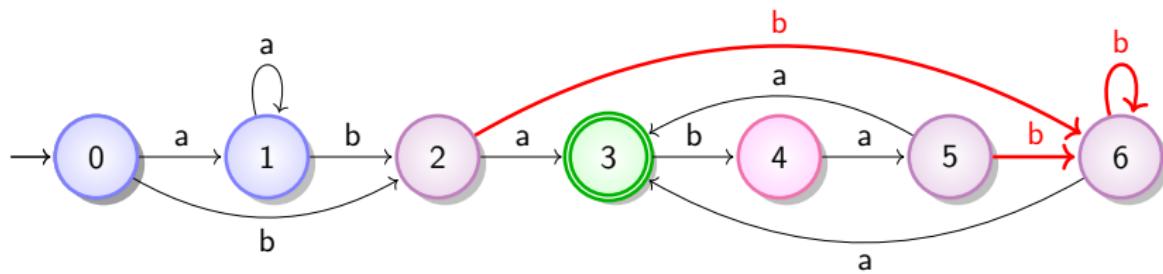
Inny przykład:



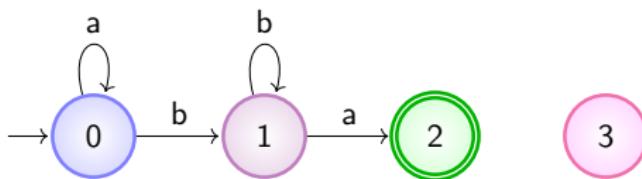
Po minimalizacji:



Inny przykład:

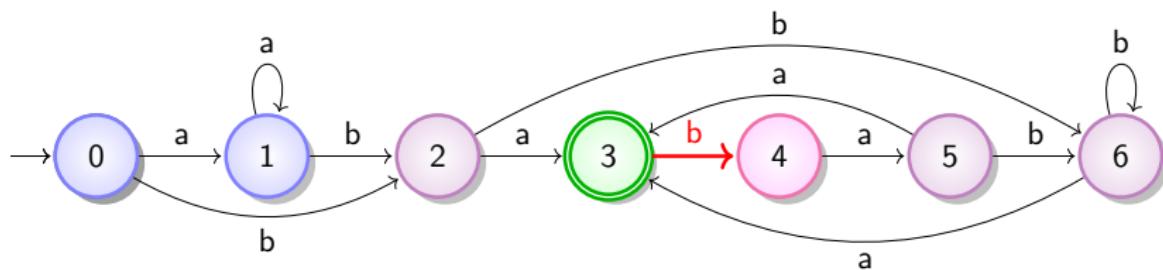


Po minimalizacji:

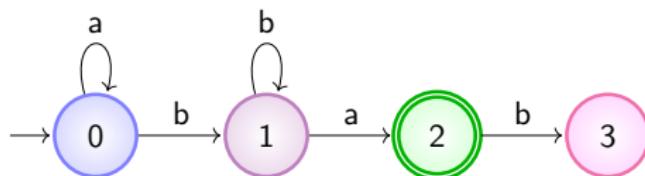


0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓

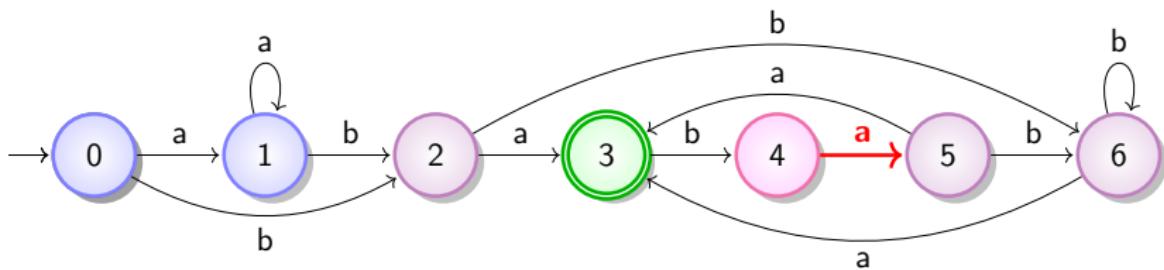
Inny przykład:



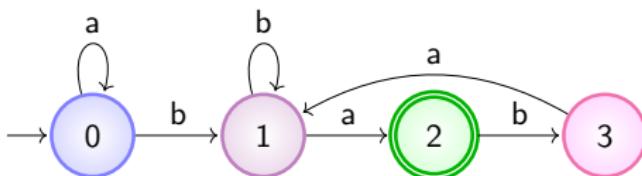
Po minimalizacji:



Inny przykład:

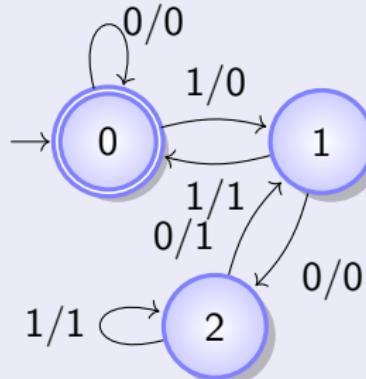


Po minimalizacji:

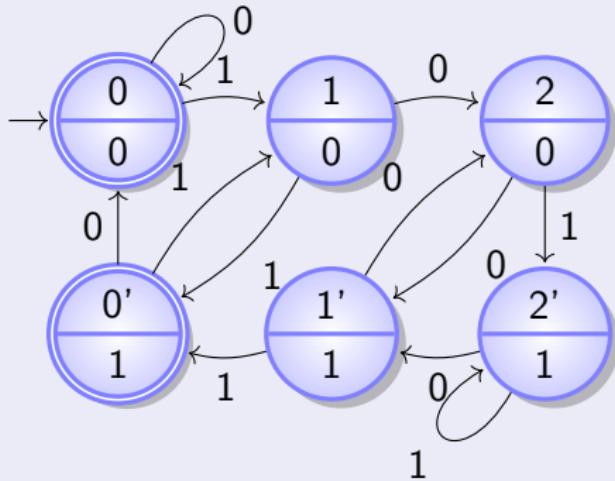


0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓

automat Mealy'ego



automat Moore'a

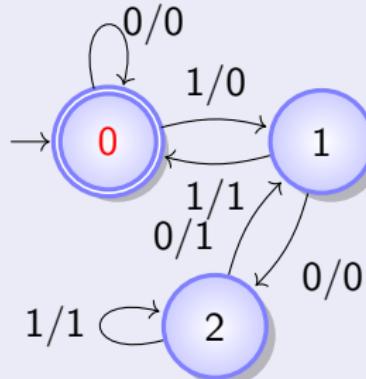


## Działanie

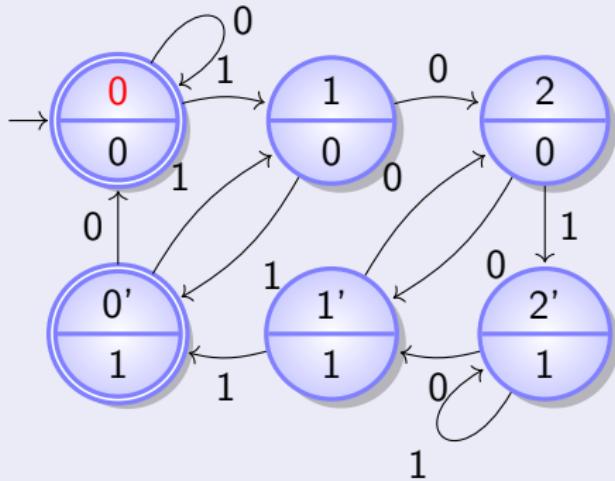
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

automat Mealy'ego



automat Moore'a

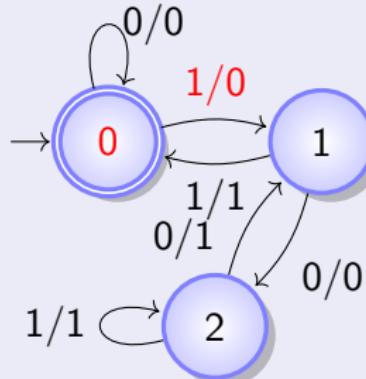


## Działanie

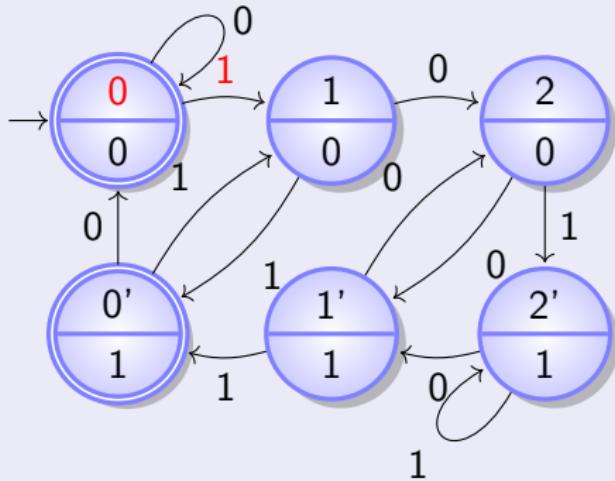
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1

automat Mealy'ego



automat Moore'a

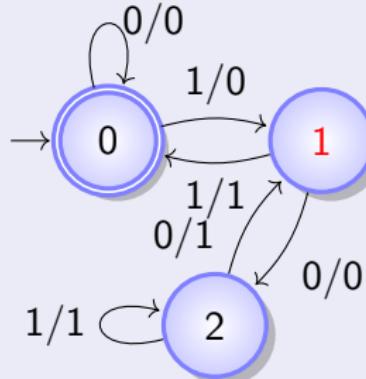


## Działanie

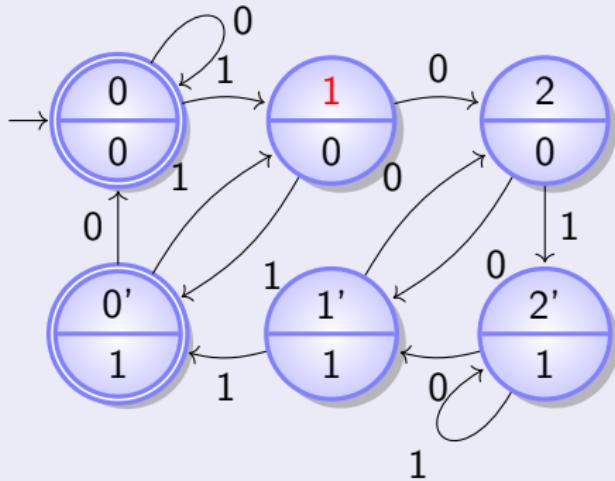
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1  
0

automat Mealy'ego



automat Moore'a

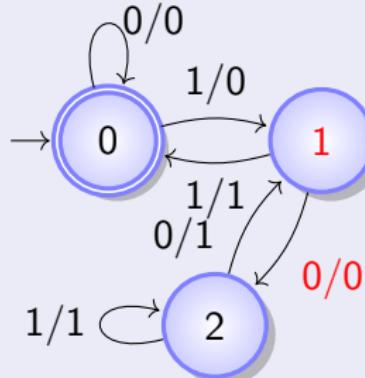


## Działanie

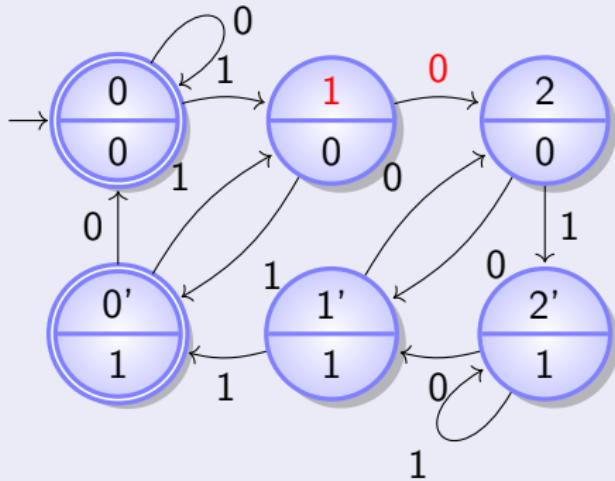
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

1 0 1 1 1 0 0 1 0 1  
0

automat Mealy'ego



automat Moore'a

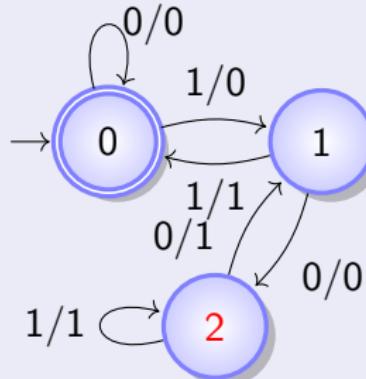


## Działanie

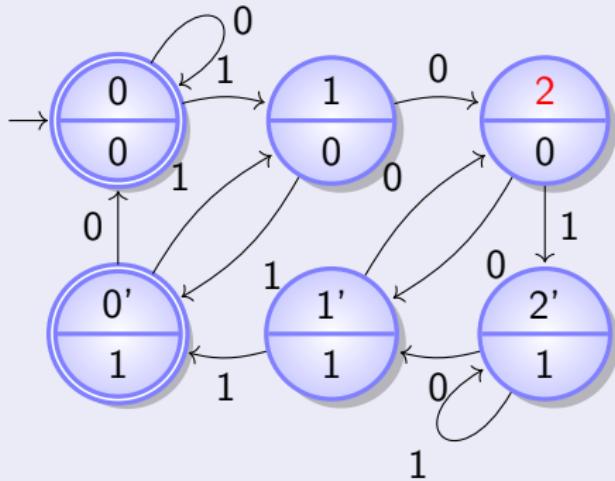
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

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

automat Mealy'ego



automat Moore'a



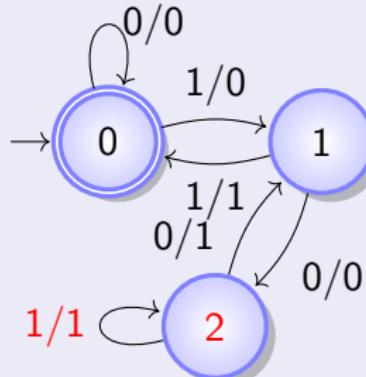
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

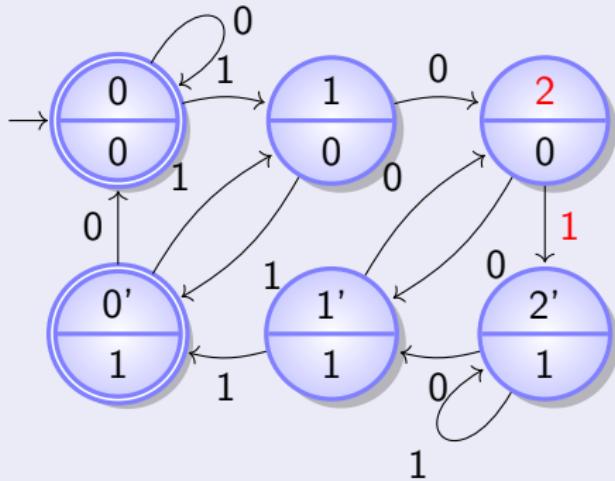
1 0 1 1 1 0 0 1 0 1

0 0

automat Mealy'ego



automat Moore'a



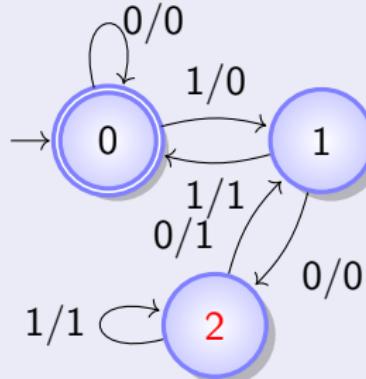
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

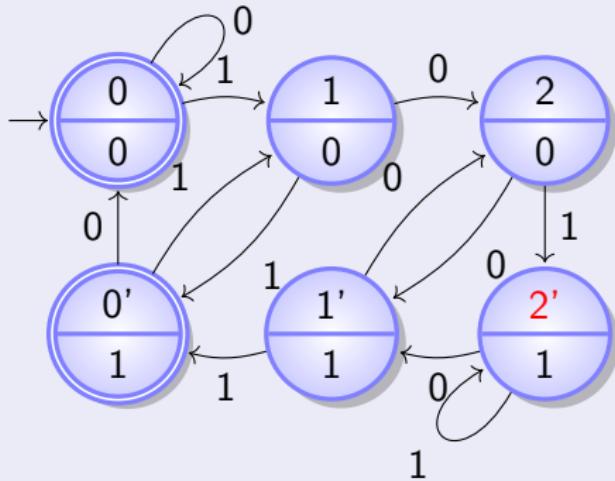
1 0 1 1 1 0 0 1 0 1

0 0 1

automat Mealy'ego



automat Moore'a



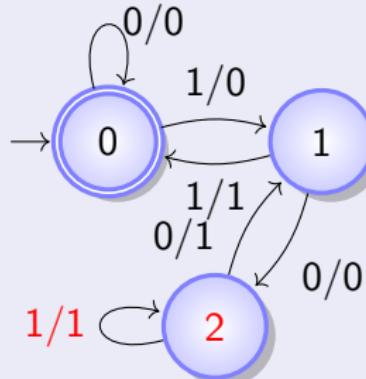
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

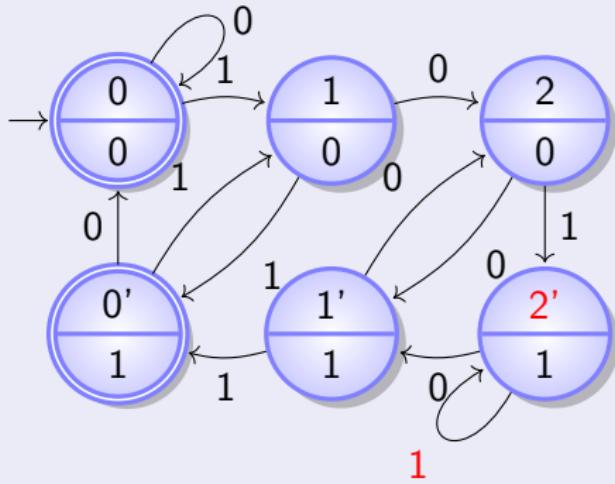
1 0 1 1 1 0 0 1 0 1

0 0 1

automat Mealy'ego



automat Moore'a

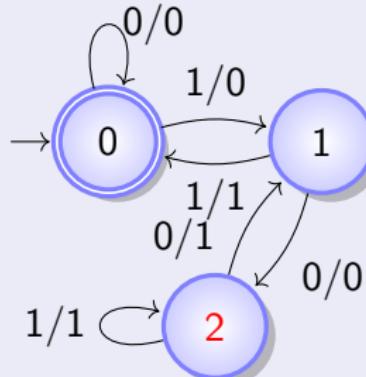


## Działanie

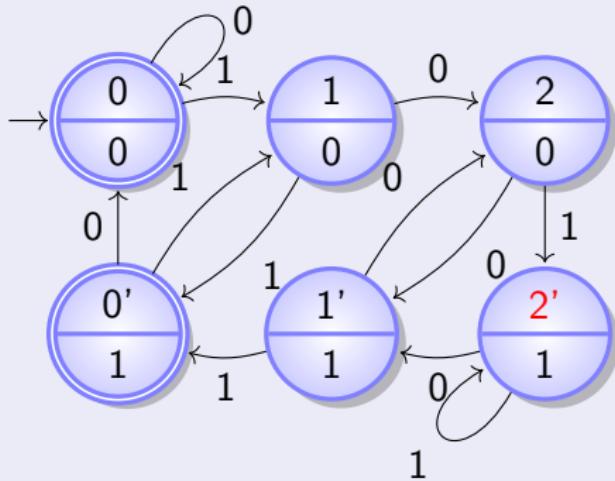
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

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

automat Mealy'ego



automat Moore'a



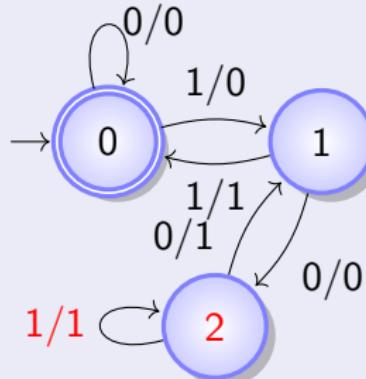
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

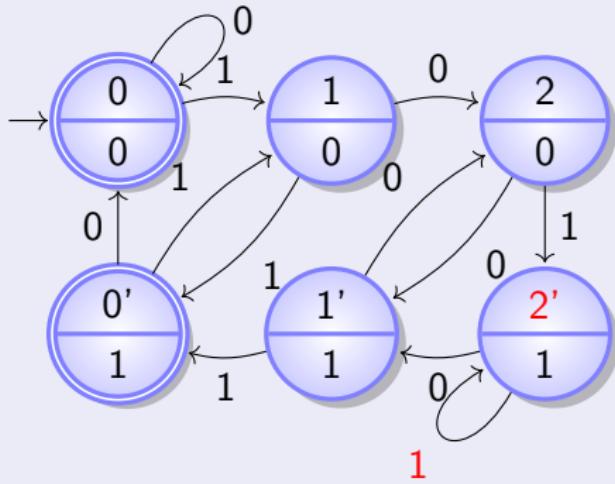
1 0 1 1 1 0 0 1 0 1

0 0 1 1

automat Mealy'ego



automat Moore'a



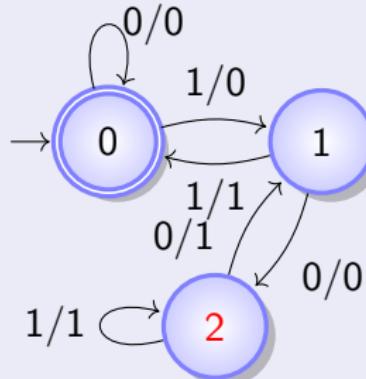
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

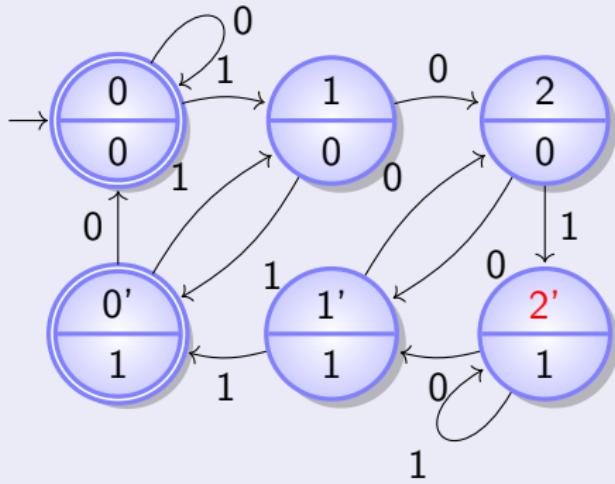
1 0 1 1 1 0 0 1 0 1

0 0 1 1 1

automat Mealy'ego



automat Moore'a



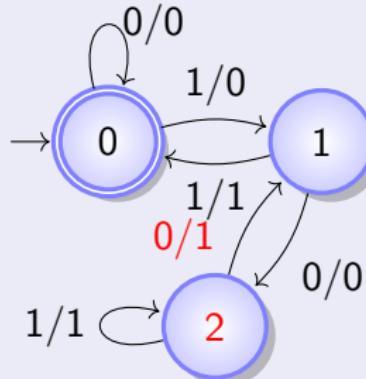
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

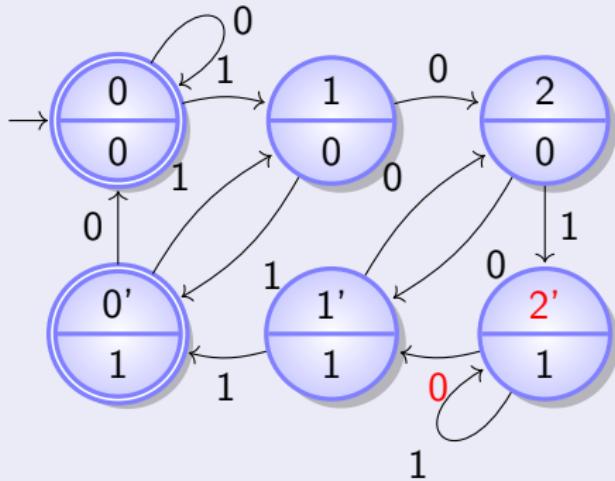
1 0 1 1 1 0 0 1 0 1

0 0 1 1 1

automat Mealy'ego



automat Moore'a

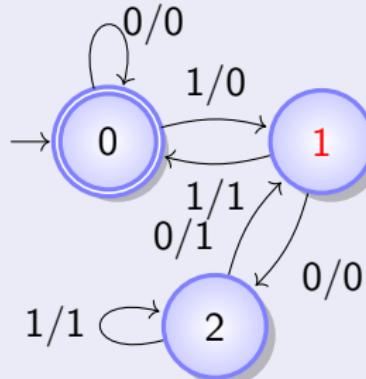


## Działanie

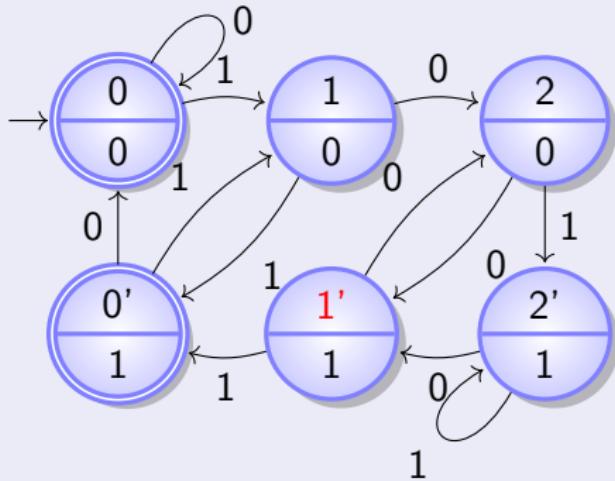
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

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

automat Mealy'ego



automat Moore'a



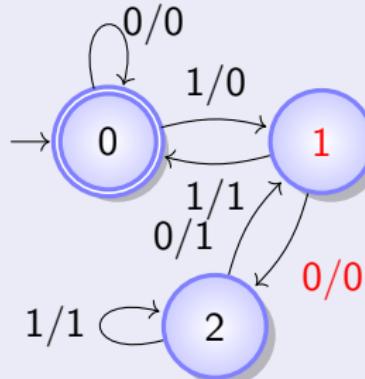
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

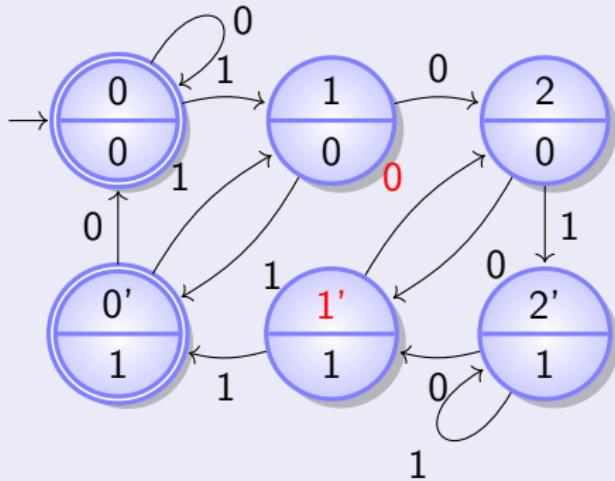
1 0 1 1 1 0 0 1 0 1

0 0 1 1 1 1

automat Mealy'ego



automat Moore'a



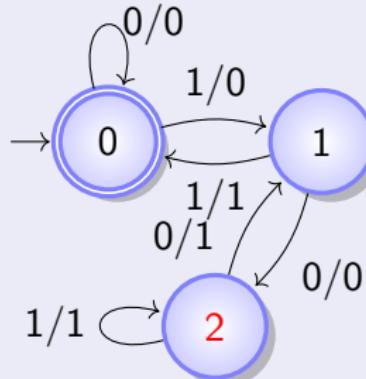
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

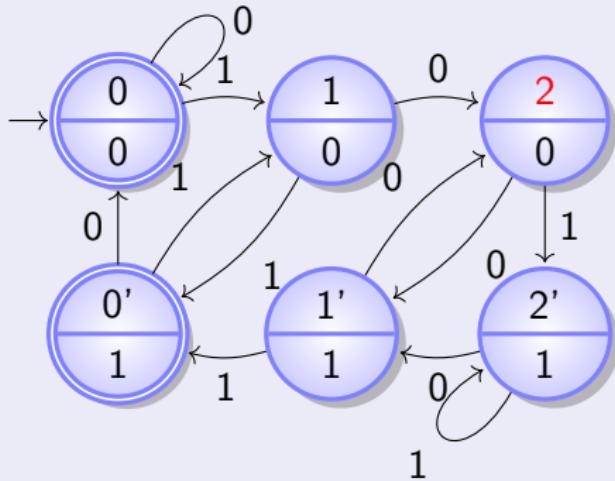
1 0 1 1 1 0 0 1 0 1

0 0 1 1 1 1 0

automat Mealy'ego



automat Moore'a



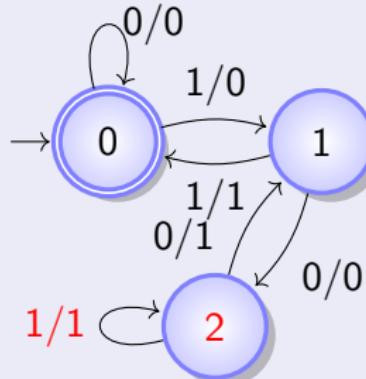
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

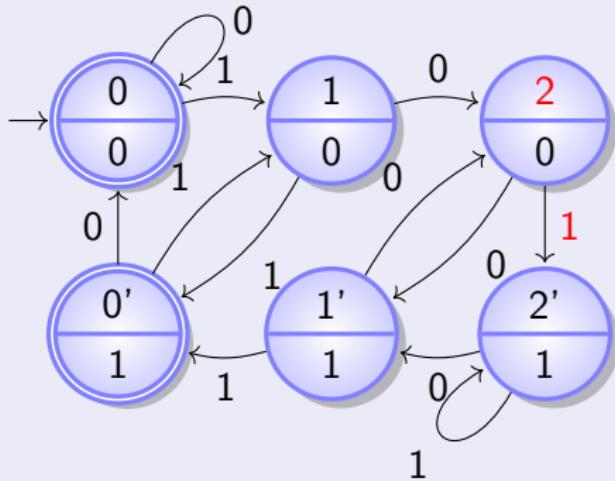
1 0 1 1 1 0 0 1 0 1

0 0 1 1 1 1 0

automat Mealy'ego



automat Moore'a



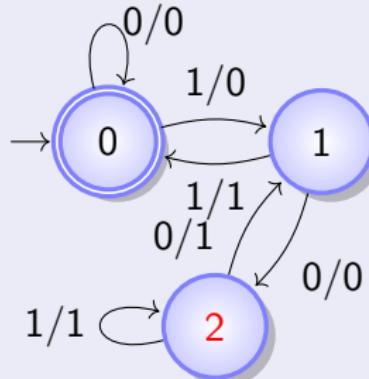
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

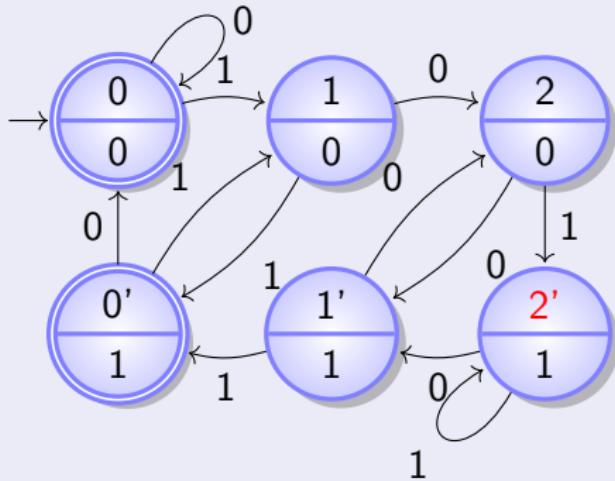
1 0 1 1 1 0 0 1 0 1

0 0 1 1 1 1 0 1

automat Mealy'ego



automat Moore'a



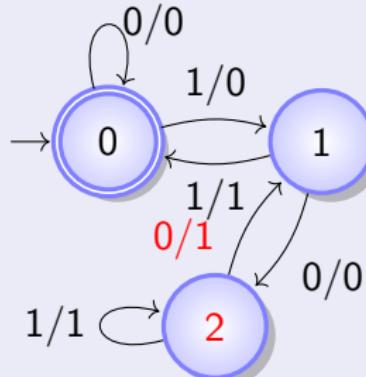
## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

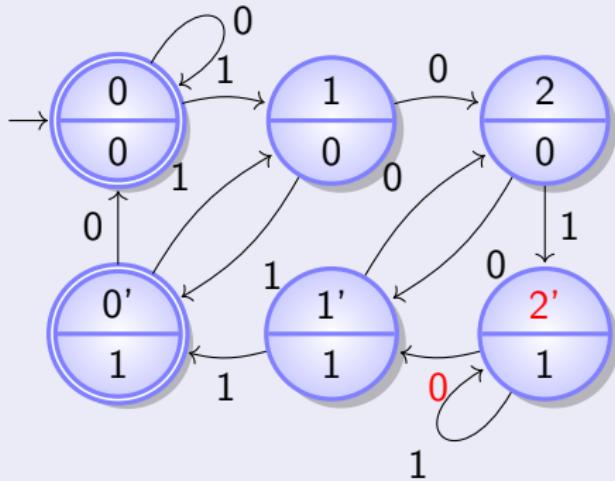
1 0 1 1 1 0 0 1 0 1

0 0 1 1 1 1 0 1

automat Mealy'ego



automat Moore'a

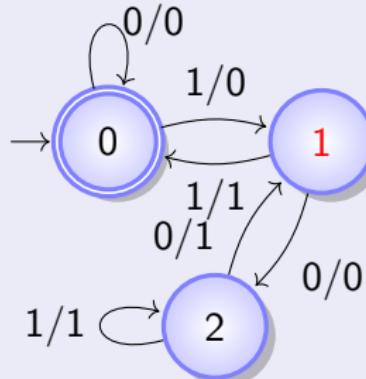


## Działanie

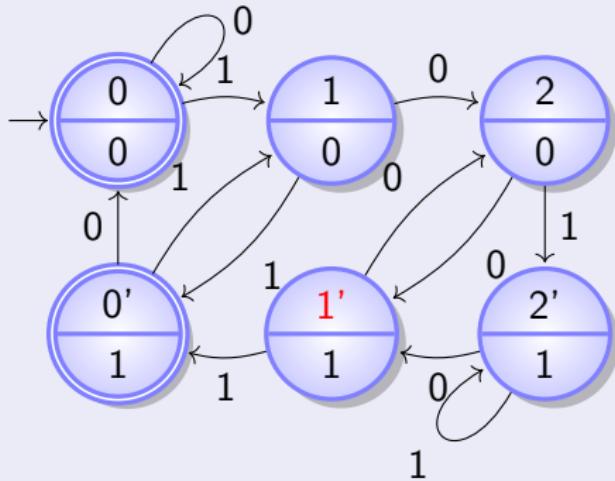
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

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

automat Mealy'ego



automat Moore'a

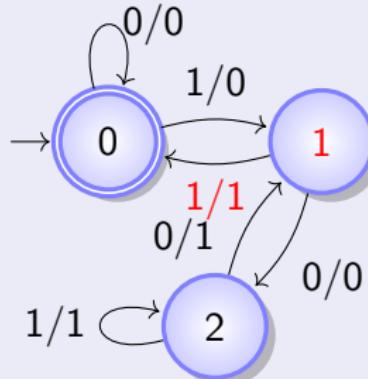


## Działanie

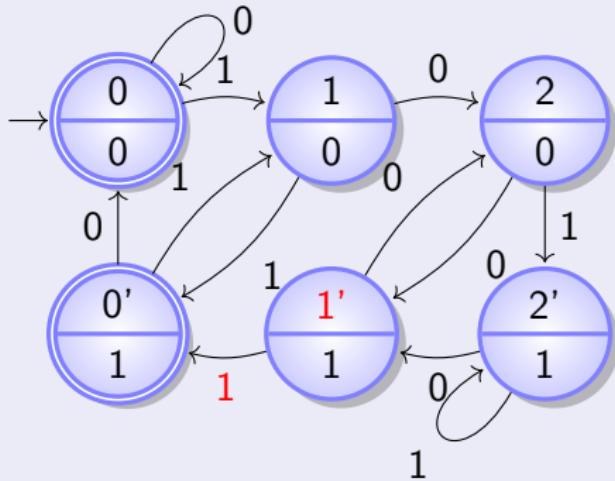
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

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

automat Mealy'ego



automat Moore'a

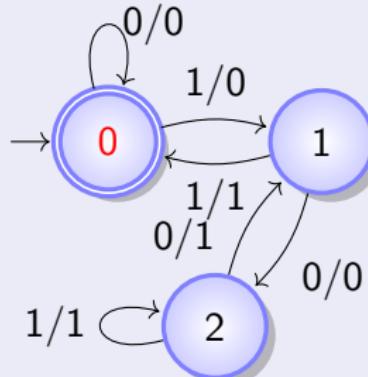


## Działanie

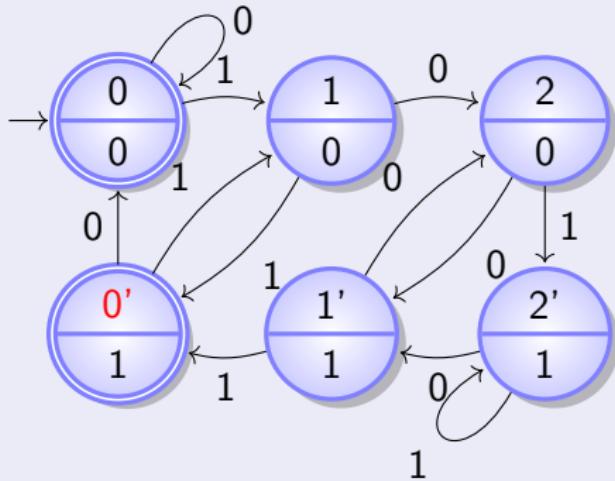
Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

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

automat Mealy'ego



automat Moore'a



## Działanie

Prześledźmy działanie automatów dla przykładowego ciągu wejściowego:

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



# Wyrażenia regularne

Wyrażeniem regularnym nad alfabetem  $\Sigma$  jest:

- ① zbiór pusty  $\emptyset$
- ② pusty ciąg symboli  $\varepsilon$  (ciąg o długości zero)
- ③ symbol  $\sigma$  z alfabetu  $\Sigma$
- ④ sklejenie  $R_1 R_2$  dwóch wyrażeń regularnych  $R_1$  i  $R_2$
- ⑤ alternatywa  $R_1 | R_2$  dwóch wyrażeń regularnych  $R_1$  i  $R_2$
- ⑥ domknięcie przechodnie  $R^*$  wyrażenia regularnego  $R$
- ⑦ wyrażenie regularne ujęte w nawiasy
- ⑧ nic ponadto

Najczęściej mamy do czynienia z rozszerzonymi wyrażeniami regularnymi.

Większość rozszerzeń nie zwiększa mocy rozpoznawanego języka.

Wyrażenie regularne opisuje język regularny i jest rozpoznawane przez automat skończony. Taki sam język może być opisany gramatyką regularną.



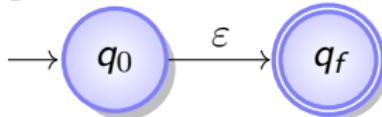
Konstrukcja Thompsona służy do budowy automatu skońzonego ( $\varepsilon$ -NFA) na podstawie wyrażenia regularnego. Jej cechą charakterystyczną jest to, że powstające automaty pośrednie i automat wynikowy mają szczególną postać — mają nie tylko jeden stan początkowy (jak każdy automat rozpoznający łańcuchy znaków), ale też (tylko) jeden stan końcowy. Wyrażenie rozkładane jest na podwyrażenia, a automat dla całego wyrażenia budowany jest na podstawie automatów zbudowanych dla podwyrażeń.

Podstawowe cegiełki:

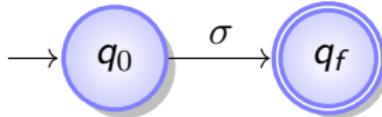
- $\emptyset \in RE$



- $\varepsilon \in RE$

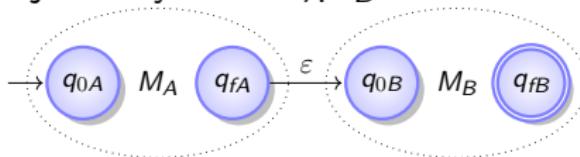


- $\sigma \in \Sigma \Rightarrow \sigma \in RE$

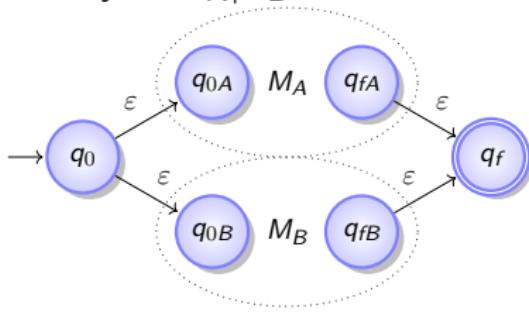


Składanie większych konstrukcji z  $R_A, R_B \in RE$ :

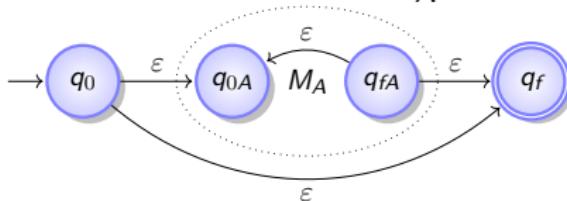
- sklejenie wyrażeń  $R_A R_B$



- alternatywa  $R_A | R_B$



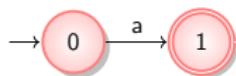
- domknięcie przechodnie  $R_A^*$





Przykład:  $a(a|b)^*a$

Przykład:  $a(a|b)^*a$



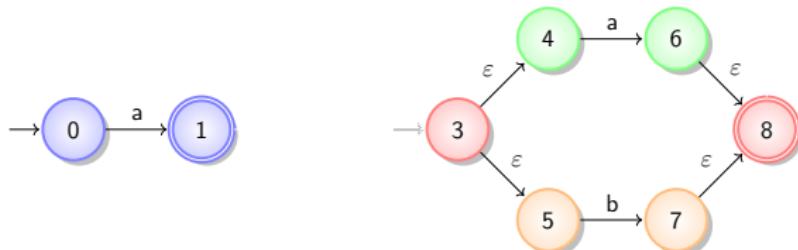
Przykład:  $a(a|b)^*a$



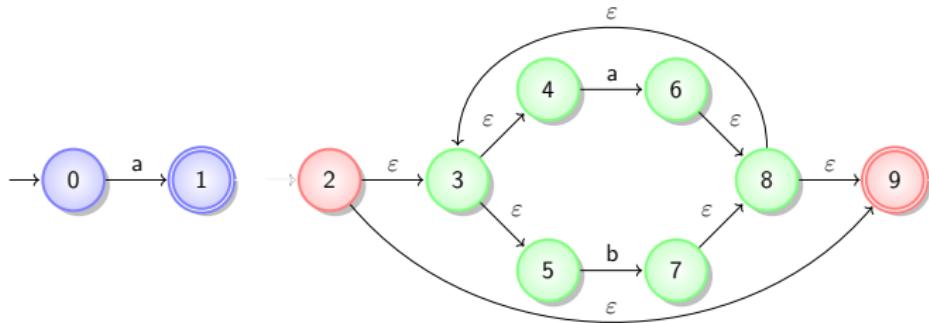
Przykład:  $a(a|b)^*a$



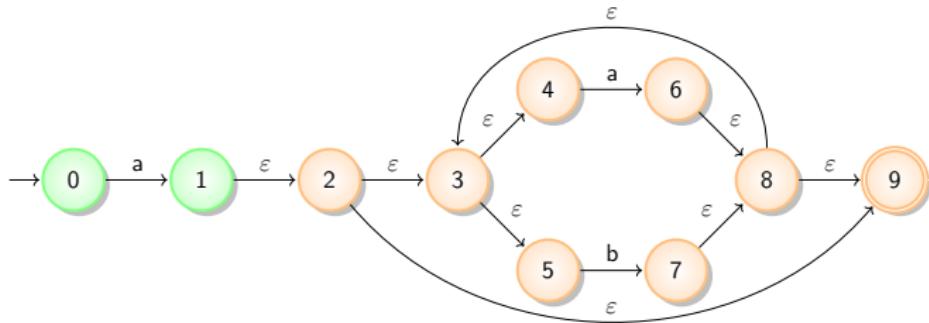
Przykład:  $a(a|b)^*a$



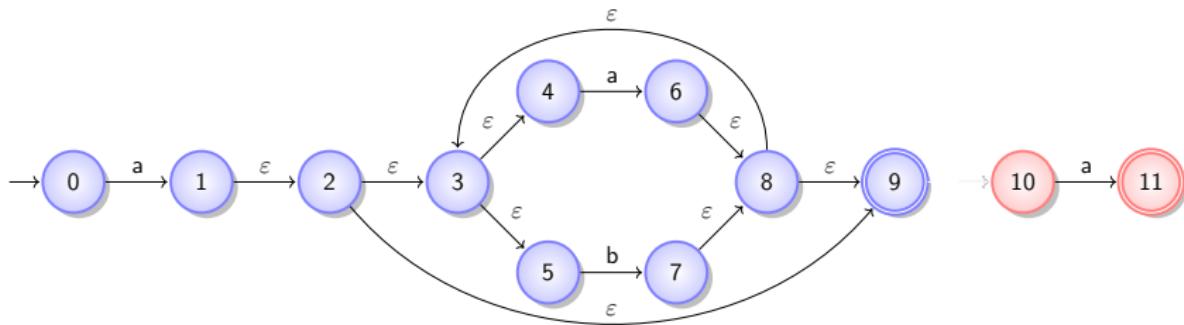
Przykład:  $a(a|b)^*a$



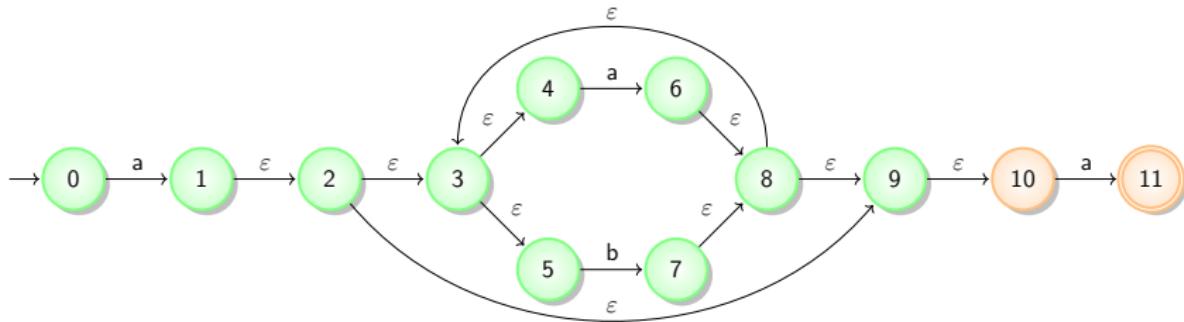
Przykład:  $a(a|b)^*a$



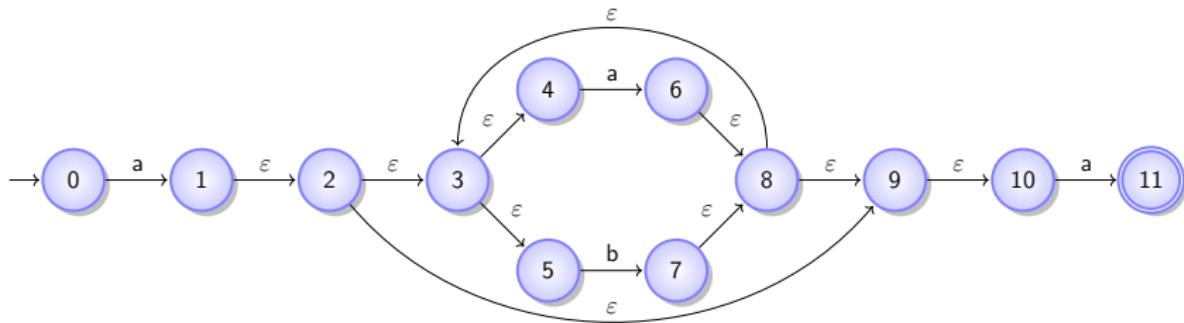
Przykład:  $a(a|b)^*a$



Przykład:  $a(a|b)^*a$



Przykład:  $a(a|b)^*a$





Cecha charakterystyczną automatu będącego wynikiem tej konstrukcji jest to, że wszystkie przejścia przychodzące do danego stanu mają tę samą etykietę. Wszystkie symbole występujące w wyrażeniu regularnym są numerowane, by odróżnić różne wystąpienia tych samych symboli. Dla każdego numerowanego symbolu tworzymy stan. Tworzymy także dodatkowo stan początkowy. Przejścia i końcowość stanów określane są za pomocą 4 elementów. Dla wyrażenia  $R$ :

- $\text{Null}(R) = \text{true} \Leftrightarrow \varepsilon \in L(R)$  — czy stan początkowy jest końcowy
- $\text{First}(R) = \{\sigma \in \Sigma : \exists_{x \in \Sigma^*} \sigma x \in L(R)\}$  — przejścia ze stanu początkowego
- $\text{Last}(R) = \{\sigma \in \Sigma : \exists_{x \in \Sigma^*} x\sigma \in L(R)\}$  — stany końcowe
- $\text{Follow}(R) = \{\sigma_1\sigma_2 \in \Sigma^2 : \exists_{x,y \in \Sigma^*} x\sigma_1\sigma_2 y \in L(R)\}$  — pozostałe przejścia



Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:



Automat ma tyle stanów, ile numerowanych symboli w wyrażeniu, plus dodatkowo stan początkowy  $q_0$ .



Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:



Ponieważ  $\varepsilon$  nie jest rozpoznawany przez to wyrażenie, to:

$$\text{Null}(x) = \text{false}$$

i stan początkowy nie jest końcowym.



Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:



$$\text{First}(x) = \{a_1\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

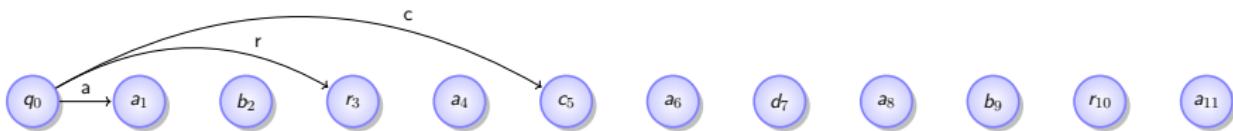


$$\text{First}(x) = \{a_1, r_3\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (\mathbf{a_1}b_2|\mathbf{r_3}a_4)^* \mathbf{c_5}a_6d_7(\mathbf{a_8}b_9|\mathbf{r_{10}}a_{11})^*$$

Zbudujmy z niego automat:

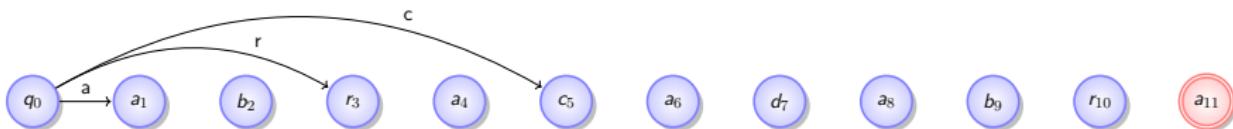


$$\text{First}(x) = \{a_1, r_3, c_5\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

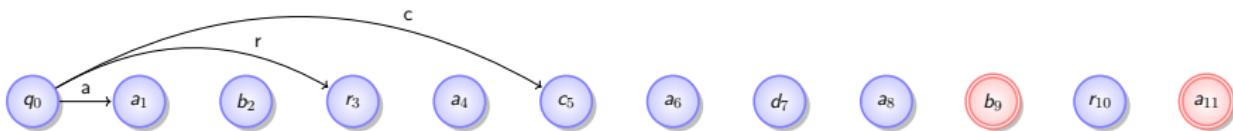


$$\text{Last}(x) = \{a_{11}\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

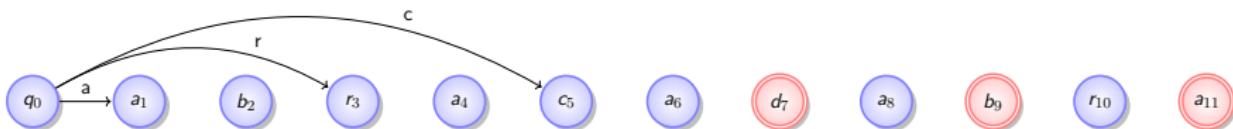


$$\text{Last}(x) = \{a_{11}, b_9\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

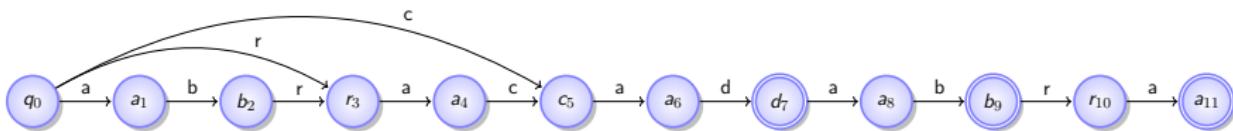


$$\text{Last}(x) = \{a_{11}, b_9, d_7\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

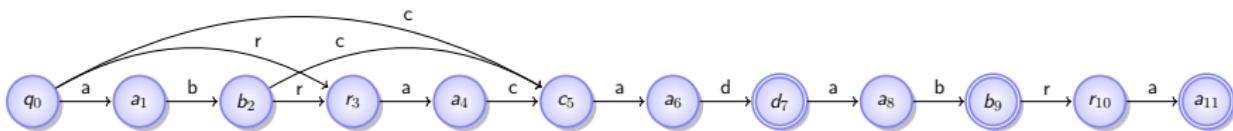


$$\text{Follow}(x) = \{a_1 b_2, b_2 r_3, r_3 a_4, a_4 c_5, c_5 a_6, a_6 d_7, d_7, a_8, a_8 b_9, b_9 r_{10}, r_{10} a_{11}\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

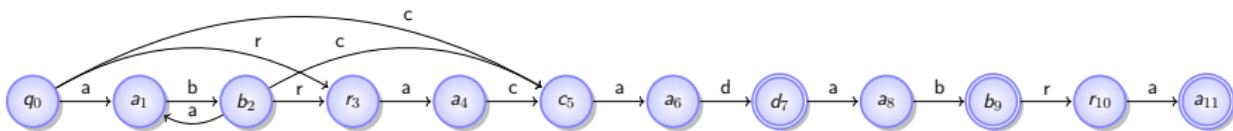


$$\text{Follow}(x) = \{a_1 b_2, b_2 r_3, r_3 a_4, a_4 c_5, c_5 a_6, a_6 d_7, d_7, a_8, a_8 b_9, b_9 r_{10}, r_{10} a_{11}, b_2 c_5\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (\mathbf{a_1b_2} | \mathbf{r_3a_4})^* c_5 a_6 d_7 (\mathbf{a_8b_9} | \mathbf{r_{10}a_{11}})^*$$

Zbudujmy z niego automat:



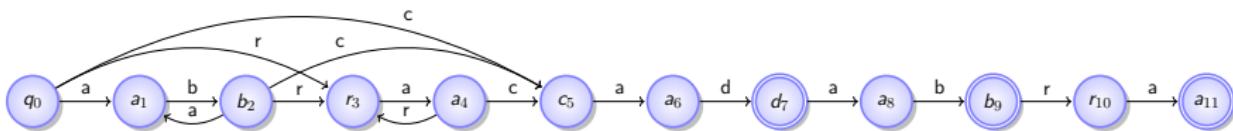
$$\text{Follow}(x) = \{a_1b_2, b_2r_3, r_3a_4, a_4c_5, c_5a_6, a_6d_7, d_7, a_8, a_8b_9, b_9r_{10}, r_{10}a_{11}, b_2c_5, b_2a_1\}$$



Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | \color{red}{r_3 a_4})^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

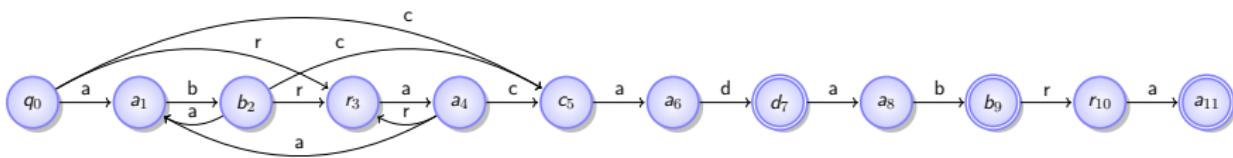


$$\text{Follow}(x) = \{a_1 b_2, b_2 r_3, r_3 a_4, a_4 c_5, c_5 a_6, a_6 d_7, d_7, a_8, a_8 b_9, b_9 r_{10}, r_{10} a_{11}, b_2 c_5, b_2 a_1, a_4 r_3\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

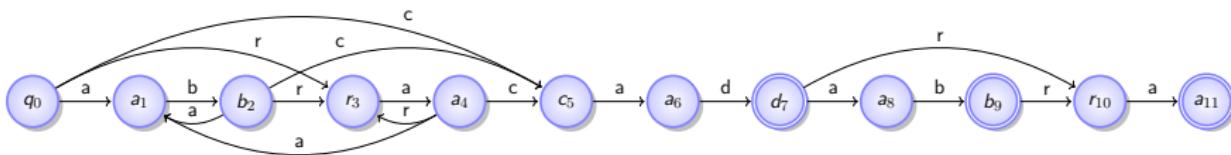


$$\text{Follow}(x) = \{a_1 b_2, b_2 r_3, r_3 a_4, a_4 c_5, c_5 a_6, a_6 d_7, d_7, a_8, a_8 b_9, b_9 r_{10}, r_{10} a_{11}, b_2 c_5, b_2 a_1, a_4 r_3, a_4 a_1\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

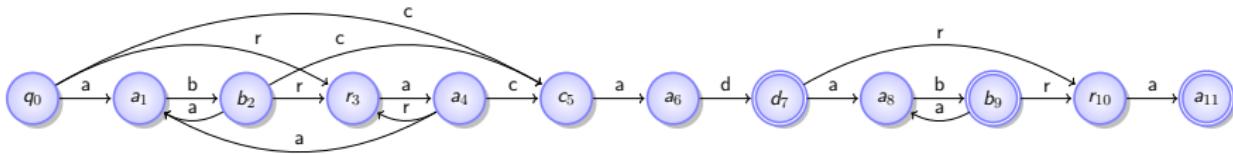


$$\text{Follow}(x) = \{a_1 b_2, b_2 r_3, r_3 a_4, a_4 c_5, c_5 a_6, a_6 d_7, d_7, a_8, a_8 b_9, b_9 r_{10}, r_{10} a_{11}, b_2 c_5, b_2 a_1, a_4 r_3, a_4 a_1, d_7 r_{10}\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:

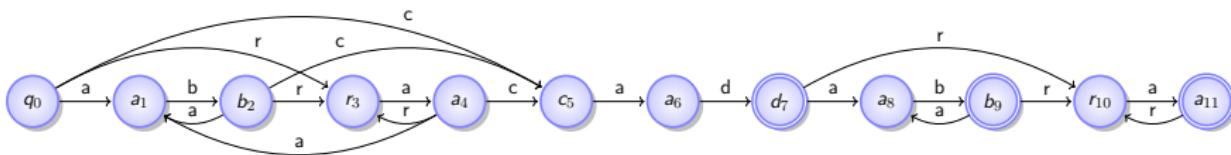


$$\text{Follow}(x) = \{a_1 b_2, b_2 r_3, r_3 a_4, a_4 c_5, c_5 a_6, a_6 d_7, d_7, a_8, a_8 b_9, b_9 r_{10}, r_{10} a_{11}, b_2 c_5, b_2 a_1, a_4 r_3, a_4 a_1, d_7 r_{10}, b_9 a_8\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | \textcolor{red}{r_{10} a_{11}})^*$$

Zbudujmy z niego automat:

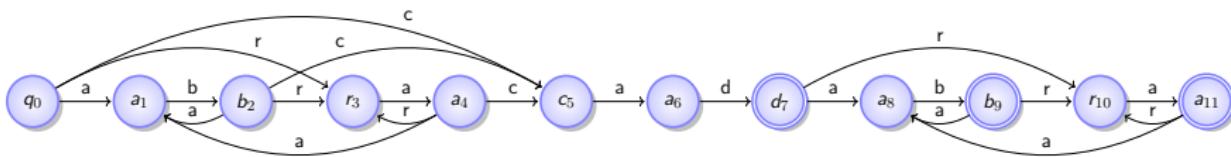


$$\text{Follow}(x) = \{a_1 b_2, b_2 r_3, r_3 a_4, a_4 c_5, c_5 a_6, a_6 d_7, d_7, a_8, a_8 b_9, \\ b_9 r_{10}, r_{10} a_{11}, b_2 c_5, b_2 a_1, a_4 r_3, a_4 a_1, d_7 r_{10}, \\ b_9 a_8, a_{11} r_{10}\}$$

Weźmy dla przykładu wyrażenie regularne  $(ab|ra)^*cad(ab|ra)^*$ . Po ponumerowaniu otrzymujemy:

$$x = (a_1 b_2 | r_3 a_4)^* c_5 a_6 d_7 (a_8 b_9 | r_{10} a_{11})^*$$

Zbudujmy z niego automat:



$$\text{Follow}(x) = \{a_1 b_2, b_2 r_3, r_3 a_4, a_4 c_5, c_5 a_6, a_6 d_7, d_7, a_8, a_8 b_9, b_9 r_{10}, r_{10} a_{11}, b_2 c_5, b_2 a_1, a_4 r_3, a_4 a_1, d_7 r_{10}, b_9 a_8, a_{11} r_{10}, a_{11} a_8\}$$



Dla podstawowych elementów wyrażeń regularnych cecha Null wynosi:

- $\text{Null}(\emptyset) = \text{false}$
- $\text{Null}(\varepsilon) = \text{true}$
- $\text{Null}(\sigma \in \Sigma) = \text{false}$

Dla złożonych wyrażeń składających się z podwyrażeń  $f$  i  $g$  Null obliczamy w następujący sposób:

- $\text{Null}(fg) = \text{Null}(f) \wedge \text{Null}(g)$
- $\text{Null}(f|g) = \text{Null}(f) \vee \text{Null}(g)$
- $\text{Null}(f^*) = \text{true}$



Zbiór First dla podstawowych elementów wyrażeń regularnych obliczamy następująco:

- $\text{First}(\emptyset) = \emptyset$
- $\text{First}(\varepsilon) = \emptyset$
- $\text{First}(\sigma \in \Sigma) = \{\sigma\}$

Dla złożonych wyrażeń składających się z podwyrażeń  $f$  i  $g$  First obliczamy w następujący sposób:

- $\text{First}(fg) = \begin{cases} \text{First}(f) & \text{jeśli } \neg\text{Null}(f) \\ \text{First}(f) \cup \text{First}(g) & \text{jeśli } \text{Null}(f) \end{cases}$
- $\text{First}(f|g) = \text{First}(f) \cup \text{First}(g)$
- $\text{First}(f^*) = \text{First}(f)$



Zbiór Last dla podstawowych elementów wyrażeń regularnych obliczamy następująco:

- $\text{Last}(\emptyset) = \emptyset$
- $\text{Last}(\varepsilon) = \emptyset$
- $\text{Last}(\sigma \in \Sigma) = \{\sigma\}$

Dla złożonych wyrażeń składających się z podwyrażeń  $f$  i  $g$  Last obliczamy w następujący sposób:

- $\text{Last}(fg) = \begin{cases} \text{Last}(g) & \text{jeśli } \neg \text{Null}(g) \\ \text{Last}(f) \cup \text{Last}(g) & \text{jeśli } \text{Null}(g) \end{cases}$
- $\text{Last}(f|g) = \text{Last}(f) \cup \text{Last}(g)$
- $\text{Last}(f^*) = \text{Last}(f)$



Zbiór Follow dla podstawowych elementów wyrażeń regularnych obliczamy następująco:

- $\text{Follow}(\emptyset) = \emptyset$
- $\text{Follow}(\varepsilon) = \emptyset$
- $\text{Follow}(\sigma \in \Sigma) = \emptyset$

Dla złożonych wyrażeń składających się z podwyrażeń  $f$  i  $g$  Follow obliczamy w następujący sposób:

- $\text{Follow}(fg) = \text{Follow}(f) \cup \text{Follow}(g) \cup (\text{Last}(f) \times \text{First}(g))$
- $\text{Follow}(f|g) = \text{Follow}(f) \cup \text{Follow}(g)$
- $\text{Follow}(f^*) = \text{Follow}(f) \cup (\text{Last}(f) \times \text{First}(f))$

# Konstrukcja McNaughton-Yamada-Głuszkow

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

RE		Null	First	Last	Follow
----	--	------	-------	------	--------

# Konstrukcja McNaughton-Yamada-Głuszkow

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

RE	Null	First	Last	Follow
$a_1$	false	$\{a_1\}$	$\{a_1\}$	$\emptyset$

# Konstrukcja McNaughton-Yamada-Głuszkow

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

RE	Null	First	Last	Follow
$a_1$	false	$\{a_1\}$	$\{a_1\}$	$\emptyset$
$a_2$	false	$\{a_2\}$	$\{a_2\}$	$\emptyset$

# Konstrukcja McNaughton-Yamada-Głuszkow

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

RE	Null	First	Last	Follow
$a_1$	false	$\{a_1\}$	$\{a_1\}$	$\emptyset$
$a_2$	false	$\{a_2\}$	$\{a_2\}$	$\emptyset$
$b_3$	false	$\{b_3\}$	$\{b_3\}$	$\emptyset$

# Konstrukcja McNaughton-Yamada-Głuszkow

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

RE	Null	First	Last	Follow
$a_1$	false	$\{a_1\}$	$\{a_1\}$	$\emptyset$
$a_2$	false	$\{a_2\}$	$\{a_2\}$	$\emptyset$
$b_3$	false	$\{b_3\}$	$\{b_3\}$	$\emptyset$
$a_2 b_3$	false	$\{a_2, b_3\}$	$\{a_2, b_3\}$	$\emptyset$

# Konstrukcja McNaughton-Yamada-Głuszkow

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

RE	Null	First	Last	Follow
$a_1$	false	$\{a_1\}$	$\{a_1\}$	$\emptyset$
$a_2$	false	$\{a_2\}$	$\{a_2\}$	$\emptyset$
$b_3$	false	$\{b_3\}$	$\{b_3\}$	$\emptyset$
$a_2 b_3$	false	$\{a_2, b_3\}$	$\{a_2, b_3\}$	$\emptyset$
$(a_2 b_3)^*$	true	$\{a_2, b_3\}$	$\{a_2, b_3\}$	$\{a_2a_2, a_2b_3, b_3a_2, b_3b_3\}$

# Konstrukcja McNaughton-Yamada-Głuszkow

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

RE	Null	First	Last	Follow
$a_1$	false	$\{a_1\}$	$\{a_1\}$	$\emptyset$
$a_2$	false	$\{a_2\}$	$\{a_2\}$	$\emptyset$
$b_3$	false	$\{b_3\}$	$\{b_3\}$	$\emptyset$
$a_2 b_3$	false	$\{a_2, b_3\}$	$\{a_2, b_3\}$	$\emptyset$
$(a_2 b_3)^*$	true	$\{a_2, b_3\}$	$\{a_2, b_3\}$	$\{a_2a_2, a_2b_3, b_3a_2, b_3b_3\}$
$a_1(a_2 b_3)^*$	false	$\{a_1\}$	$\{a_1, a_2, b_3\}$	$\{a_1a_2, a_1b_3, a_2a_2, a_2b_3, b_3a_2, b_3b_3\}$

# Konstrukcja McNaughton-Yamada-Głuszkow

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

RE	Null	First	Last	Follow
$a_1$	false	$\{a_1\}$	$\{a_1\}$	$\emptyset$
$a_2$	false	$\{a_2\}$	$\{a_2\}$	$\emptyset$
$b_3$	false	$\{b_3\}$	$\{b_3\}$	$\emptyset$
$a_2 b_3$	false	$\{a_2, b_3\}$	$\{a_2, b_3\}$	$\emptyset$
$(a_2 b_3)^*$	true	$\{a_2, b_3\}$	$\{a_2, b_3\}$	$\{a_2a_2, a_2b_3, b_3a_2, b_3b_3\}$
$a_1(a_2 b_3)^*$	false	$\{a_1\}$	$\{a_1, a_2, b_3\}$	$\{a_1a_2, a_1b_3, a_2a_2, a_2b_3, b_3a_2, b_3b_3\}$
$a_4$	false	$\{a_4\}$	$\{a_4\}$	$\emptyset$



Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$

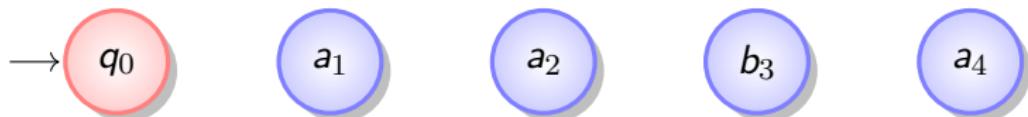
RE	Null	First	Last	Follow
$a_1$	false	{ $a_1$ }	{ $a_1$ }	$\emptyset$
$a_2$	false	{ $a_2$ }	{ $a_2$ }	$\emptyset$
$b_3$	false	{ $b_3$ }	{ $b_3$ }	$\emptyset$
$a_2 b_3$	false	{ $a_2, b_3$ }	{ $a_2, b_3$ }	$\emptyset$
$(a_2 b_3)^*$	true	{ $a_2, b_3$ }	{ $a_2, b_3$ }	{ $a_2a_2, a_2b_3, b_3a_2, b_3b_3$ }
$a_1(a_2 b_3)^*$	false	{ $a_1$ }	{ $a_1, a_2, b_3$ }	{ $a_1a_2, a_1b_3, a_2a_2, a_2b_3, b_3a_2, b_3b_3$ }
$a_4$	false	{ $a_4$ }	{ $a_4$ }	$\emptyset$
$a_1(a_2 b_3)^*a_4$	false	{ $a_1$ }	{ $a_4$ }	{ $a_1a_2, a_1b_3, a_2a_2, a_2b_3, b_3a_2, b_3b_3, a_1a_4, a_2a_4, b_3a_4$ }

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$



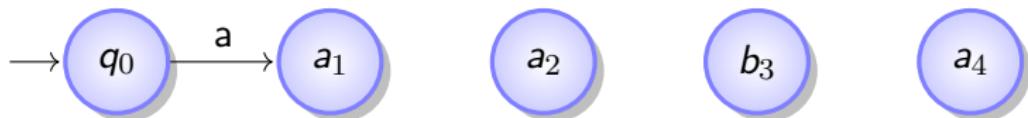
Automat ma tyle stanów, ile numerowanych symboli w wyrażeniu, plus dodatkowo stan początkowy  $q_0$ .

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$



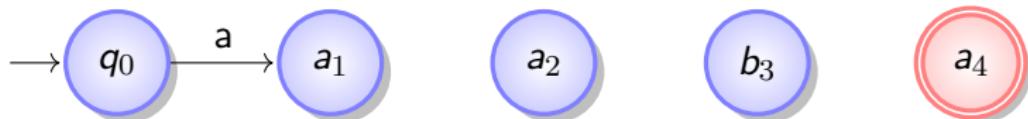
Ponieważ  $\varepsilon$  nie jest rozpoznawany przez to wyrażenie, to:  
 $\text{Null}(a_1(a_2|b_3)^*a_4) = \text{false}$   
i stan początkowy nie jest końcowym.

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$



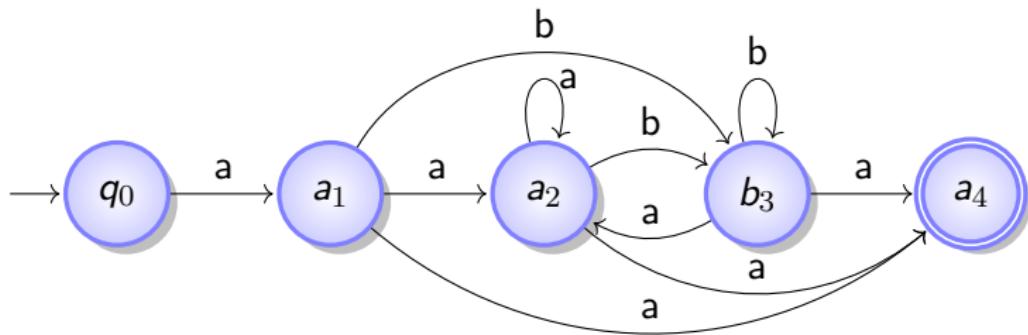
$$\text{First}(a_1(a_2|b_3)^*a_4) = \{a_1\}$$

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$



$$\text{Last}(a_1(a_2|b_3)^*a_4) = \{a_4\}$$

Przykład:  $a(a|b)^*a$ , po ponumerowaniu symboli:  $a_1(a_2|b_3)^*a_4$



$$\text{Follow}(a_1(a_2|b_3)^*a_4) = \{a_1a_2, a_1b_3, a_2a_2, a_2b_3, b_3a_2, b_3b_3, a_1a_4, a_2a_4, b_3a_4\}$$

Nie jest to minimalny automat deterministyczny.



```
do {
    switch (stan) {
        case 0:
            switch (znak) {
                case '0':
                    ...
                case '9':
                    *bufor++ = znak; stan = 7;
                    exit;
                case '+':
                    stan = 11;
                    exit;
                    ...
                ...
            }
            znak = czytaj_znak();
    } while (znak != EOF);
```



# Jak zbudować analizator leksykalny

- ① Na końcu każdego wzorca doklejamy separator i znak identyfikujący daną regułę (np. kolejną literę).
- ② Wzorce łączymy w wielką alternatywę, tak jakby były rozdzielone znakami „|”.
- ③ Stosujemy algorytm zamiany wyrażenia regularnego na automat niedeterministyczny.
- ④ Determinizujemy automat.
- ⑤ Minimalizujemy automat.
- ⑥ W trakcie rozpoznania wzorca, przechodzimy kolejnymi przejściami etykietowanymi symbolami wejściowymi, dopóki jest to możliwe.
- ⑦ Jeśli z osiągniętego stanu nie prowadzi żadne przejście etykietowane separatorem, wróć do ostatniego takiego stanu na ścieżce.
- ⑧ Przejdź przejściem etykietowanym separatorem.
- ⑨ Etykieta pierwszego przejścia określa regułę do wykonania.



# Własności języków regularnych



Jeśli  $R_1$  i  $R_2$  są językami regularnymi, a  $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$  i  $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$  rozpoznającymi je automatami, to językami regularnymi są także:

- $R = R_1 \cup R_2$ ,  $M = (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F_1 \times Q_2 \cup Q_1 \times F_2)$ ,  
 $\delta((p, q), \sigma) = (\delta_1(p, \sigma), \delta_2(q, \sigma))$
- $R = R_1 \cap R_2$ ,  $M = (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F_1 \times F_2)$ ,  
 $\delta((p, q), \sigma) = (\delta_1(p, \sigma), \delta_2(q, \sigma))$
- $R = \overline{R_1} = \Sigma^* - R_1$  — dopełnienie języka,  $M = (Q_1, \Sigma, \delta_1, q_0, Q \setminus F_1)$
- $R = R_1 \setminus R_2$ ,  $M = (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F_1 \times (Q_2 \setminus F_2))$
- $R = R_1^{R_1}$  — odwrócenie języka (słowa czytamy od tyłu)
- $R = R_1^*$  — domknięcie języka
- obraz homomorficzny  $R_1$  — podstawienie łańcuchów za symbole
- przeciwbraz homomorficzny  $R_1$



Gramatyką bezkontekstową nazywamy czwórkę uporządkowaną  $(V, \Sigma, P, S)$ , gdzie:

- $V$  to zbiór **zmiennych** (inaczej — symboli niekońcowych)
- $\Sigma$  (oznaczany też jako  $T$ ) to zbiór **symboli końcowych** (ang. *terminals*)
- $P$  to zbiór **reguł produkcji** postaci  $\alpha \rightarrow \beta$ , gdzie  $\alpha \in V$ , zaś  $\beta$  to dowolny (także pusty) ciąg symboli końcowych i niekońcowych
- $S$  to symbol wyróżniony — **symbol początkowy** gramatyki



Zapis **BNF** — Backus-Naur Form lub Backus Normal Form to inny zapis gramatyki bezkontekstowej:

- zmienne zapisujemy w nawiasach kątowych, np. `<zmienna>`
- symbole końcowe zapisujemy w cudzysłowach
- $\epsilon$  zapisujemy jako `" "`
- symbol przepisania ( $\rightarrow$ ) w regule produkcji zapisujemy jako `::=`
- różne reguły produkcji z tą samą zmienną po lewej stronie można łączyć w jedną, gdzie prawe strony rozdzielone są znakiem `|`

Przykład: gramatykę  $E \rightarrow 0 \mid 1 \mid E + E \mid E * E$

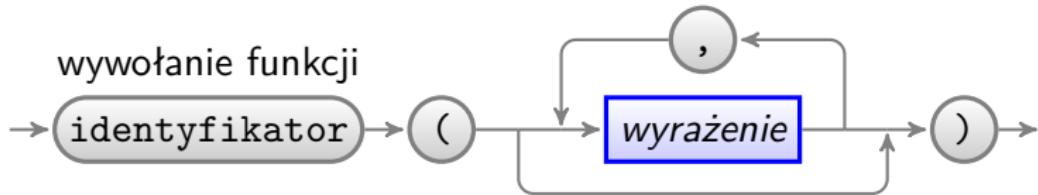
zapisujemy jako:

`<E> ::= "0" | "1" | <E> "+" <E> | <E> "*" <E>`

Dla gramatyki:

```
<wywołanie funkcji> := "identyfikator" "(" <parametry> ")"
<parametry> := "" | <lista wyrażeń>
<lista wyrażeń> := <wyrażenie>
                  | <lista wyrażeń> "," <wyrażenie>
```

możemy narysować następujący diagram składniowy:

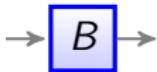


Każda reguła produkcji w zapisie BNF może zostać przekształcona na diagram składniowy o strukturze wynikającej z następujących przekształceń:

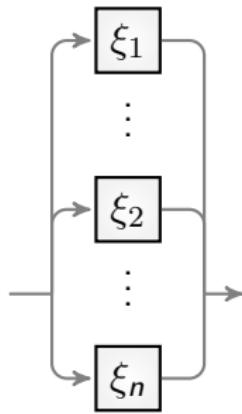
- każdy symbol końcowy  $x$  zostaje przekształcony w następujący schemat (kółko może zostać zastąpione prostokątem o zaokrąglonych bokach, gdy nazwa symbolu jest długa):



- każde wystąpienie zmiennej  $B$  jest reprezentowane przez łuk z etykietą w kwadracie:



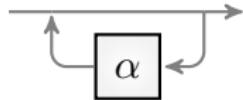
- Produkcja postaci  $\langle A \rangle ::= \xi_1 \mid \dots \mid \xi_n$  jest zamieniania na poniższy diagram, w którym każde  $\xi_i$  jest zastąpione przez odpowiedni diagram składowy zgodnie z regułami:



- Ciąg  $\xi$  postaci  $\alpha_1\alpha_2\dots\alpha_m$  jest przekształcany na diagram:



- Rozszerzenie BNF:  $\{\alpha\}$  oznacza dowolną liczbę powtórzeń  $\alpha$  ( $\alpha^*$  jako wyrażenie regularne). Każde  $\xi = \{\alpha\}$  przekształcamy w diagram:





Mamy daną gramatykę bezkontekstową  $G = (V, \Sigma, P, S)$  i regułę produkcji  $A \rightarrow \gamma$ , gdzie  $A \in V$ ,  $\gamma \in (V \cup \Sigma)^*$ . Przekształcenie ciągu symboli  $\alpha A \beta$  na ciąg  $\alpha \gamma \beta$ , gdzie  $\alpha, \beta \in (V \cup \Sigma)^*$ , zapisywane jest jako  $\alpha A \beta \xrightarrow[G]{\gamma} \alpha \gamma \beta$ . Ciąg takich przekształceń oznaczamy za pomocą symbolu  $\Rightarrow_G^*$ . Jeśli nie powoduje to niejednoznaczności, indeks  $G$  można pominąć.

Ciąg przekształceń  $S \xrightarrow[G]{\gamma} w$ ,  $w \in \Sigma^*$  nazywamy **wyprowadzeniem**. Jeśli dla każdego przekształcenia  $\alpha \in \Sigma^*$  (rozwijamy zawsze pierwszą od lewej zmienną), to wyprowadzenie nazywamy **lewostronnym**. Jeśli dla każdego przekształcenia  $\beta \in \Sigma^*$  (rozwijamy pierwszą od prawej zmienną), to wyprowadzenie nazywamy **prawostronnym**.



**Językiem bezkontekstowym** nazywamy zbiór wszystkich słów, które dadzą się wyprowadzić z gramatyki bezkontekstowej  $G = (V, \Sigma, P, S)$ :

$$L(G) = \{w \in \Sigma^* : S \xrightarrow[G]{*} w\}$$

Każdy ciąg symboli  $\alpha \in (\Sigma \cup V)^*$  taki że  $S \xrightarrow[G]{*} \alpha$  nazywamy **formą zdaniową**. Analogicznie jak dla wyprowadzeń, istnieją **lewostronne** i **prawostonne** formy zdaniowe. Powstają przez rozwijanie pierwszej od odpowiednio lewej lub prawej strony zmiennej we wcześniejszym wyprowadzeniu.



## Drzewo wyprowadzenia

**Drzewo wyprowadzenia** dla gramatyki  $G = (V, \Sigma, P, S)$  to dowolne drzewo spełniające następujące warunki:

- ① każdy wierzchołek wewnętrzny etykietowany jest zmienną z  $V$
- ② każdy liść etykietowany jest symbolem końcowym lub  $\varepsilon$ , przy czym liść etykietowany  $\varepsilon$  musi być jedynym dzieckiem swojego rodzica
- ③ dla każdego wierzchołka wewnętrznego etykietowanego zmienną  $A \in V$  i jego dzieci  $X_1, X_2, \dots, X_k$  istnieje produkcja  $A \rightarrow X_1, X_2, \dots, X_k$  w  $P$ .

Ciąg liści drzewa wyprowadzenia nazywamy **plonem**.

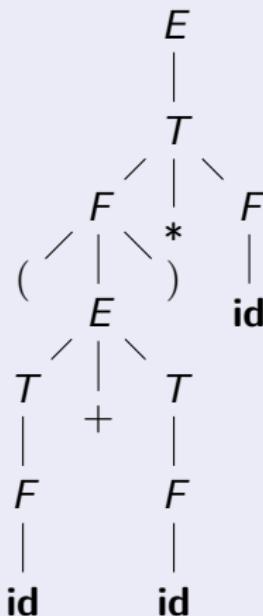
## gramatyka

$$\begin{aligned} E &\rightarrow T + T \mid T \\ T &\rightarrow F * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## zdanie

$$(\text{id} + \text{id}) * \text{id}$$

## drzewo wyprowadzenia





**Gramatyka jednoznaczna** to taka gramatyka, w której dla każdego słowa należącego do języka generowanego przez tę gramatykę istnieje jedno drzewo wyprowadzenia, którego plon jest tym słowem.

Gramatyka, która nie jest jednoznaczna jest **wieloznaczna**.

Dla wielu gramatyk wieloznacznych można znaleźć gramatyki jednoznaczne, które generują ten sam język bezkontekstowy. Jednak istnieją języki, dla których wszystkie gramatyki są wieloznaczne. Są to języki **ściśle wieloznaczne**.

Analizator składniowy otrzymuje od analizatora leksykalnego ciąg symboli końcowych i sprawdza, czy i jak można je otrzymać z symbolu początkowego gramatyki stosując reguły produkcji. Jeśli ciągu wejściowego nie da się wyprowadzić, analizator składniowy powinien wskazać przyczynę błędu i kontynuować analizę w taki sposób, by wykryć ewentualne inne błędy składniowe.

Analiza może być:

- **zstępująca** — zaczynamy od symbolu początkowego gramatyki i stosujemy reguły produkcji do czasu otrzymania ciągu symboli końcowych; ten rodzaj analizy stosowany jest głównie w analizatorach budowanych ręcznie
- **wstępująca** — zaczynamy od ciągu symboli końcowych i stosujemy redukcje ciągu symboli występujących po prawej stronie reguły produkcji zastępując je zmienną występującą po lewej stronie tej reguły produkcji; ten rodzaj analizy stosowany jest głównie w analizatorach tworzonych z użyciem narzędzi takich jak `bison`.



## Analiza zstępująca

Program analizatora składniowego zstępującego można tworzyć bezpośrednio z diagramu składniowego. W praktyce stosuje się gramatyki spełniające następujące wymagania:

- ① Dla każdej zadanej produkcji

$$A ::= \xi_1 \mid \xi_2 \mid \dots \mid \xi_n$$

zbiory pierwszych symboli w zdaniach, które mogą być wyprowadzone z  $\xi_i$  muszą być rozłączne, tzn.

$$\text{pierw}(\xi_i) \cap \text{pierw}(\xi_j) = \emptyset \text{ dla wszystkich } i \neq j$$

- ② Dla każdego symbolu  $A \in V$ , z którego można wyprowadzić pusty ciąg symboli, tzn.  $A \Rightarrow^* \varepsilon$ , zbiór jego pierwszych symboli musi być rozłączny ze zbiorem symboli, które mogą występować po dowolnym ciągu wyprowadzonym z  $A$ , tzn.,  
 $\text{pierw}(A) \cap \text{nast}(A) = \emptyset$

Gramatyka taka nazywa się  $LL(1)$ , gdzie pierwsze  $L$  pochodzi od analizy od lewej (Left), drugie od lewostronnego (Leftmost) wyprowadzenia, a 1 od podglądu 1 symbolu.



Zbiór  $\text{pierw}(\xi)$  jest zbiorem wszystkich symboli końcowych, które mogą wystąpić na pierwszej pozycji w formach zdaniowych wyprowadzonych z  $\xi$ . Zbiór  $\text{pierw}(\xi)$  wyznaczamy na podstawie następujących zasad:

- ① Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi) = \{a\}$
- ② Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- ③ Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1 \xi_2 \cdots \xi_k$  jest produkcją, to należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .



Do obliczenia  $nast(\xi)$  stosujemy następujące reguły

- ① W  $nast(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- ② Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\alpha, \beta \in (V \cup \Sigma)^*$ , to wszystkie symbole z  $pierw(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $nast(B)$ .
- ③ Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in pierw(\beta)$ , to wszystkie symbole z  $nast(A)$  są w  $nast(B)$ .

Konstrukcja analizatora zstępującego dla diagramu składniowego przebiega według następujących reguł, w których  $T(S)$  oznacza instrukcję otrzymaną z diagramu  $S$ :

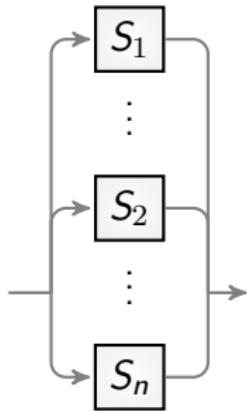
- ① Zredukuj, na ile to możliwe, liczbę diagramów, stosując odpowiednie podstawienia.
- ② Każdy diagram zastąp deklaracją procedury zgodnie z dalszymi regułami.
- ③ Ciąg elementów postaci:



jest przekształcany na instrukcję postaci:

{  $T(S_1); T(S_2); \dots ; T(S_n);$  }

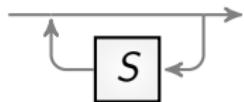
- ④ Diagram alternatywy jest zamieniany na instrukcję wyboru



```
switch (sym) {  
    case L1: T(S1);  
    case L2: T(S2);  
    ...  
    case Ln: T(Sn);  
}
```

gdzie  $L_i$  oznacza zbiór symboli początkowych konstrukcji  $S_i$ , tzn.  $L_i = \text{pierw}(S_i)$ . Oczywiście zamiast instrukcji wyboru można użyć instrukcji warunkowej.

- ④ Diagram pętli zamieniamy na instrukcję while



**while** (*sym* **in** *L*) *T(S)*;

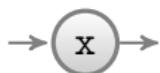
gdzie  $L = \text{pierw}(S)$ , zaś **in** przynależność do zbioru.

- ⑤ Element diagramu oznaczający inny diagram *A* zastępujemy wywołaniem procedury *A*



*A()*;

- ⑥ Element diagramu oznaczający symbol końcowy *x* zastępujemy poniższą instrukcją



**if** (*sym* == *x*) *sym* = *pobsym()*; **else**  
*błąd*;



Gramatyka  $LL(1)$  wprowadza ograniczenia na postać reguł produkcji. Jeśli by tych ograniczeń nie było, mogłyby to spowodować kłopoty.

Kłopoty może sprawiać np. lewostronna rekurencja. Rozważmy regułę:

$$\text{lista} \rightarrow \text{lista } ', ' \text{ element} \mid \text{element}$$

Na wejściu mamy `element`. Rozwijamy `lista` używając pierwszego wariantu i otrzymując `lista ', ' element`. Na pierwszym miejscu mamy znów `lista`. Rozwijamy go i znów na pierwszym miejscu mamy `lista...`. W regule są dwa warianty, ale w pierwszym wariantie `element` należy do `pierw(lista)`.

W tym przypadku można zamienić rekurencję na prawostronną, która nie sprawia kłopotu dla analizy zstępującej metodą zejść rekurencyjnych:

$$\text{lista} \rightarrow \text{element } ', ' \text{ lista} \mid \text{element}$$



Lewostronną rekurencję można usunąć przepisując odpowiednio reguły.  
Rozważmy regułę postaci:

$$A \rightarrow A\alpha \mid \beta$$

gdzie  $\alpha, \beta \in (\Sigma \cup V)$  to takie ciągi zmiennych i symboli końcowych, które nie zaczynają się od  $A$ . Lewostronną rekurencję można usunąć przepisując reguły następująco:

$$A \rightarrow \beta R$$

$$R \rightarrow \alpha R \mid \varepsilon$$



Rozważmy jeszcze raz gramatykę listy:

$\text{lista} \rightarrow \text{lista} \cdot \cdot \text{element} \mid \text{element}$

mamy  $A = \text{lista}$ ,  $\alpha = \cdot \cdot \text{element}$  i  $\beta = \text{element}$ . Możemy zatem przepisać gramatykę do postaci:

$\text{lista} \rightarrow \text{element } R$

$R \rightarrow \cdot \cdot \text{element } R \mid \varepsilon$

Ta gramatyka poprawnie rozpoznaje składnię listy.



## Ogólny algorytm usuwania lewostronnej rekurencji

- 1: Ustaw zmienne w pewnym porządku  $A_1, A_2, \dots, A_n$
- 2: **for**  $i := 1$  **to**  $n$  **do**
- 3:     **for**  $j := 1$  **to**  $n - 1$  **do**
- 4:         każdą produkcję postaci  $A_i \rightarrow A_j \gamma$  zamień na produkcje postaci  $A_i \rightarrow \delta_1 \gamma | \delta_2 \gamma | \dots | \delta_k \gamma$ , gdzie  $A_j \rightarrow \delta_1 | \delta_2 | \dots | \delta_k$  są wszystkimi aktualnymi produkcjami dla  $A_j$
- 5:     **end for**
- 6:     Usuń lewostronną rekurencję dla  $A_i$
- 7: **end for**



Innym kłopotem dla analizatora zstępującego używającego metody zejść rekurencyjnych jest niejednoznaczność występująca wtedy, gdy ten sam symbol końcowy może wystąpić jako pierwszy w różnych wariantach reguły produkcji dla określonej zmiennej. Rozwiązaniem jest opóźnianie decyzji do chwili, gdy możemy ją podjąć, tzn. dopiero po odczytaniu symbolu, gdy na wejściu mamy symbol, który pozwoli nam wybrać odpowiednią gałąź.

Metoda ta nazywa się **faktoryzacją lewostronną**. Polega na przekształceniu reguł produkcji postaci:

$$A \rightarrow \alpha\beta_1|\alpha\beta_2| \dots |\alpha\beta_n|\gamma$$

na reguły postaci:

$$A \rightarrow \alpha A' |\gamma$$

$$A' \rightarrow \beta_1|\beta_2| \dots |\beta_n$$



Rozpatrzmy składnię języka C. Na najwyższym poziomie mogą występować zarówno deklaracje funkcji (prototypy), jak i definicje funkcji. Możemy więc napisać:

$S \rightarrow \text{deklaracja\_funkcji} \mid \text{definicja\_funkcji} \mid \dots$

ale obie konstrukcje zaczynają się tak samo:

$\text{deklaracja\_funkcji} \rightarrow \text{nagłówek\_funkcji} ;$

$\text{definicja\_funkcji} \rightarrow \text{nagłówek\_funkcji} \text{ deklaracje\_blok}$

stąd lepiej napisać:

$S \rightarrow \text{nagłówek\_funkcji} \text{ reszta\_funkcji} \mid \dots$

$\text{reszta\_funkcji} \rightarrow ; \mid \text{reszta\_definicji\_funkcji}$



Analizator nie powinien przerywać analizy po napotkaniu pierwszego błędu. Jednak jeden błąd może spowodować lawinę dalszych, sztucznych błędów wynikających z ponownego rozpoczęcia analizy w nieodpowiednim miejscu. Dlatego ważnym jest, by analizę wznowić od miejsca, gdzie jest to najlepsze.

Do obsługi błędów używamy procedury TEST wywoływanej na końcu procedury obsługi danej konstrukcji składniowej. Procedura ma trzy parametry:

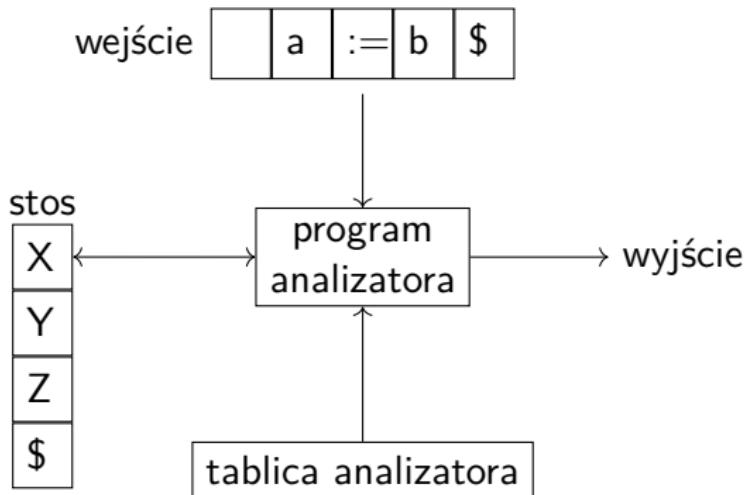
- ①  $s1$  — zbiór symboli, które mogą wystąpić po danej konstrukcji
- ②  $s2$  — zbiór symboli, które zaczynają ważne konstrukcje składniowe i których nie można pominąć
- ③  $n$  — identyfikator (numer) błędu

## procedura obsługi błędów

```
1: procedure TEST(zbiorsym s1, zbiorsym s2, const int n)
2:   if sym  $\notin$  s1 then
3:     BŁĄD(n)
4:     s1 = s1  $\cup$  s2
5:   while sym  $\notin$  s1 do
6:     POBSYM
7:   end while
8:   end if
9: end procedure
```

Zmienna *sym* jest zmienną globalną. Procedura POBSYM pobiera kolejny symbol z wejścia i umieszcza w zmiennej *sym*.

W analizie zchodzącej metoda zejść rekurencyjnych nie jest jedyną możliwą. Rezygnując z rekurencji należy wprowadzić jawnego stos.





Części analizatora:

- bufor wejściowy — zawiera ciąg symboli do analizowania, zakończony symbolem końca ciągu wejściowego (tutaj oznaczony jako \$), który nie należy do alfabetu
- stos — zawiera zmienne gramatyki oraz symbol \$; w momencie rozpoczętania analizy zawiera symbol \$, który w tym miejscu oznacza koniec stosu, a nad nim symbol początkowy gramatyki
- tablica analizatora — jest dwuwymiarową tablicą  $M[A, a]$ , gdzie  $A$  to zmienna gramatyki, a  $a$  jest symbolem końcowym lub symbolem końca danych \$, zaś wartościami są reguły produkcji

## Algorytm wypełniania tablicy analizatora

```
1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\varepsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:      dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:    end if
12:  end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję  $M$ 
```



## Algorytm analizy składniowej analizatora nierekurencyjnego

Niech  $X$  będzie symbolem na wierzchołku stosu

Niech  $a$  będzie bieżącym symbolem na wejściu

```
1: repeat
2:   if  $X \in \Sigma \vee X = \$$  then
3:     if  $X = a$  then
4:       Zdemij  $X$  ze stosu. Przesuń wejście o jeden symbol do przodu
5:     else
6:       Zgłoś błąd
7:     end if
8:   else
9:     if  $M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$  then
10:      zdejmij  $X$  ze stosu
11:      odłóż  $Y_k Y_{k-1} \dots Y_1$  na stos ( $Y_1$  na wierzchołku)
12:    else
13:      Zgłoś błąd
14:    end if
15:  end if
16: until  $X = \$$ 
```

Rozważmy uproszczoną gramatykę wyrażeń arytmetycznych:

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

Po usunięciu lewostronnej rekurencji otrzymamy:

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi) = \{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw(A)*

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## *nast(A)*

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw(A)*

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast(A)*

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw(A)*

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ ( \quad ) \} \end{aligned}$$

## *nast(A)*

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw(A)*

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast(A)*

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw(A)*

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ (, \text{id}) \} \end{aligned}$$

## *nast(A)*

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw(A)*

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast(A)*

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ (, \text{id}) \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ (, \text{id}) \} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi) = \{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1 \xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{(, \text{id}\} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{(, \text{id}\} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{(, \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi) = \{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1 \xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{(, \text{id}\} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{(, \text{id}\} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{(, \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow + \quad | \quad \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \quad | \quad \varepsilon \\ F &\rightarrow (E) \quad | \quad \text{id} \end{aligned}$$

## *pierw(A)*

$$\begin{aligned} E &\mapsto \{( , \text{id} \} \\ E' &\mapsto \{ + \} \\ T &\mapsto \{( , \text{id} \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{( , \text{id} \} \end{aligned}$$

## *nast(A)*

$$\begin{aligned} E &\mapsto \{ \} \\ E' &\mapsto \{ \} \\ T &\mapsto \{ \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{ \} \end{aligned}$$

## Obliczanie *pierw(A)*

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast(A)*

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id} \} \\ E' &\mapsto \{ +, \varepsilon \} \\ T &\mapsto \{( , \text{id} \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{( , \text{id} \} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \} \\ E' &\mapsto \{ \} \\ T &\mapsto \{ \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{ \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi) = \{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1 \xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{(, \text{id}\}) \\ E' &\mapsto \{+, \varepsilon\} \\ T &\mapsto \{(, \text{id}\}) \\ T' &\mapsto \{\} \\ F &\mapsto \{(, \text{id}\}) \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{\} \\ E' &\mapsto \{\} \\ T &\mapsto \{\} \\ T' &\mapsto \{\} \\ F &\mapsto \{\} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ * \} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \} \\ E' &\mapsto \{ \} \\ T &\mapsto \{ \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{ \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{+ , \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{^* , \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{(, \text{id}\}) \\ E' &\mapsto \{+, \varepsilon\} \\ T &\mapsto \{(, \text{id}\}) \\ T' &\mapsto \{*, \varepsilon\} \\ F &\mapsto \{(, \text{id}\}) \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \quad \} \\ E' &\mapsto \{ \quad \} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{(, \text{id}\}) \\ E' &\mapsto \{+, \varepsilon\} \\ T &\mapsto \{(, \text{id}\}) \\ T' &\mapsto \{*, \varepsilon\} \\ F &\mapsto \{(, \text{id}\}) \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{\$ \ } \\ E' &\mapsto \{ \ } \\ T &\mapsto \{ \ } \\ T' &\mapsto \{ \ } \\ F &\mapsto \{ \ } \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{+ , \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{^* , \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$ , )\} \\ E' &\mapsto \{ \} \\ T &\mapsto \{ \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{ \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ *, \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$, )\} \\ E' &\mapsto \{ \} \\ T &\mapsto \{ \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{ \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ *, \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$, )\} \\ E' &\mapsto \{ \$, )\} \\ T &\mapsto \{ \quad \} \\ T' &\mapsto \{ \quad \} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ *, \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$, )\} \\ E' &\mapsto \{ \$, )\} \\ T &\mapsto \{ \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{ \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ *, \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$, )\} \\ E' &\mapsto \{ \$, )\} \\ T &\mapsto \{ + \} \\ T' &\mapsto \{ \} \\ F &\mapsto \{ \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow \textcolor{red}{TE} \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{+ , \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{^* , \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$ , ) \} \\ E' &\mapsto \{ \$ , ) \} \\ T &\mapsto \{ + , ) , \$ \} \\ T' &\mapsto \{ \quad \quad \quad \} \\ F &\mapsto \{ \quad \quad \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ *, \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$, )\} \\ E' &\mapsto \{ \$, )\} \\ T &\mapsto \{ +, ), \$\} \\ T' &\mapsto \{ \quad \quad \quad \} \\ F &\mapsto \{ \quad \quad \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ *, \varepsilon\} \\ F &\mapsto \{ ( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$, )\} \\ E' &\mapsto \{ \$, )\} \\ T &\mapsto \{ +, ), \$\} \\ T' &\mapsto \{ +, ), \$\} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ *, \varepsilon\} \\ F &\mapsto \{ ( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$, )\} \\ E' &\mapsto \{ \$, )\} \\ T &\mapsto \{ +, ), \$\} \\ T' &\mapsto \{ +, ), \$\} \\ F &\mapsto \{ \quad \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{+ , \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{^* , \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$ , ) \} \\ E' &\mapsto \{ \$ , ) \} \\ T &\mapsto \{ + , ) , \$ \} \\ T' &\mapsto \{ + , ) , \$ \} \\ F &\mapsto \{ ^* \} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon \in \text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

Policzmy zbiory *pierw* i *nast*.

## Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

## *pierw*(A)

$$\begin{aligned} E &\mapsto \{( , \text{id}\} \\ E' &\mapsto \{ +, \varepsilon\} \\ T &\mapsto \{( , \text{id}\} \\ T' &\mapsto \{ *, \varepsilon\} \\ F &\mapsto \{( , \text{id}\} \end{aligned}$$

## *nast*(A)

$$\begin{aligned} E &\mapsto \{ \$, )\} \\ E' &\mapsto \{ \$, )\} \\ T &\mapsto \{ +, ), \$\} \\ T' &\mapsto \{ +, ), \$\} \\ F &\mapsto \{ *, +, ), \$\} \end{aligned}$$

## Obliczanie *pierw*(A)

- 1 Jeśli pierwszy symbol argumentu jest symbolem końcowym  $a$ , to  $\text{pierw}(a\xi)=\{a\}$
- 2 Jeśli  $\xi \rightarrow \varepsilon$  jest produkcją, to należy dodać  $\varepsilon$  do  $\text{pierw}(\xi)$
- 3 Jeśli  $\xi$  jest zmienną i  $\xi \rightarrow \xi_1\xi_2 \dots \xi_k$  jest produkcją, to  $a$  należy umieścić w  $\text{pierw}(\xi)$ , jeśli istnieje takie  $i$ , że  $a$  jest w  $\text{pierw}(\xi_i)$ , a  $\varepsilon$  jest we wszystkich  $\text{pierw}(\xi_1), \dots, \text{pierw}(\xi_{i-1})$ . Jeśli  $\varepsilon$  jest w  $\text{pierw}(\xi_j)$  dla wszystkich  $j = 1, \dots, k$ , to do  $\text{pierw}(\xi)$  należy dodać  $\varepsilon$ .

## Obliczanie *nast*(A)

- 1 W  $\text{nast}(S)$ , gdzie  $S$  jest symbolem początkowym, należy umieścić znacznik końca wejścia  $\$$ .
- 2 Jeśli mamy produkcję  $A \rightarrow \alpha B \beta$ , to wszystkie symbole z  $\text{pierw}(\beta)$  z wyjątkiem  $\varepsilon$  należy umieścić w  $\text{nast}(B)$ .
- 3 Jeśli mamy produkcję  $A \rightarrow \alpha B$  albo produkcję  $A \rightarrow \alpha B \beta$ , gdzie  $\varepsilon \in \text{pierw}(\beta)$ , to wszystkie symbole z  $\text{nast}(A)$  są w  $\text{nast}(B)$ .

## Tablica analizatora

zm.	symbol wejściowy						
	<b>id</b>	+	*	(	)	\$	

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\varepsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:       dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:     end if
12:   end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \varepsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \varepsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ . , +, \varepsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ . , *, \varepsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, . \} \\
 E' &\mapsto \{ \$, . , + \} \\
 T &\mapsto \{ . , +, \$ \} \\
 T' &\mapsto \{ . , +, \$ \} \\
 F &\mapsto \{ . , +, \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy						
	id	+	*	(	)	\$	
E							

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\varepsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:       dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:     end if
12:   end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \varepsilon$$

$$F \rightarrow (E) \mid id$$

### pierw( $A$ )

$$E \mapsto \{ , id \}$$

$$E' \mapsto \{ +, \varepsilon \}$$

$$T \mapsto \{ , id \}$$

$$T' \mapsto \{ *, \varepsilon \}$$

$$F \mapsto \{ (, id \}$$

### nast( $A$ )

$$E \mapsto \{ \$, \} \}$$

$$E' \mapsto \{ \$, ) \}$$

$$T \mapsto \{ +, \$, \}$$

$$T' \mapsto \{ +, ), \$ \}$$

$$F \mapsto \{ *, +, ), \$ \}$$

## Tablica analizatora

zm.	symbol wejściowy					
	id	+	*	(	)	\$
$E$					$E \rightarrow TE'$	

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\varepsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:       dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:     end if
12:   end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

### pierw( $A$ )

$E \mapsto \{ (, id\}$

$E' \mapsto \{ +, \varepsilon \}$

$T \mapsto \{ (, id\}$

$T' \mapsto \{ *, \varepsilon \}$

$F \mapsto \{ (, id\}$

### nast( $A$ )

$E \mapsto \{ \$, \}$

$E' \mapsto \{ \$, ) \}$

$T \mapsto \{ +, \$ \}$

$T' \mapsto \{ +, ), \$ \}$

$F \mapsto \{ *, +, ), \$ \}$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$				$E \rightarrow TE'$	

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\varepsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:       dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:     end if
12:   end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \varepsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \varepsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ . , +, \varepsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ . , *, \varepsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$ \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ . , +, \$ \} \\
 T' &\mapsto \{ . , , \$ \} \\
 F &\mapsto \{ . , +, , \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$				$E \rightarrow TE'$	
$E'$						

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\varepsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:   if  $\$ \in nast(A)$  then
10:    dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:  end if
12: end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \varepsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \varepsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ . , + \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ . , *, \varepsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$ \} \\
 E' &\mapsto \{ \$ , ) \} \\
 T &\mapsto \{ + , \$ \} \\
 T' &\mapsto \{ + , ) , \$ \} \\
 F &\mapsto \{ * , + , ) , \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	id	+	*	(	)	\$
$E$	$E \rightarrow TE'$				$E \rightarrow TE'$	
$E'$		$E' \rightarrow +TE'$				

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\varepsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:   if  $\$ \in nast(A)$  then
10:    dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:  end if
12: end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \varepsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \varepsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ . , +, \varepsilon \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$ \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ . , +, \$ \} \\
 F &\mapsto \{ . , id \} \\
 F &\mapsto \{ . , *, +, \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$				$E \rightarrow TE'$	
$E'$		$E' \rightarrow +TE'$				$E' \rightarrow \varepsilon$

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\varepsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:      dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:    end if
12:  end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \varepsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \varepsilon \\ F &\rightarrow (E) \mid id \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned} E &\mapsto \{ . , id \} \\ E' &\mapsto \{ . , +, \varepsilon \} \\ T &\mapsto \{ . , id \} \\ T' &\mapsto \{ . , *, \varepsilon \} \\ F &\mapsto \{ . , id \} \end{aligned}$$

### nast( $A$ )

$$\begin{aligned} E &\mapsto \{ \$ \} \\ E' &\mapsto \{ \$, ) \} \\ T &\mapsto \{ . , +, \$ \} \\ T' &\mapsto \{ . , +, \$ \} \\ F &\mapsto \{ . , +, \$ \} \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:   end for
9:   if  $\$ \in nast(A)$  then
10:    dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:  end if
12: end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \epsilon \\ F &\rightarrow (E) \mid id \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned} E &\mapsto \{ . , id \} \\ E' &\mapsto \{ +, \epsilon \} \\ T &\mapsto \{ . , id \} \\ T' &\mapsto \{ *, \epsilon \} \\ F &\mapsto \{ . , id \} \end{aligned}$$

### nast( $A$ )

$$\begin{aligned} E &\mapsto \{ \$, \} \\ E' &\mapsto \{ \$, ) \} \\ T &\mapsto \{ +, \$ \} \\ T' &\mapsto \{ +, \$ \} \\ F &\mapsto \{ *, +, \$ \} \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	
$T$						$E' \rightarrow \epsilon$

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:   if  $\$ \in nast(A)$  then
10:    dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:  end if
12: end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, . , \$ \} \\
 T' &\mapsto \{ *, . , \$ \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	
$T$				$T \rightarrow FT'$		$E' \rightarrow \epsilon$

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:   if  $\$ \in nast(A)$  then
10:    dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:  end if
12: end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ (, id \} \\
 T' &\mapsto \{ *, \epsilon \} \\
 F &\mapsto \{ (, id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, \$ \} \\
 T' &\mapsto \{ +, \$ \} \\
 F &\mapsto \{ *, +, \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		$E' \rightarrow \epsilon$

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:   if  $\$ \in nast(A)$  then
10:    dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:  end if
12: end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ *, \epsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, \$ \} \\
 T' &\mapsto \{ +, \$ \} \\
 F &\mapsto \{ *, +, \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$					$E' \rightarrow \epsilon$	

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:      dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:    end if
12:  end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ *, \epsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, . , \$ \} \\
 T' &\mapsto \{ +, . , \$ \} \\
 F &\mapsto \{ *, +, . , \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$			$T' \rightarrow *FT'$			

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:      dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:    end if
12:  end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F \rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ *, \epsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, . , \$ \} \\
 T' &\mapsto \{ +, . , \$ \} \\
 F &\mapsto \{ *, +, . , \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$			

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:   end for
9:   if  $\$ \in nast(A)$  then
10:    dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:  end if
12: end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F \rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E \mapsto \{ . , id \} \\
 E' \mapsto \{ +, \epsilon \} \\
 T \mapsto \{ . , id \} \\
 T' \mapsto \{ *, \epsilon \} \\
 F \mapsto \{ (, id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E \mapsto \{ \$, \} \\
 E' \mapsto \{ \$, ) \} \\
 T \mapsto \{ +, \$, \} \\
 T' \mapsto \{ +, ), \$ \} \\
 F \mapsto \{ *, +, \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:   end for
9:   if  $\$ \in nast(A)$  then
10:    dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:  end if
12: end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M

```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F \rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ *, \epsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, \$, \} \\
 T' &\mapsto \{ +, ), \$ \} \\
 F &\mapsto \{ *, +, \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:      dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:    end if
12:  end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M
  
```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F &\rightarrow (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ *, \epsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, \$, \} \\
 T' &\mapsto \{ +, ), \$ \} \\
 F &\mapsto \{ *, +, \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
$F$						

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:      dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:    end if
12:  end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M
  
```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F \xrightarrow{\text{red}} (E) &\mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ *, \epsilon \} \\
 F &\mapsto \{ . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, \$, \} \\
 T' &\mapsto \{ +, ), \$ \} \\
 F &\mapsto \{ *, +, ), \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
$F$				$F \rightarrow (E)$		

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in pierw(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in pierw(\alpha)$  then
6:     for all  $b \in \Sigma : b \in nast(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in nast(A)$  then
10:      dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:    end if
12:  end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M
  
```

### Gramatyka

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F &\xrightarrow{\text{(red)}} (E) \mid id
 \end{aligned}$$

### pierw( $A$ )

$$\begin{aligned}
 E &\mapsto \{ . , id \} \\
 E' &\mapsto \{ +, \epsilon \} \\
 T &\mapsto \{ . , id \} \\
 T' &\mapsto \{ *, \epsilon \} \\
 F &\mapsto \{ ( . , id \}
 \end{aligned}$$

### nast( $A$ )

$$\begin{aligned}
 E &\mapsto \{ \$, \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ +, . , \$ \} \\
 T' &\mapsto \{ +, . , \$ \} \\
 F &\mapsto \{ *, +, . , \$ \}
 \end{aligned}$$

## Tablica analizatora

zm.	symbol wejściowy					
	<b>id</b>	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
$F$	$F \rightarrow \text{id}$			$F \rightarrow (E)$		

### Algorytm wypełniania tablicy analizatora

```

1: for all reguła produkcji  $A \rightarrow \alpha$  gramatyki do
2:   for all  $a \in \Sigma : a \in \text{pierw}(\alpha)$  do
3:     dodaj  $A \rightarrow \alpha$  do  $M[A, a]$ 
4:   end for
5:   if  $\epsilon \in \text{pierw}(\alpha)$  then
6:     for all  $b \in \Sigma : b \in \text{nast}(A)$  do
7:       dodaj  $A \rightarrow \alpha$  do  $M[A, b]$ 
8:     end for
9:     if  $\$ \in \text{nast}(A)$  then
10:      dodaj  $A \rightarrow \alpha$  do  $M[A, \$]$ 
11:    end if
12:  end if
13: end for
14: Wstaw błąd na każdą niezdefiniowaną pozycję M
  
```

### Gramatyka

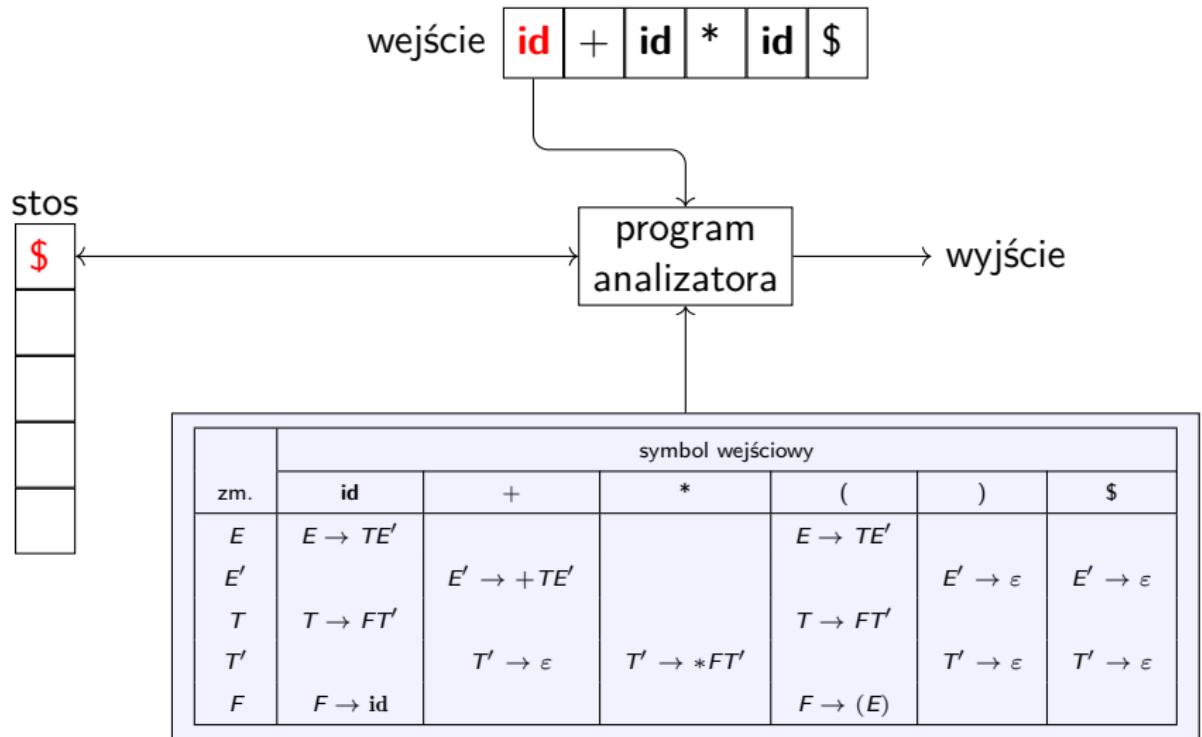
$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \epsilon \\
 F &\rightarrow (E) \mid \text{id}
 \end{aligned}$$

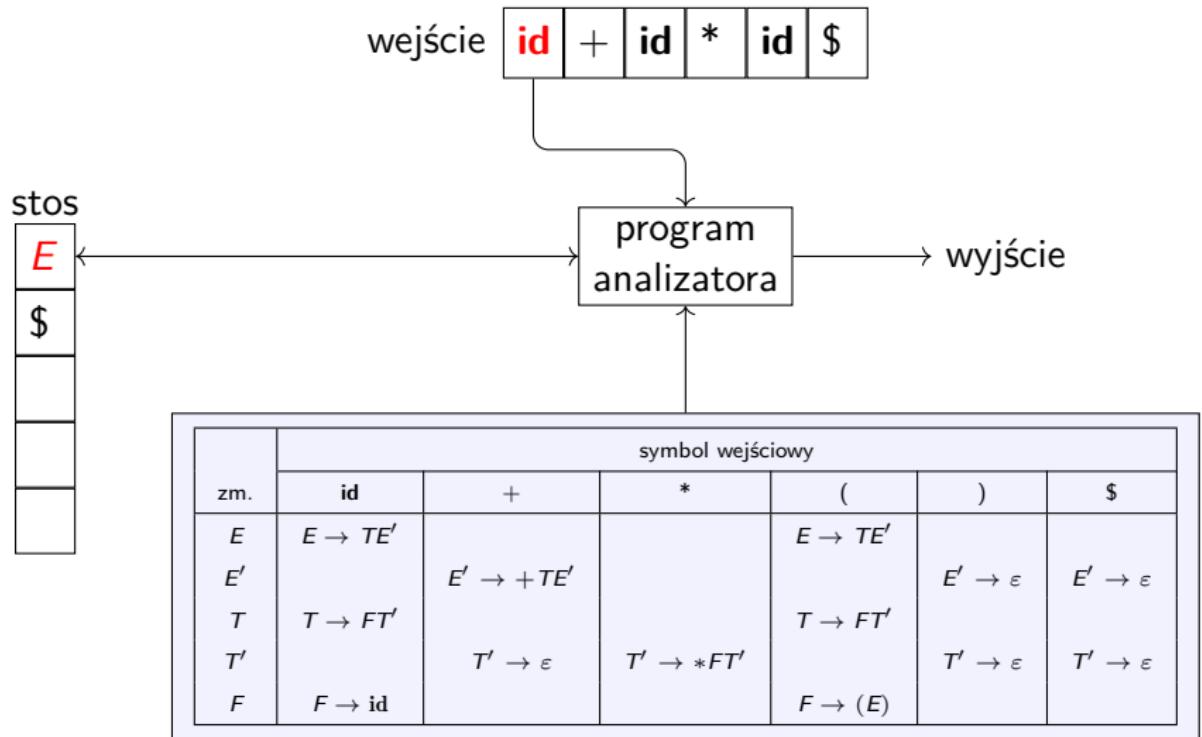
### pierw( $A$ )

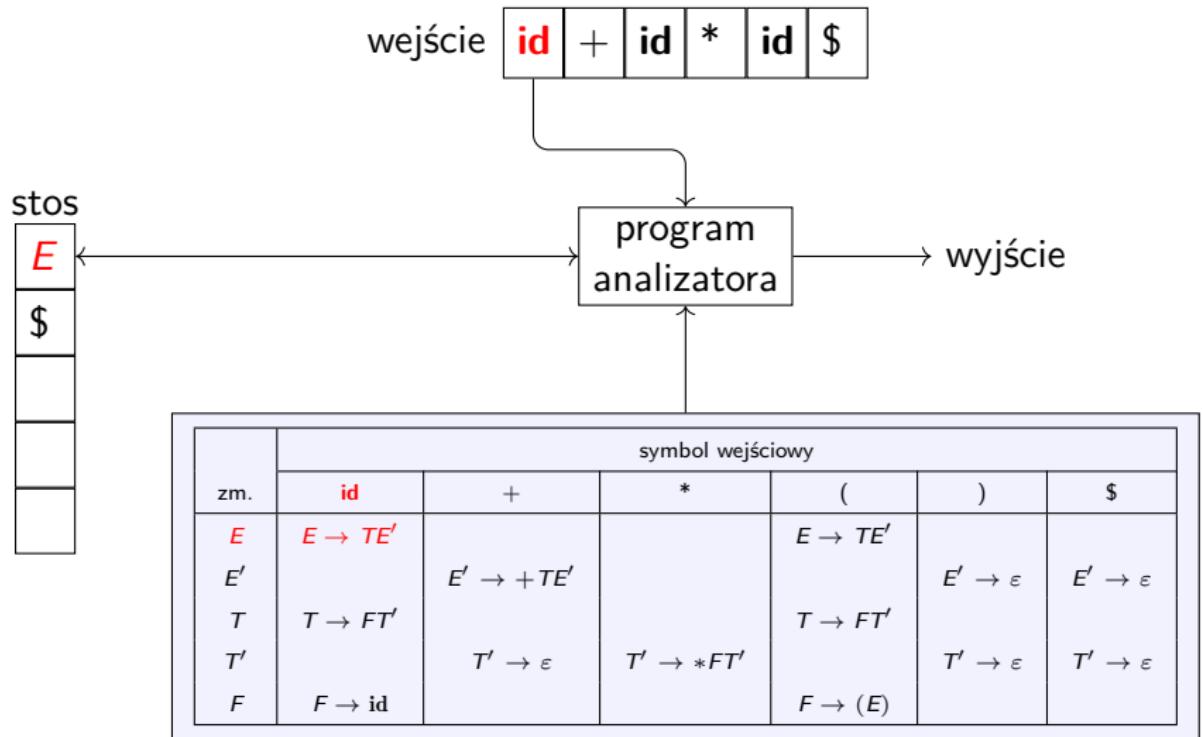
$$\begin{aligned}
 E &\mapsto \{ . , \text{id} \} \\
 E' &\mapsto \{ . , +, \epsilon \} \\
 T &\mapsto \{ . , \text{id} \} \\
 T' &\mapsto \{ . , *, \epsilon \} \\
 F &\mapsto \{ . \}
 \end{aligned}$$

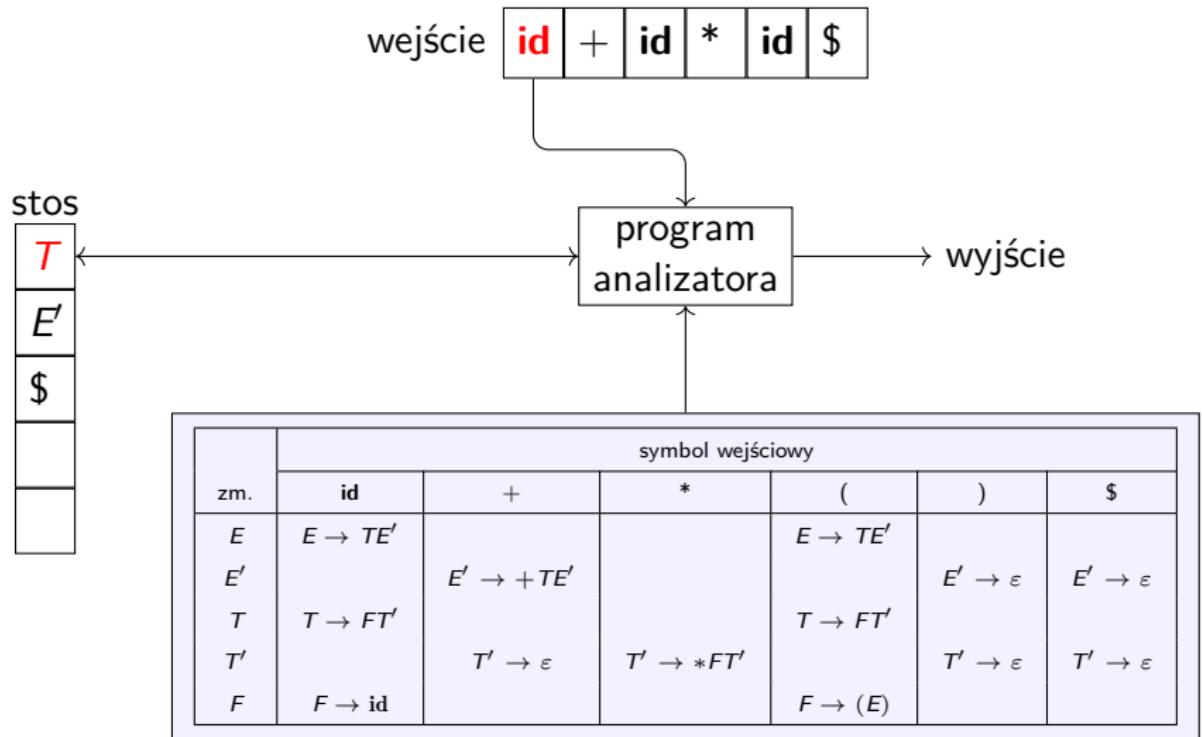
### nast( $A$ )

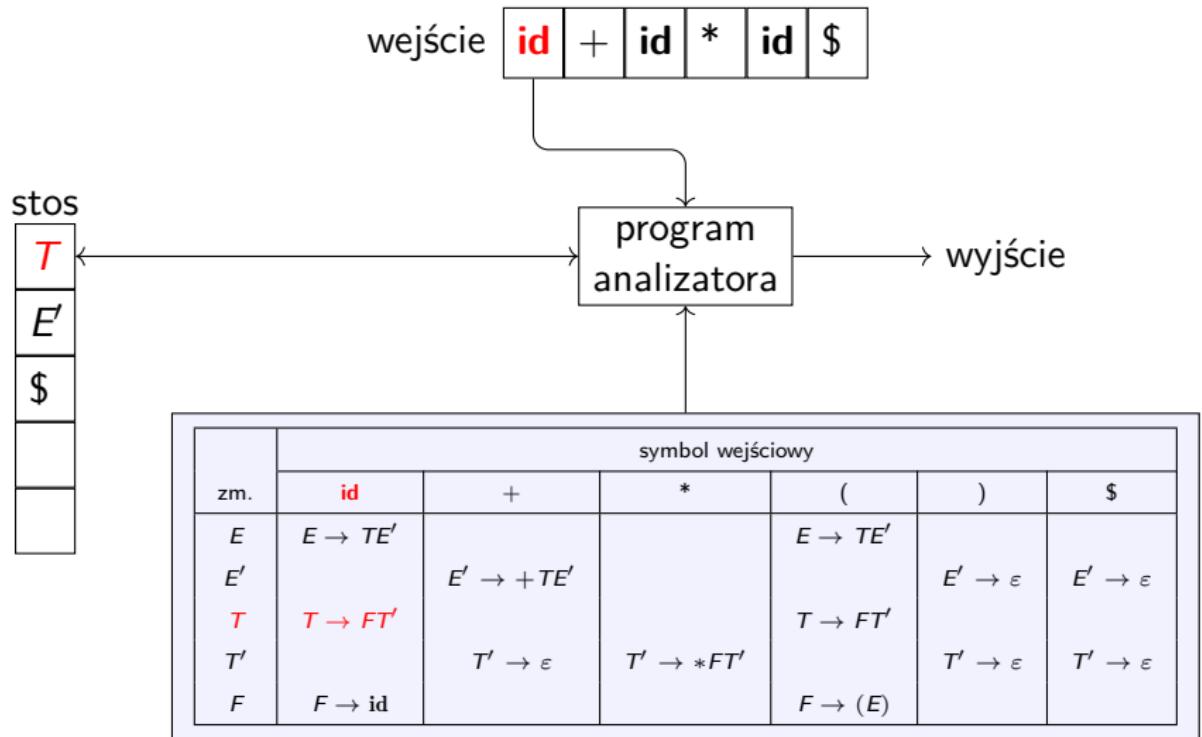
$$\begin{aligned}
 E &\mapsto \{ \$ \} \\
 E' &\mapsto \{ \$, ) \} \\
 T &\mapsto \{ . , +, \$ \} \\
 T' &\mapsto \{ . , , \$ \} \\
 F &\mapsto \{ . , +, ) \}
 \end{aligned}$$

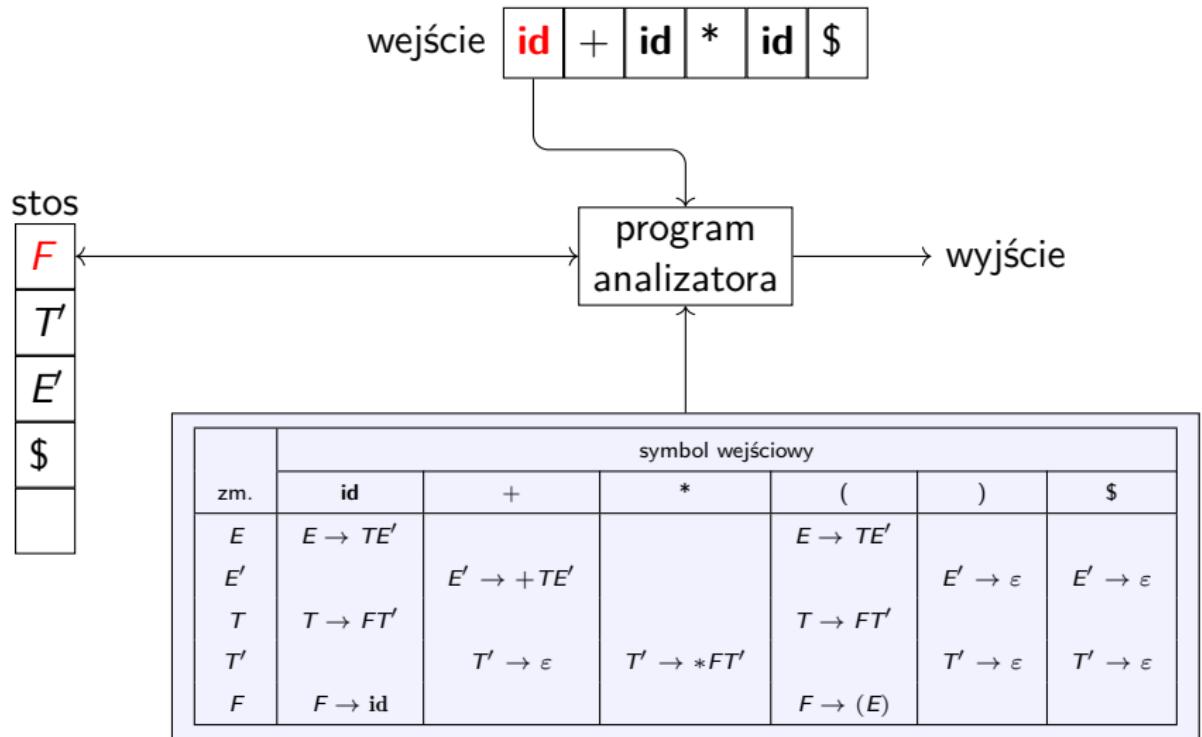


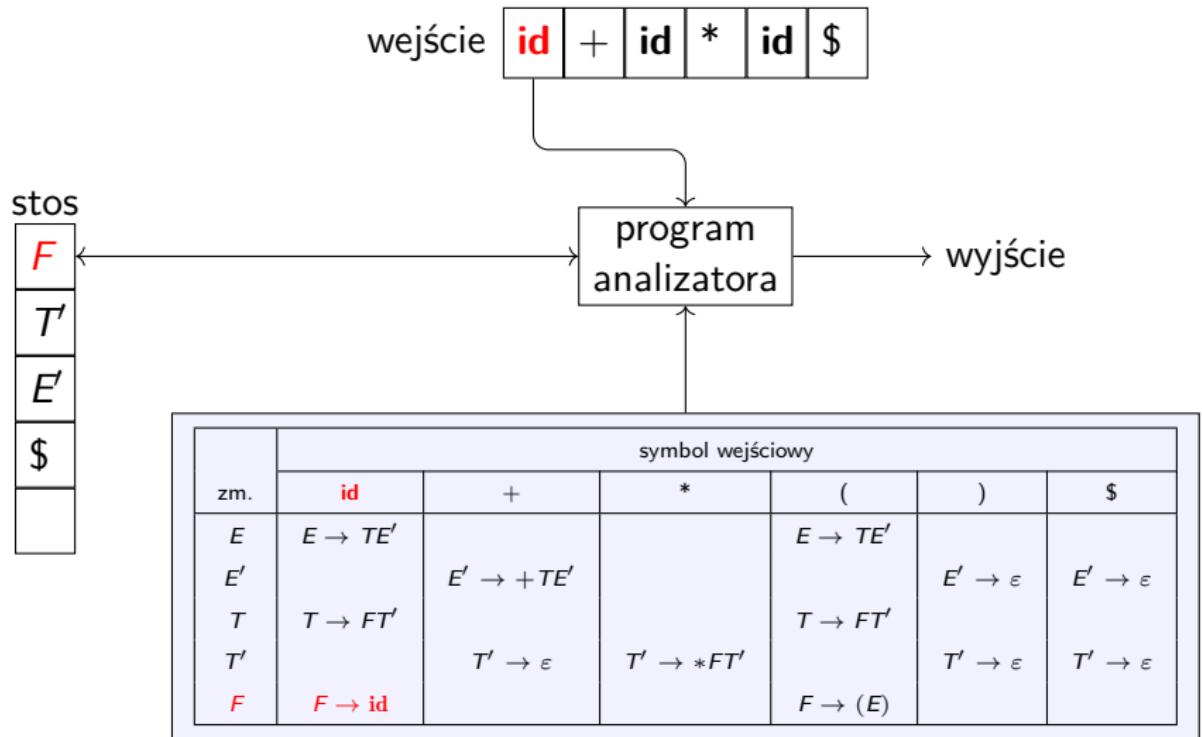


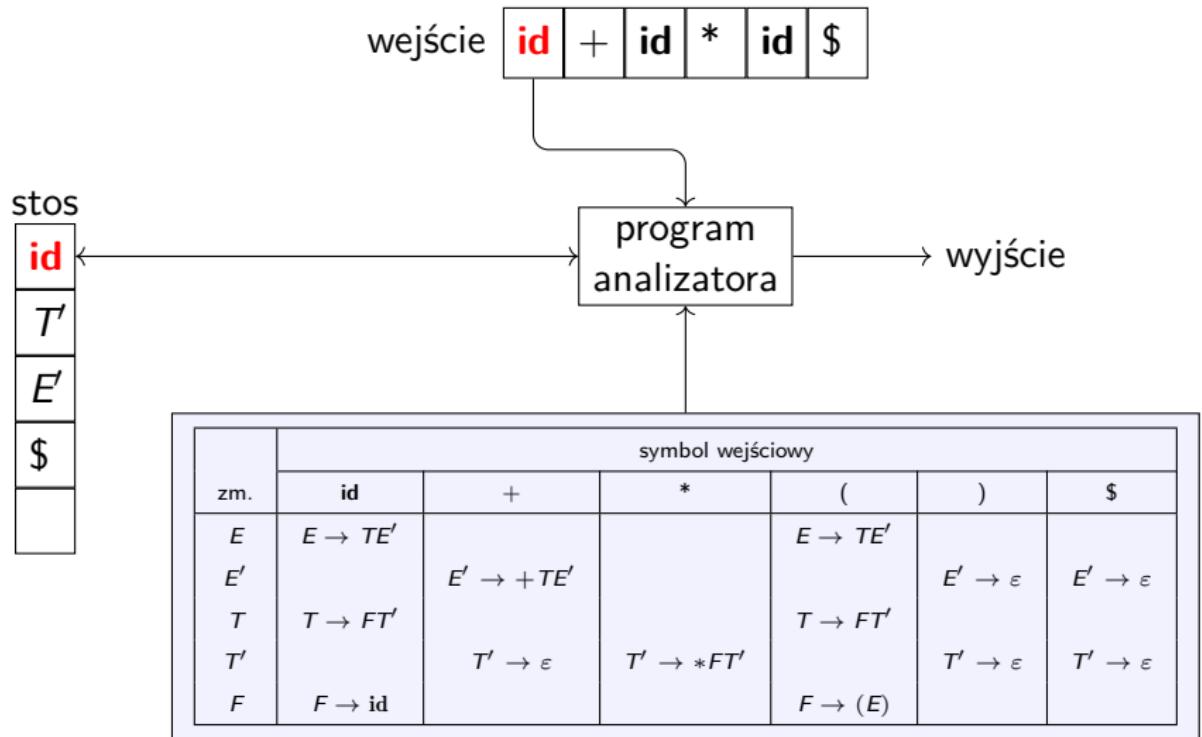


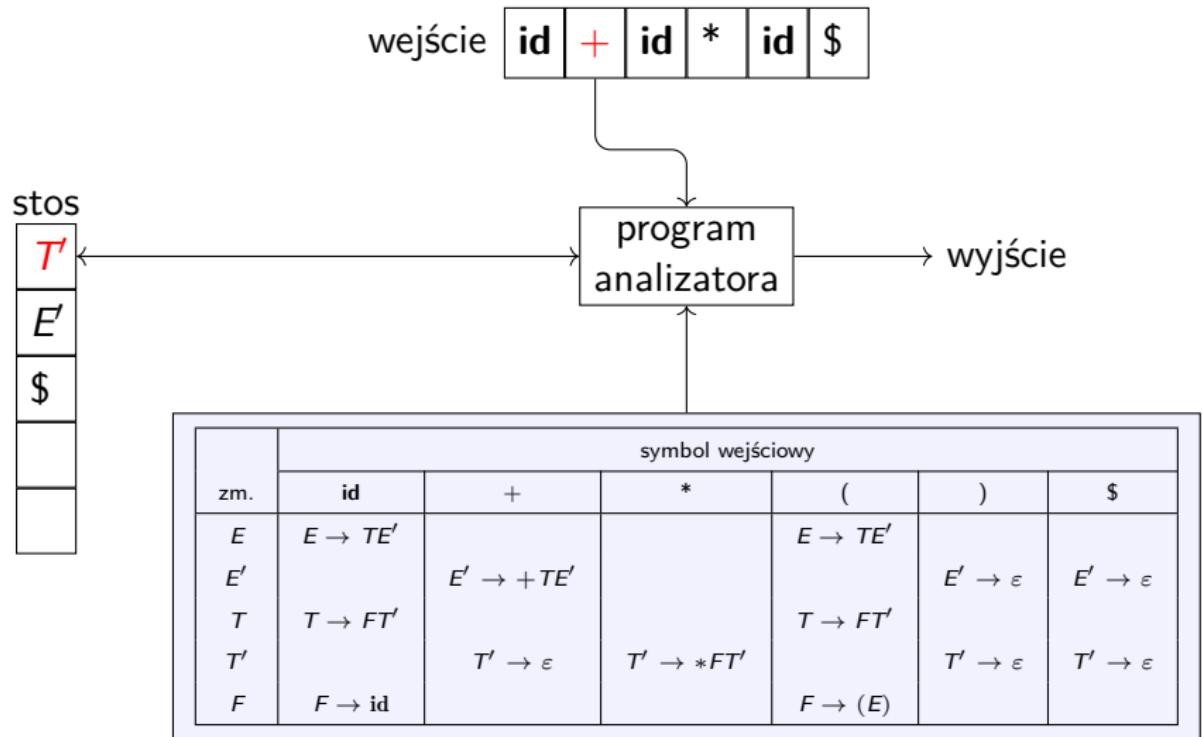


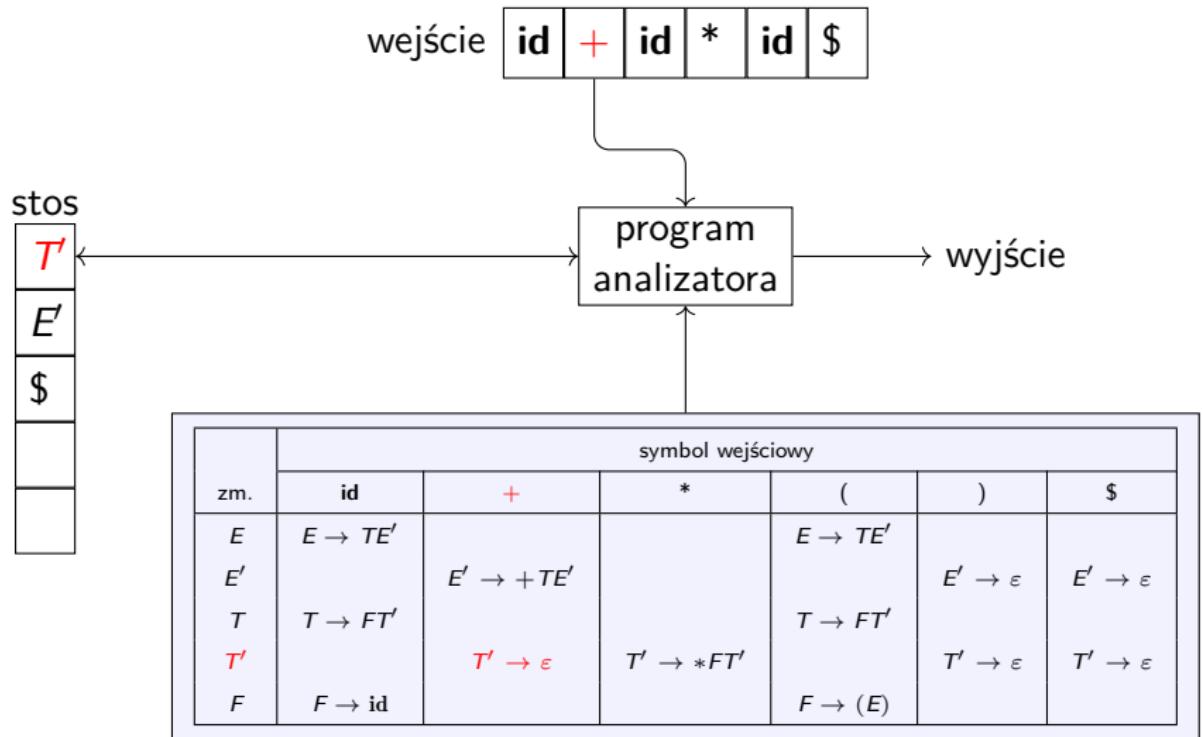


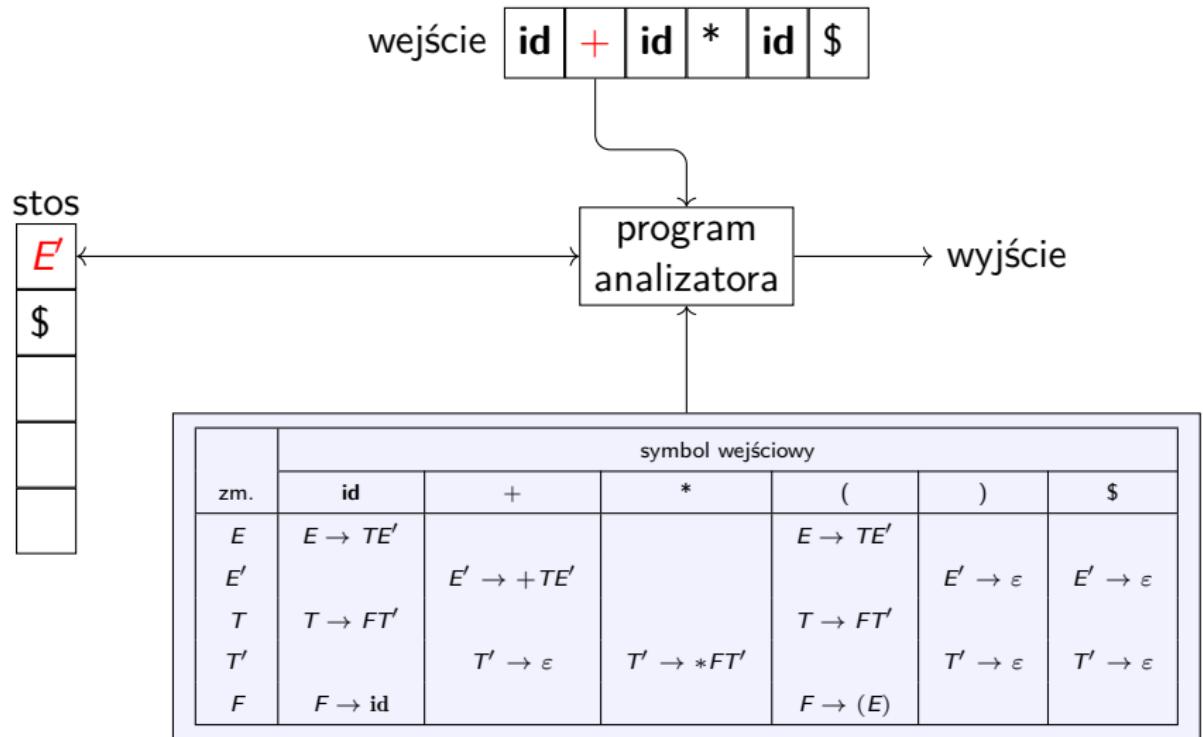


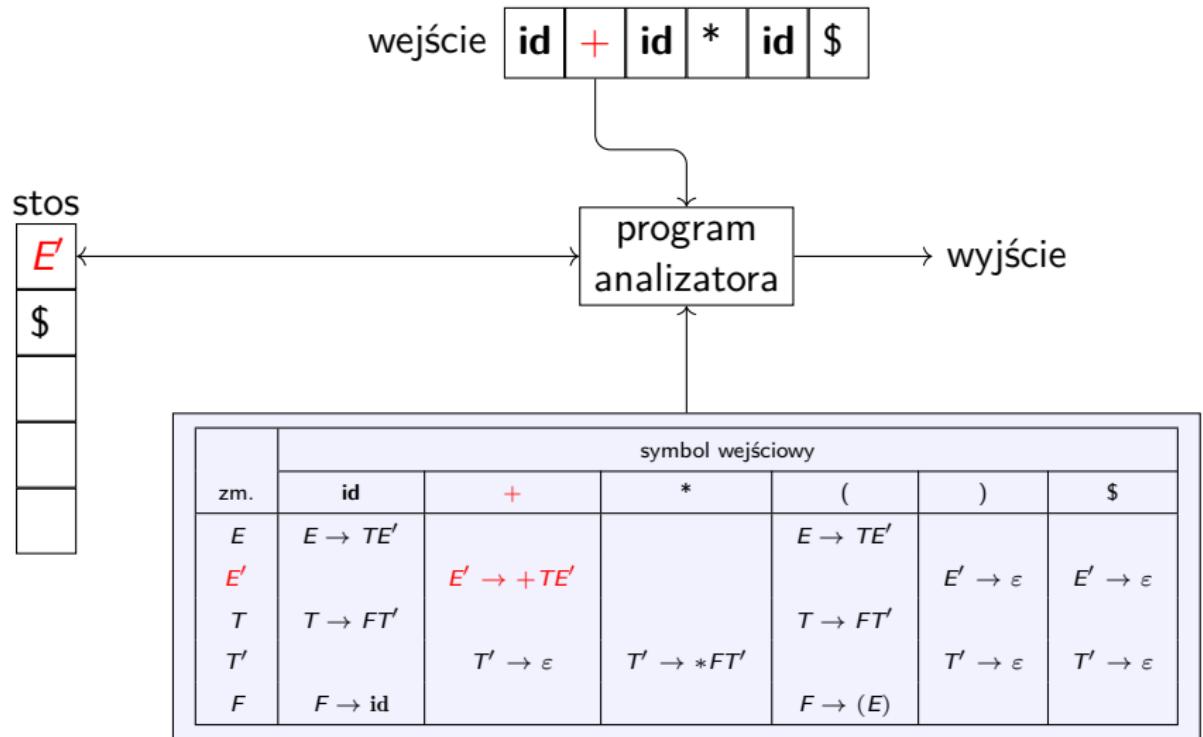


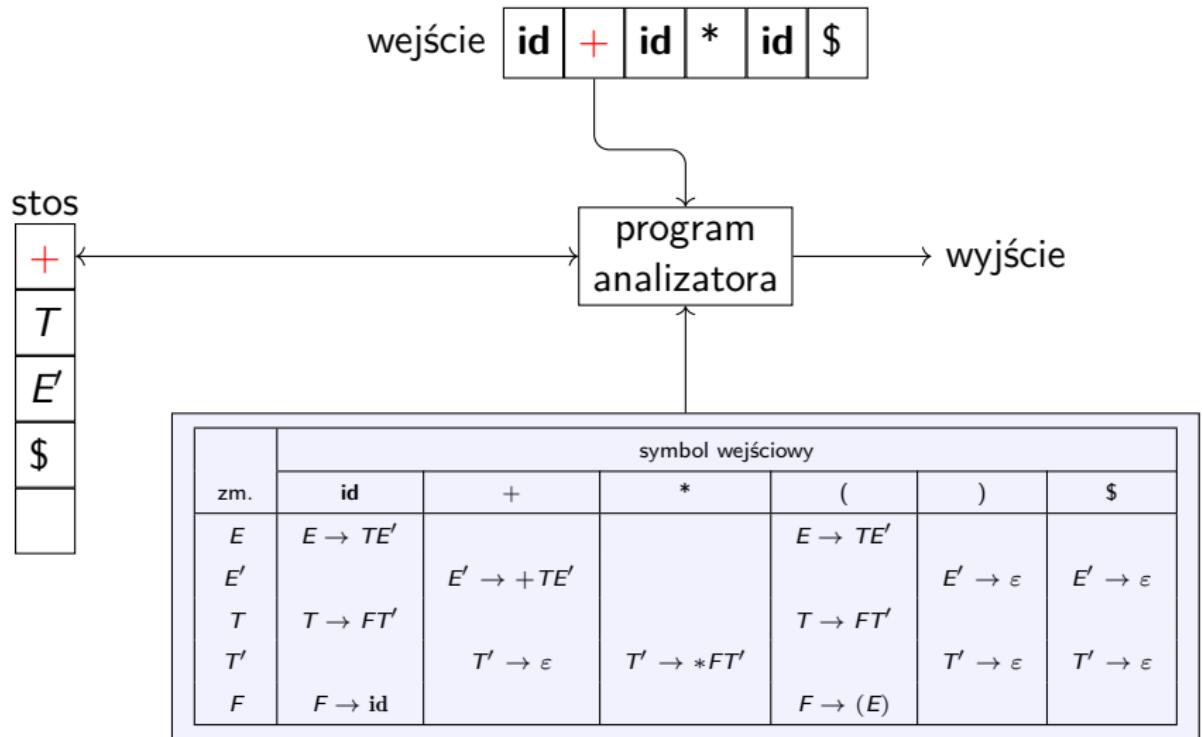


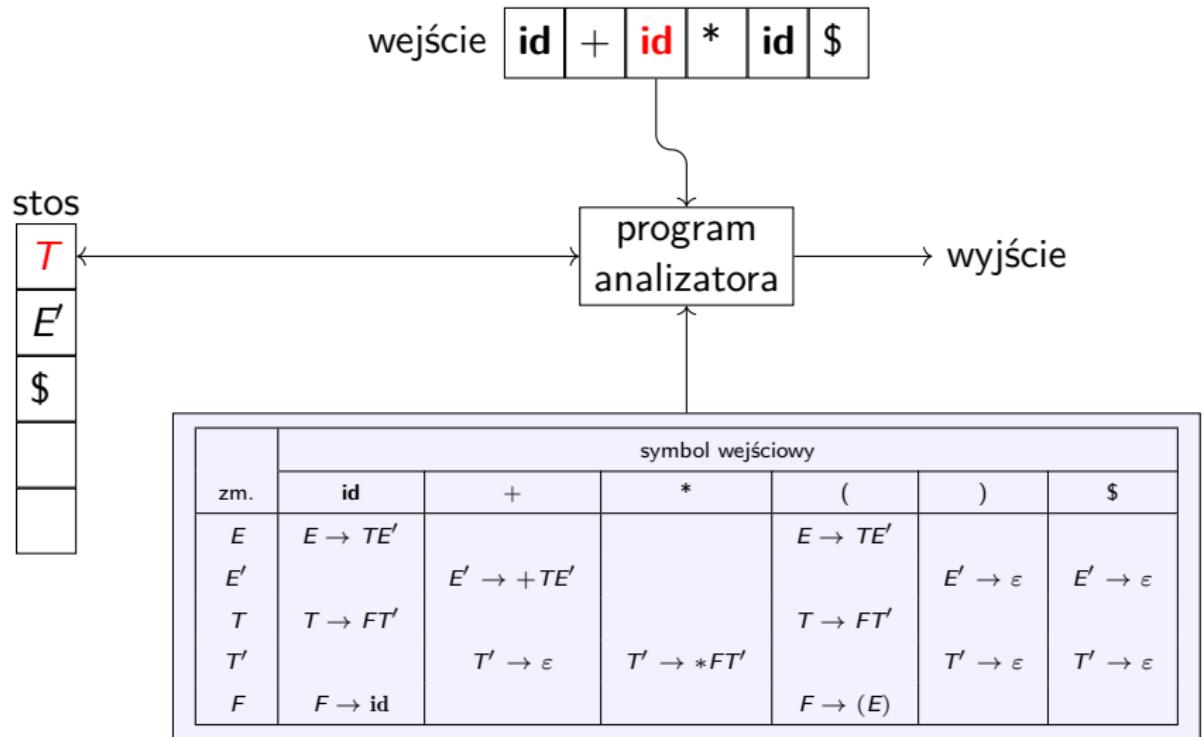


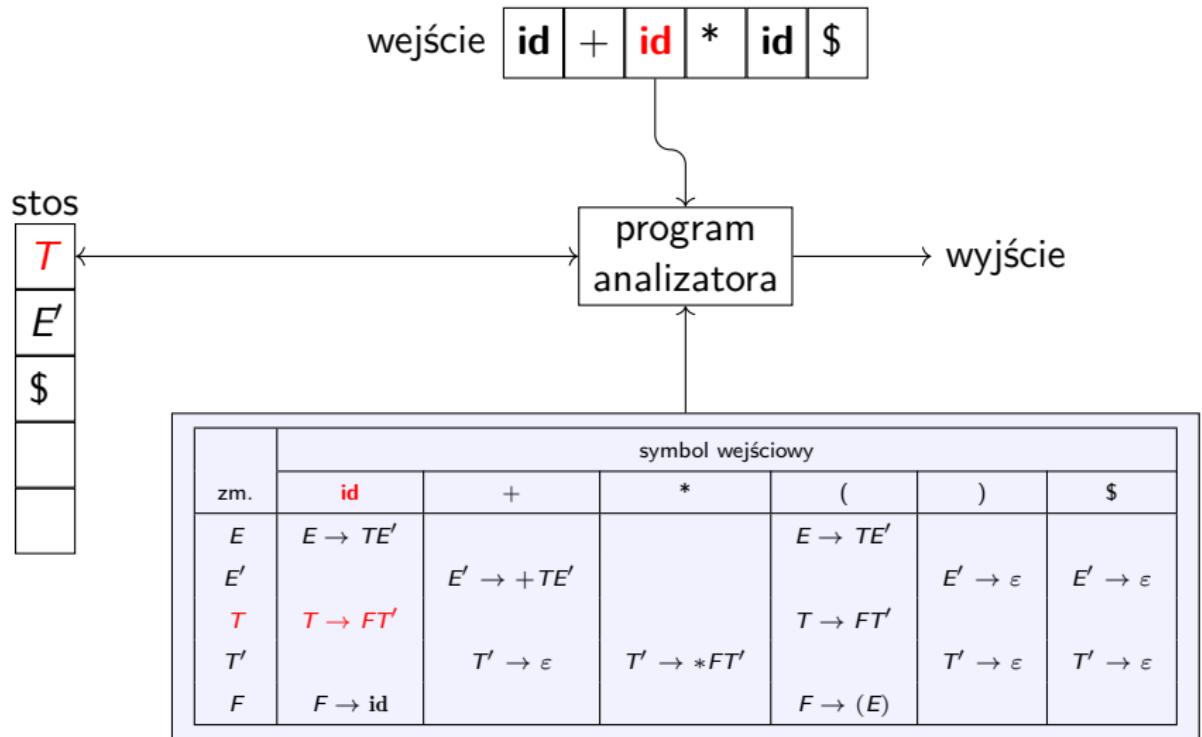


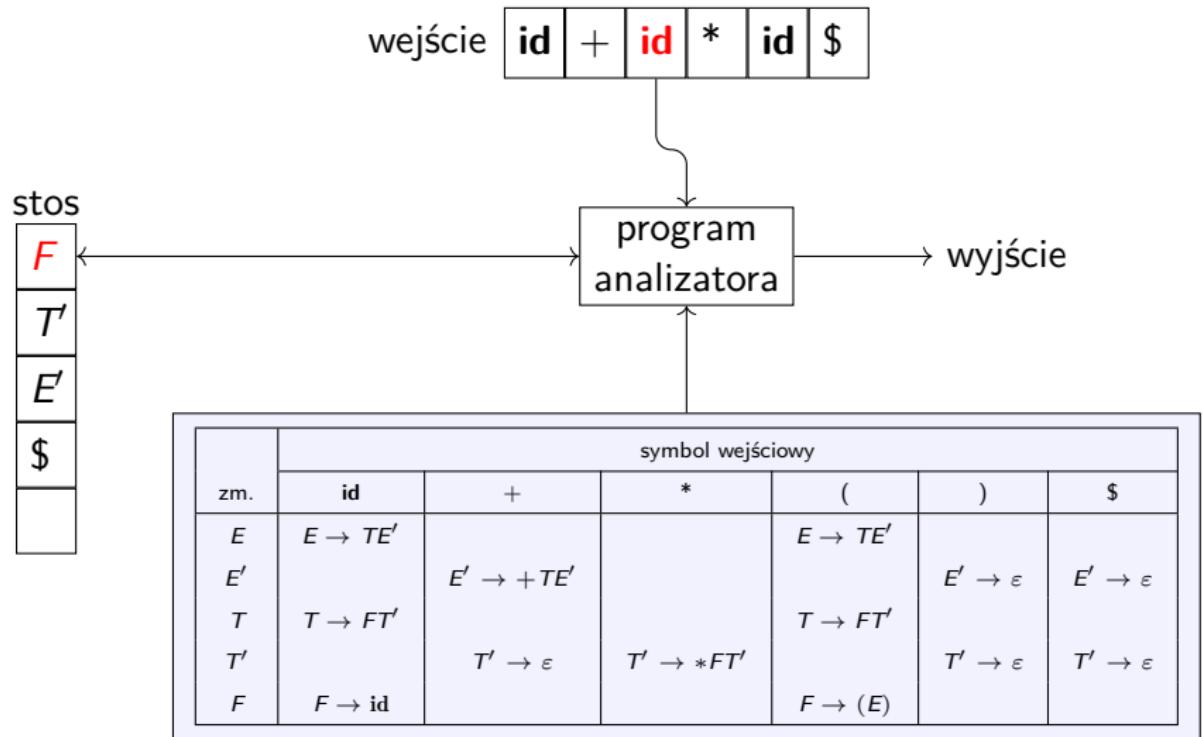


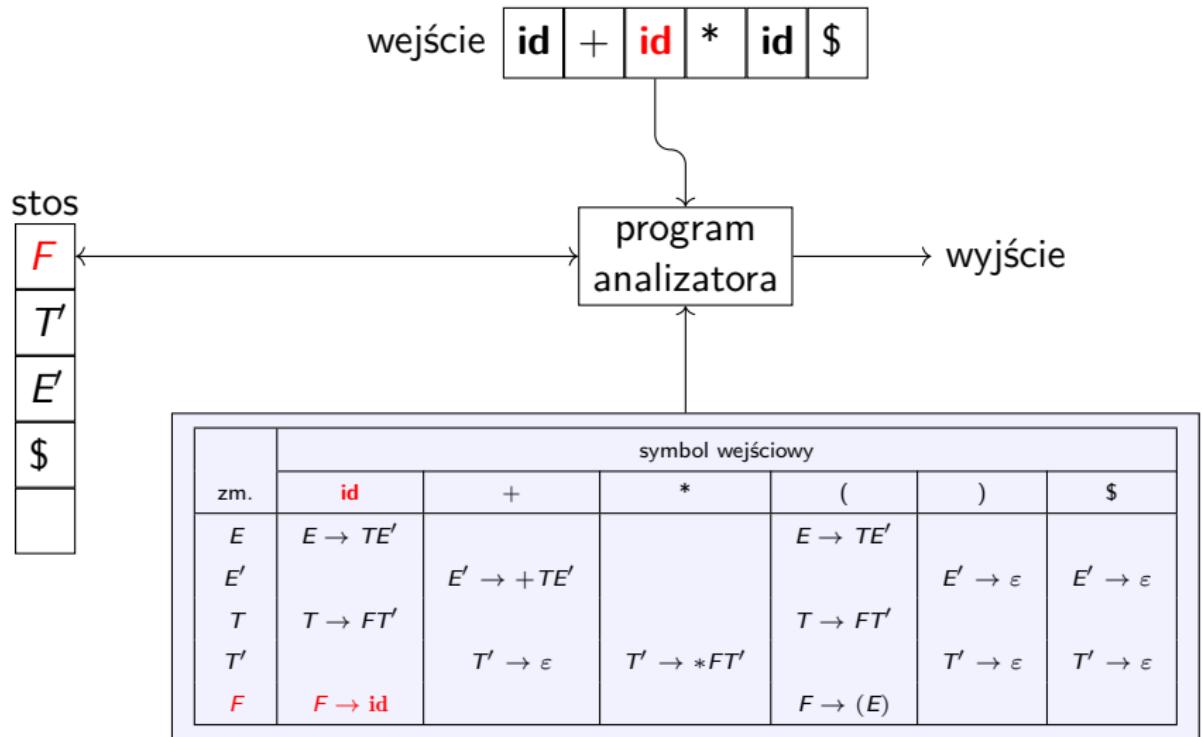


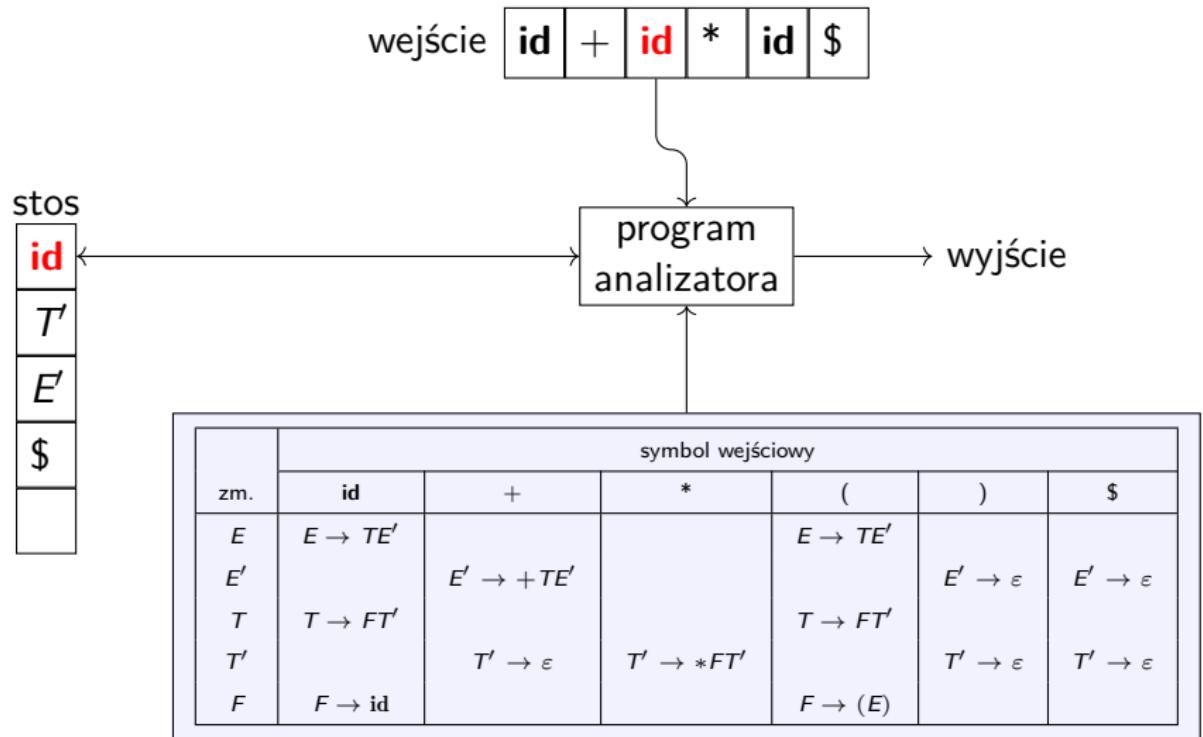


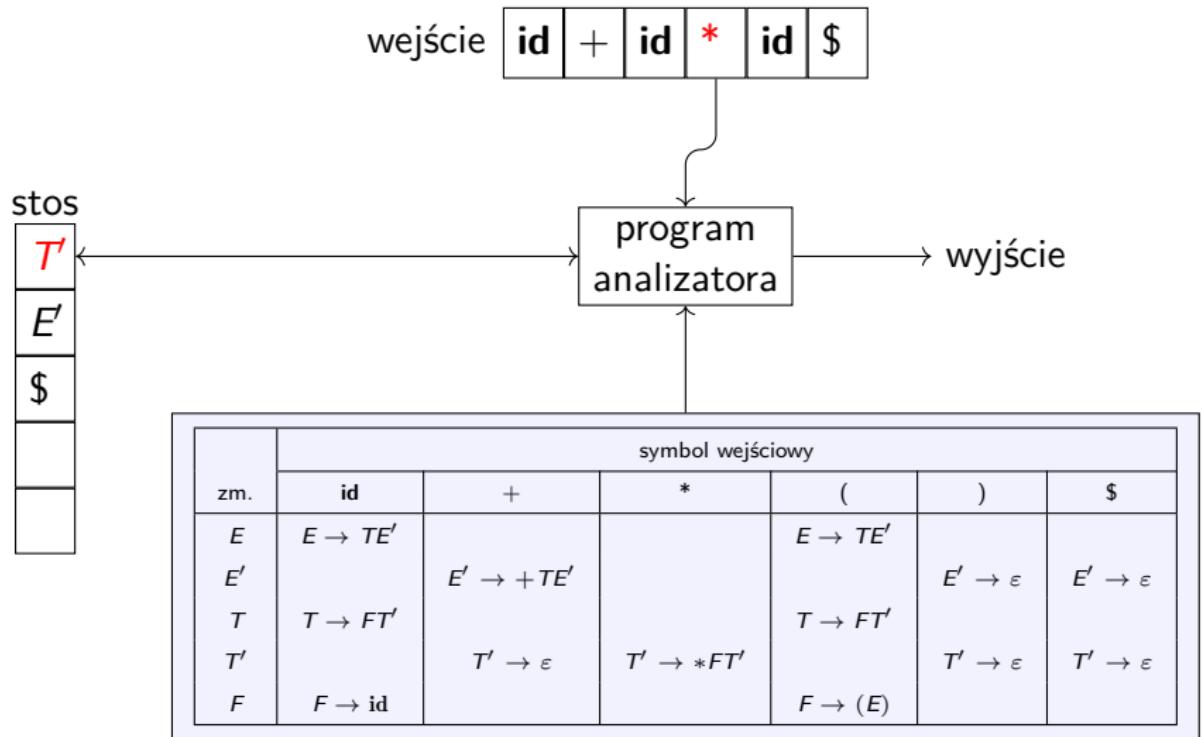


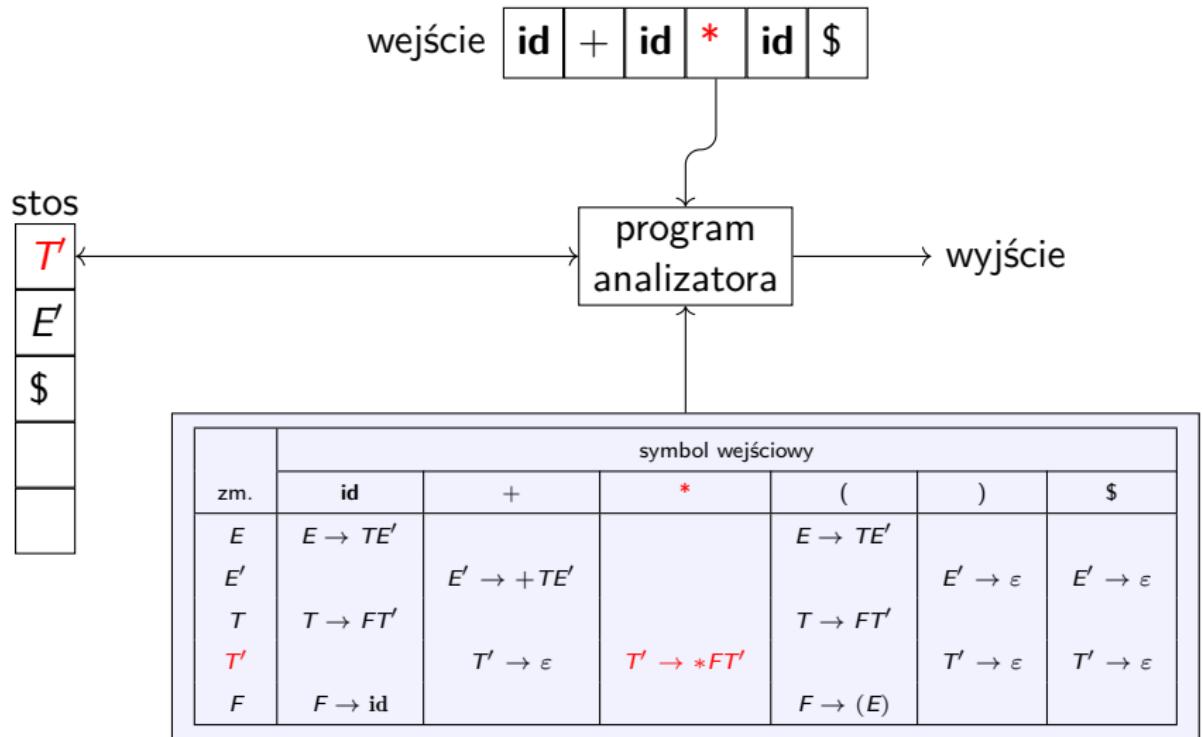


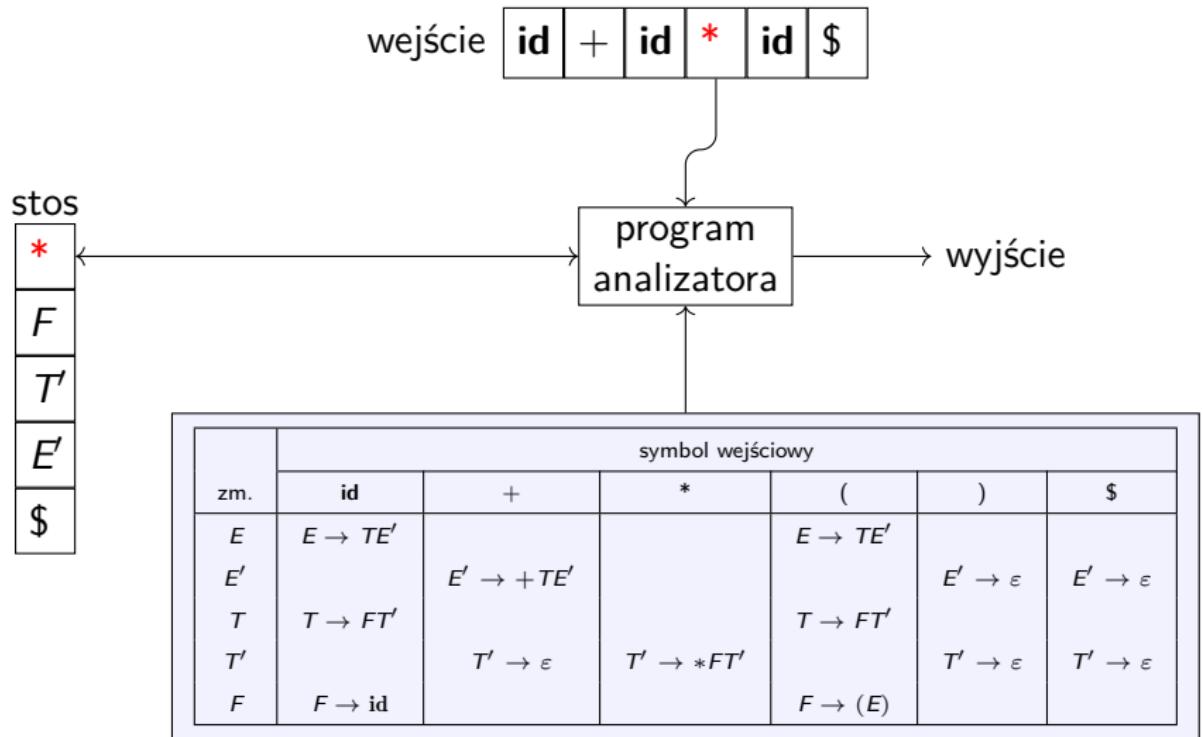


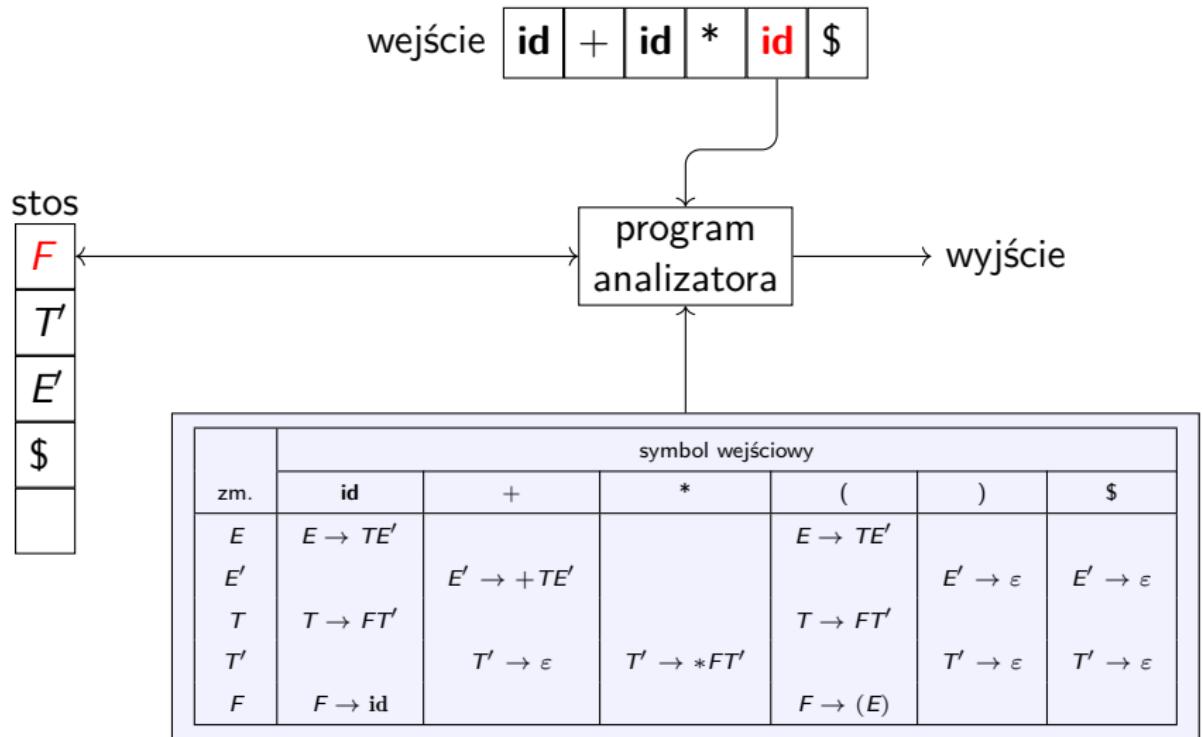


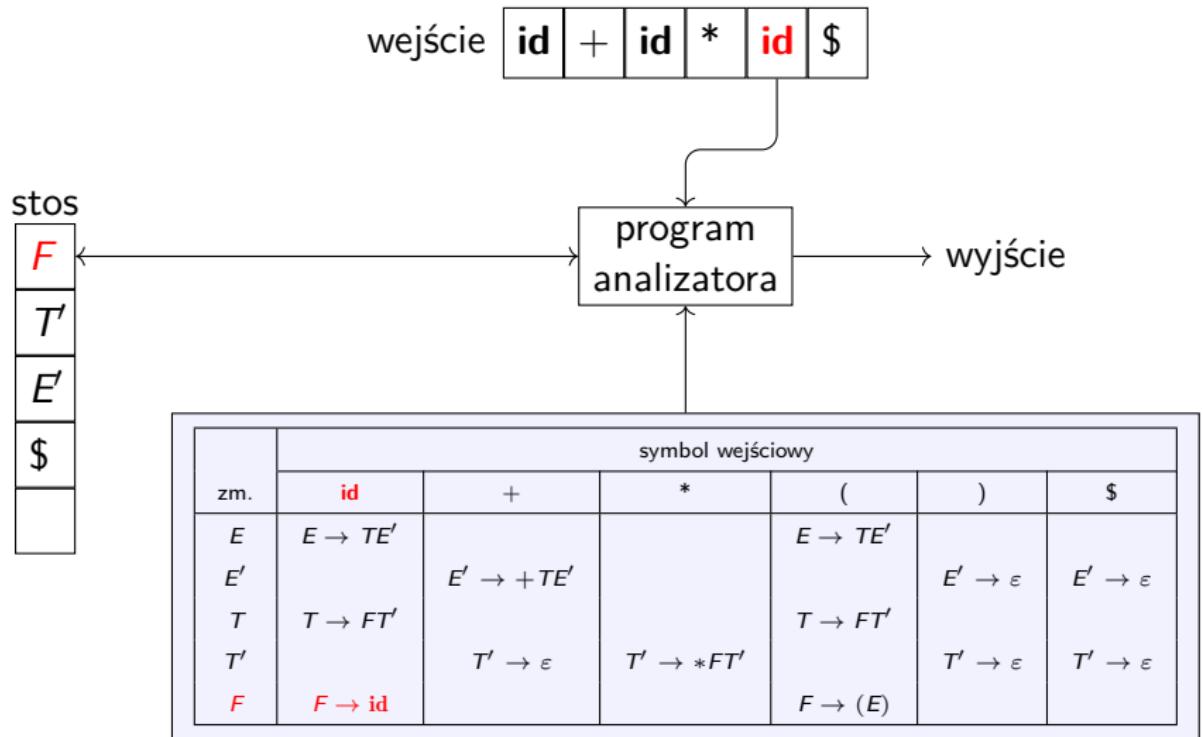


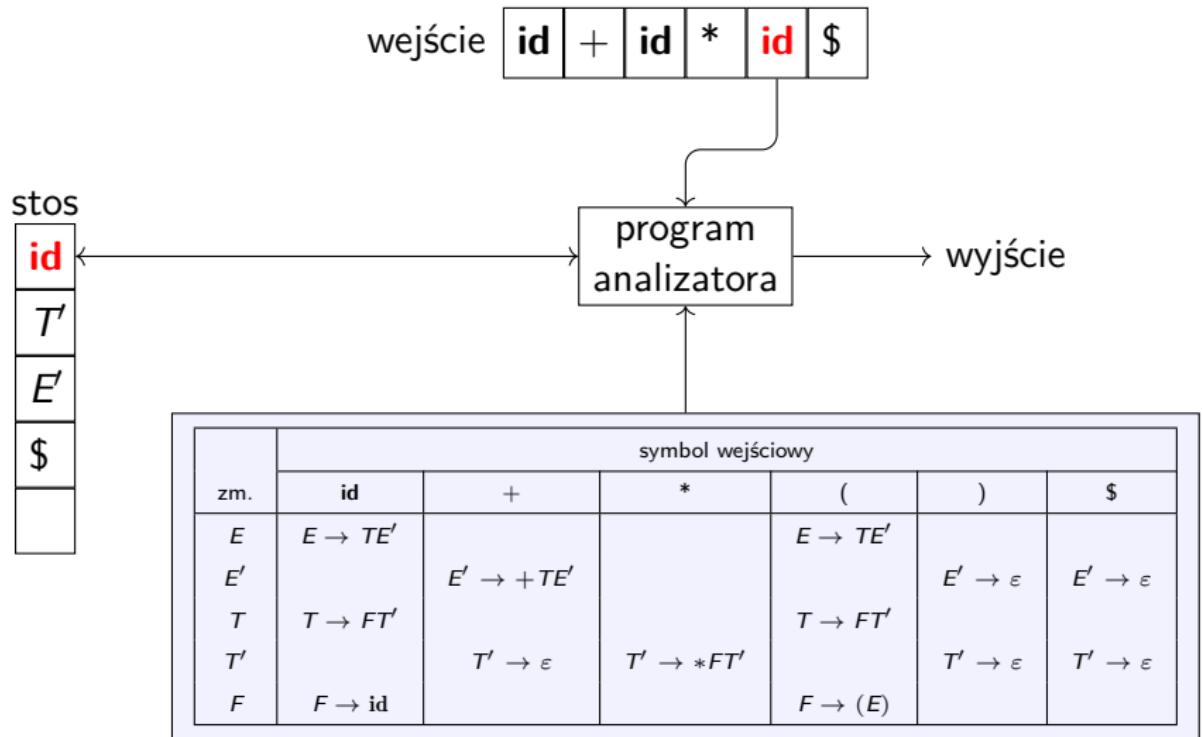


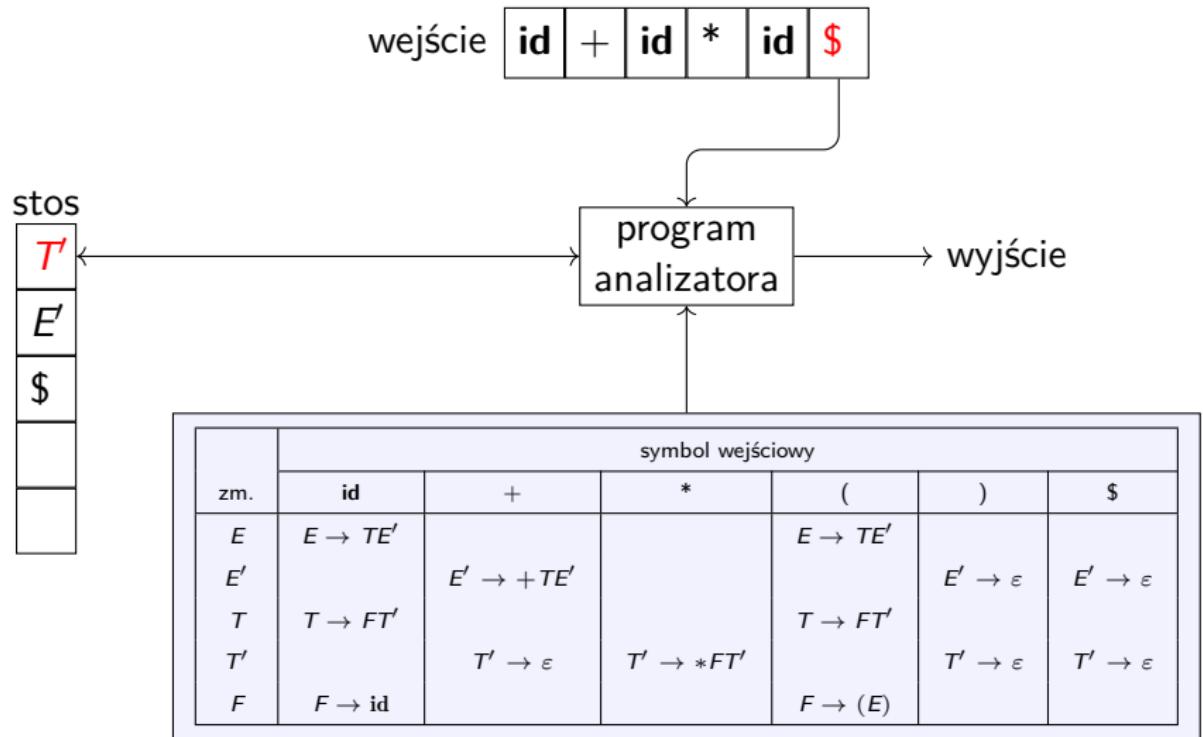


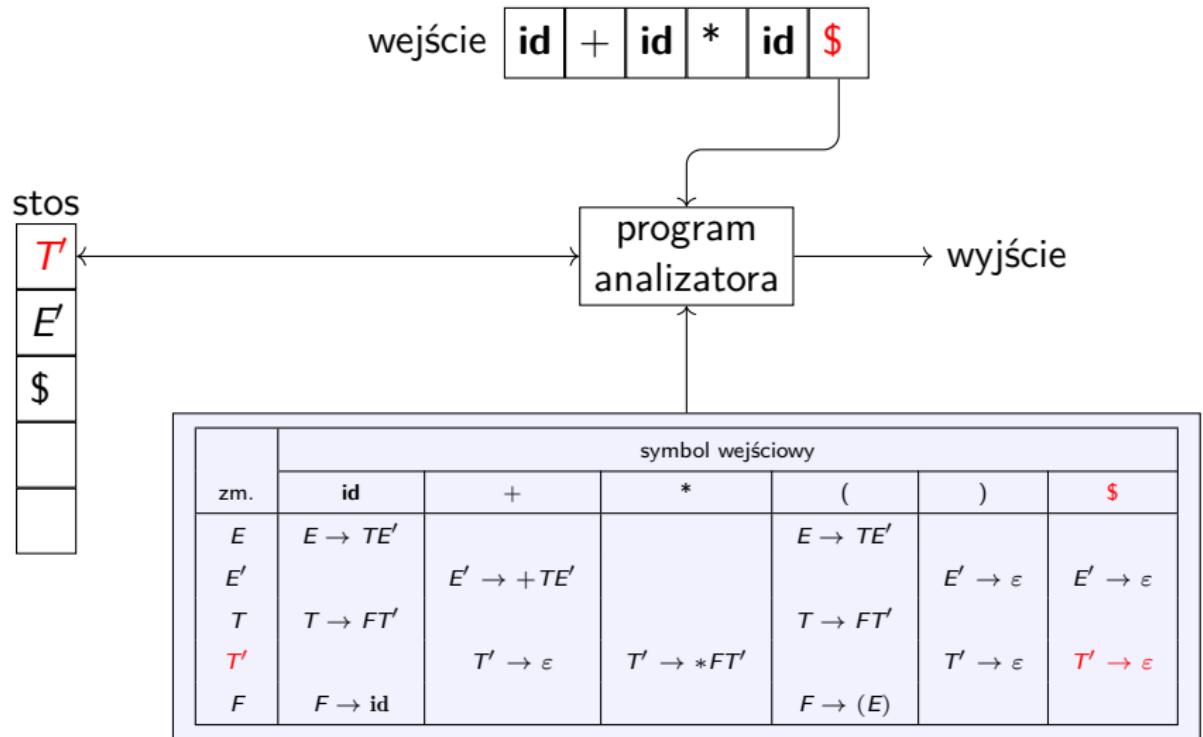


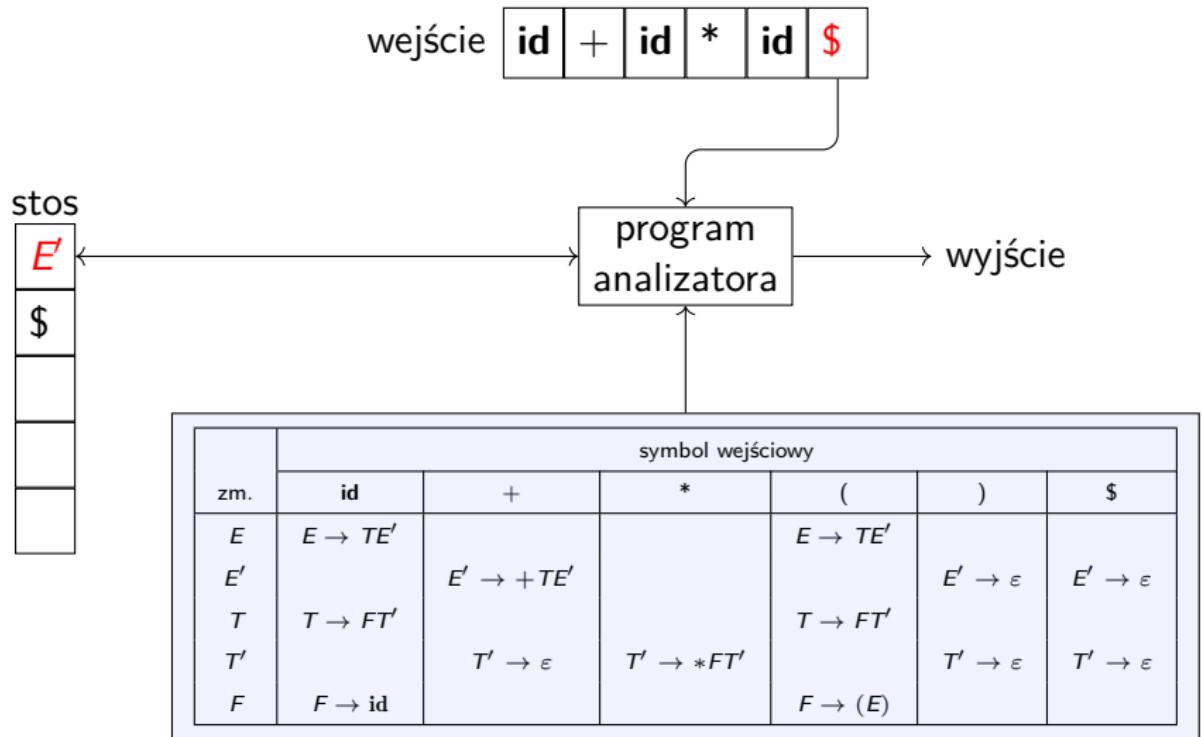


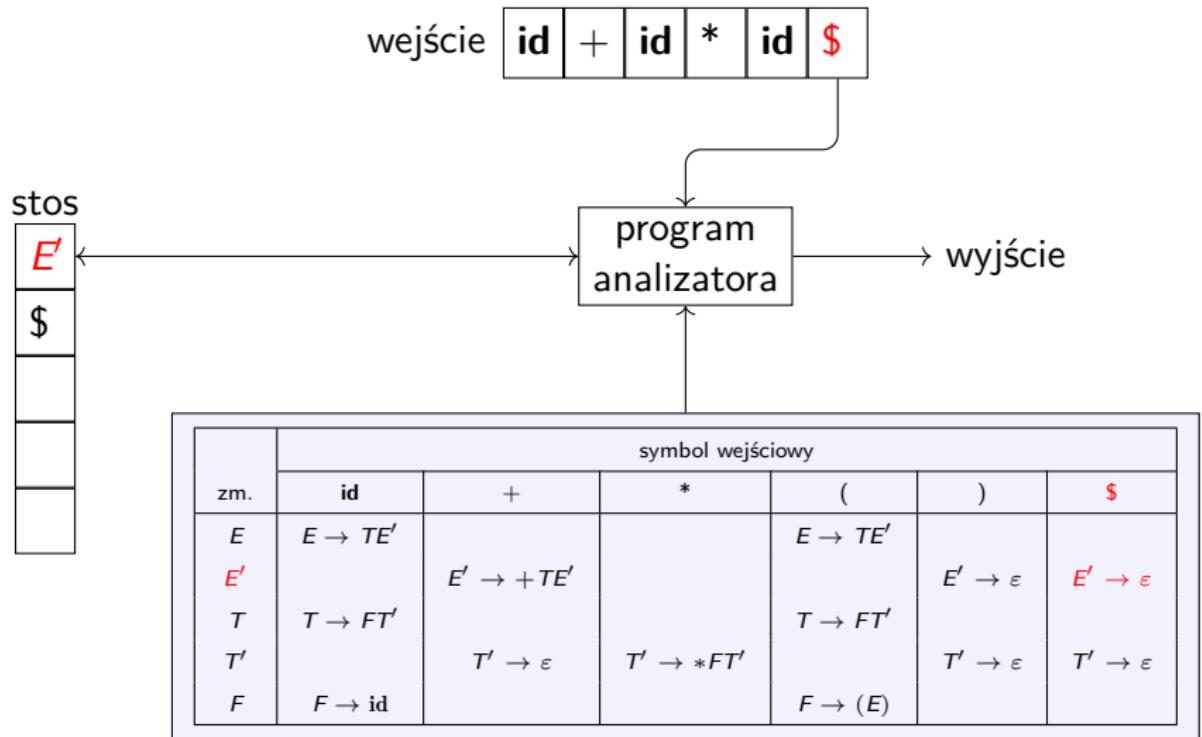


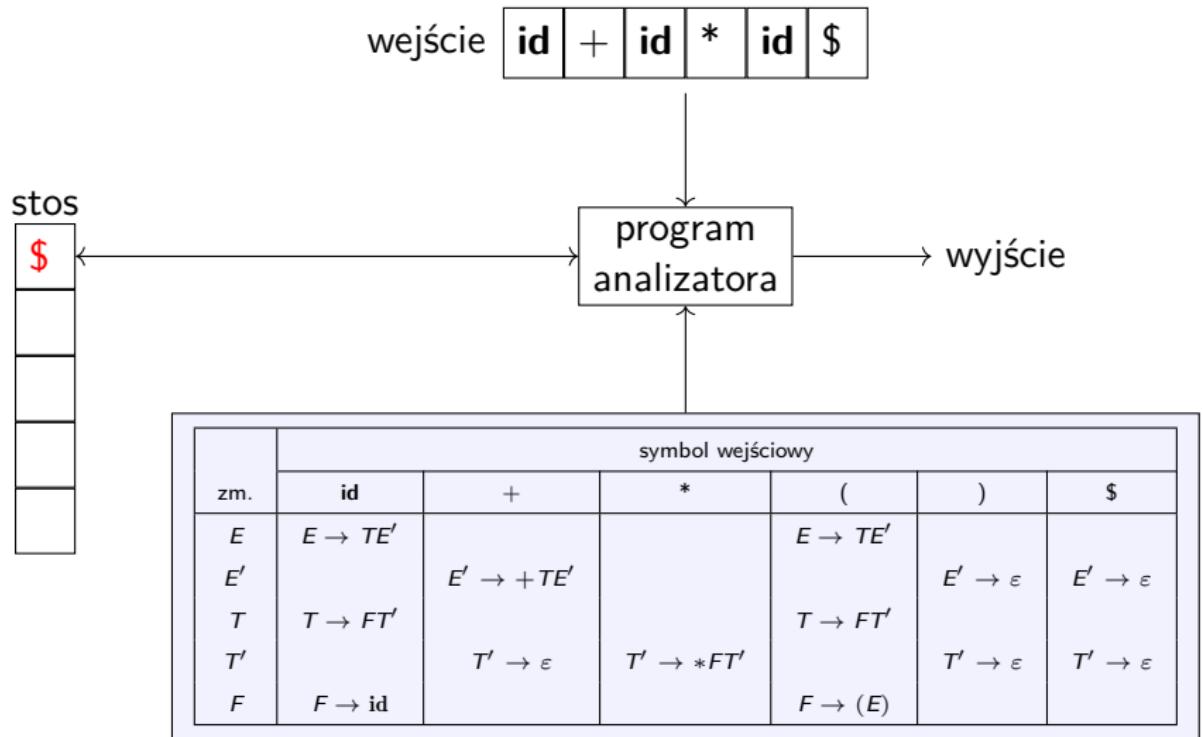












Do obsługi błędów można stosować podobną strategię jak w przypadku analizatora metodą zejść rekurencyjnych, tzn. przez pomijanie symboli końcowych aż do napotkania  $nast(A)$  lub ważnych słów kluczowych.

Można też stosować inne strategie:

- ① Dodanie  $pierw(A)$  do zbioru synchronizacyjnego zmiennej  $A$ , co spowoduje pomijanie symboli aż do rozpoczęcia analizy zgodnie z regułami dla  $A$ .
- ② Wybieranie reguł produkcji dających  $\varepsilon$  jako domyślnych
- ③ Zdejmowanie ze stosów zmiennych, których nie można dopasować do wejścia

## Przykład:

Tablica analizatora

zm.	symbol wejściowy					
	id	+	*	(	)	\$
$E$	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
$T$	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
$T'$		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
$F$	$F \rightarrow id$	synch	synch	$F \rightarrow (E)$	synch	synch

Elementy zbioru synchronizacyjnego dla rozpatrywanej wcześniej gramatyki oznaczono jako *synch*.



## Analiza wstępująca

Analiza wstępująca polega na zwijaniu ciągów symboli do pojedynczych symboli używając reguł produkcji gramatyki aż do osiągnięcia w ten sposób symbolu początkowego gramatyki.

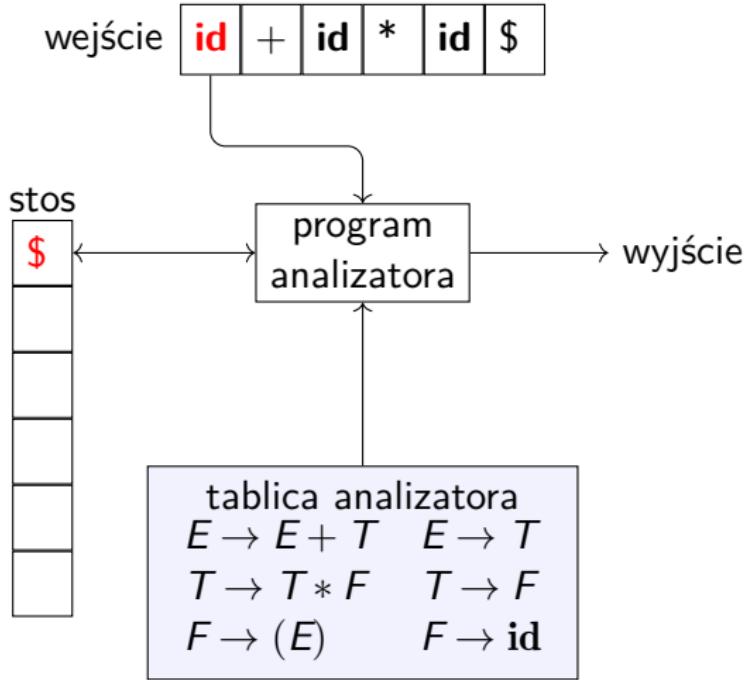
Rozpatrzmy ogólną zasadę działania takiego analizatora bez wchodzenia w szczegóły. Przyjmijmy w naszym przykładzie tę samą gramatykę, co w przykładzie analizy zstępującej z jawnym stosem (przed przekształceniem usuwającym lewostronną rekurencję):

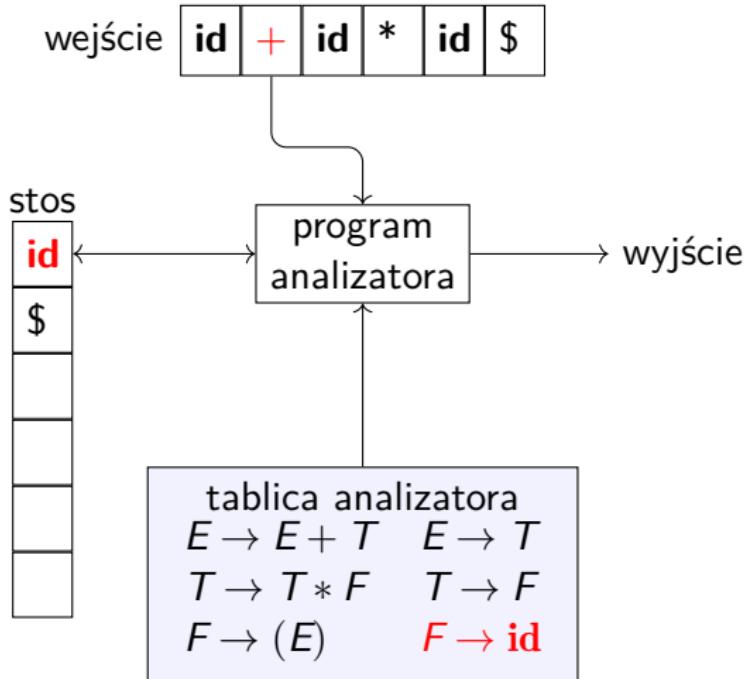
$$E \rightarrow E + T \mid T$$

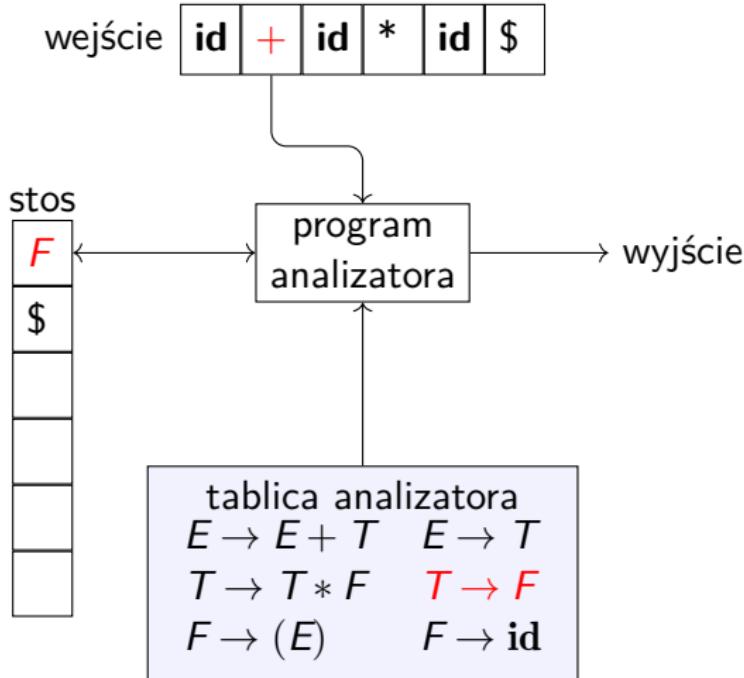
$$T \rightarrow T * F \mid F$$

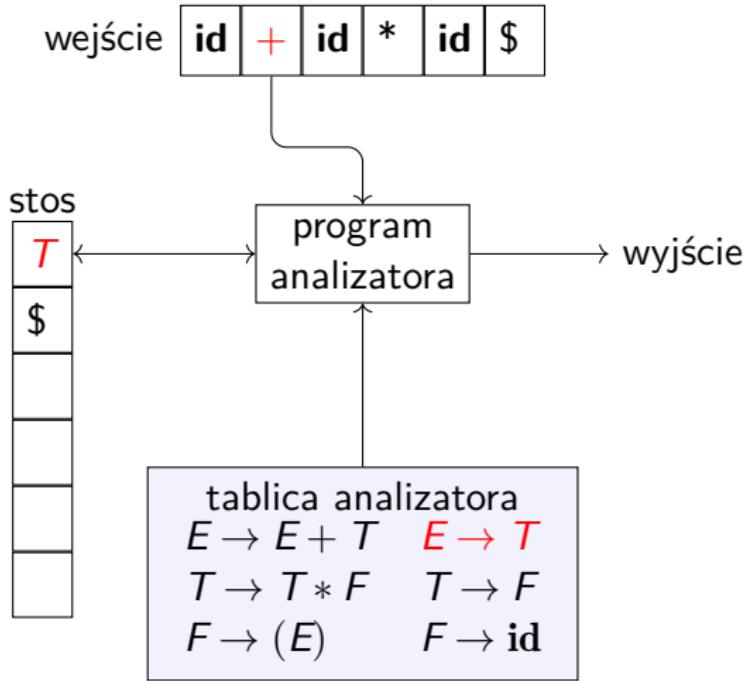
$$F \rightarrow (E) \mid \text{id}$$

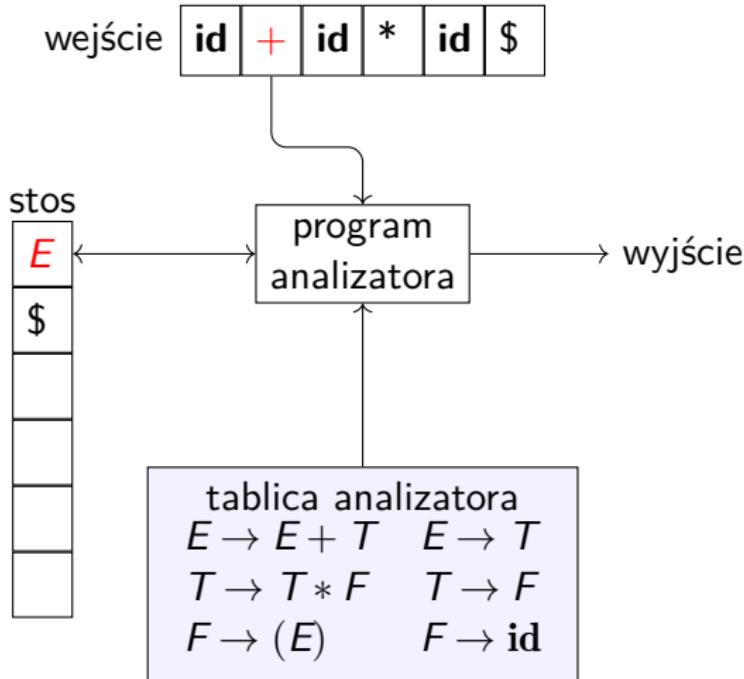
Analizator wykonuje dwa możliwe działania: **przesunięcie** (*shift*) symbolu z wejścia na stos i **zwinięcie** (*reduction*) ciągu symboli na stosie polegające na zastąpieniu ciągu występującego po prawej stronie reguły produkcji przez zmienną po lewej stronie tej reguły.

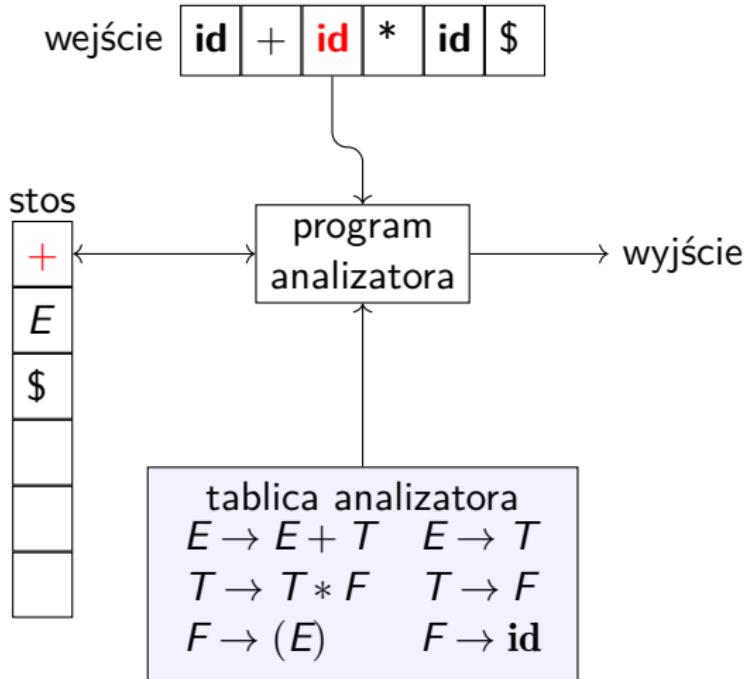


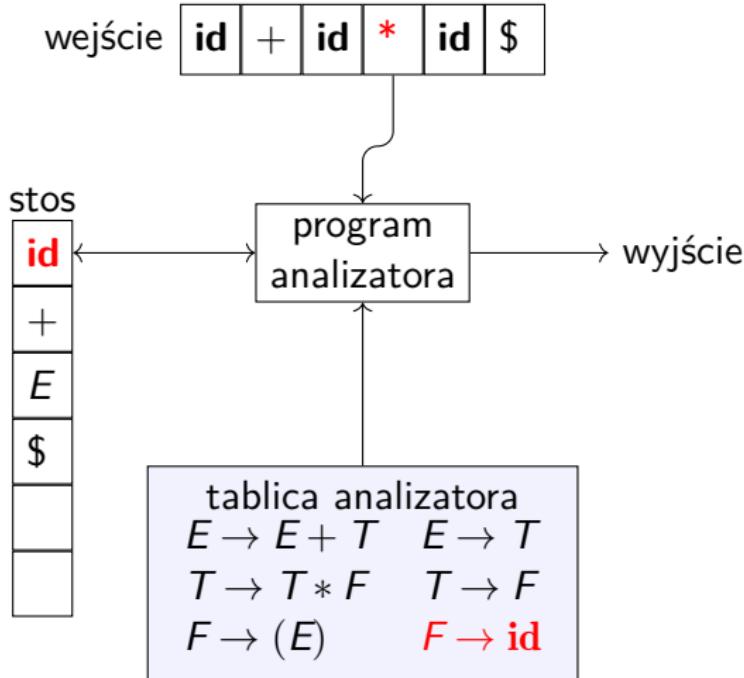


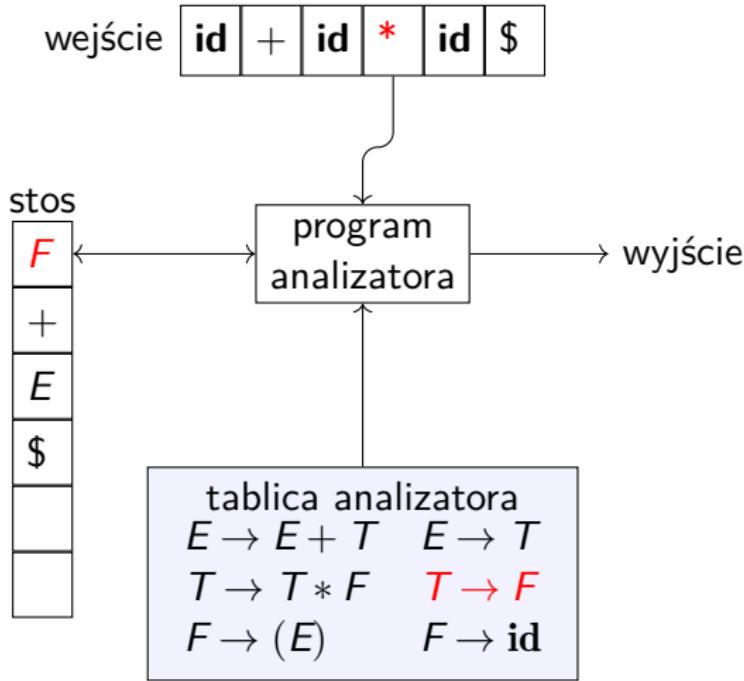


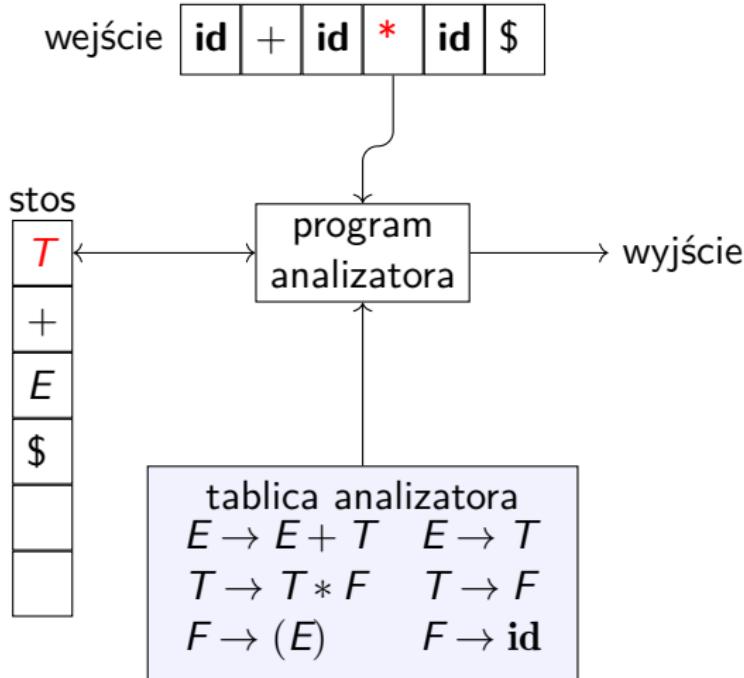


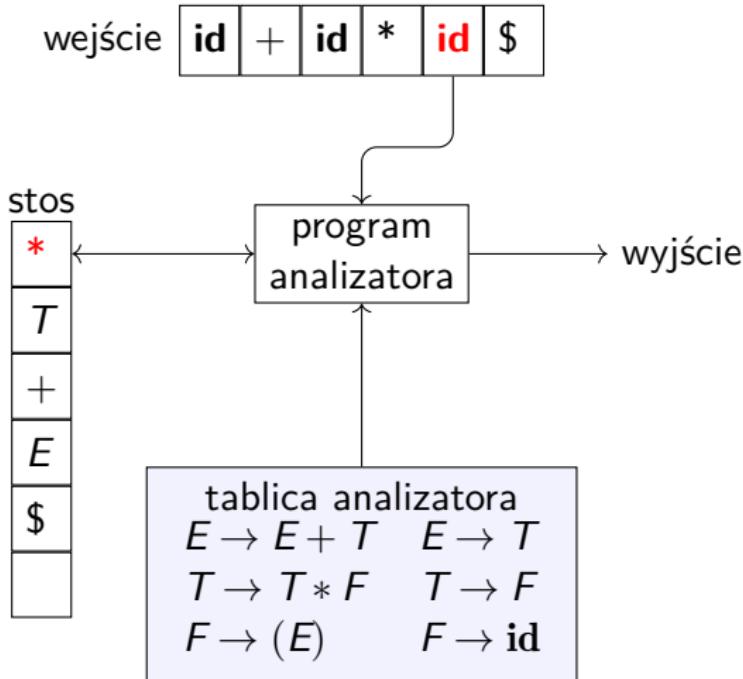


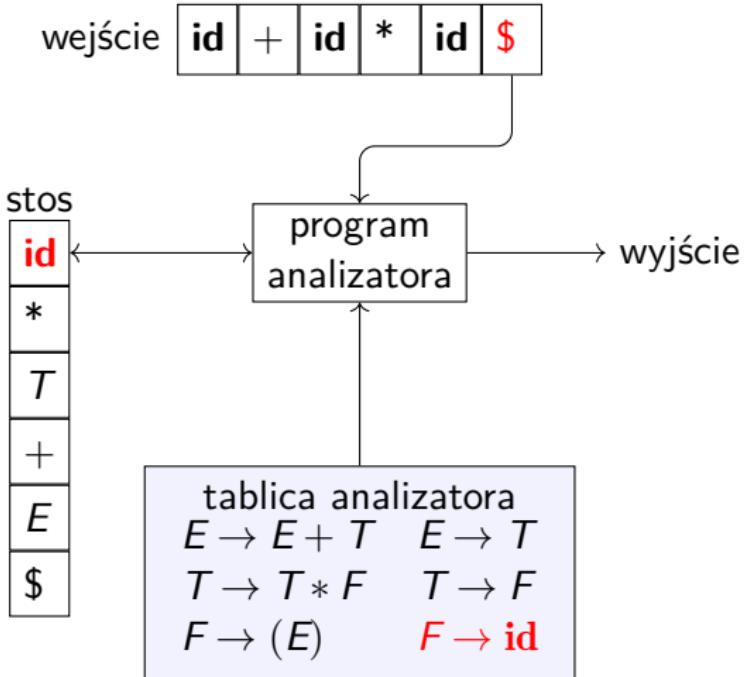


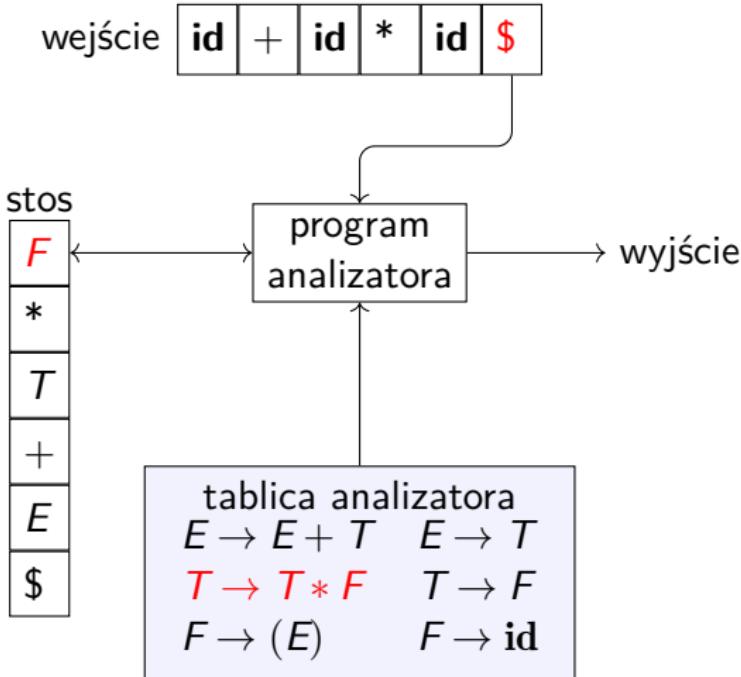


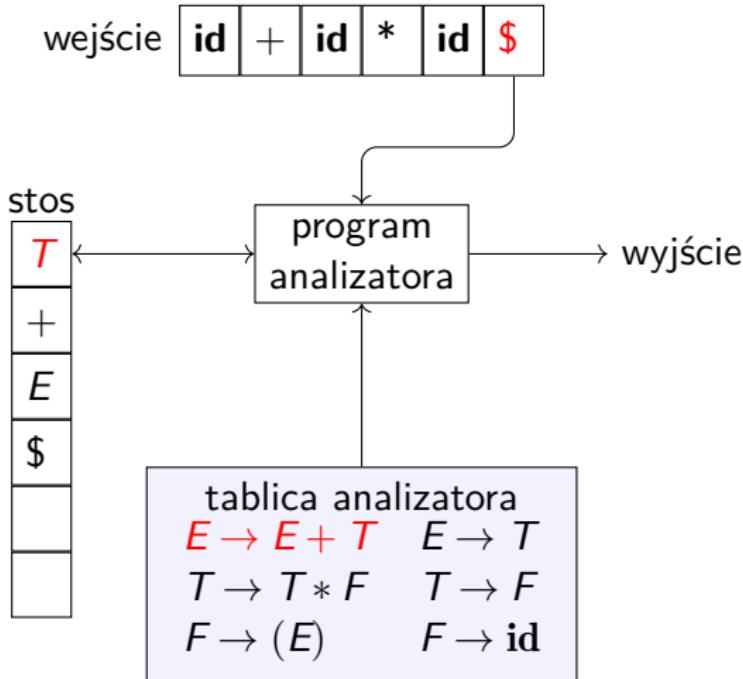


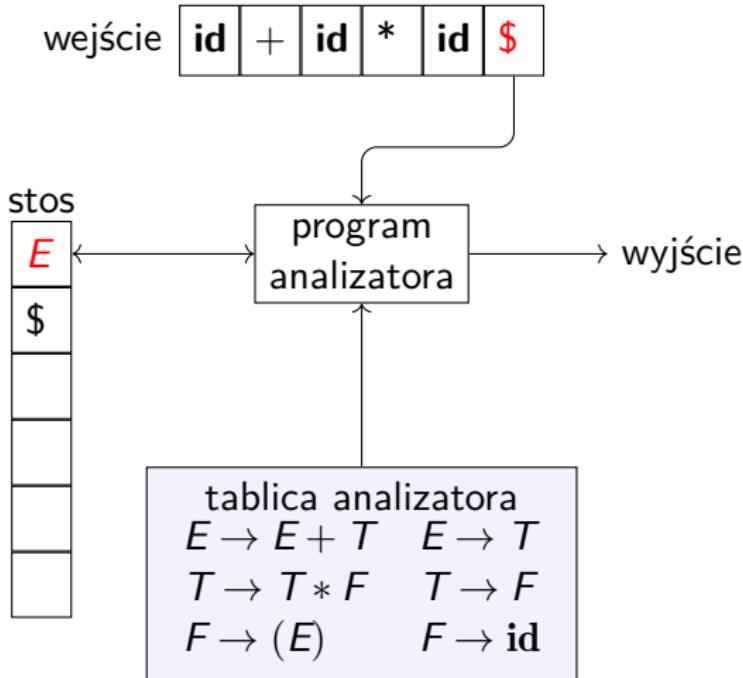












## Analiza wstępująca

Przypatrzmy się kolejności zwijania ciągów symboli:

**id**+**id**\***id**\$                               $F \rightarrow \text{id}$

## Analiza wstępująca

Przypatrzmy się kolejności zwijania ciągów symboli:

$$\text{id} + \text{id}^* \text{id} \$ \qquad\qquad F \rightarrow \text{id}$$

$$F + \text{id}^* \text{id} \qquad\qquad T \rightarrow F$$

## Analiza wstępująca

Przypatrzmy się kolejności zwijania ciągów symboli:

$$\text{id} + \text{id}^* \text{id} \$ \qquad \qquad F \rightarrow \text{id}$$

$$F + \text{id}^* \text{id} \qquad \qquad T \rightarrow F$$

$$T + \text{id}^* \text{id} \qquad \qquad E \rightarrow T$$

## Analiza wstępująca

Przypatrzmy się kolejności zwijania ciągów symboli:

$$\text{id} + \text{id}^* \text{id} \$ \quad F \rightarrow \text{id}$$

$$F + \text{id}^* \text{id} \quad T \rightarrow F$$

$$T + \text{id}^* \text{id} \quad E \rightarrow T$$

$$E + \text{id}^* \text{id} \quad F \rightarrow \text{id}$$

## Analiza wstępująca

Przypatrzmy się kolejności zwijania ciągów symboli:

$$\text{id} + \text{id}^* \text{id} \$ \quad F \rightarrow \text{id}$$

$$F + \text{id}^* \text{id} \quad T \rightarrow F$$

$$T + \text{id}^* \text{id} \quad E \rightarrow T$$

$$E + \text{id}^* \text{id} \quad F \rightarrow \text{id}$$

$$E + F^* \text{id} \quad T \rightarrow F$$

## Analiza wstępująca

Przypatrzmy się kolejności zwijania ciągów symboli:

$$\text{id} + \text{id}^* \text{id} \$ \quad F \rightarrow \text{id}$$

$$F + \text{id}^* \text{id} \quad T \rightarrow F$$

$$T + \text{id}^* \text{id} \quad E \rightarrow T$$

$$E + \text{id}^* \text{id} \quad F \rightarrow \text{id}$$

$$E + F^* \text{id} \quad T \rightarrow F$$

$$E + T^* \text{id} \quad F \rightarrow \text{id}$$

# Analiza wstępująca

Przypatrzmy się kolejności zwijania ciągów symboli:

$\text{id} + \text{id}^* \text{id} \$$	$F \rightarrow \text{id}$
$F + \text{id}^* \text{id}$	$T \rightarrow F$
$T + \text{id}^* \text{id}$	$E \rightarrow T$
$E + \text{id}^* \text{id}$	$F \rightarrow \text{id}$
$E + F^* \text{id}$	$T \rightarrow F$
$E + T^* \text{id}$	$F \rightarrow \text{id}$
$E + T^* F$	$T \rightarrow T * F$

# Analiza wstępująca

Przypatrzmy się kolejności zwijania ciągów symboli:

$\text{id} + \text{id}^* \text{id} \$$	$F \rightarrow \text{id}$
$F + \text{id}^* \text{id}$	$T \rightarrow F$
$T + \text{id}^* \text{id}$	$E \rightarrow T$
$E + \text{id}^* \text{id}$	$F \rightarrow \text{id}$
$E + F^* \text{id}$	$T \rightarrow F$
$E + T^* \text{id}$	$F \rightarrow \text{id}$
$E + T^* F$	$T \rightarrow T * F$
$E + T$	$E \rightarrow E + T$



# Analiza wstępująca



Przypatrzmy się kolejności zwijania ciągów symboli:

$\text{id} + \text{id}^* \text{id} \$$	$F \rightarrow \text{id}$
$F + \text{id}^* \text{id}$	$T \rightarrow F$
$T + \text{id}^* \text{id}$	$E \rightarrow T$
$E + \text{id}^* \text{id}$	$F \rightarrow \text{id}$
$E + F^* \text{id}$	$T \rightarrow F$
$E + T^* \text{id}$	$F \rightarrow \text{id}$
$E + T^* F$	$T \rightarrow T * F$
$E + T$	$E \rightarrow E + T$
$E$	

## Analiza wstępująca

To samo w odwrotnej kolejności, od symbolu początkowego gramatyki (tu symbolem początkowym jest  $E$ ) do liści:

$E$	$E \rightarrow E + T$
$E + T$	$T \rightarrow T * F$
$E + T * F$	$F \rightarrow \text{id}$
$E + T * \text{id}$	$T \rightarrow F$
$E + F * \text{id}$	$F \rightarrow \text{id}$
$E + \text{id} * \text{id}$	$E \rightarrow T$
$T + \text{id} * \text{id}$	$T \rightarrow F$
$F + \text{id} * \text{id}$	$F \rightarrow \text{id}$
$\text{id} + \text{id} * \text{id} \$$	

To samo w odwrotnej kolejności, od symbolu początkowego gramatyki (tu symbolem początkowym jest  $E$ ) do liści:

$E$	$E \rightarrow E + T$
$E + T$	$T \rightarrow T * F$
$E + T * F$	$F \rightarrow \text{id}$
$E + T * \text{id}$	$T \rightarrow F$
$E + F * \text{id}$	$F \rightarrow \text{id}$
$E + \text{id} * \text{id}$	$E \rightarrow T$
$T + \text{id} * \text{id}$	$T \rightarrow F$
$F + \text{id} * \text{id}$	$F \rightarrow \text{id}$
$\text{id} + \text{id} * \text{id} \$$	

Jest to wyprowadzenie prawostronne. Stąd nazwa analizy  $LR(k)$ :  $L$  od ang. *left to right* (od lewej do prawej),  $R$  od ang. *rightmost* (prawostronne),  $k$  od podglądu  $k$  symboli wejściowych do przodu.



Istnieją trzy podstawowe typy analizatorów  $LR(1)$ , ale wszystkie trzy korzystają z tablicy, która określa działanie analizatora na podstawie symbolu na szczycie stosu i następnego symbolu na wejściu. Są dostępne cztery działania:

- ① przeniesienie (przesunięcie) symbolu na stos (ang. *shift*)
- ② zwinięcie kilku symboli na szczycie stosu (ang. *reduce*) przez zastąpienie ich lewą stroną wskazanej reguły produkcji, której prawą stroną stanowią te symbole
- ③ przyjęcie wejścia
- ④ odrzucenie wejścia (błąd)

Na stos odkładamy parę (symbol, stan). Stan stanowi numer wiersza w tablicy analizatora. Początkowo na stosie znajduje się numer 0.

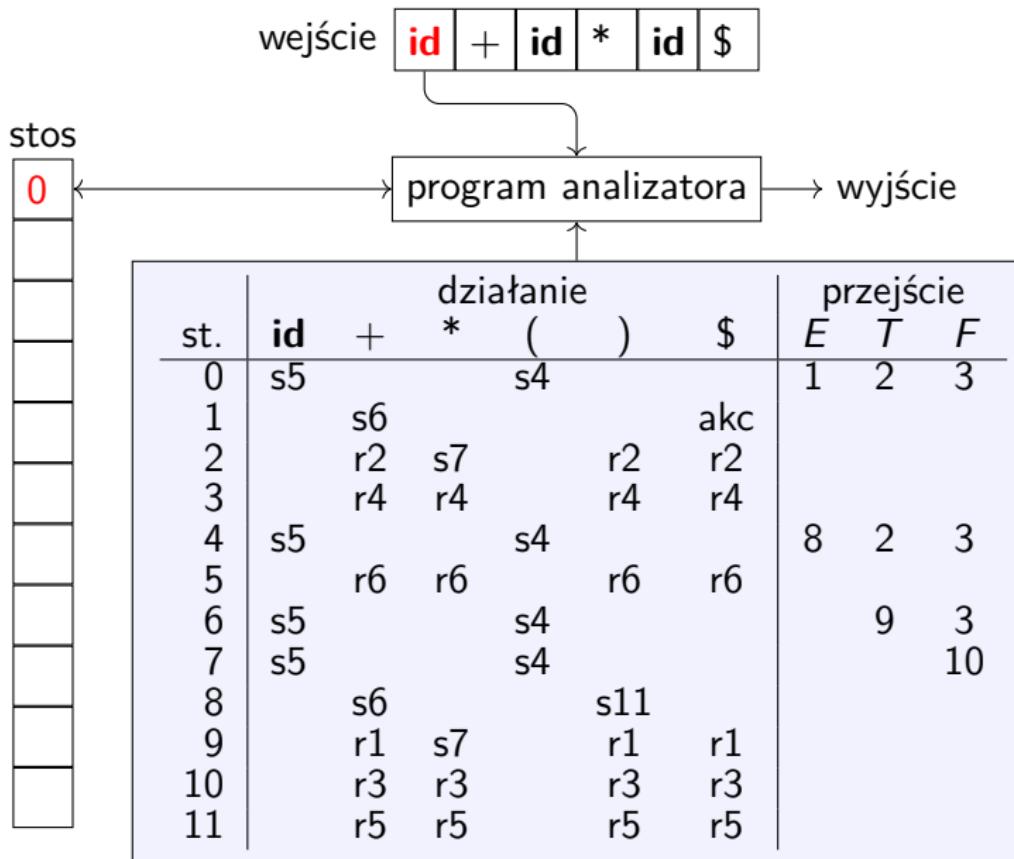
Wiersze tablicy analizatora oznaczają stany. Kolejne kolumny tablicy analizatora należą do dwóch grup: *działanie* i *przejście*. Kolumny działania etykietowane są symbolami końcowymi (symbolami z alfabetu) i znakiem końca wejścia \$. Zawartość tych kolumn oznaczamy jako literę *s* lub *r* i liczbę, przy czym:

- dla przesunięcia mamy literę *s*, a liczba jest numerem stanu docelowego — wiersza w tablicy analizatora
- dla zwinięcia mamy literę *r*, a liczba jest numerem reguły produkcji, według której zwijamy stos

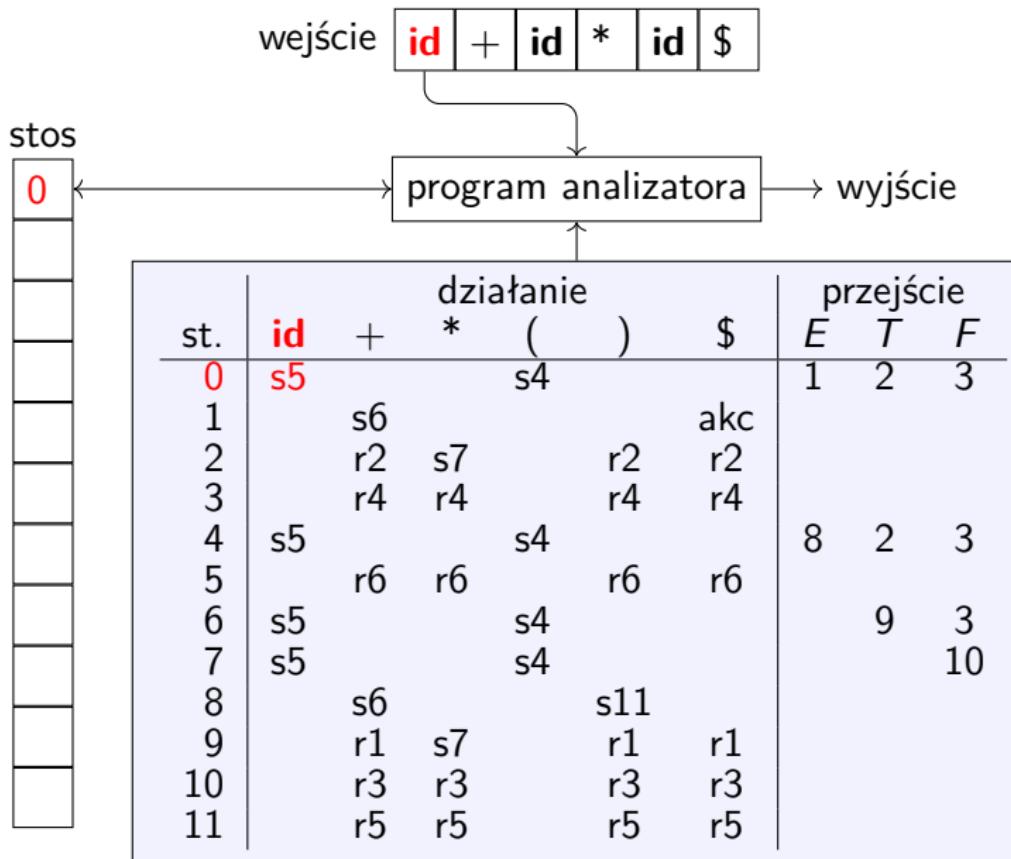
Kolumny przejścia etykietowane są zmiennymi gramatyki i zawierają numery stanów docelowych. Etykiety interpretowane są jako lewe strony reguł produkcji, według których następuje zwijanie.

Algorytm analizy  $LR(k)$ 

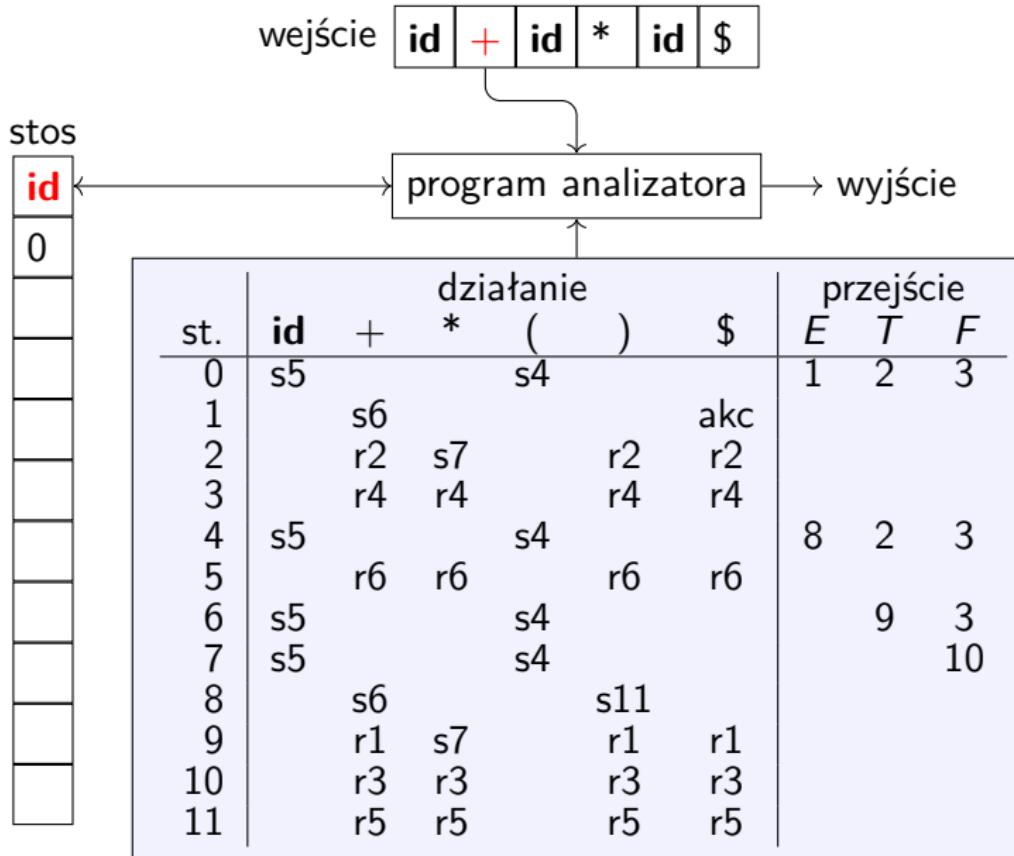
```
1: repeat
2:   niech  $a$  będzie bieżącym symbolem na wejściu
3:   niech  $s$  będzie symbolem z wierzchołka stosu
4:   if działanie[ $s, a$ ] = przesuń  $s'$  then
5:     odłóż  $a$  na stos
6:     odłóż  $s'$  na stos
7:     przesuń wskaźnik wejścia, ustaw  $a$  na nast. symbol na wejściu
8:   else if działanie[ $s, a$ ] = zwinięcie według reguły  $A \rightarrow \beta$  then
9:     zdejmij ze stosu  $|\beta|$  par (symbol, nr stanu)
10:    niech  $s'$  będzie stanem na szczycie stosu
11:    odłóż  $A$  na stos
12:    odłóż przejście[ $s', A$ ] na stos
13:   else if działanie[ $s, a$ ] = przyjmij then
14:     zakończ działanie — rozpoznano wejście
15:   else
16:     zgłoś błąd
17:   end if
18: until True
```



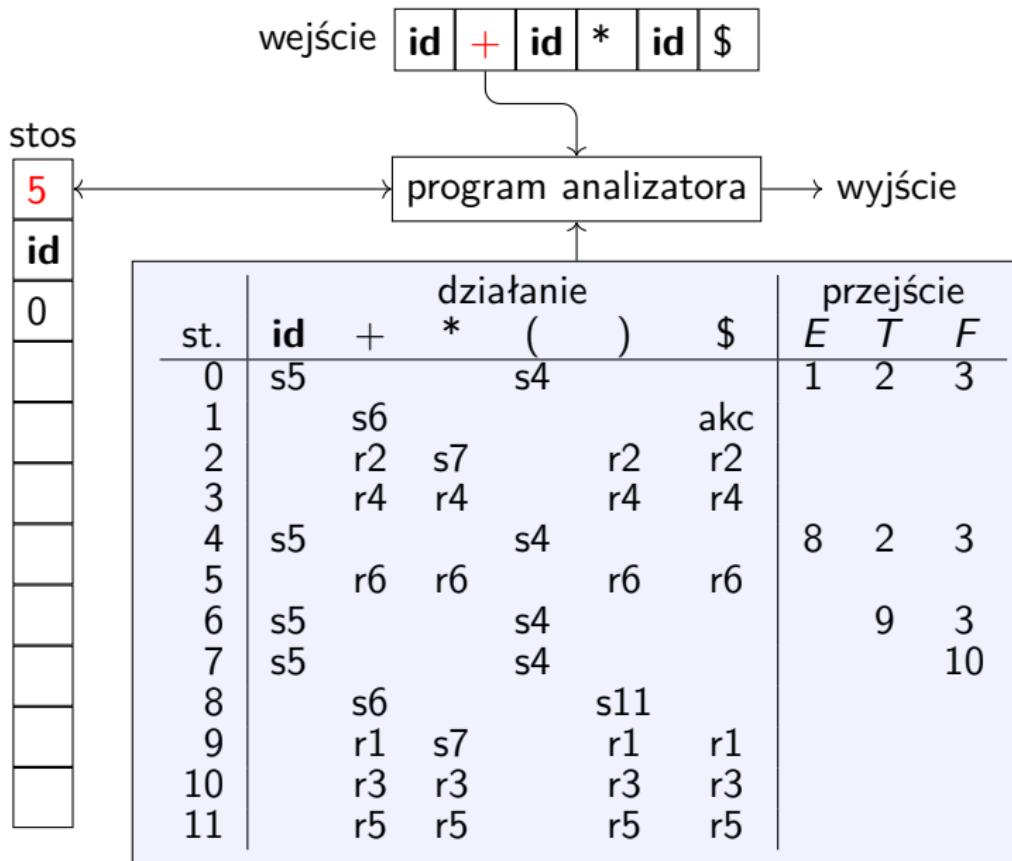
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



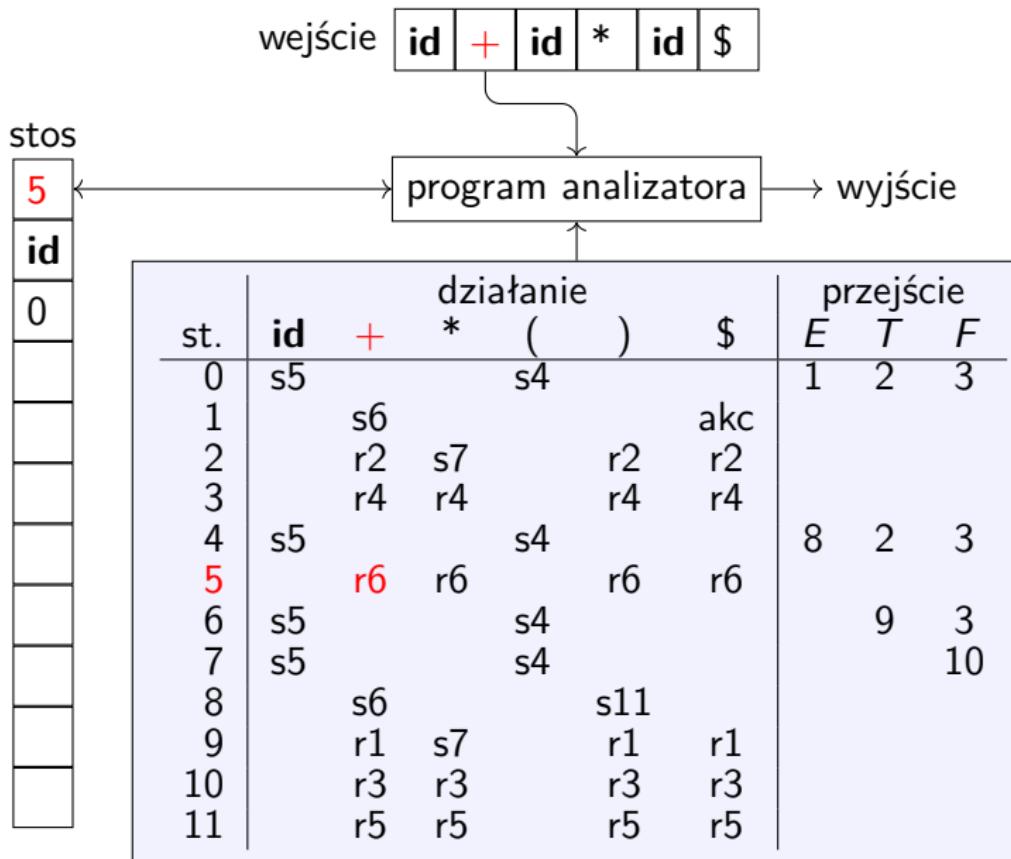
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



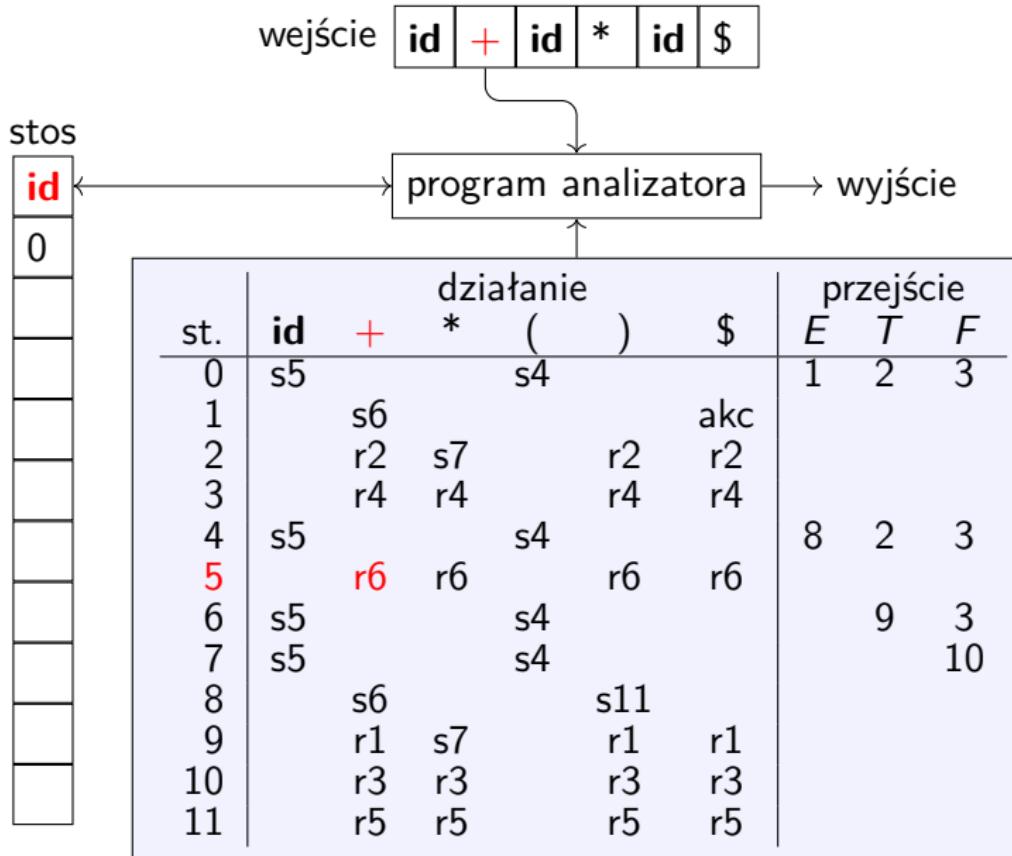
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

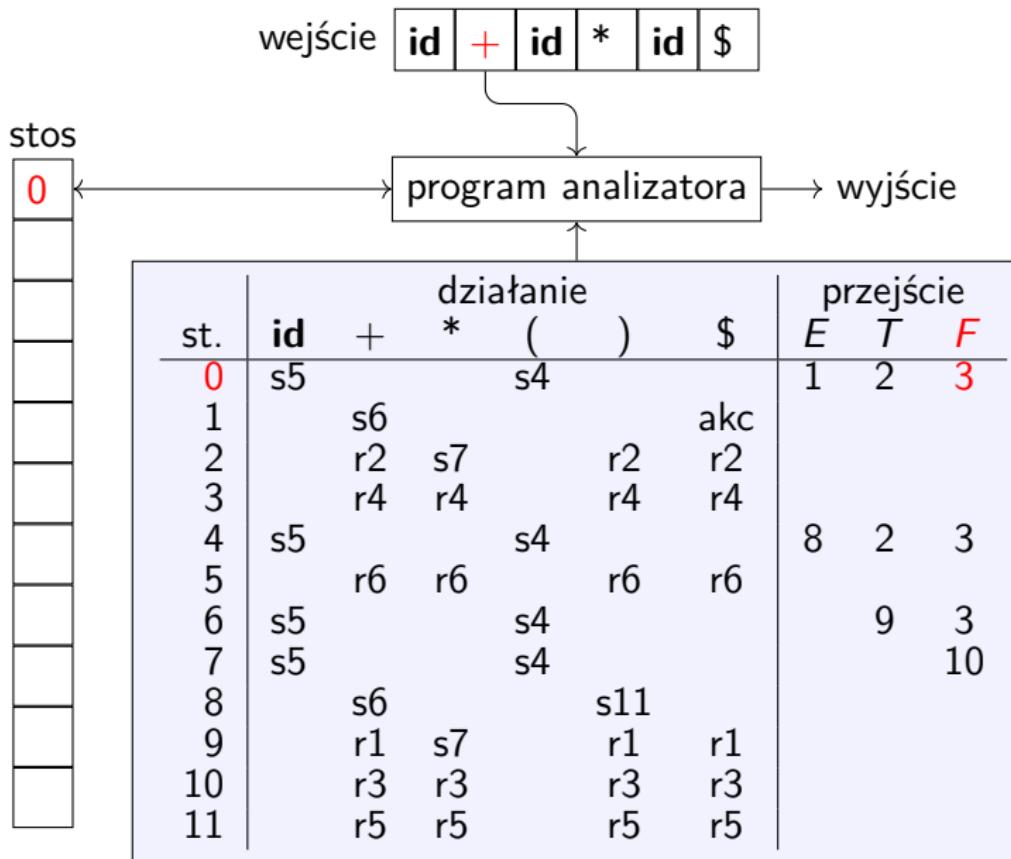


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

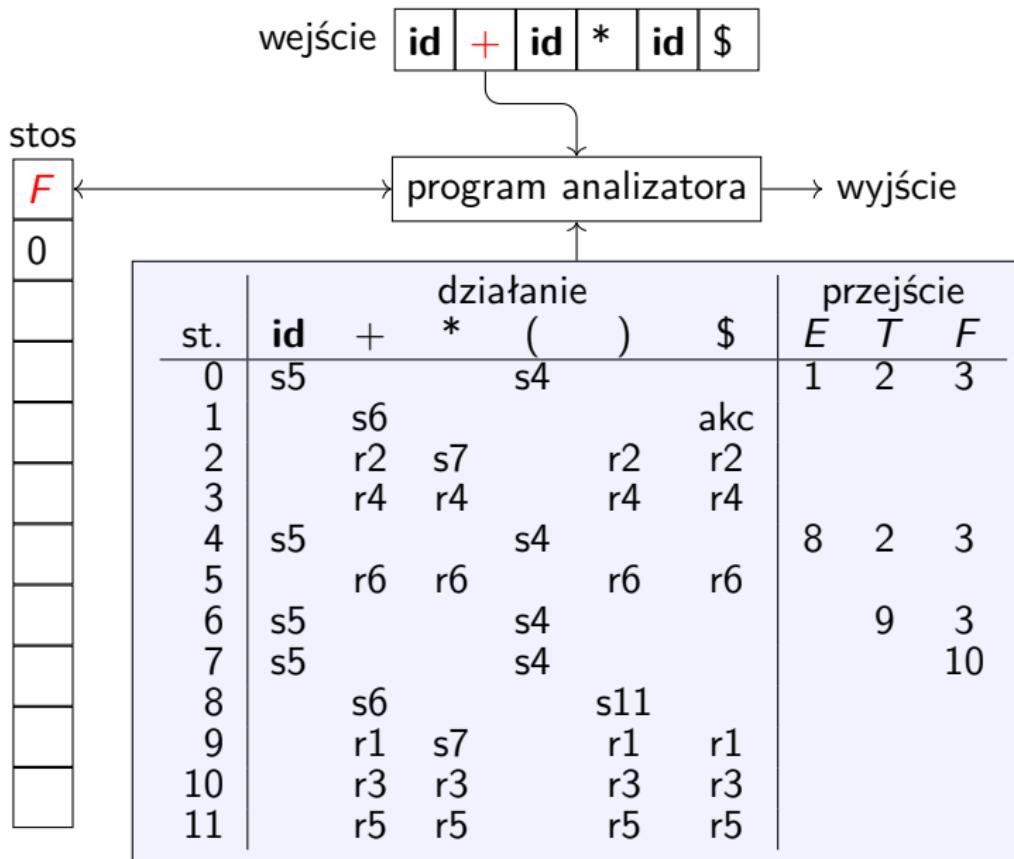


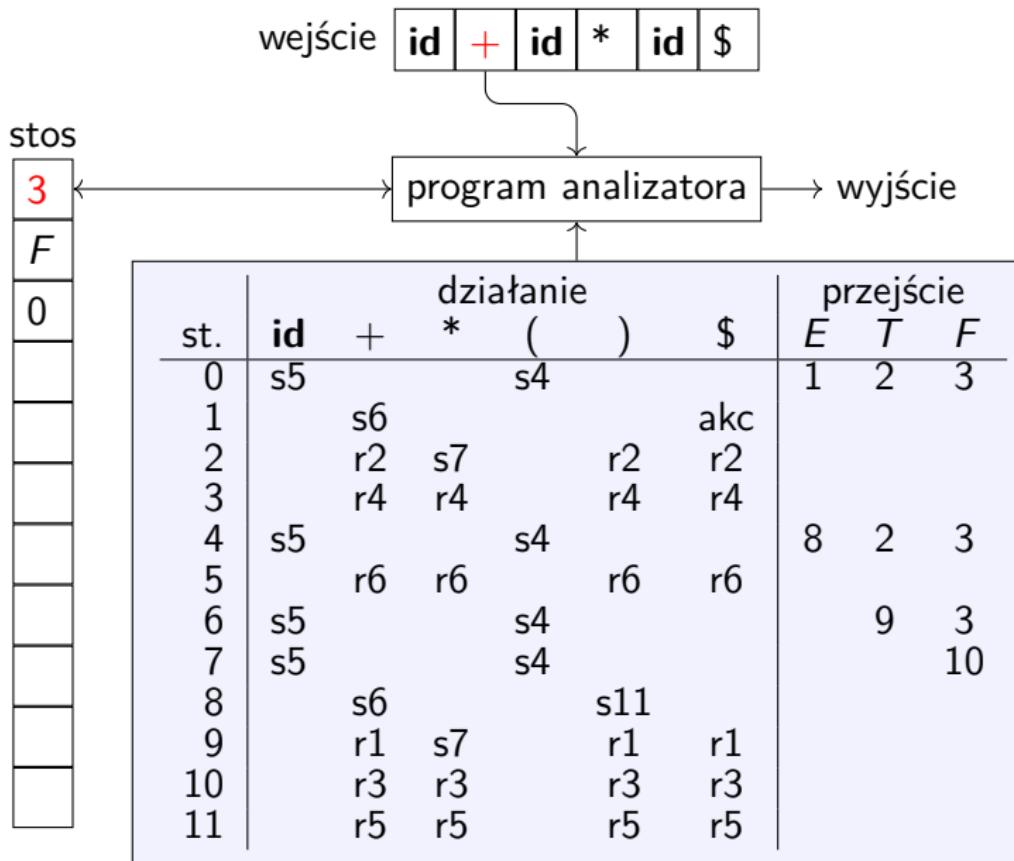
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



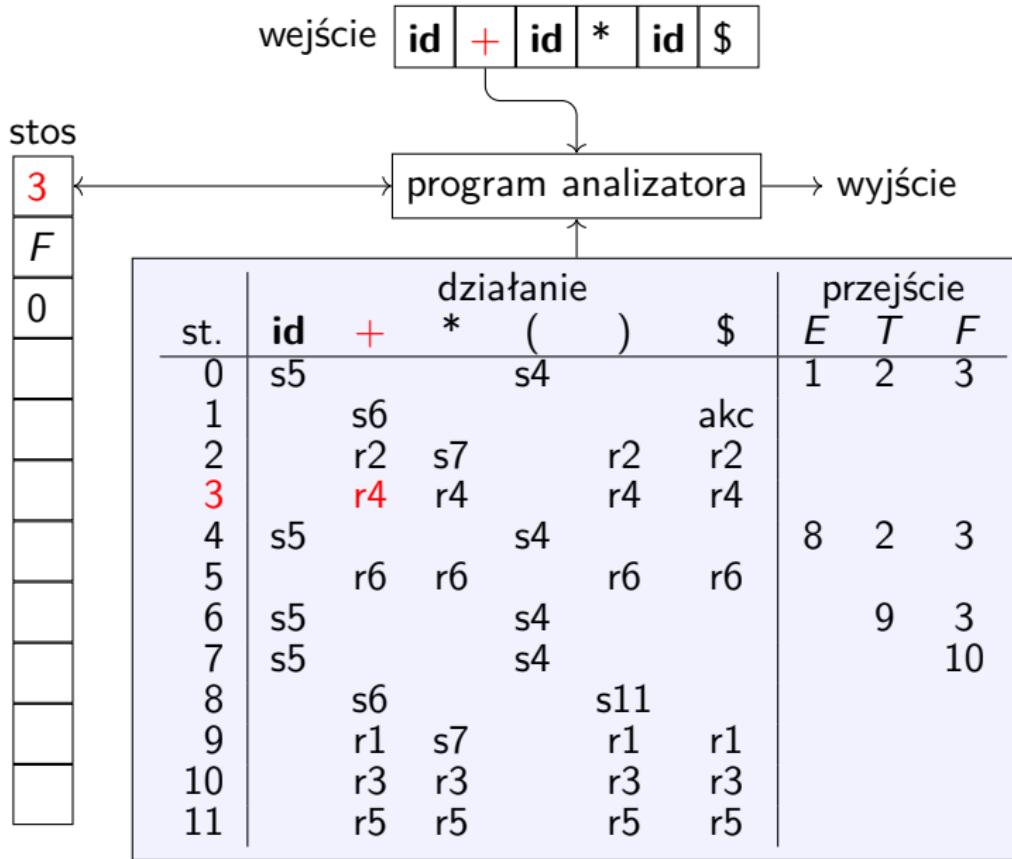


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

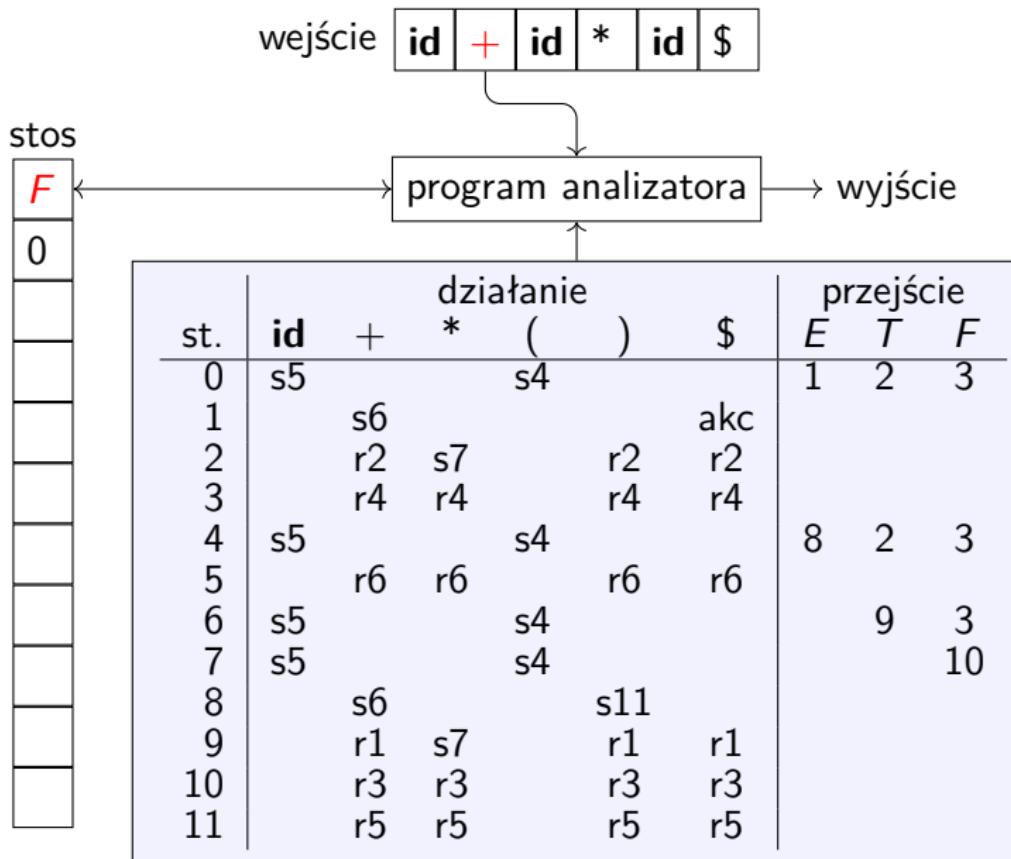




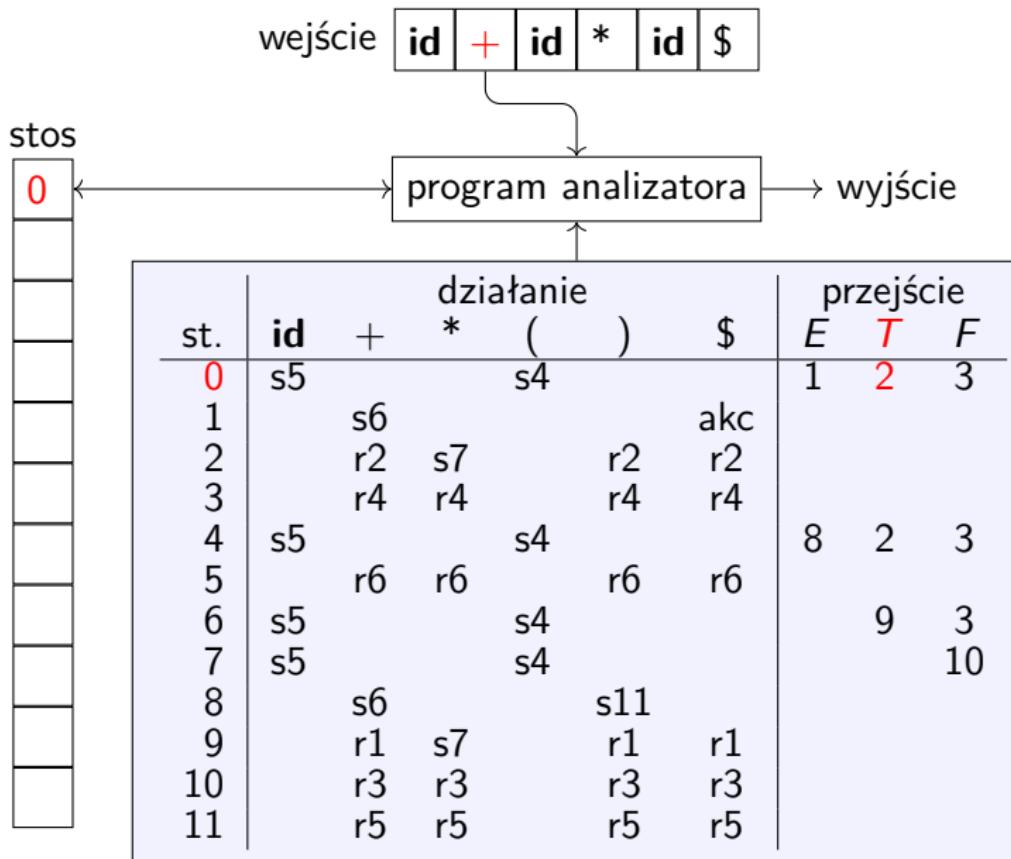
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



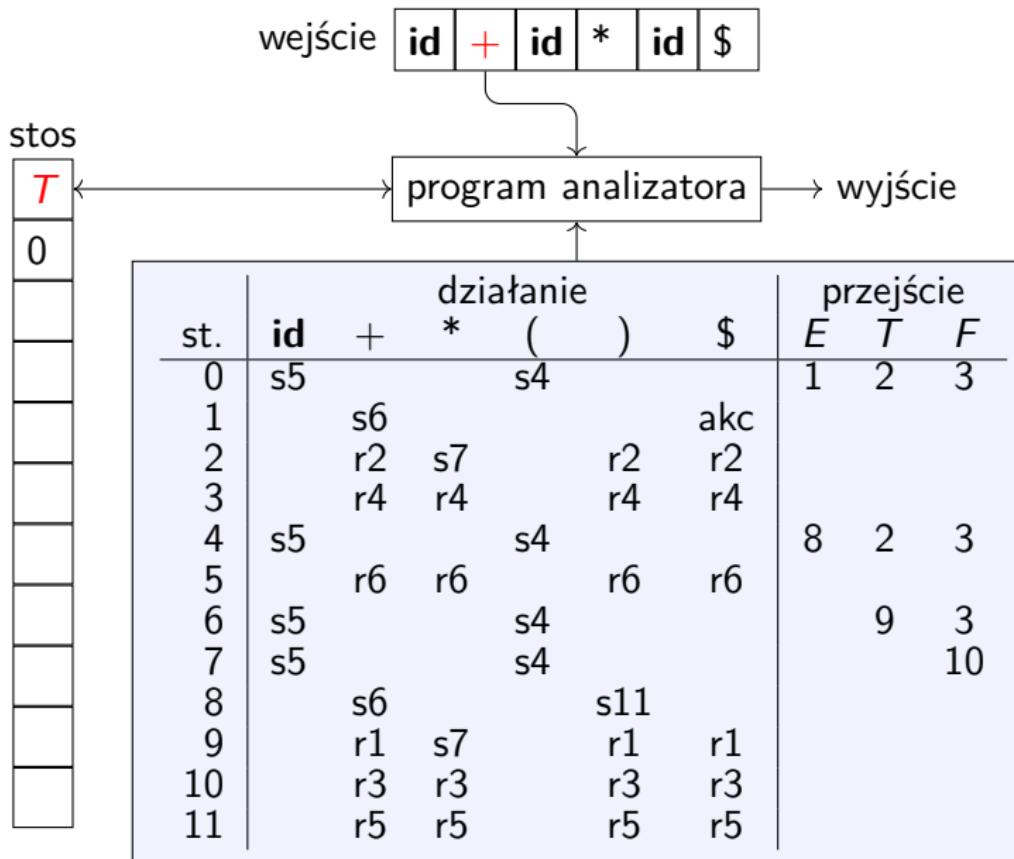
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
- 4.  $T \rightarrow F$**
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

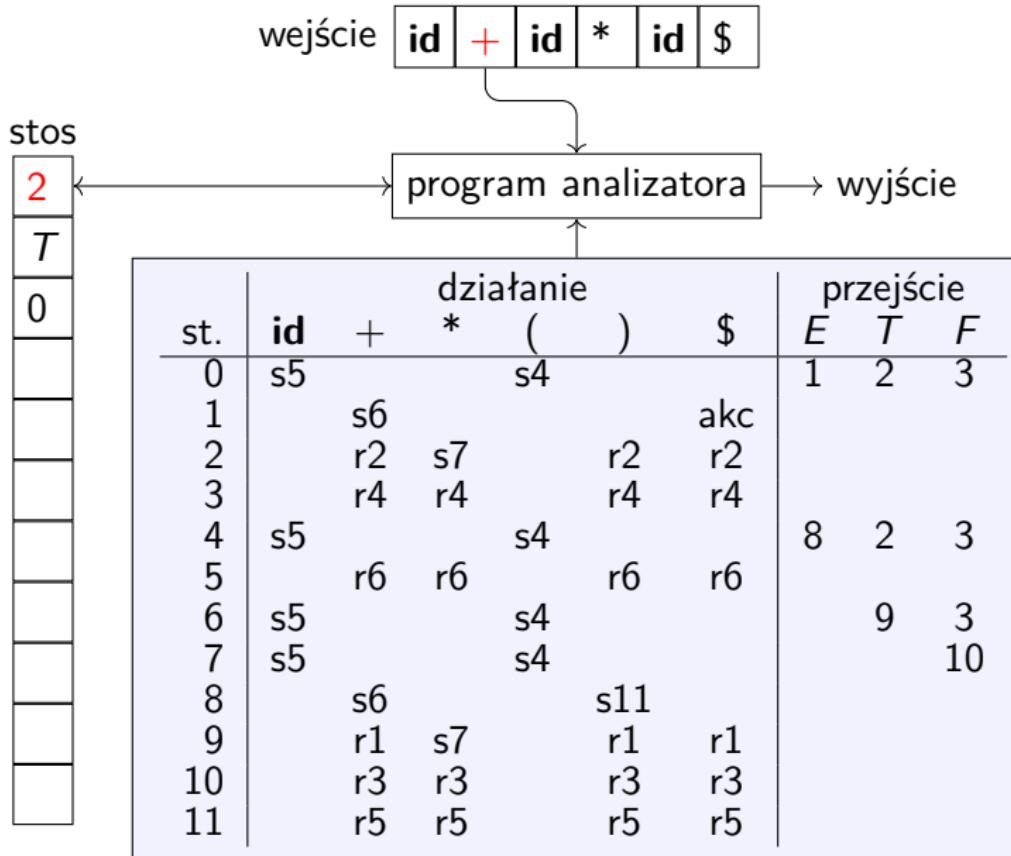


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$

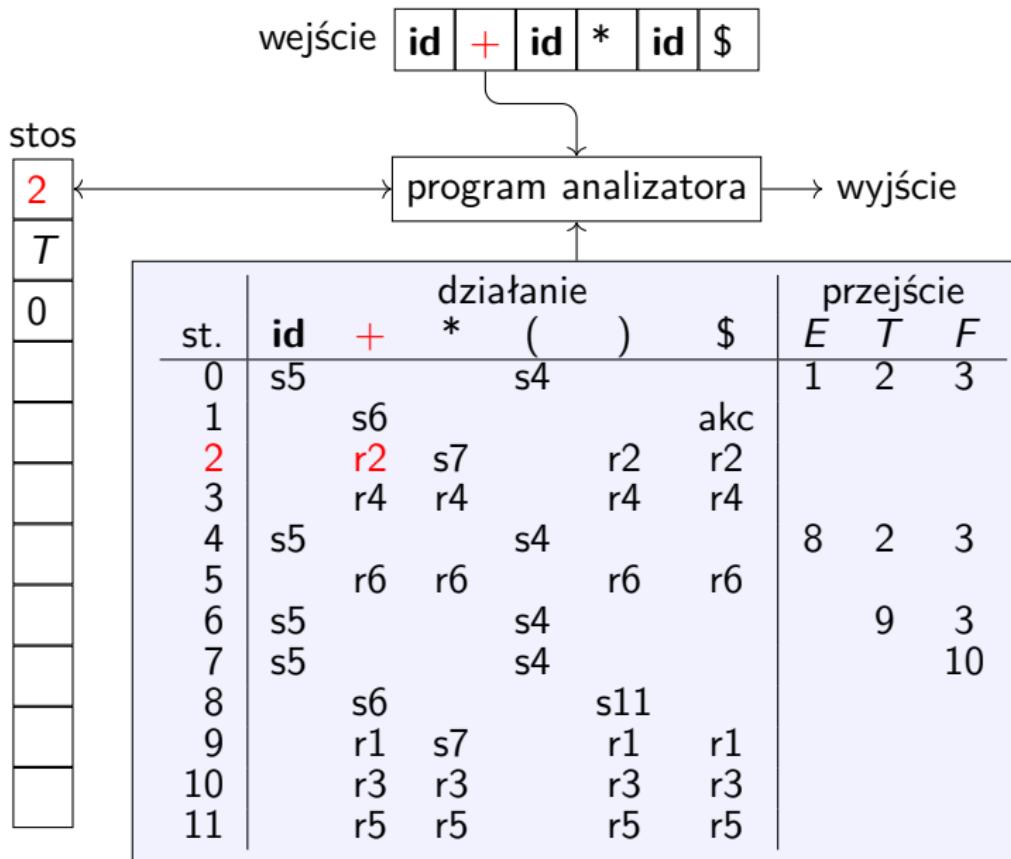


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

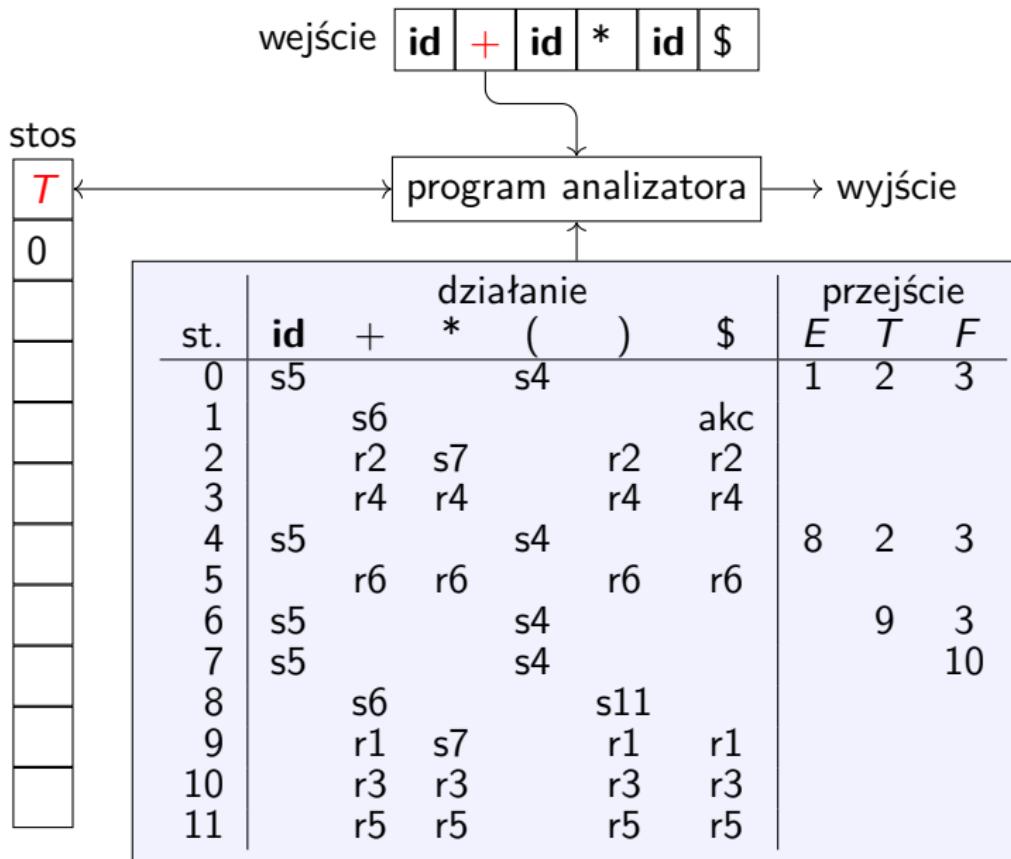


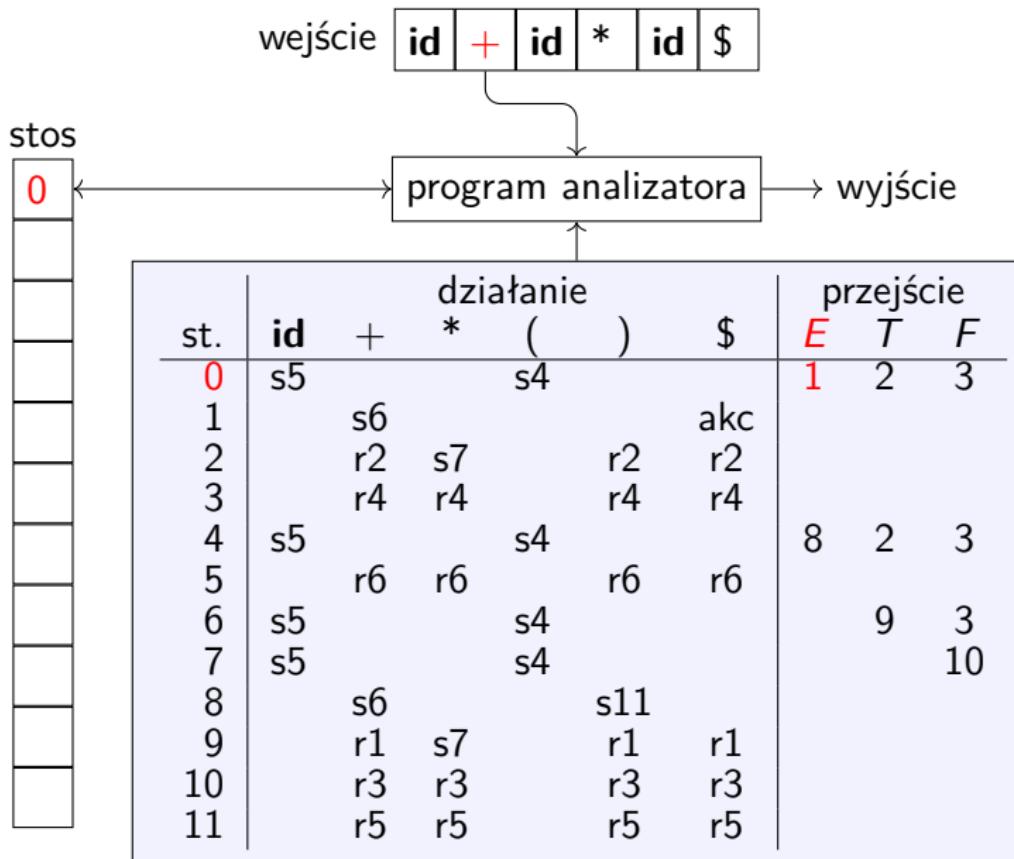


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

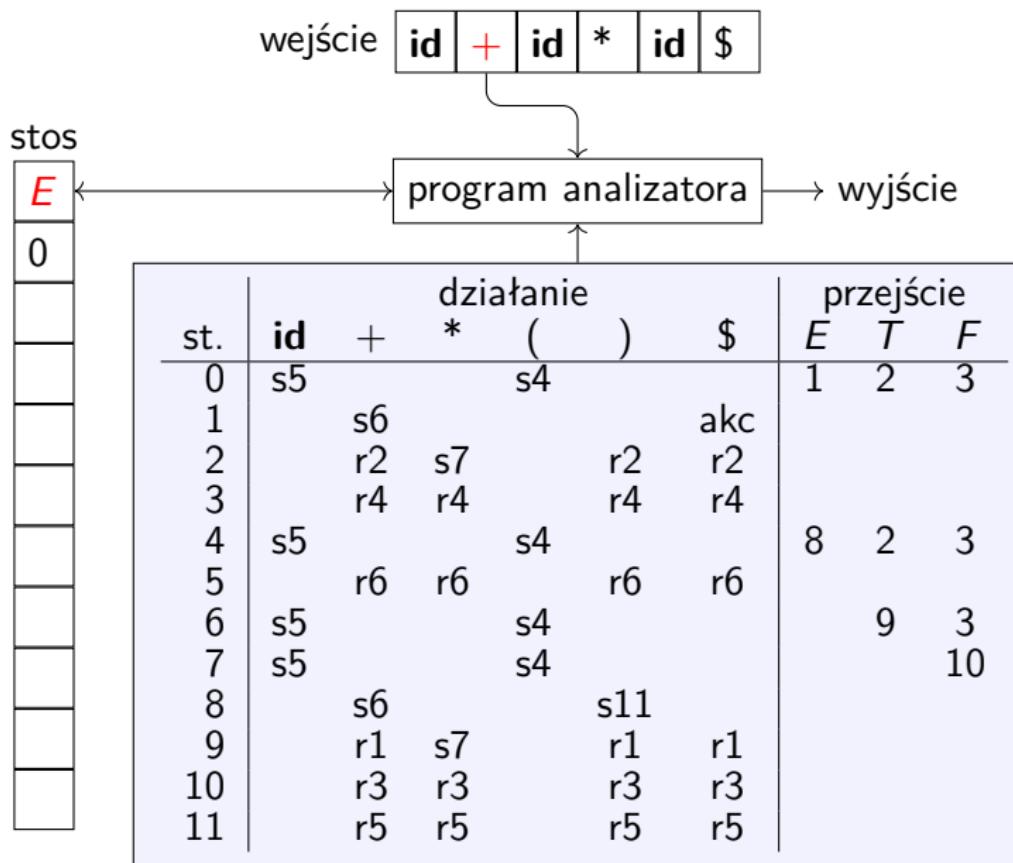


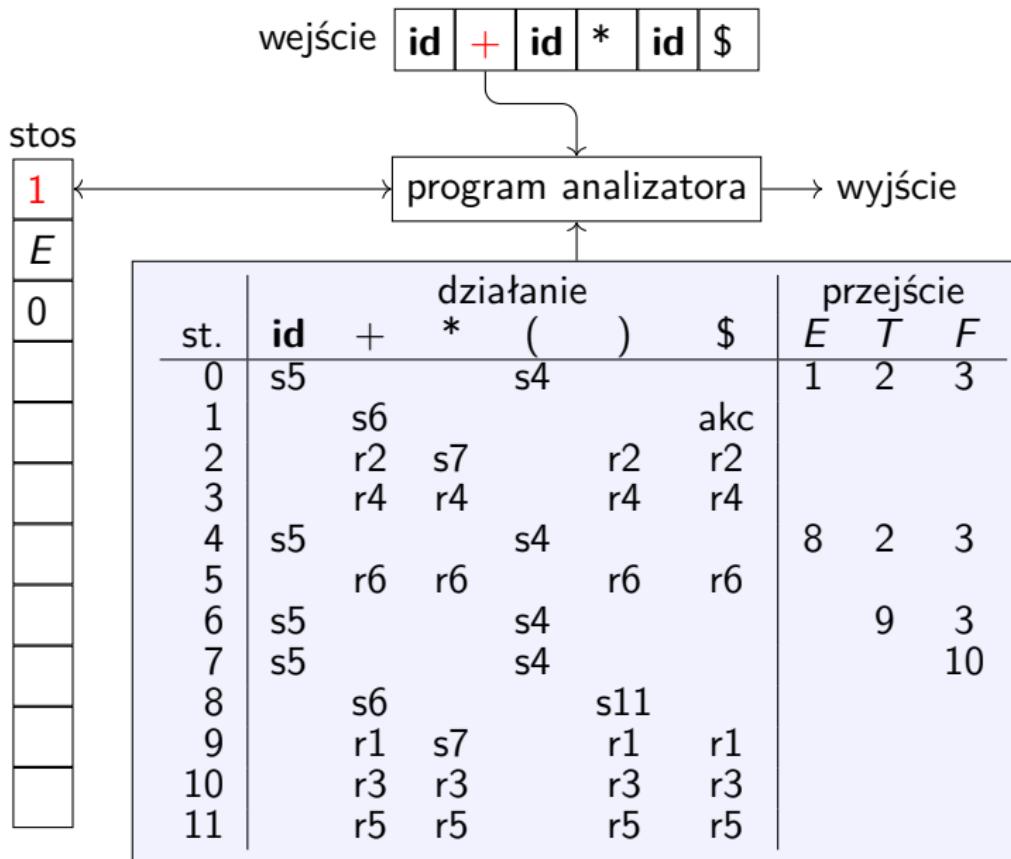
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



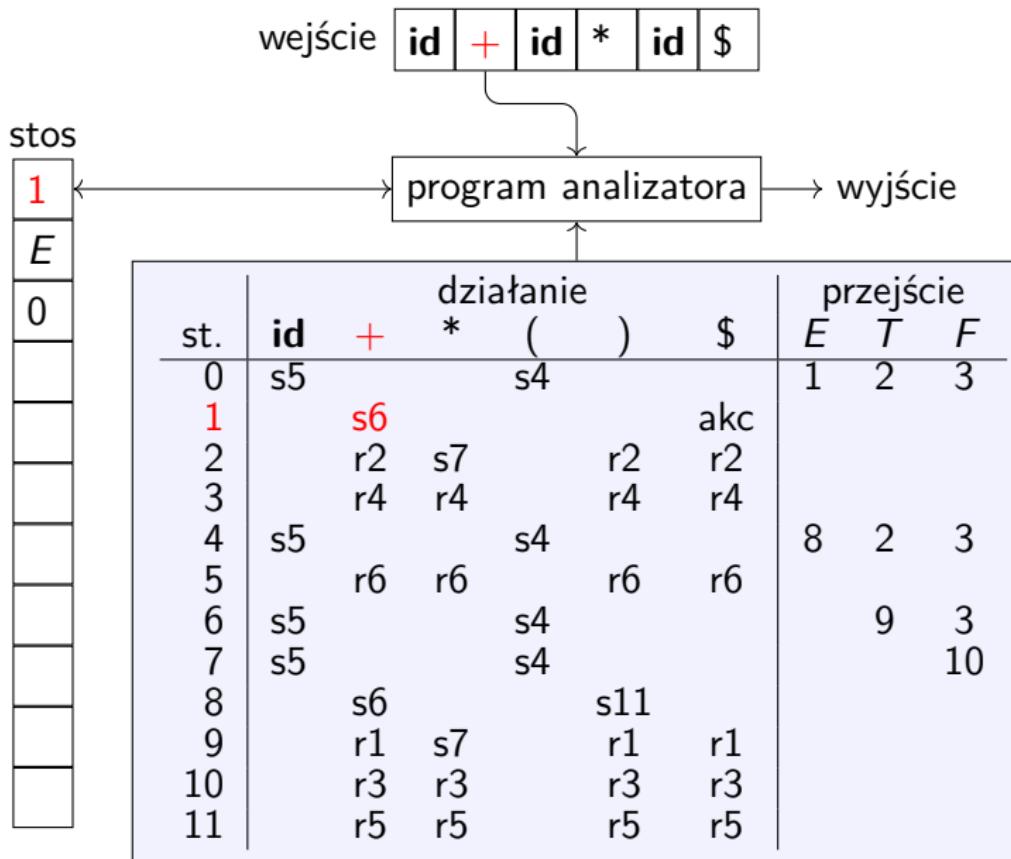


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

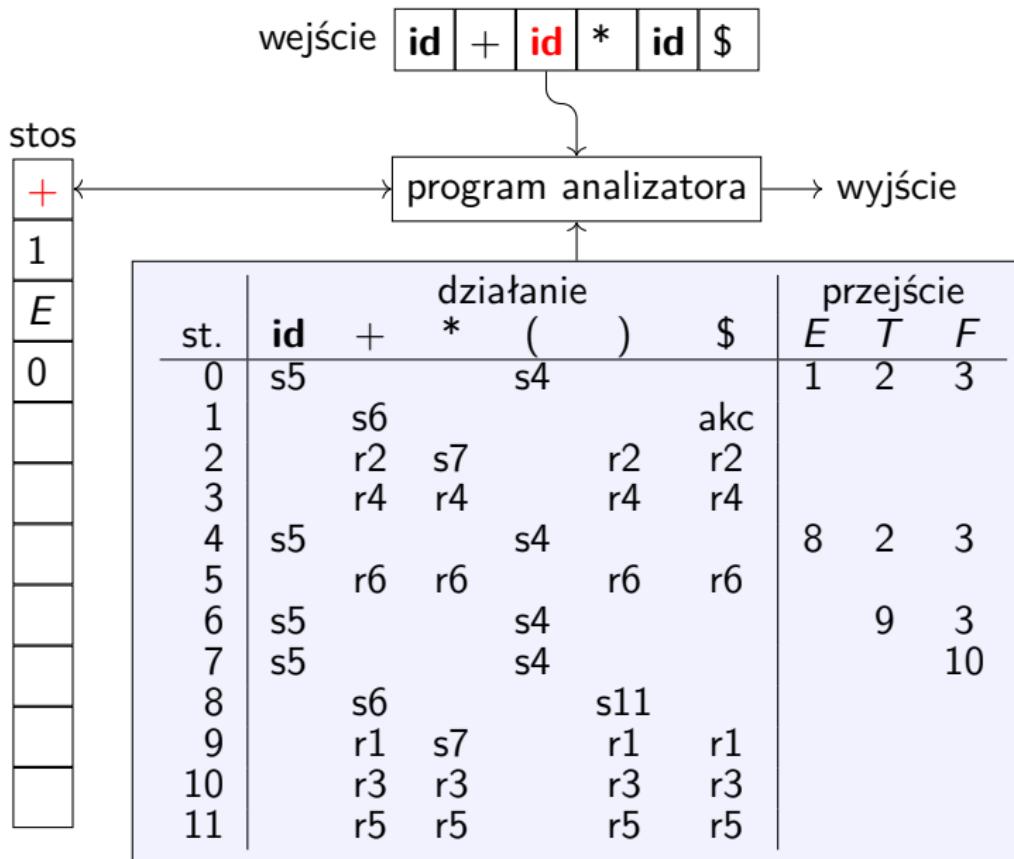




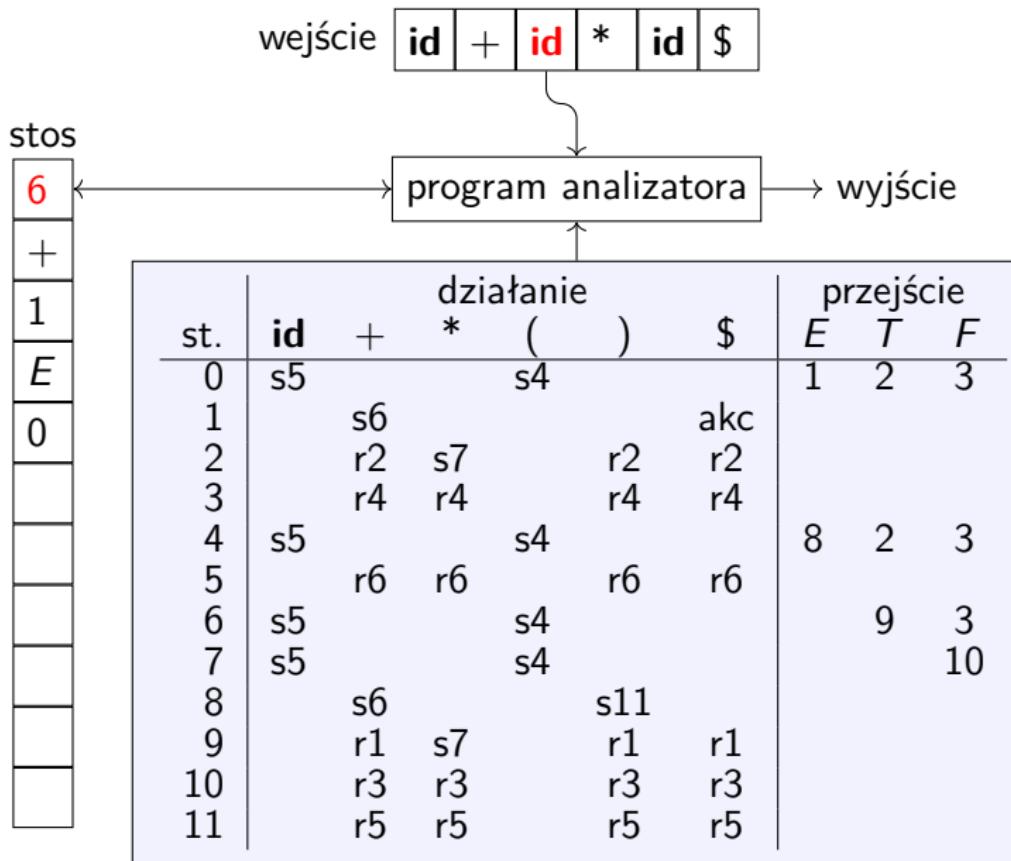
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



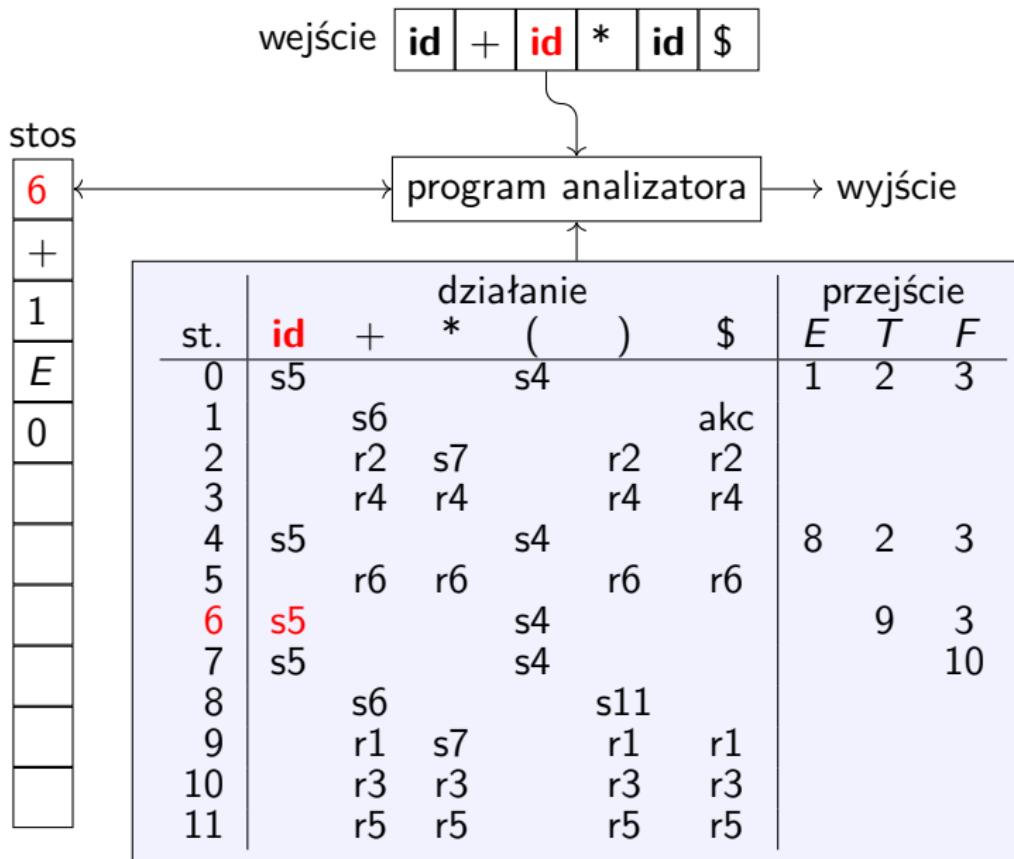
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



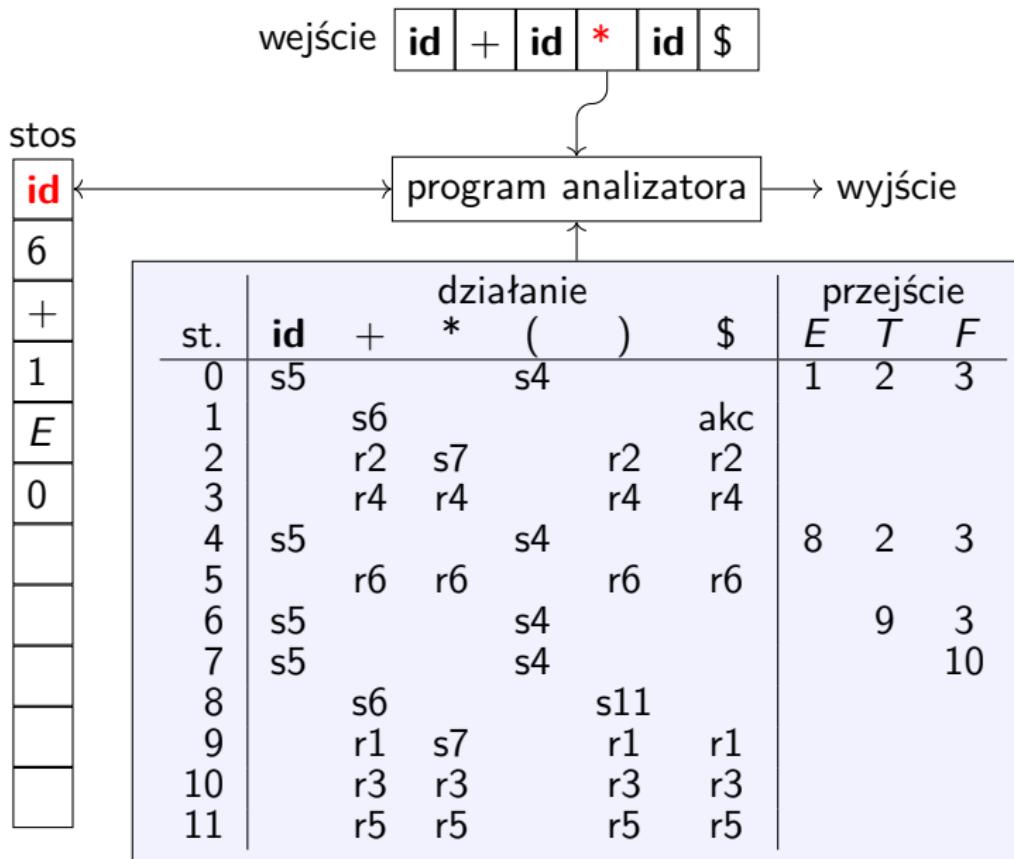
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



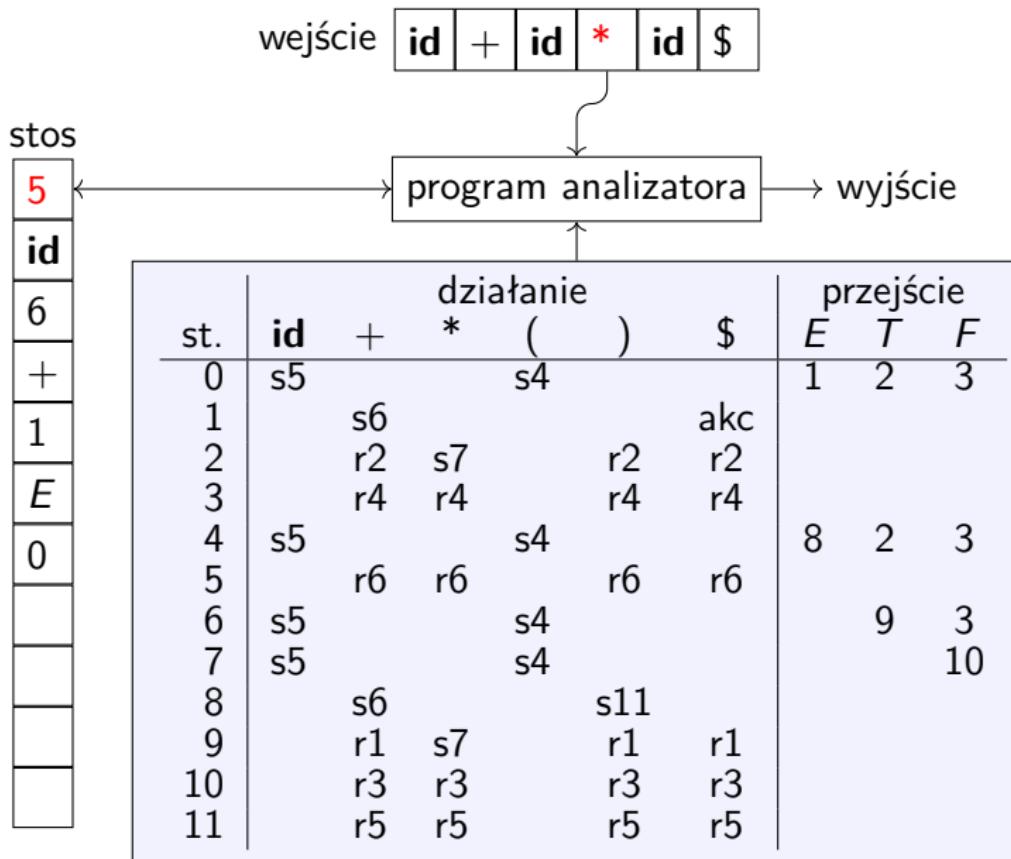
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



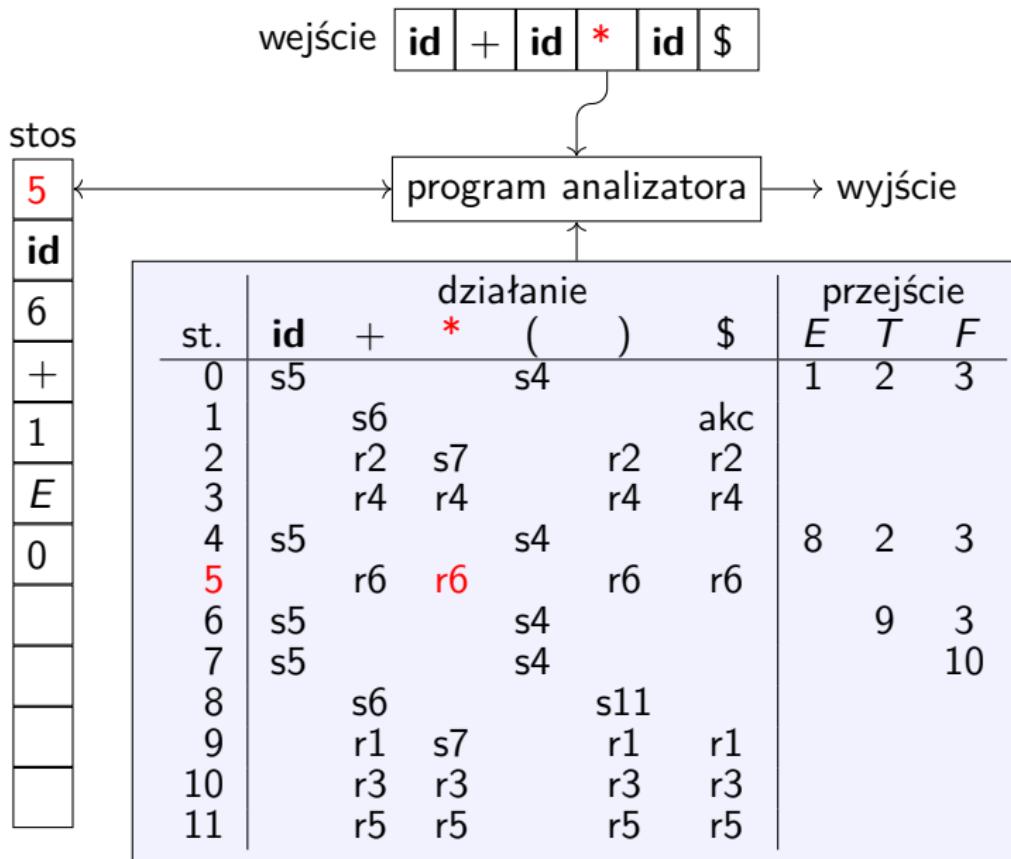
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



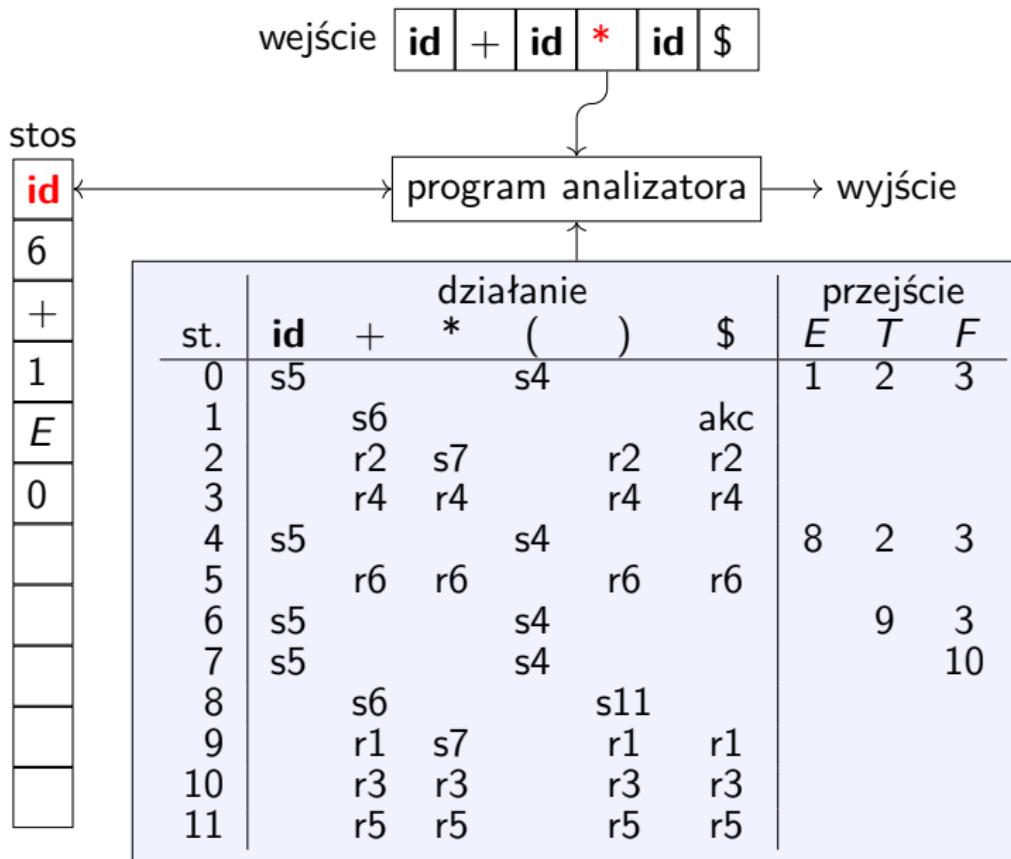
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

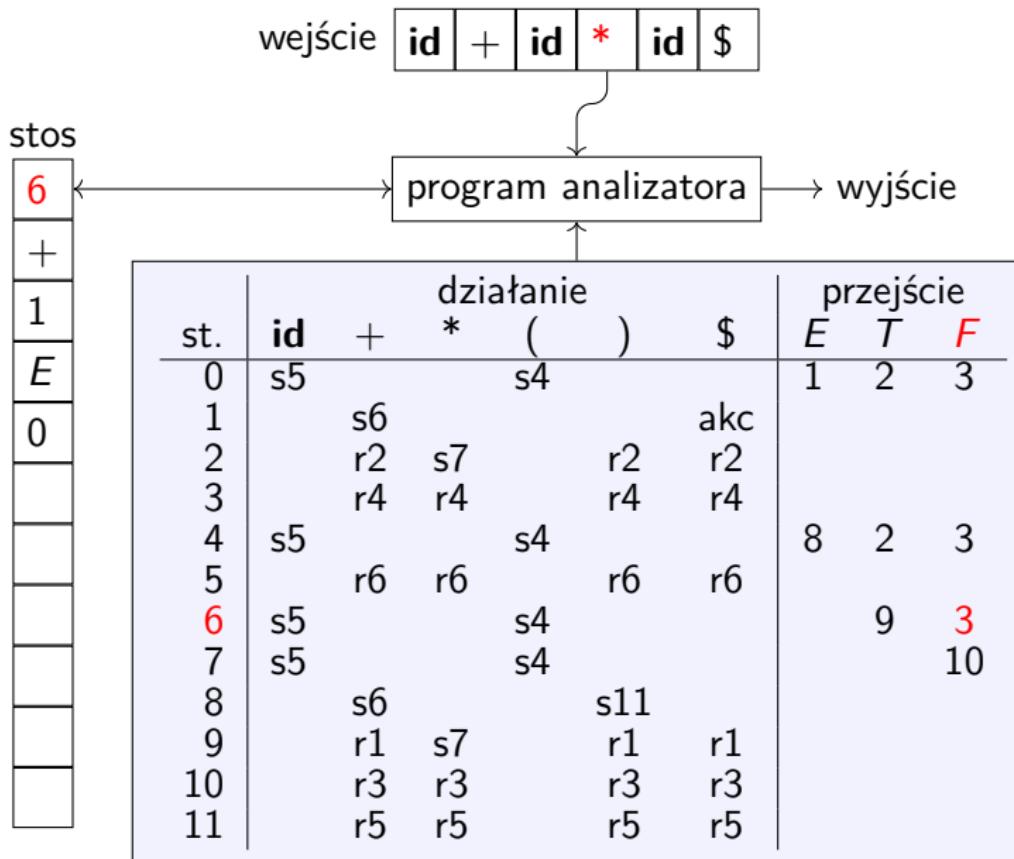


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

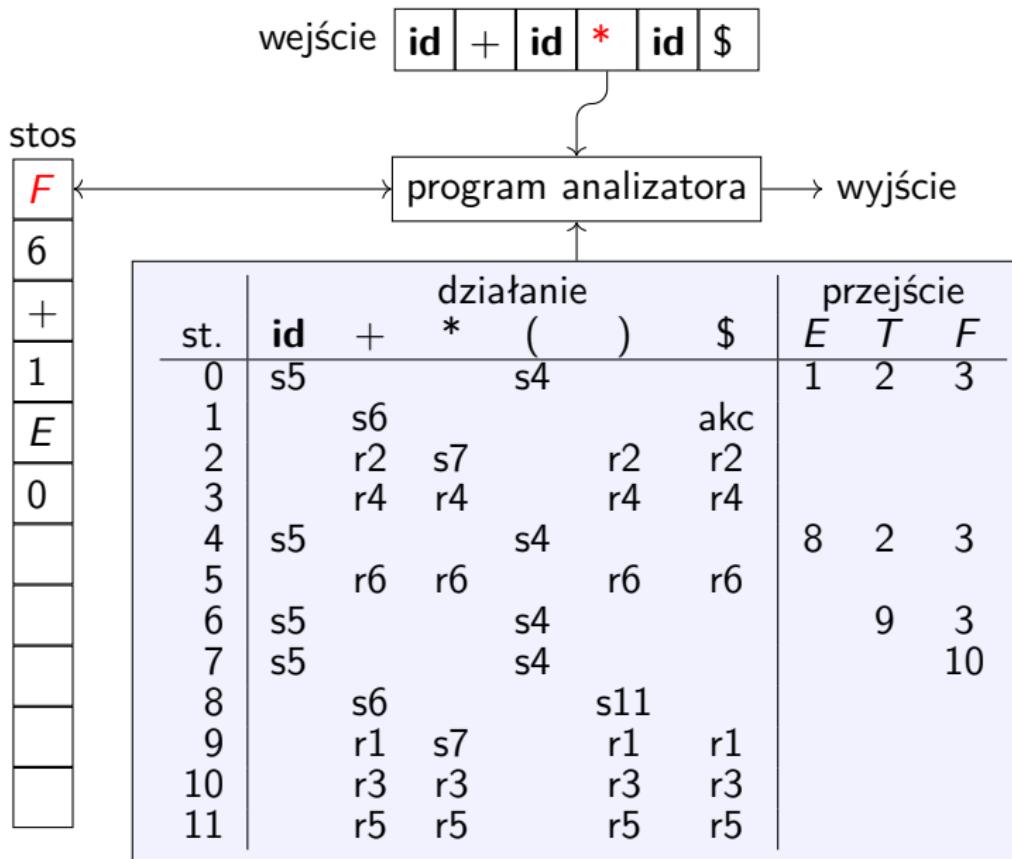


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

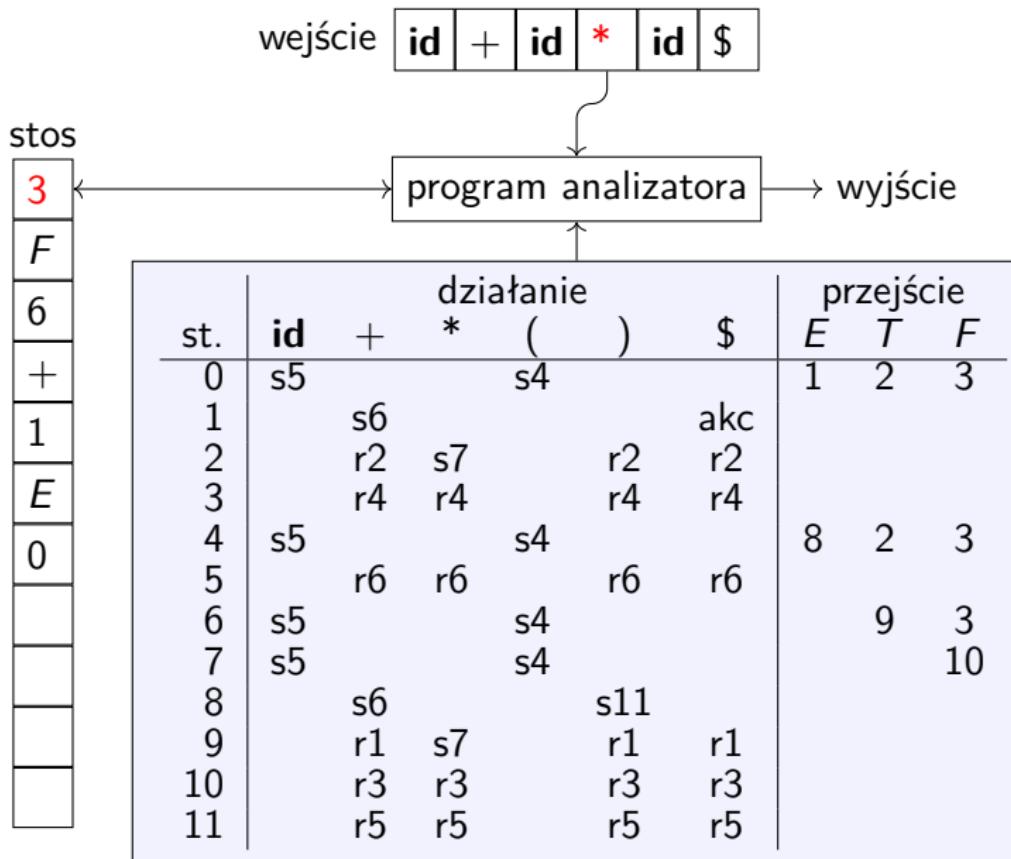




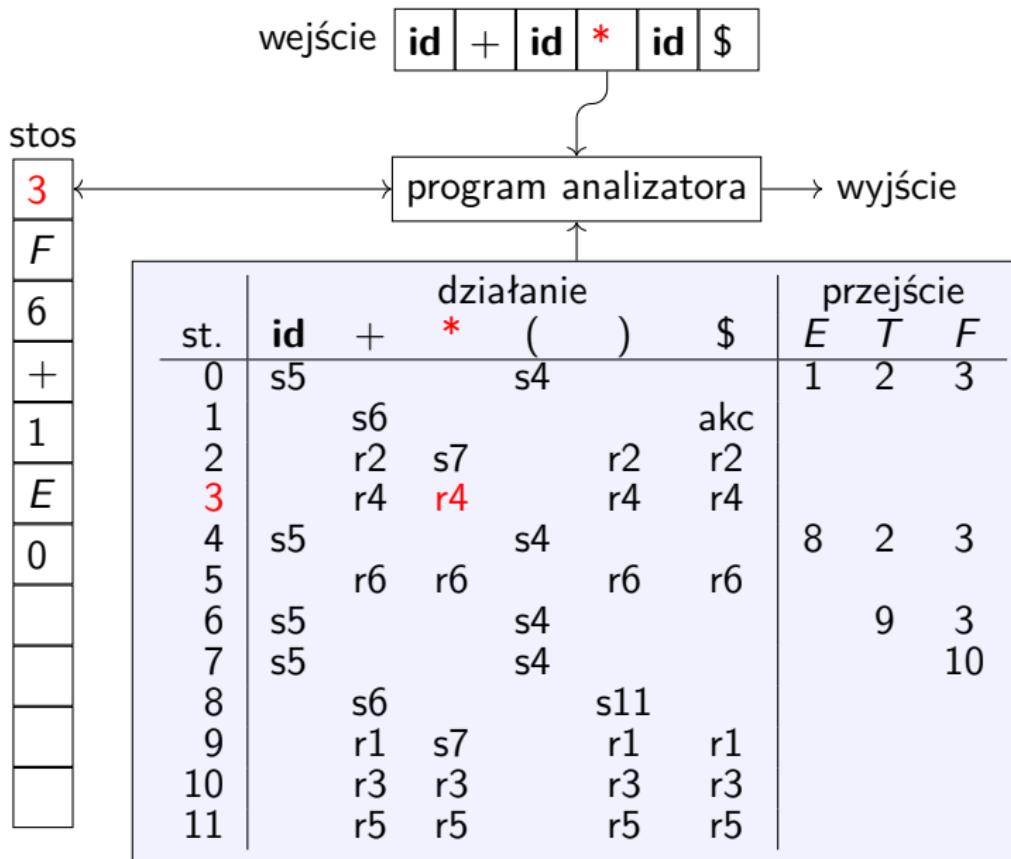
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



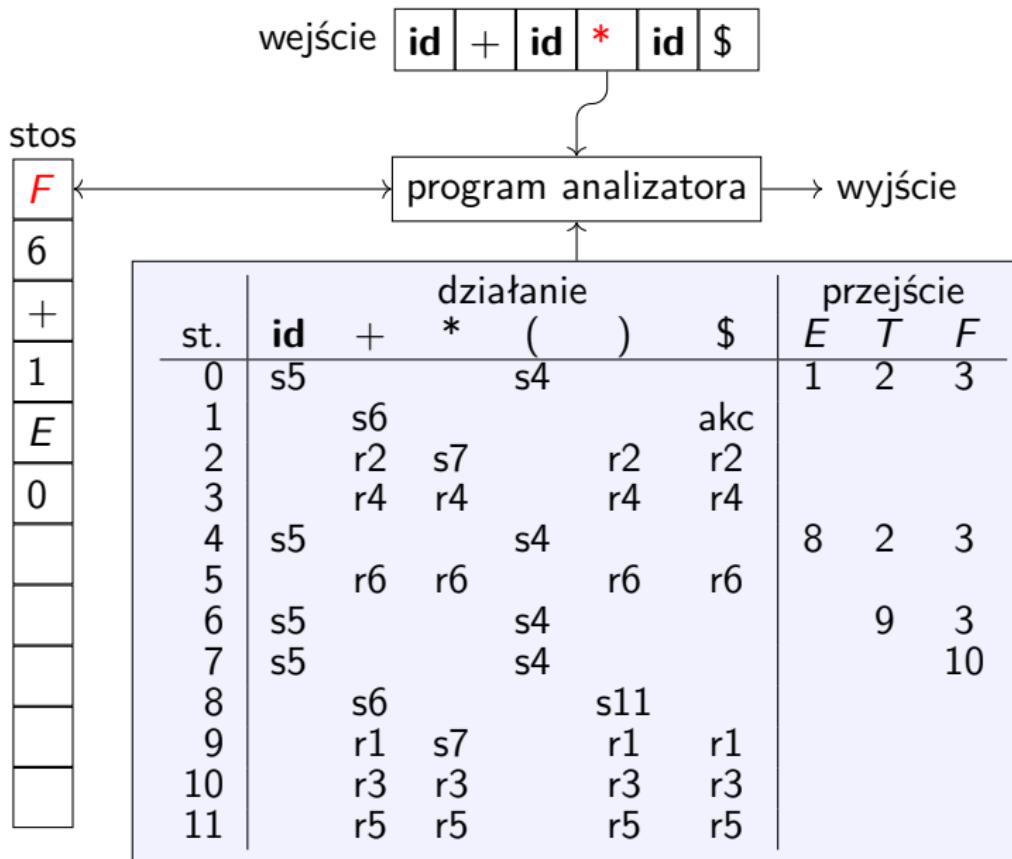
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



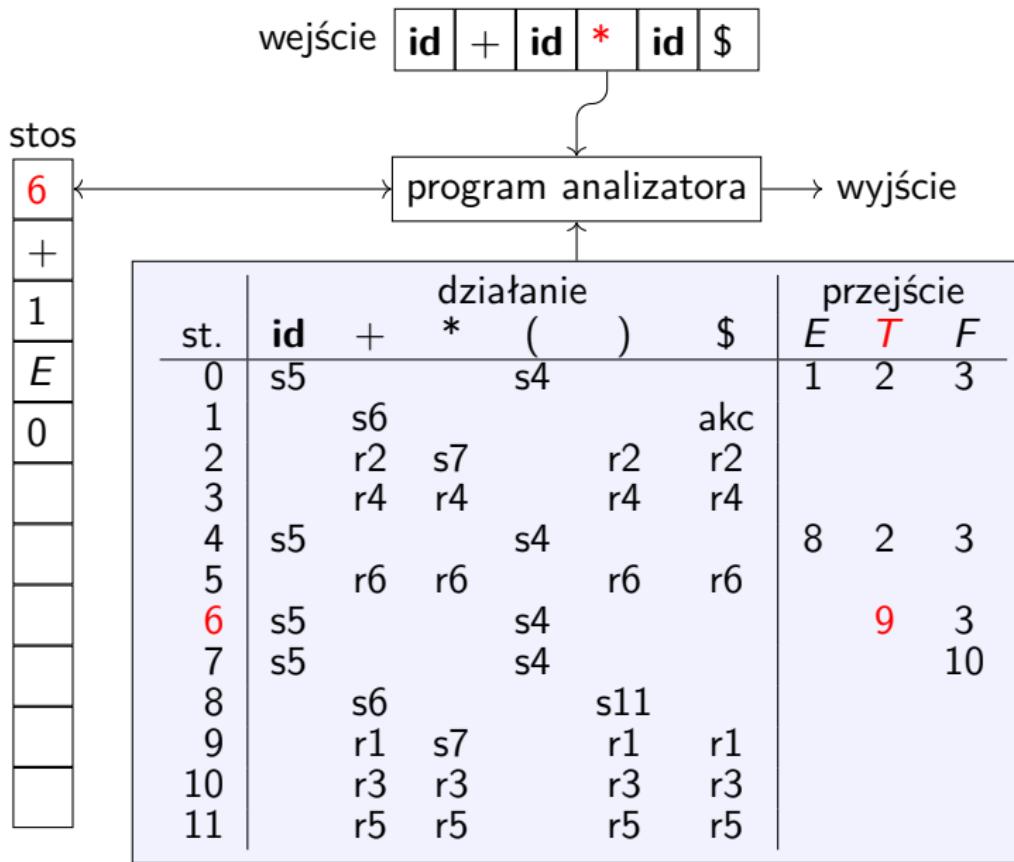
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



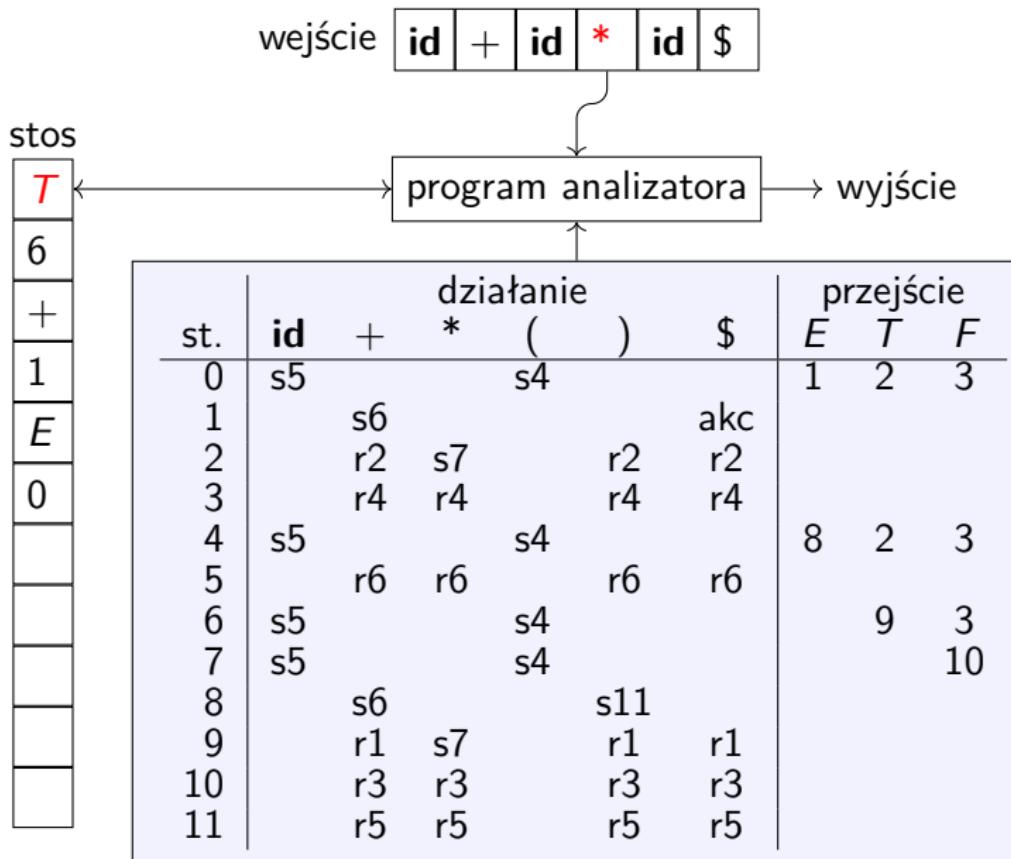
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



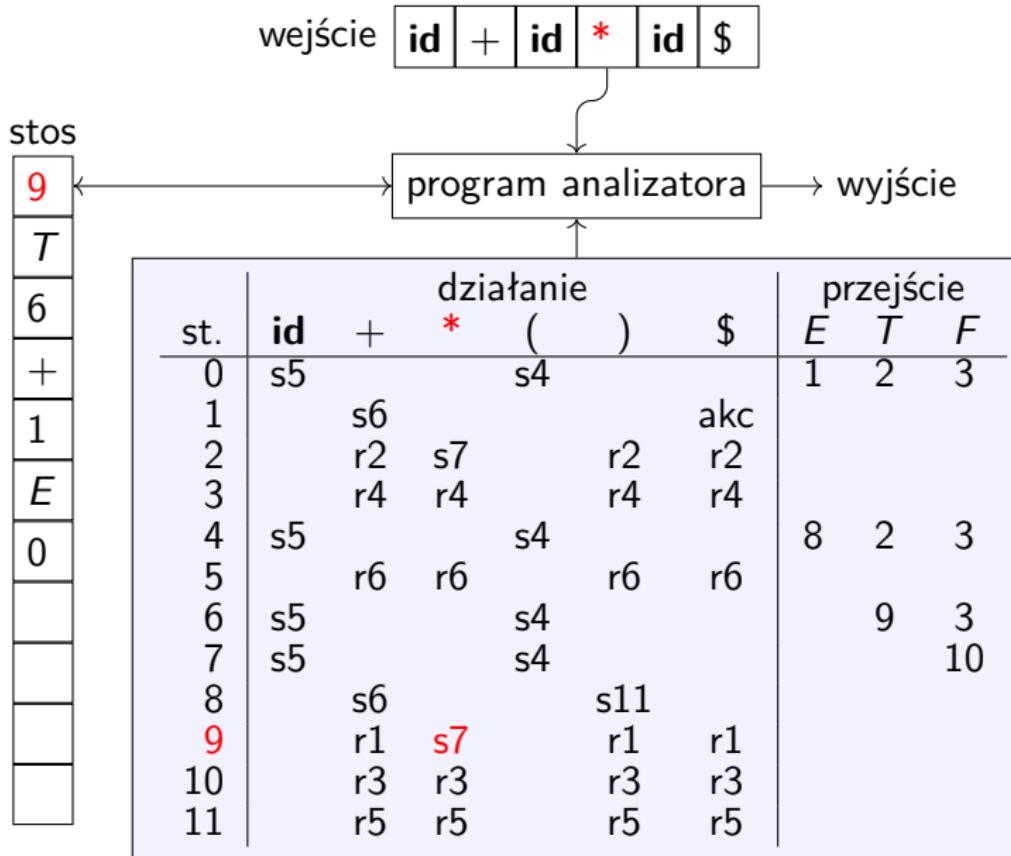
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



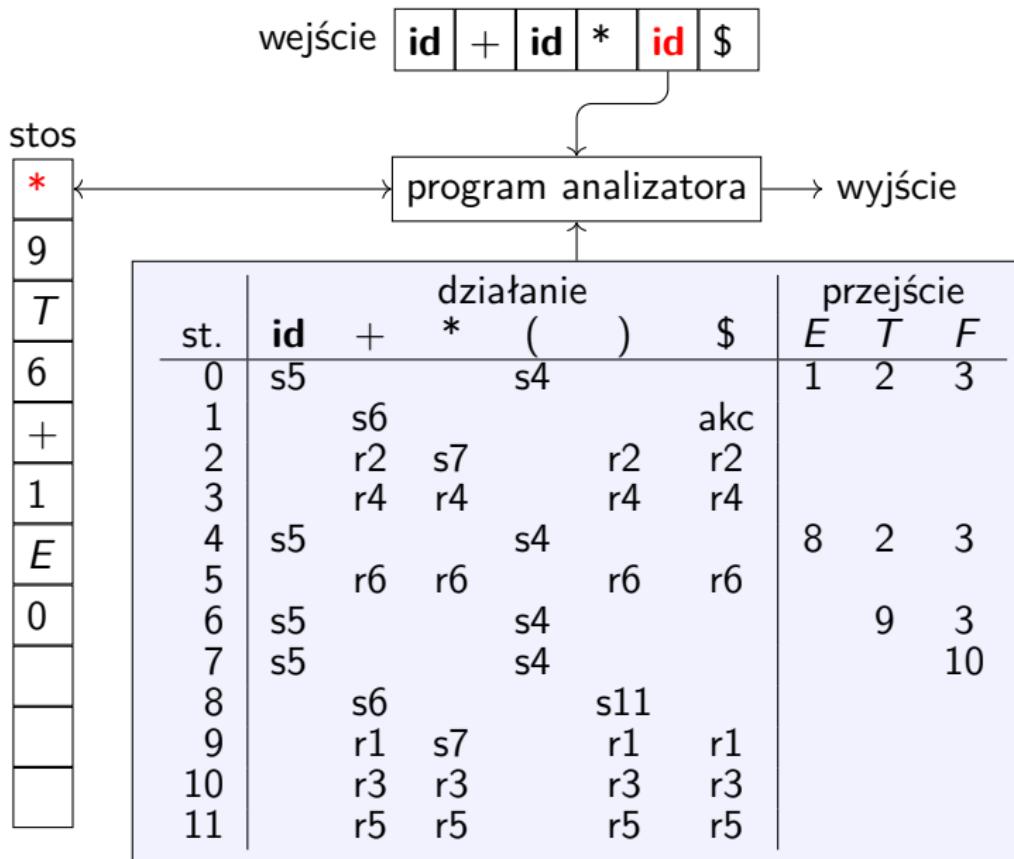
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



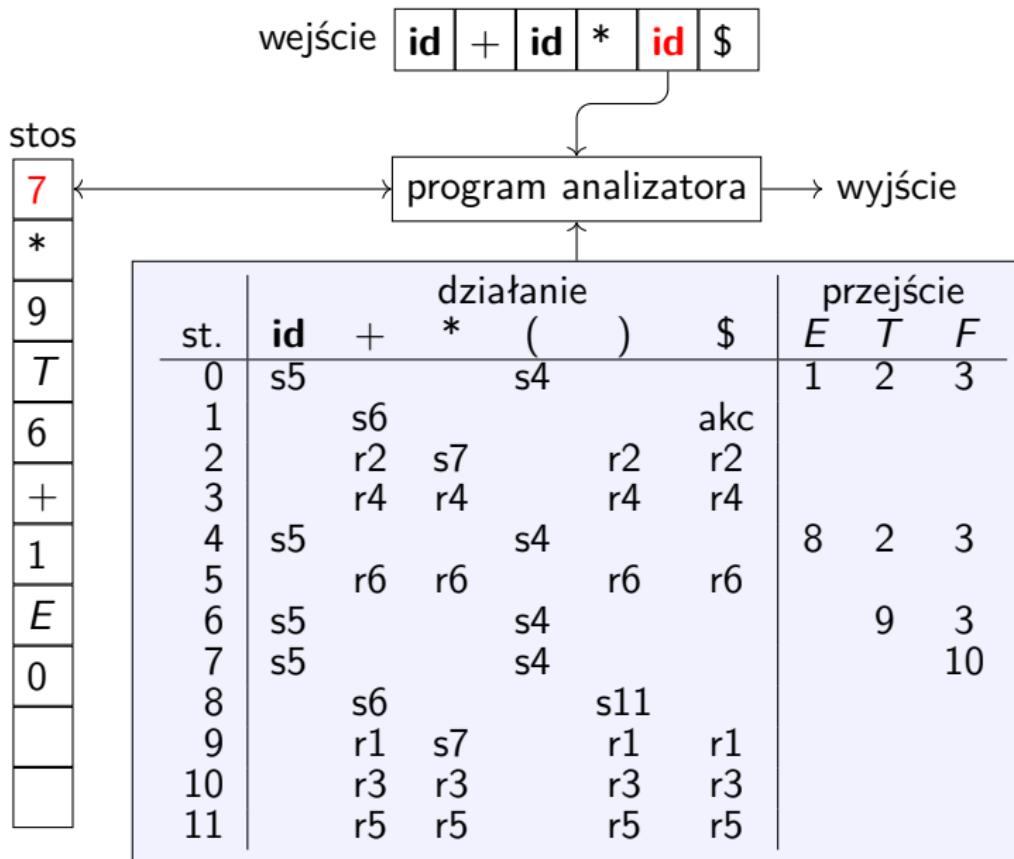
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



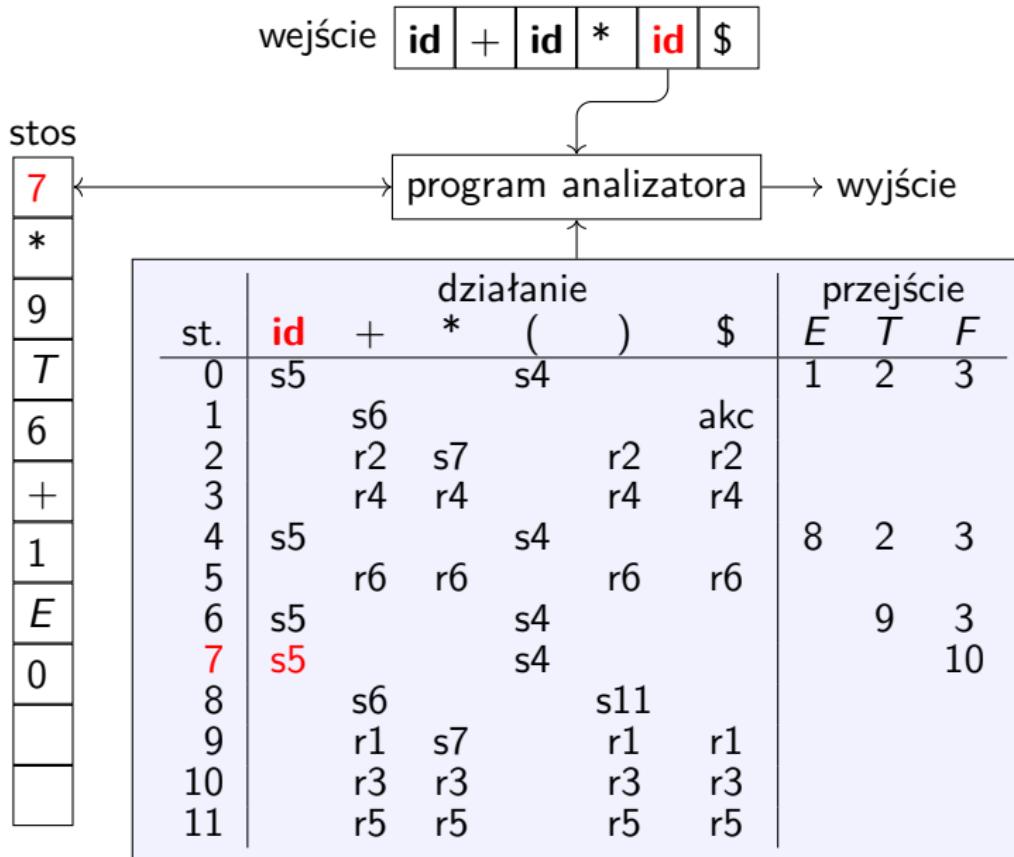
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



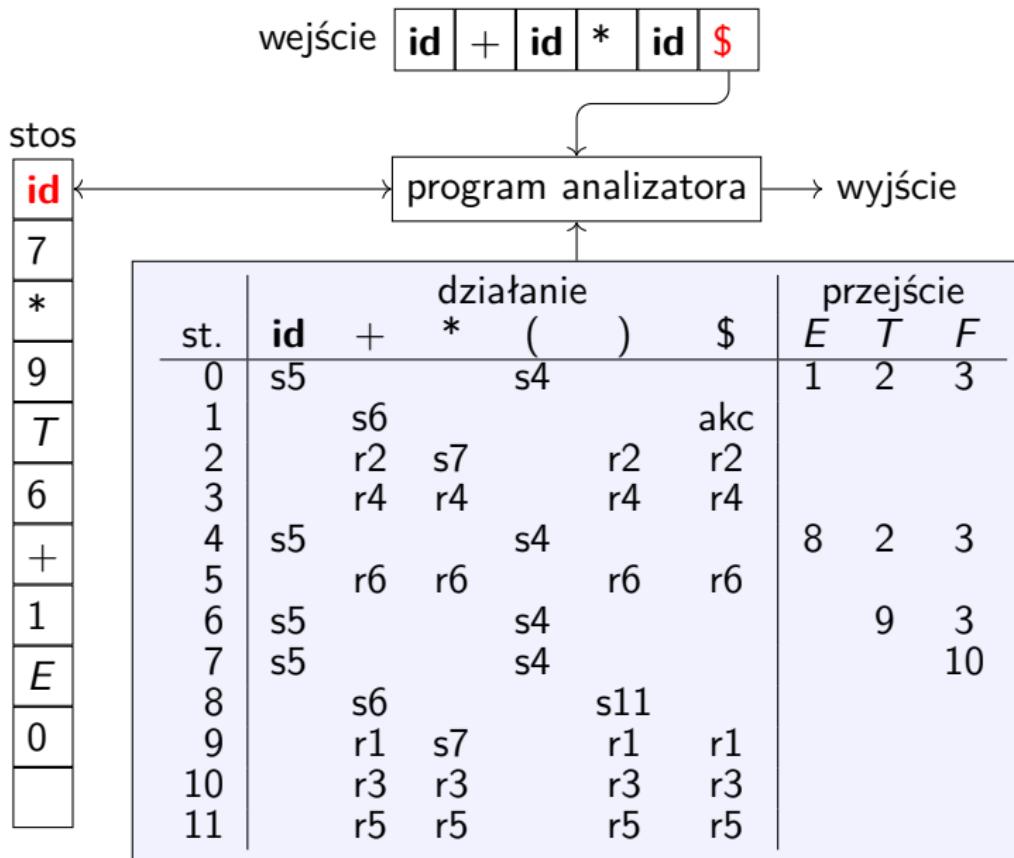
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



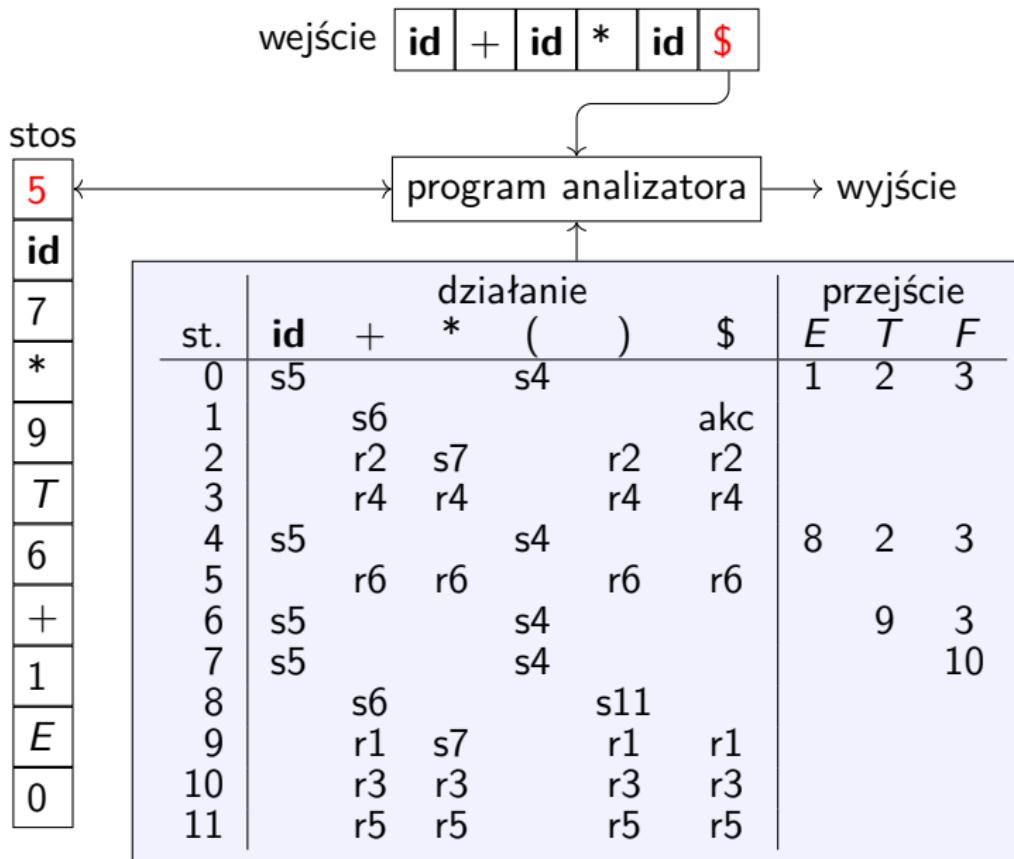
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



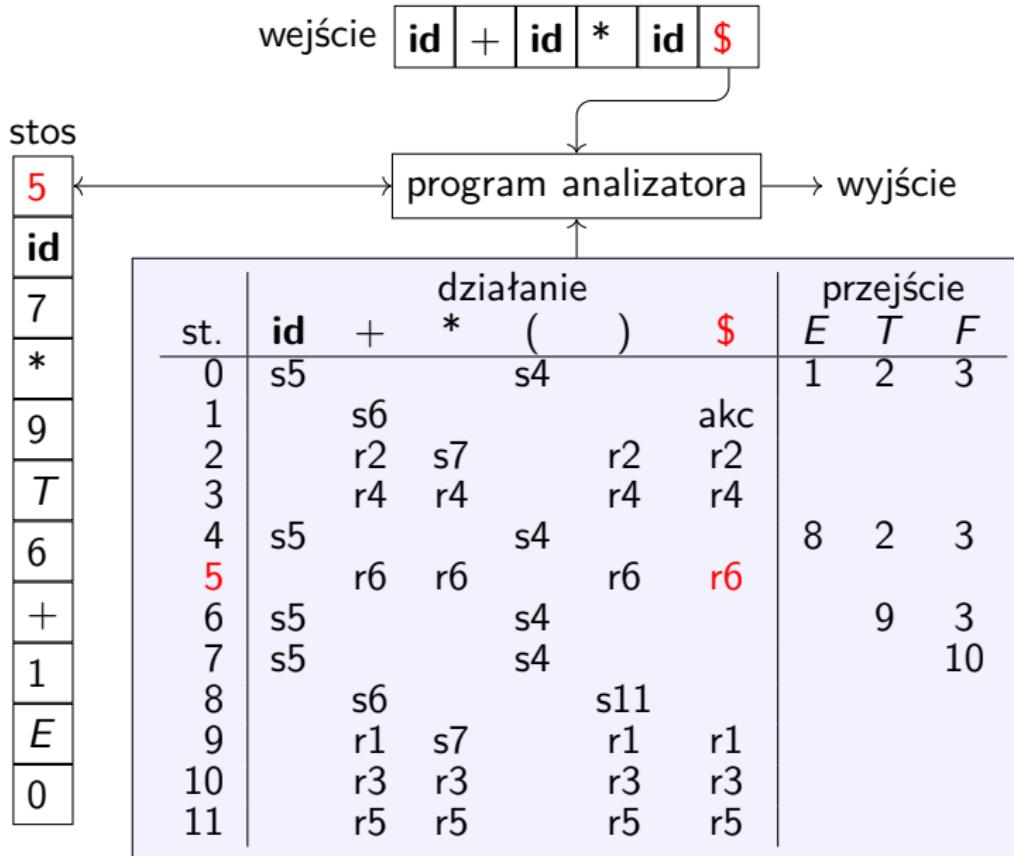
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



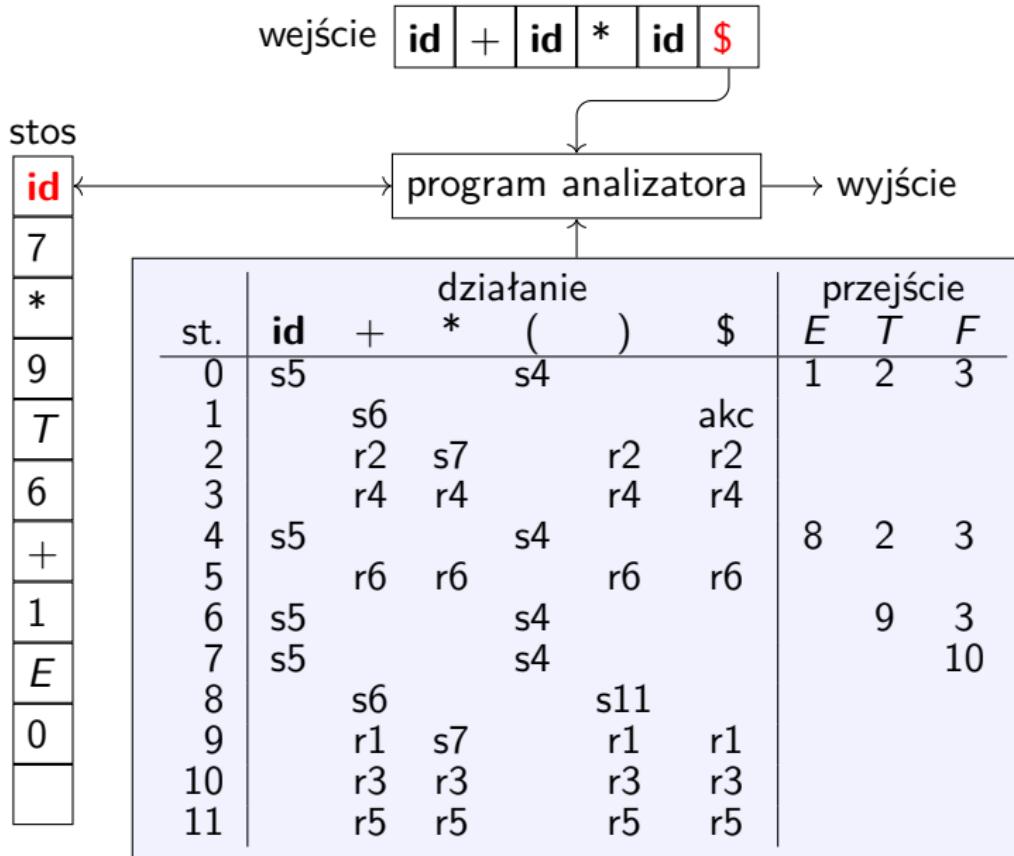
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



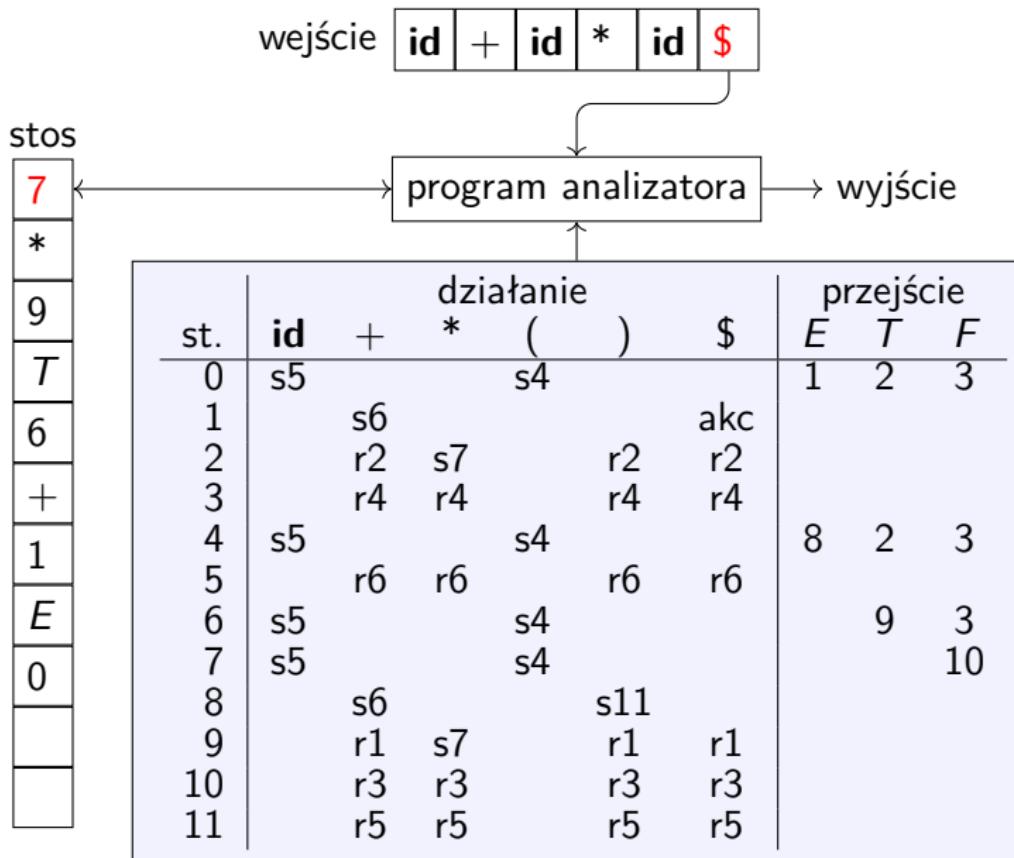
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



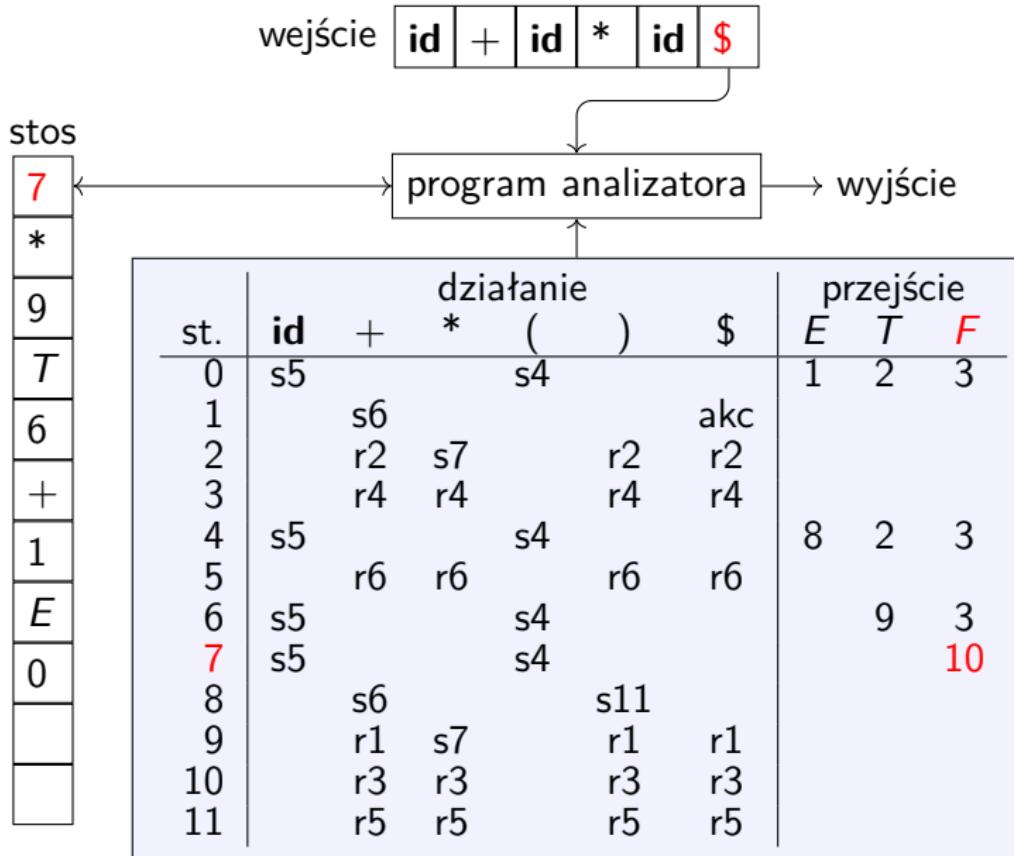
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



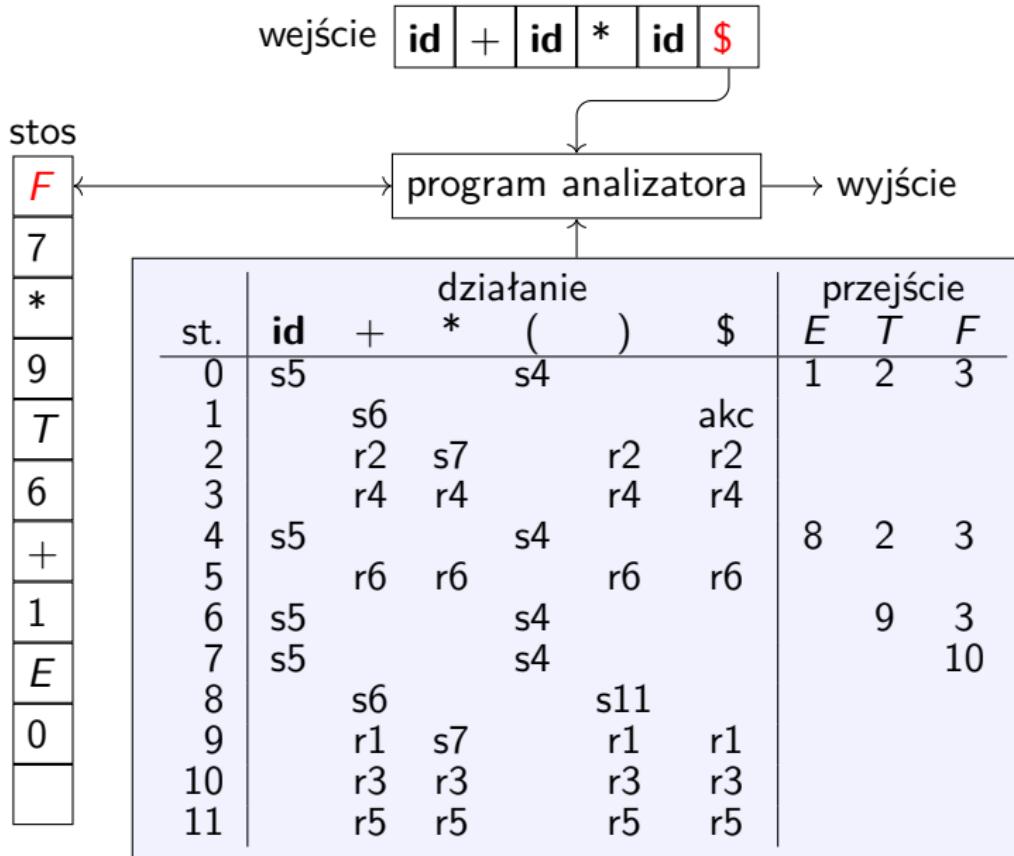
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$

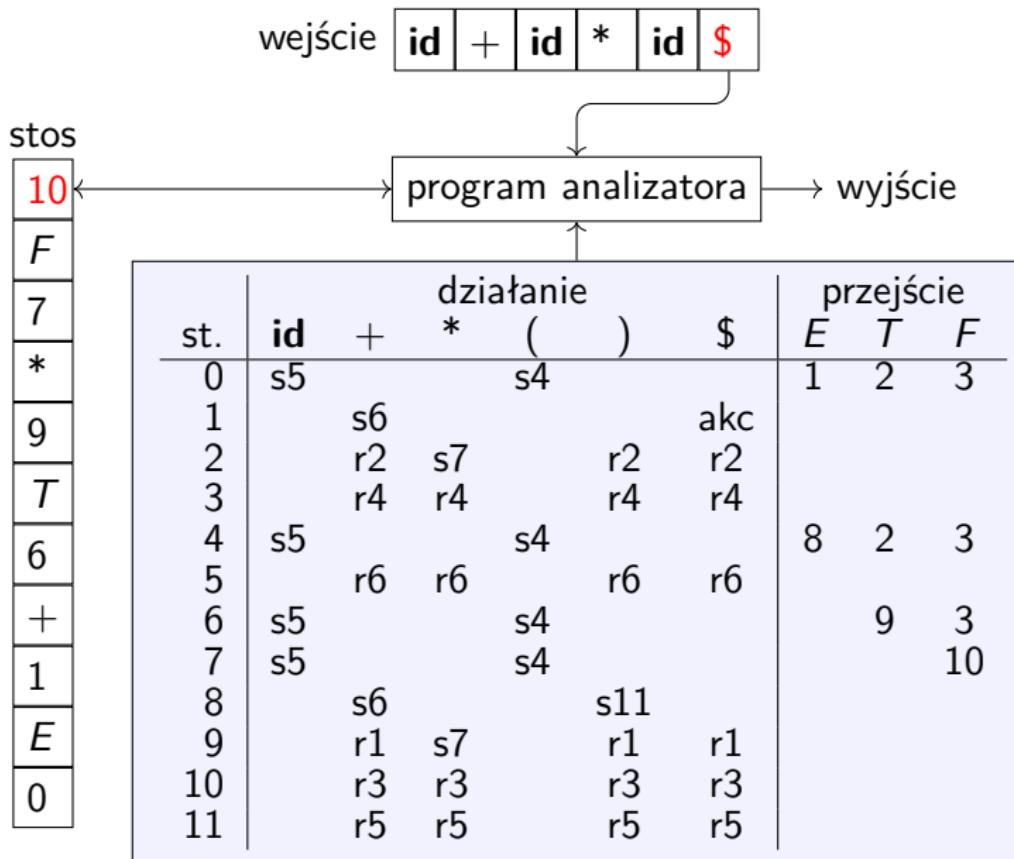


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

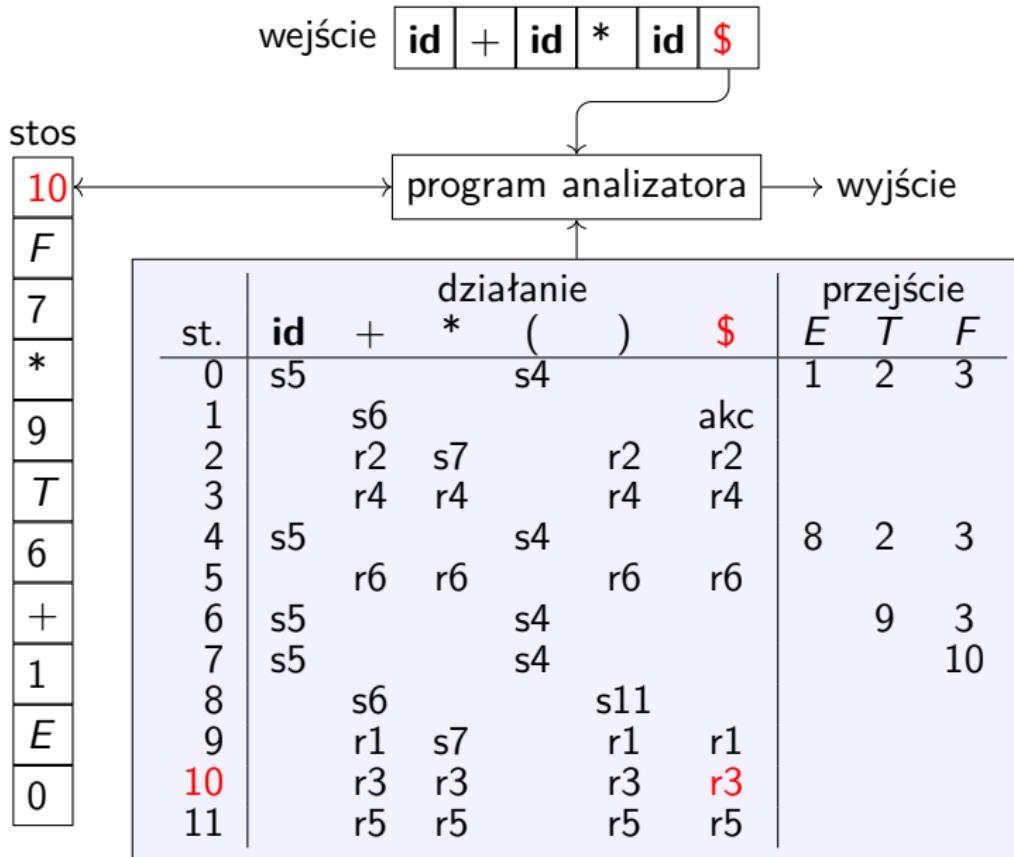


1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$

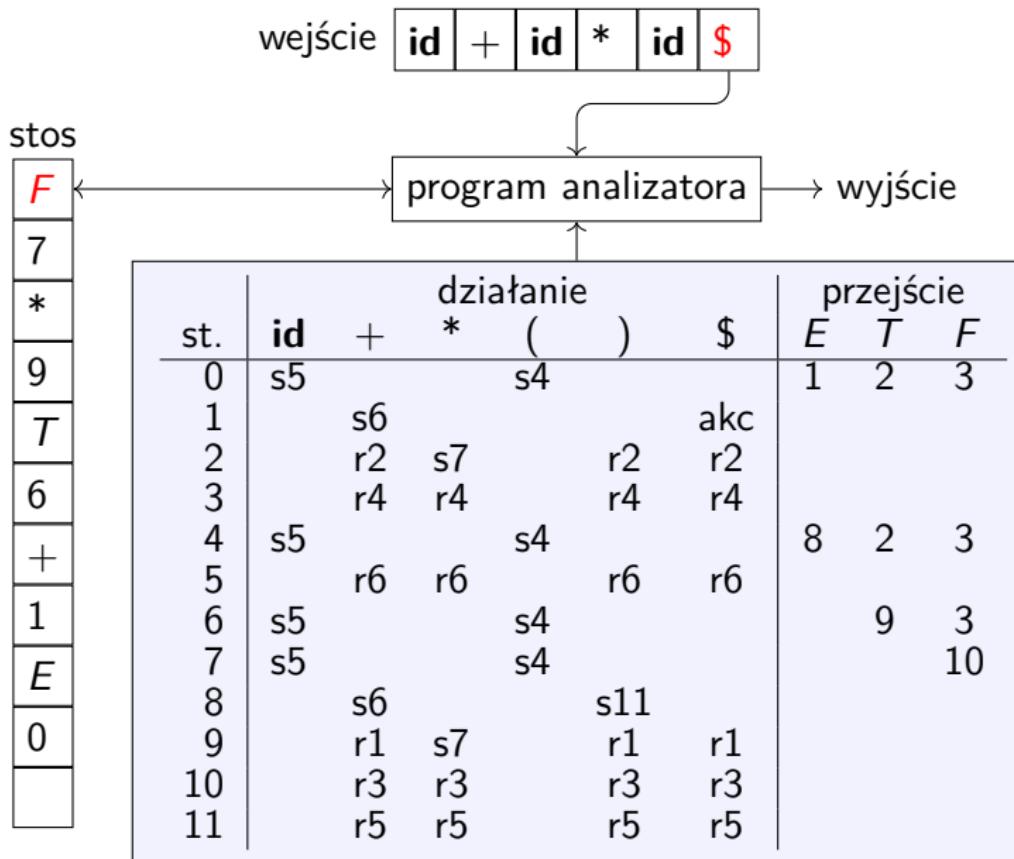




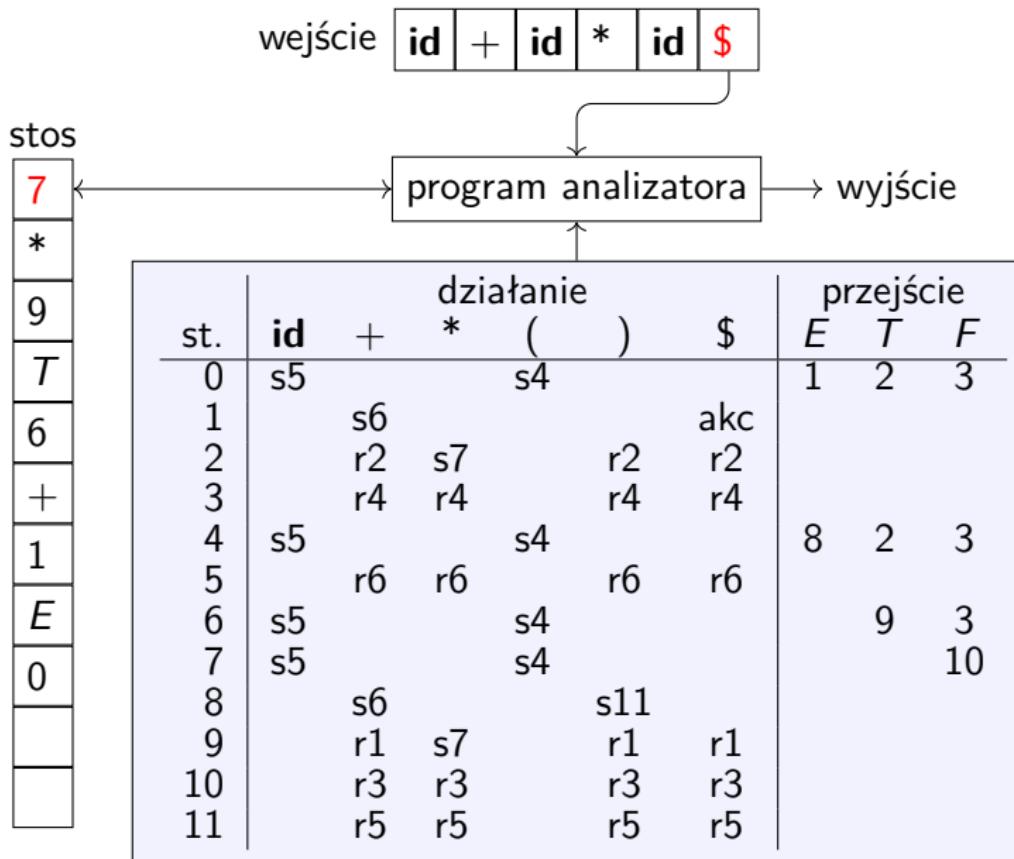
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



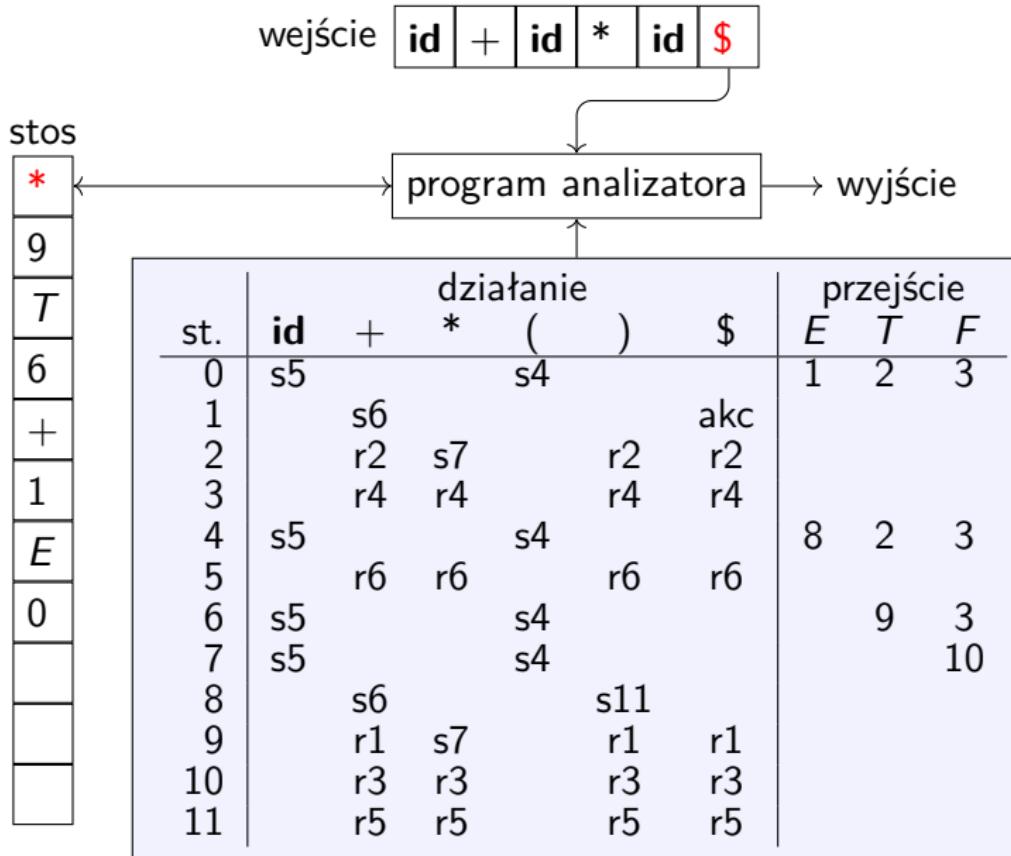
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



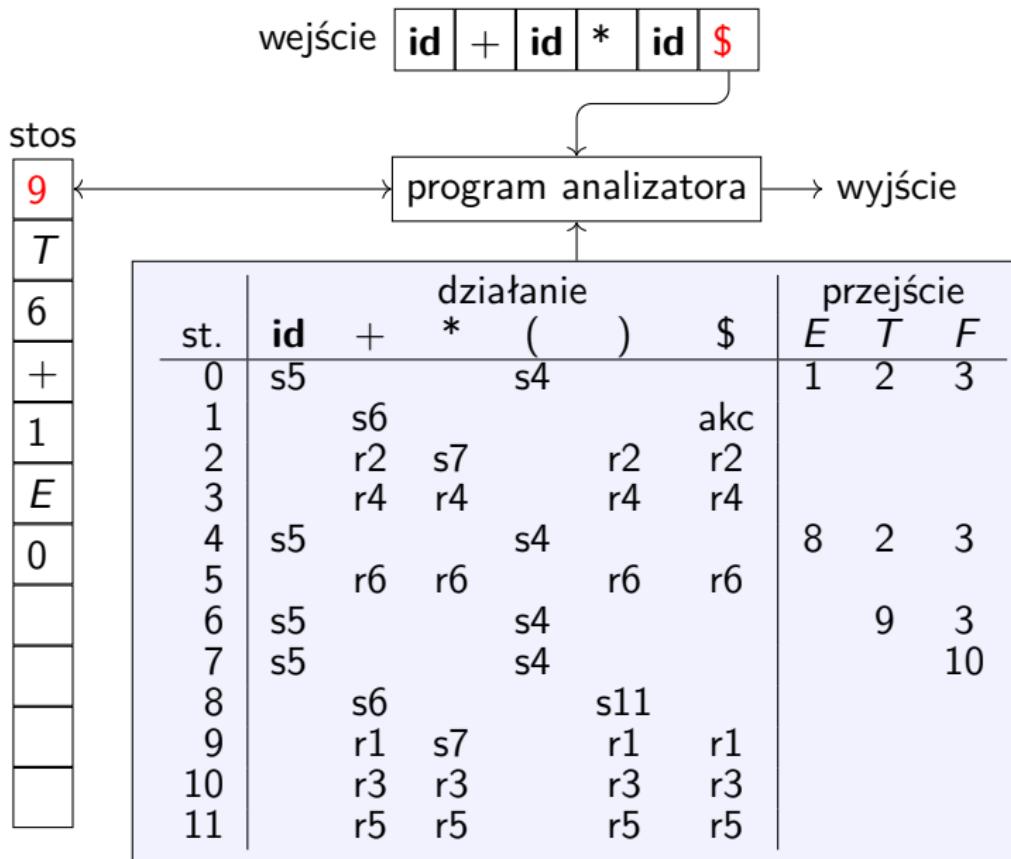
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



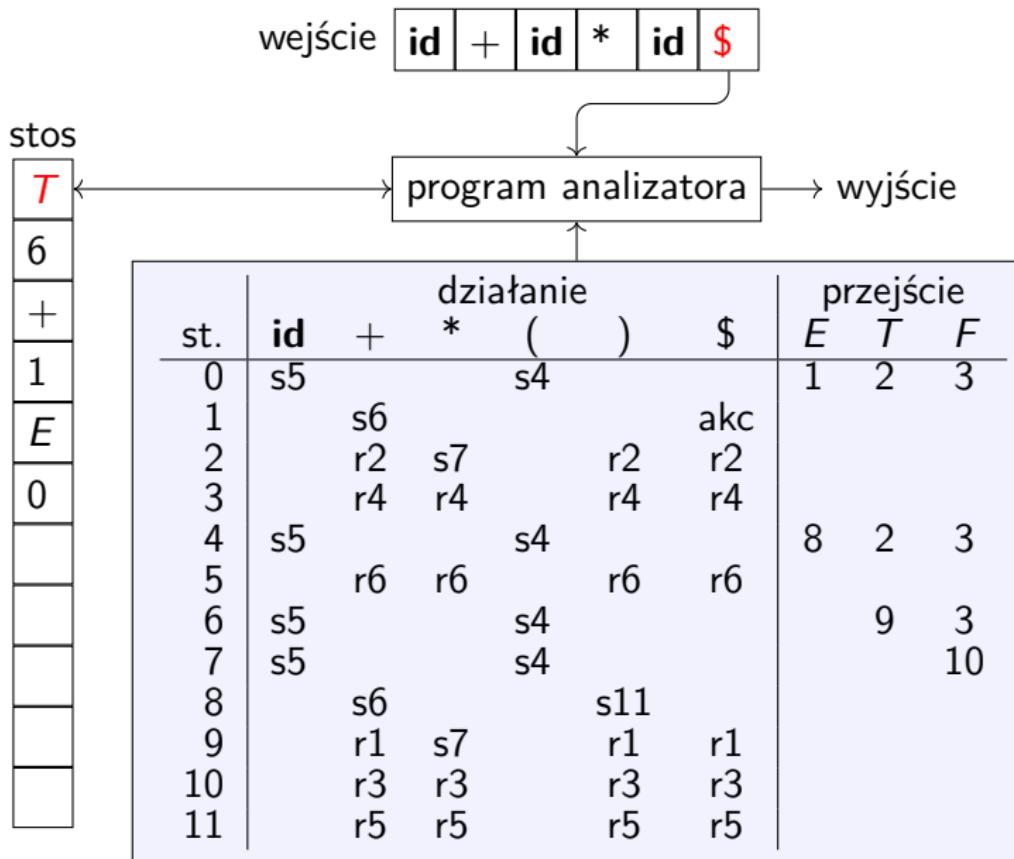
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



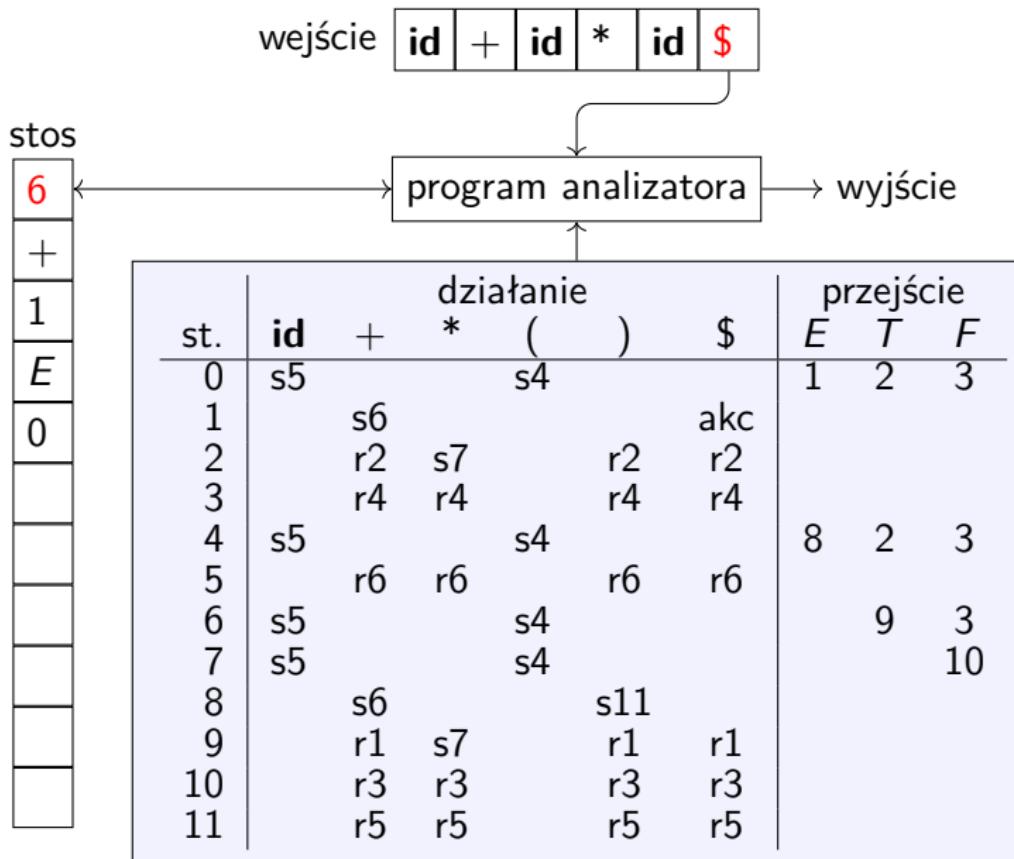
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



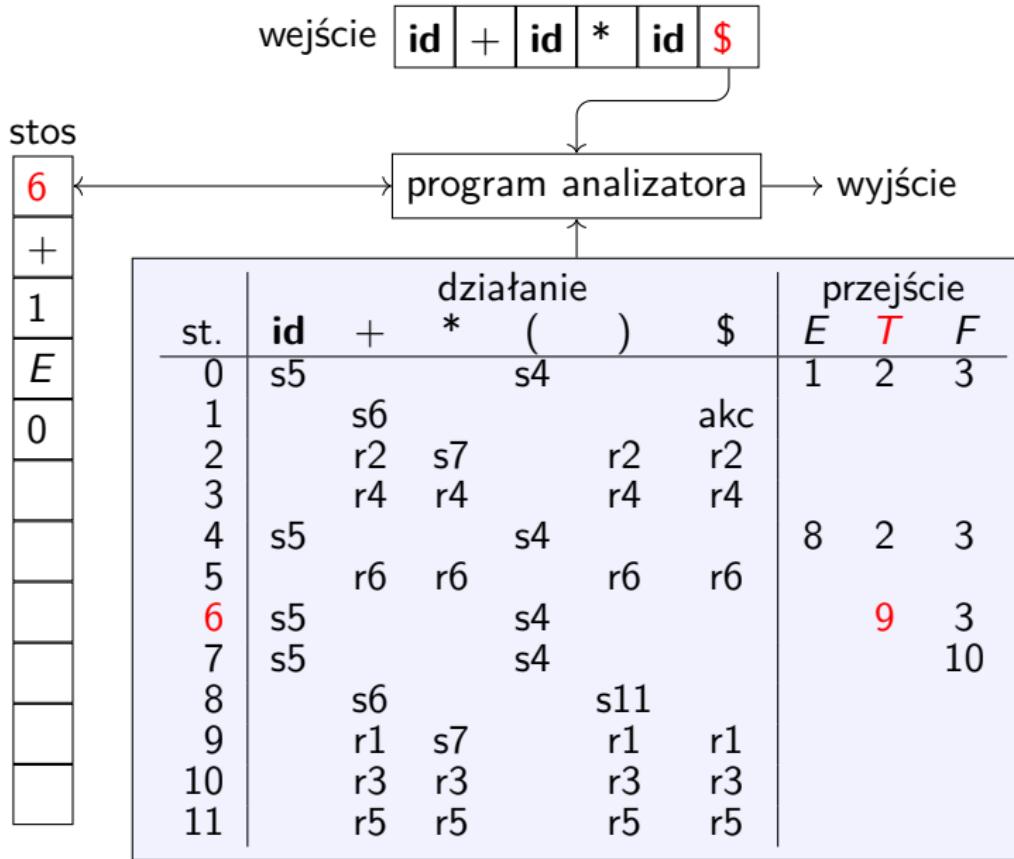
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



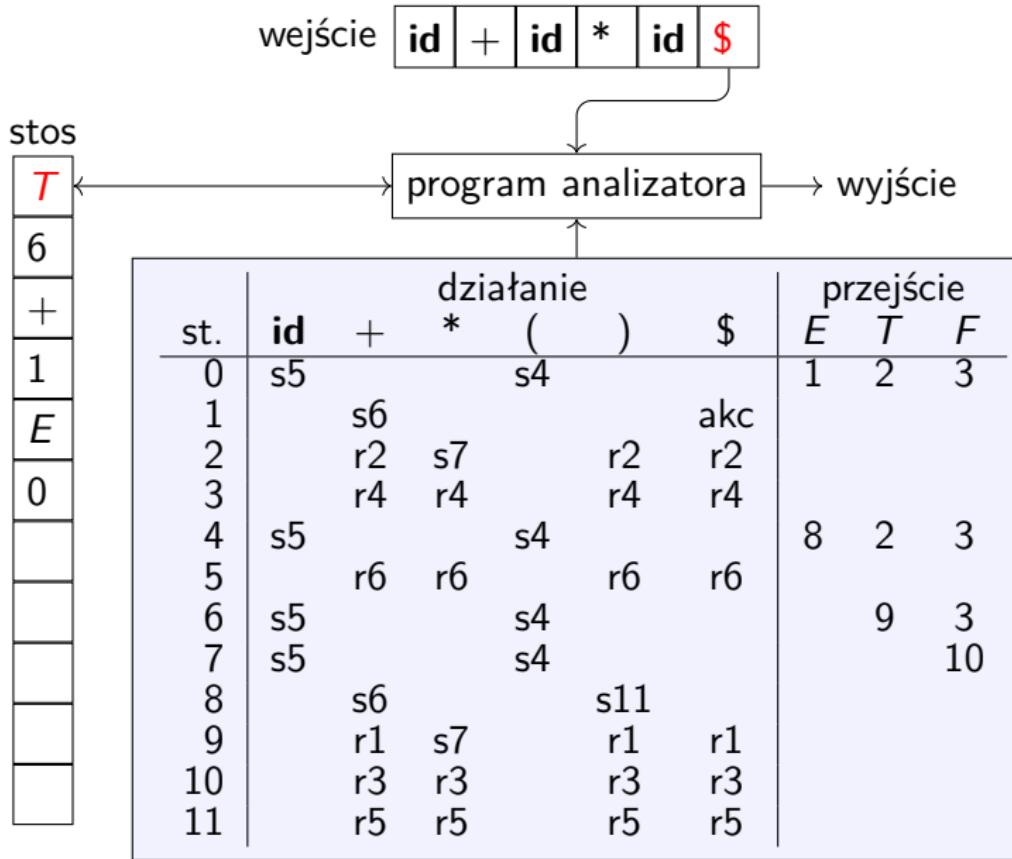
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



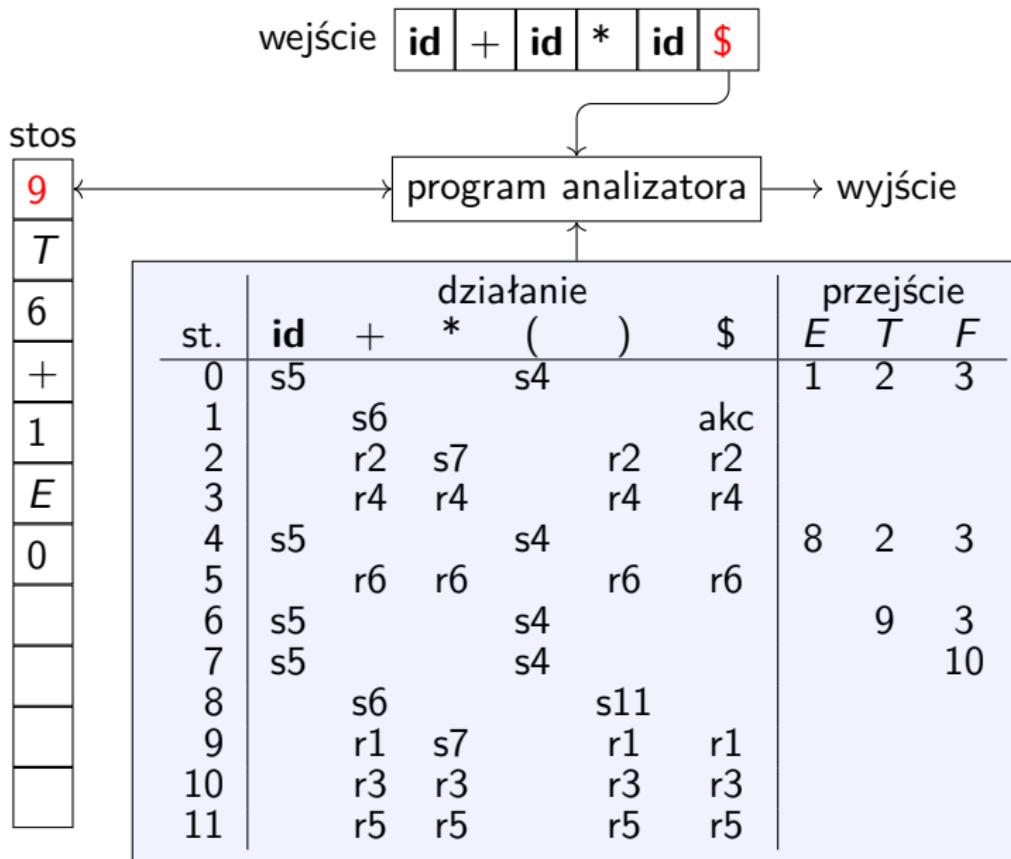
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



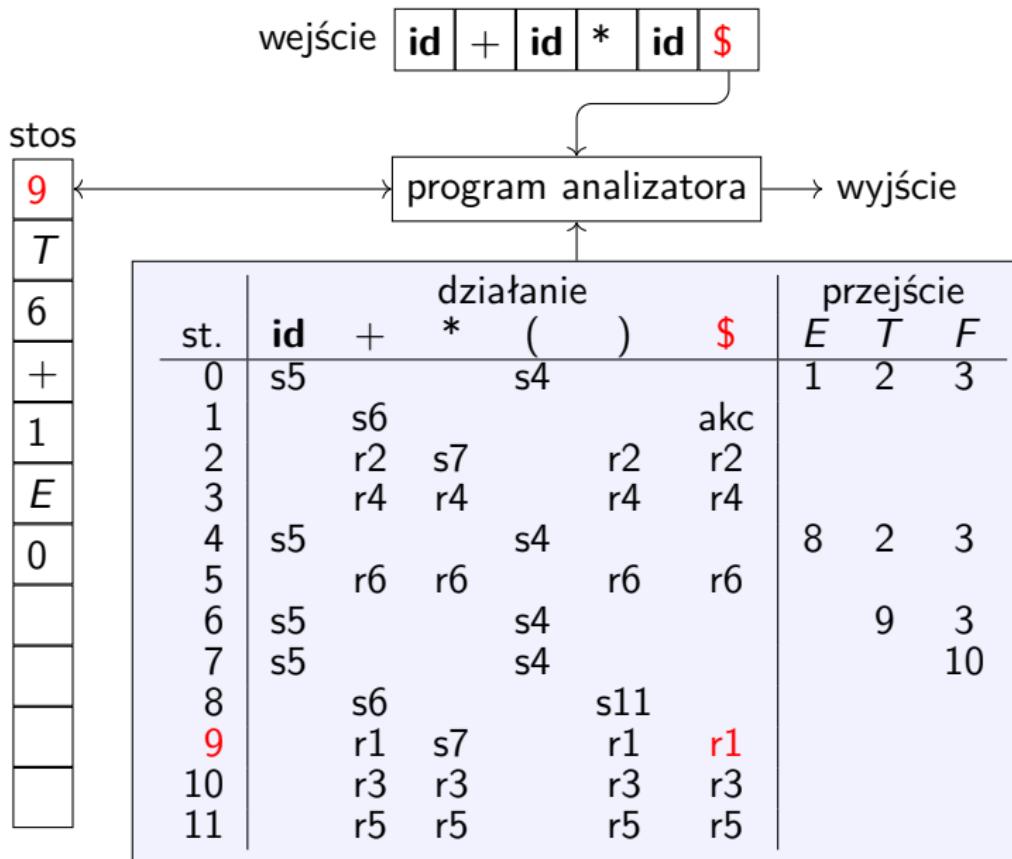
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
- 3.  $T \rightarrow T * F$**
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



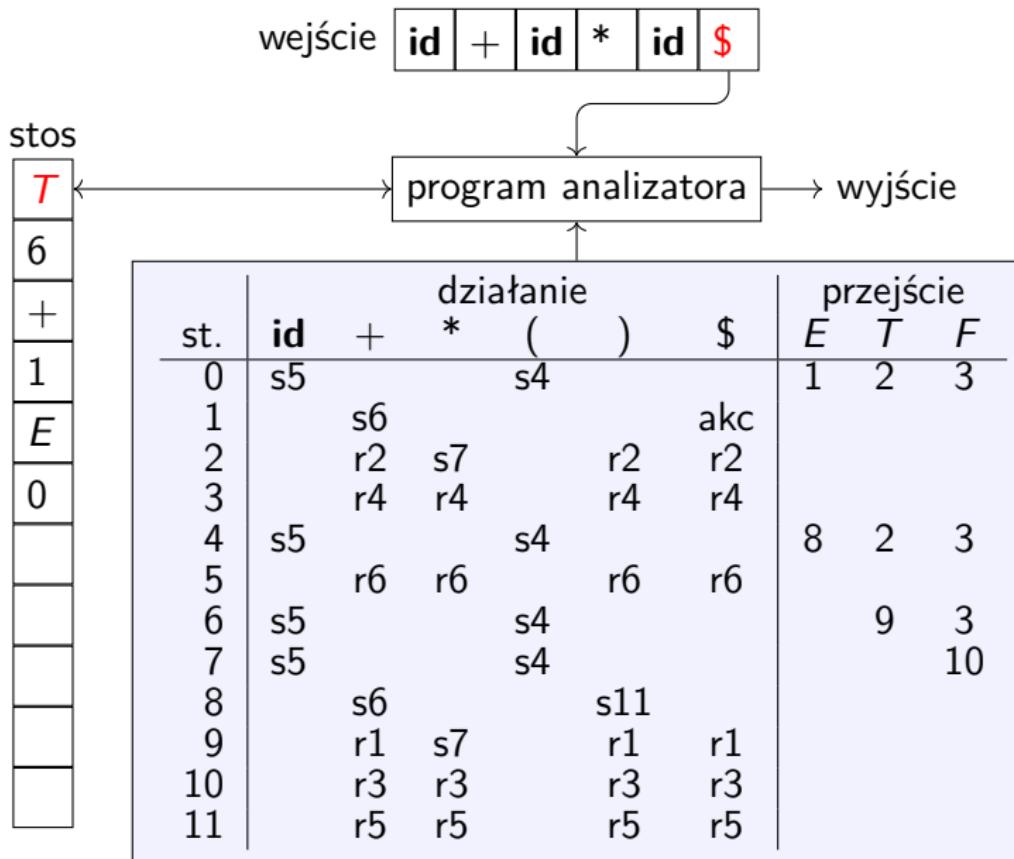
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



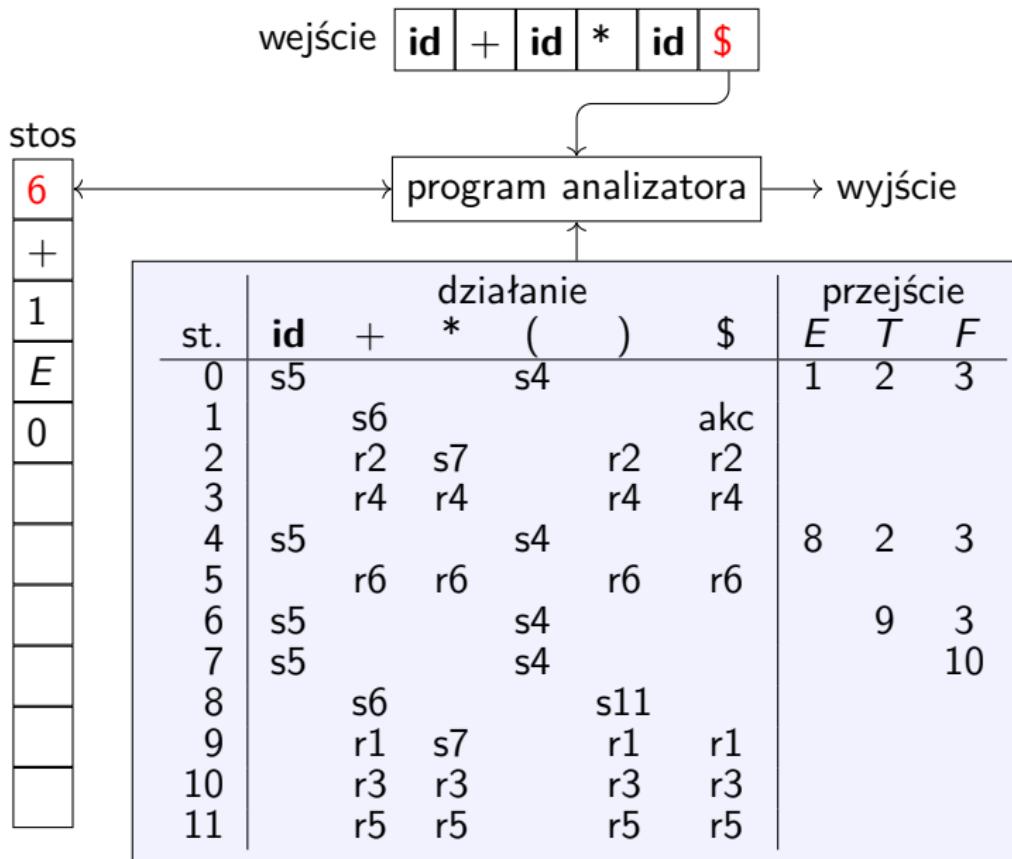
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



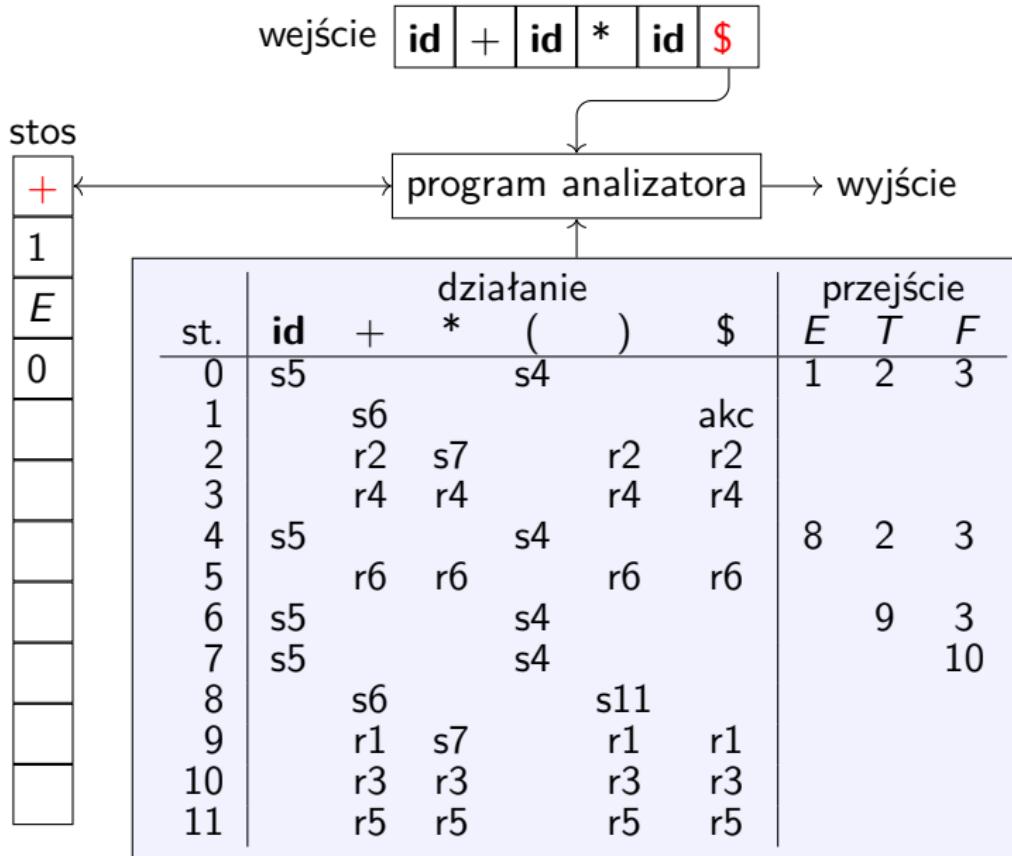
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



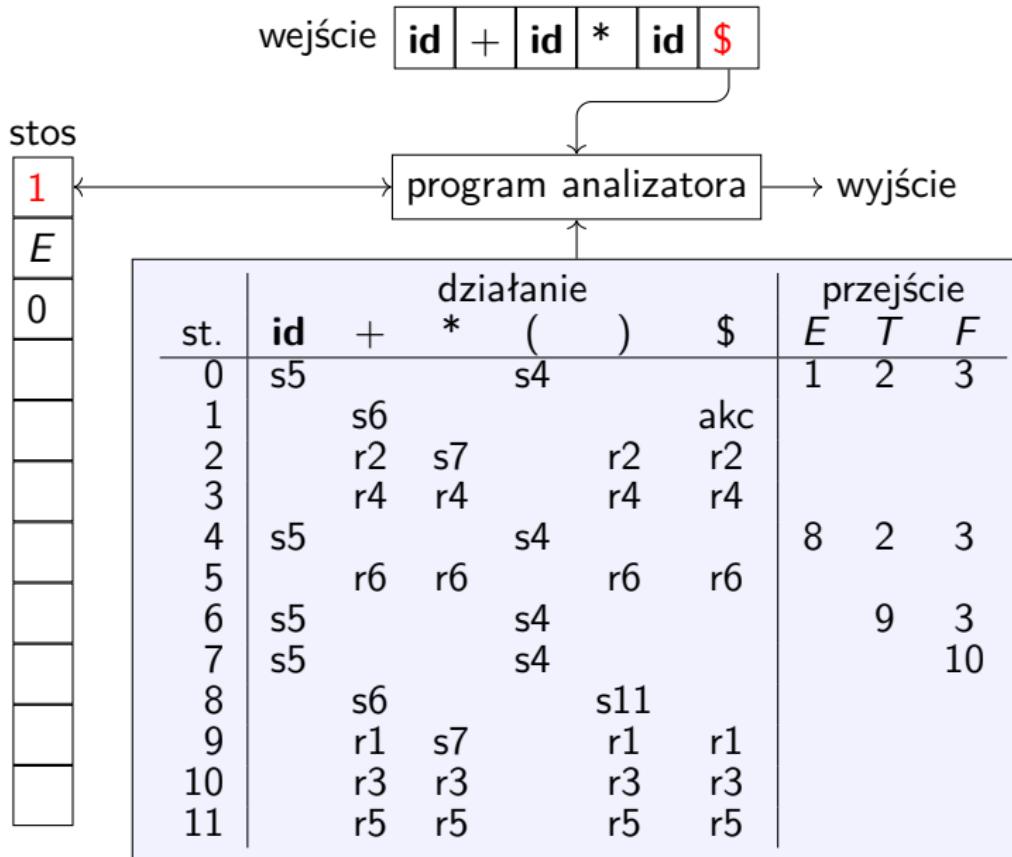
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



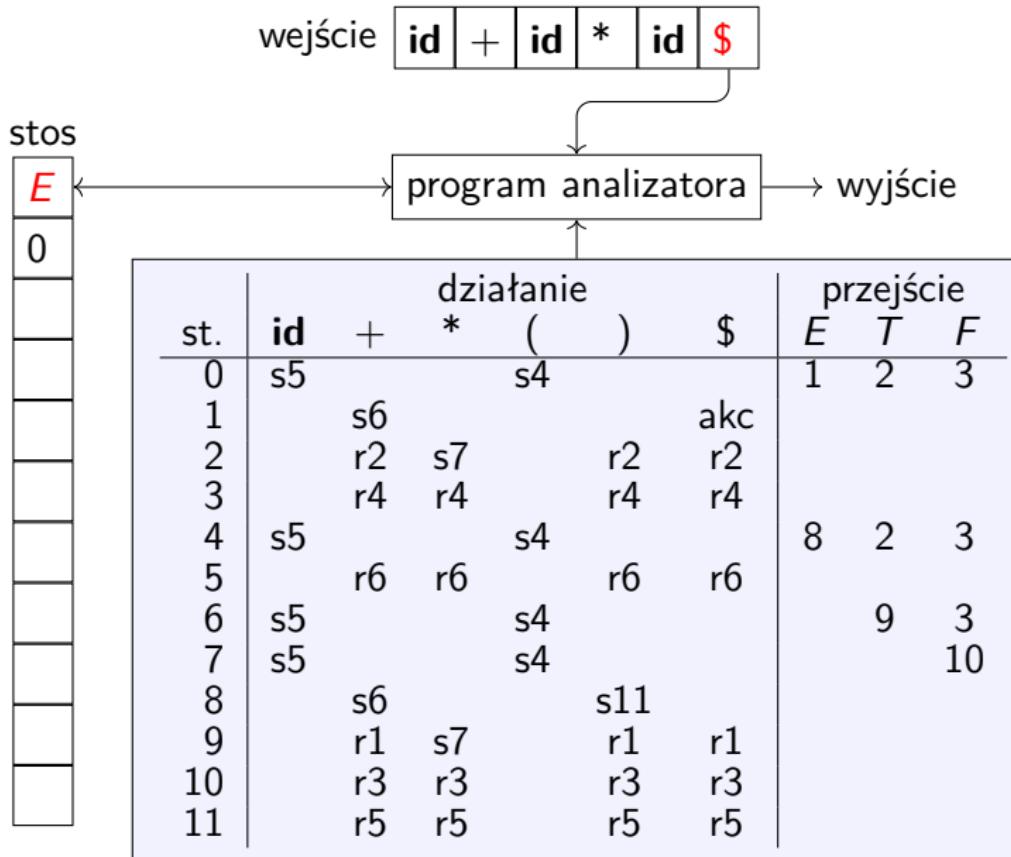
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



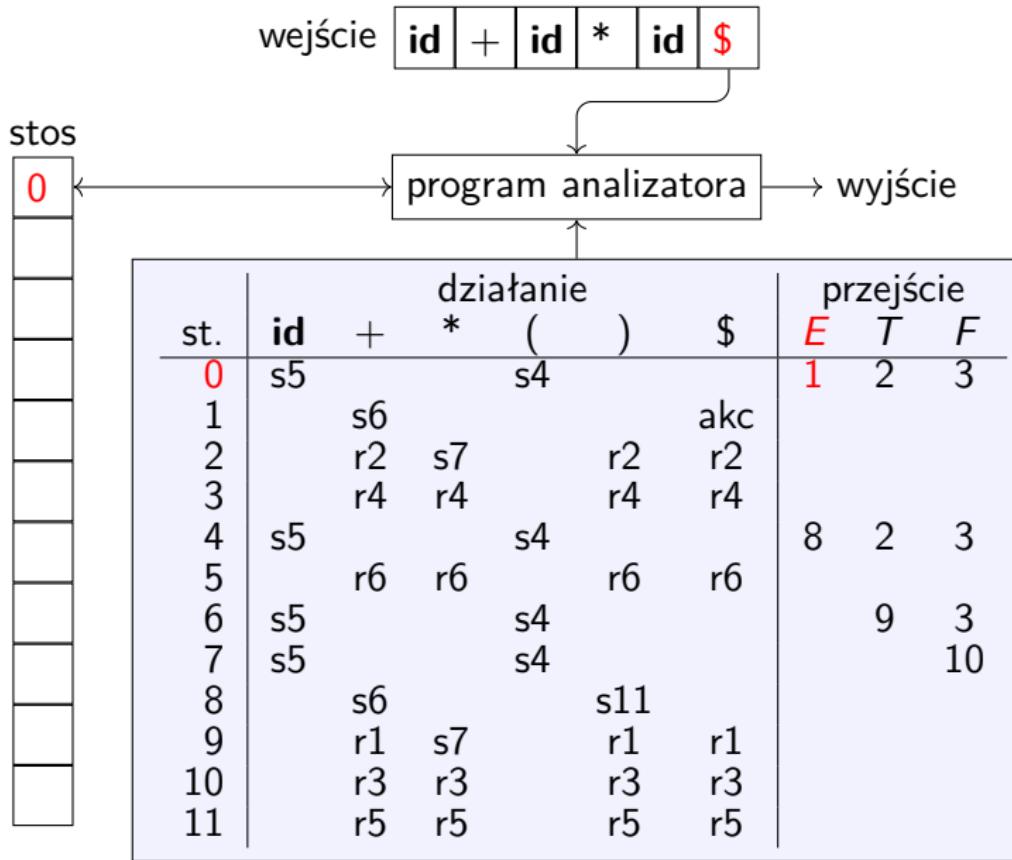
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



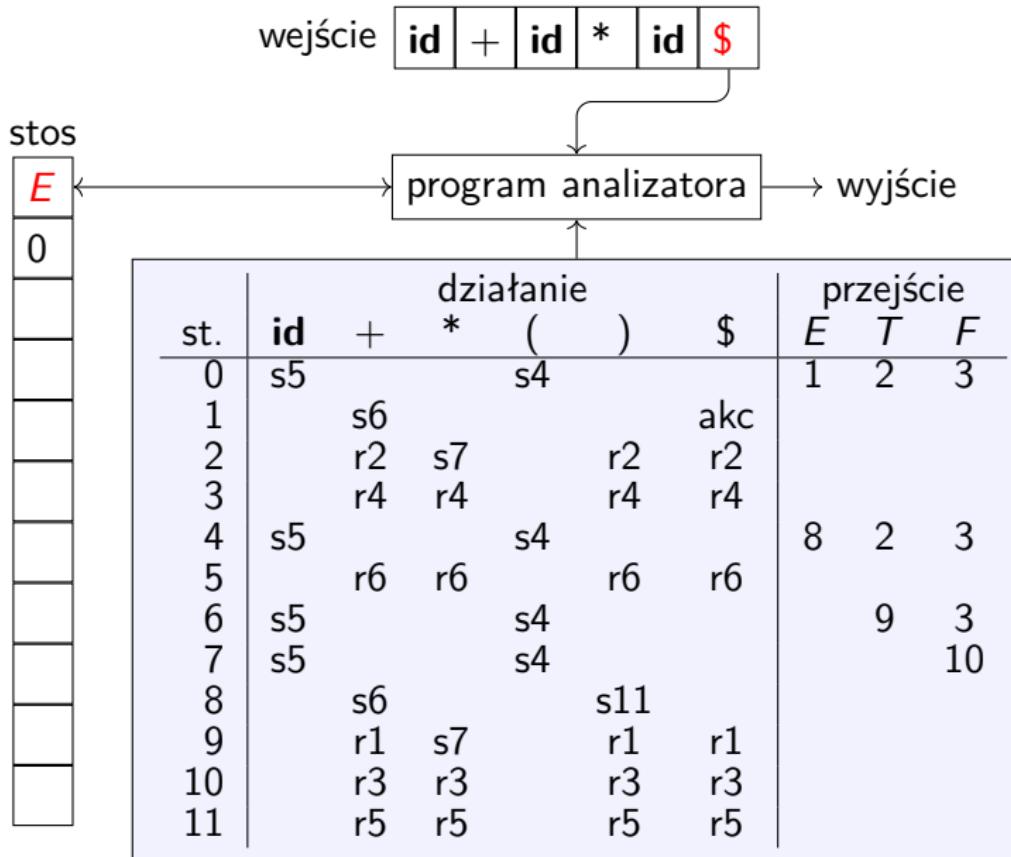
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



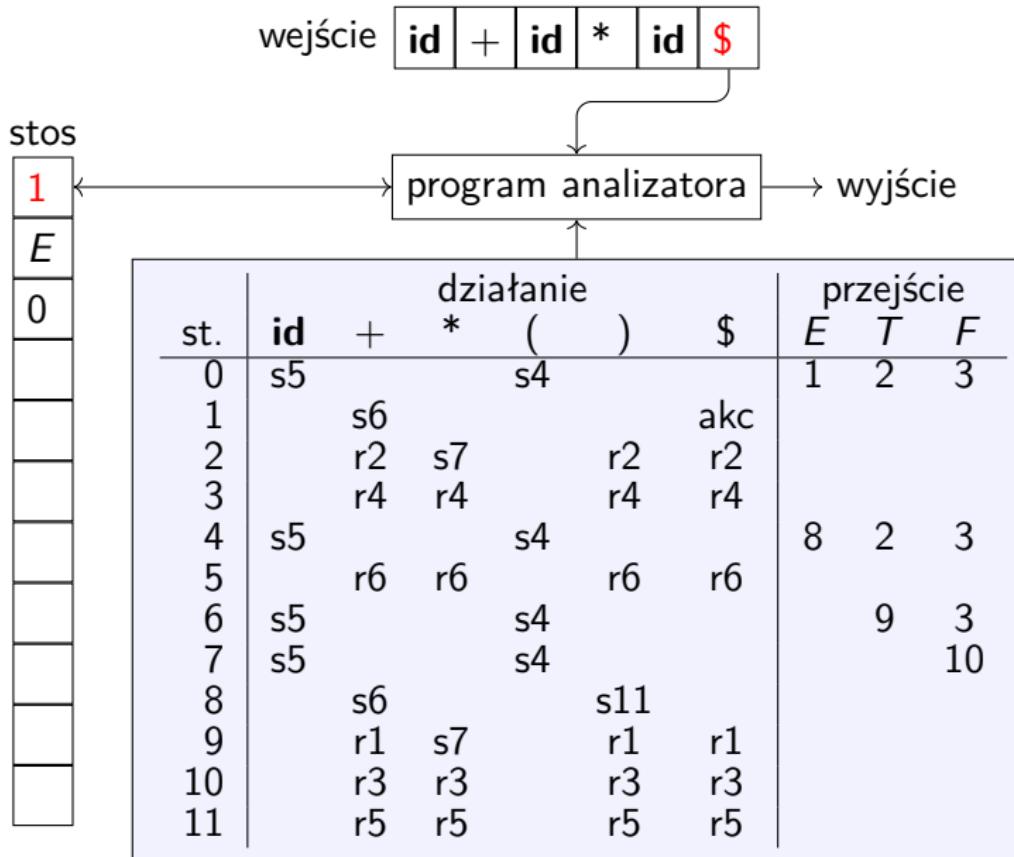
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



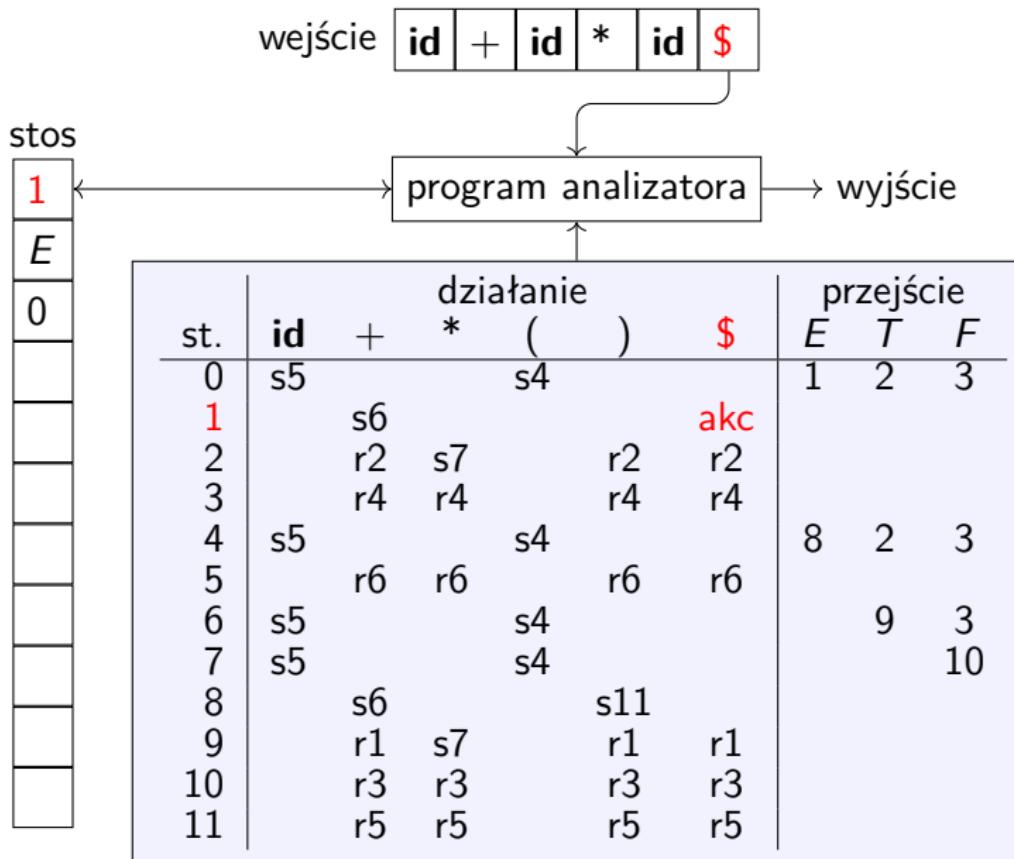
1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow id$



1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$



Istnieją różne rodzaje analizatorów  $LR(1)$ . Różnią się sposobem wypełniania tablicy i jej rozmiarem, a także złożonością wypełniania i klasą rozpoznawanych języków. Rozpatrzymy:

- ① Simple LR (SLR)
- ② Kanoniczna LR
- ③ Look-Ahead LR (LALR)

Mówimy też o gramatykach i językach SLR i LALR.



## Tablica analizatora SLR

Tablica analizatora SLR powstaje jako niedeterministyczny automat skończony z przejściami etykietowanymi symbolem  $\varepsilon$  (pustym ciągiem symboli). Stanami są sytuacje gramatyki.

**Sytuacją**  $LR(0)$  gramatyki  $G$  nazywamy regułę produkcji z kropką umieszczoną w dowolnym miejscu z prawej strony. Np. dla produkcji  $A \rightarrow XY$  mamy:

- ①  $A \rightarrow \bullet XY$
- ②  $A \rightarrow X \bullet Y$
- ③  $A \rightarrow XY \bullet$

Dla produkcji  $A \rightarrow \varepsilon$  mamy jedną sytuację:  $A \rightarrow \bullet$ .



## Tworzenie automatu dla tablicy SLR

- 1: Z gramatyki  $G = (\Sigma, V, P, S)$  utwórz gramatykę  $G' = (\Sigma, V \cup \{S'\}, P \cup \{S' \rightarrow S\}, S')$
- 2: Utwórz stany automatu z sytuacji  $G'$
- 3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$
- 4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $B \in V$ , połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\varepsilon$
- 5: Determinizuj automat



## Algorytm wypełniania tablicy analizatora SLR

- 1: Utwórz automat deterministyczny ze stanami  $I_0, I_1, \dots, I_n$
- 2: **if**  $A \rightarrow \alpha \bullet a\beta$  dla  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**
- 3:     działanie $[i, a] = s_j$
- 4: **end if**
- 5: **if**  $A \rightarrow \alpha \bullet$  jest w  $I_i$ ,  $A \neq S'$  **then**
- 6:     **for all**  $a \in nast(A)$  **do**
- 7:         działanie $[i, a] = r_m$ , gdzie  $m$  jest numerem produkcji  $A \rightarrow \alpha$
- 8:     **end for**
- 9: **end if**
- 10: **if**  $S' \rightarrow S \bullet$  jest w  $I_i$  **then**
- 11:     działanie $[i, \$] = \text{akc}$
- 12: **end if**
- 13: **for all**  $A$  takie że  $\delta(I_i, A) = I_j$  **do**
- 14:     przejście $[i, A] = j$
- 15: **end for**



**Przykład.** Weźmy jeszcze raz gramatykę dla wyrażeń arytmetycznych:

1.  $E \rightarrow E + T$
2.  $E \rightarrow T$
3.  $T \rightarrow T * F$
4.  $T \rightarrow F$
5.  $F \rightarrow (E)$
6.  $F \rightarrow \text{id}$

uzupełniamy gramatykę o dodatkową regułę produkcji:

0.  $E' \rightarrow E$



# Wypełnianie tablicy analizatora SLR

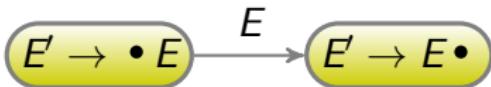


$$E' \rightarrow \bullet E$$

3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



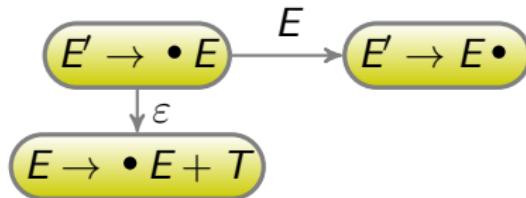
# Wypełnianie tablicy analizatora SLR



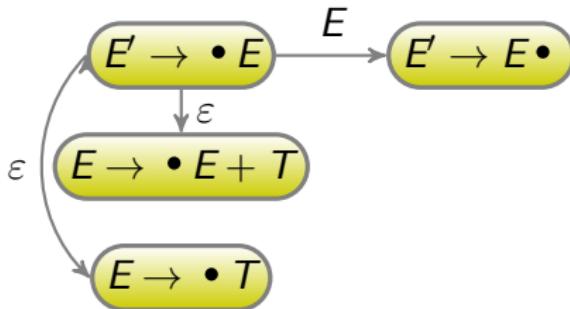
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



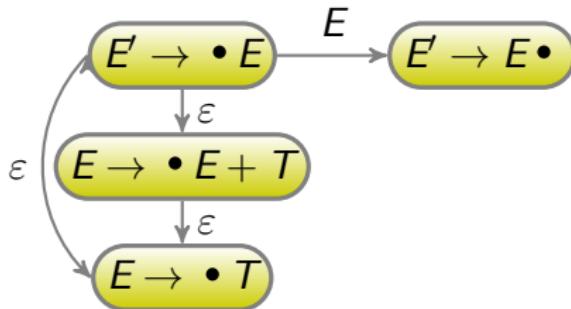
# Wypełnianie tablicy analizatora SLR



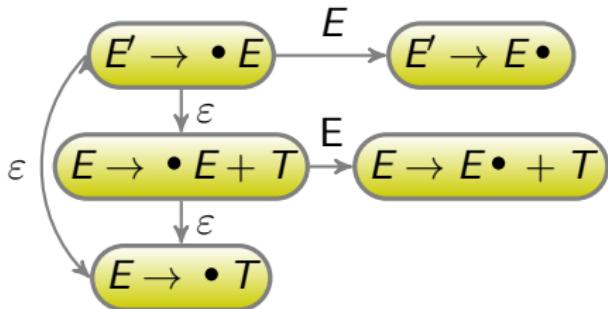
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\varepsilon$



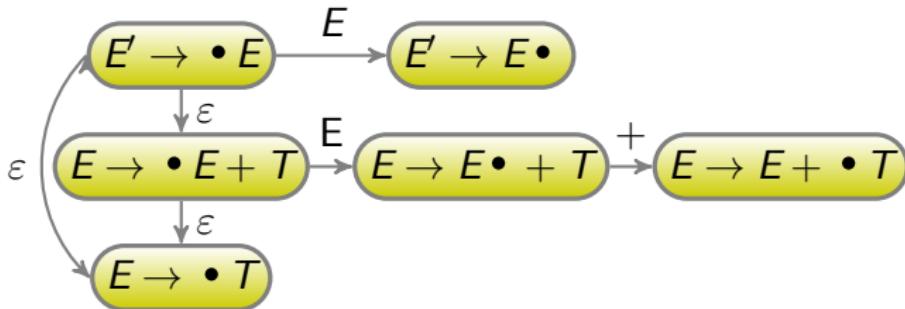
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



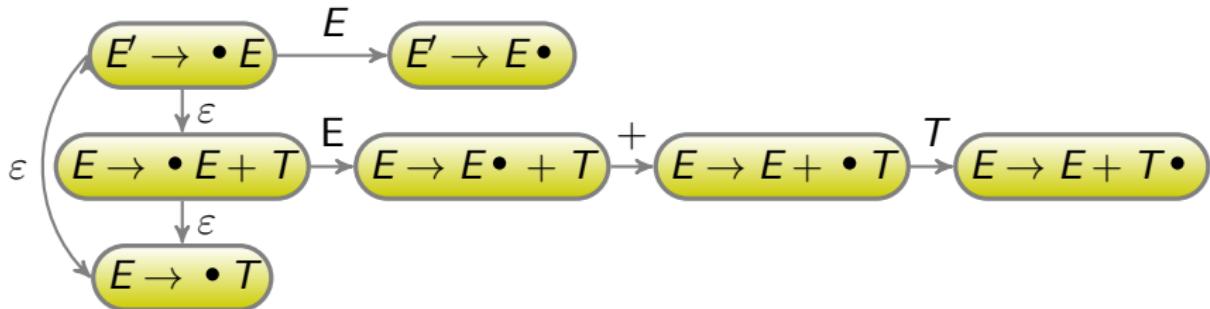
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



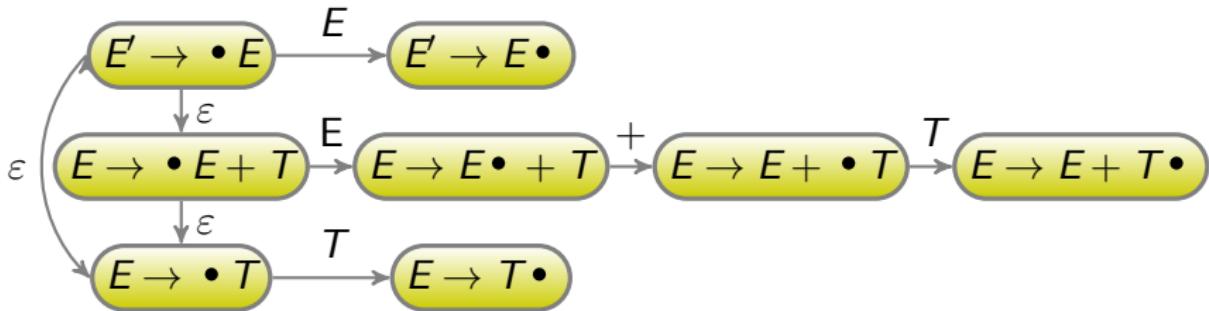
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



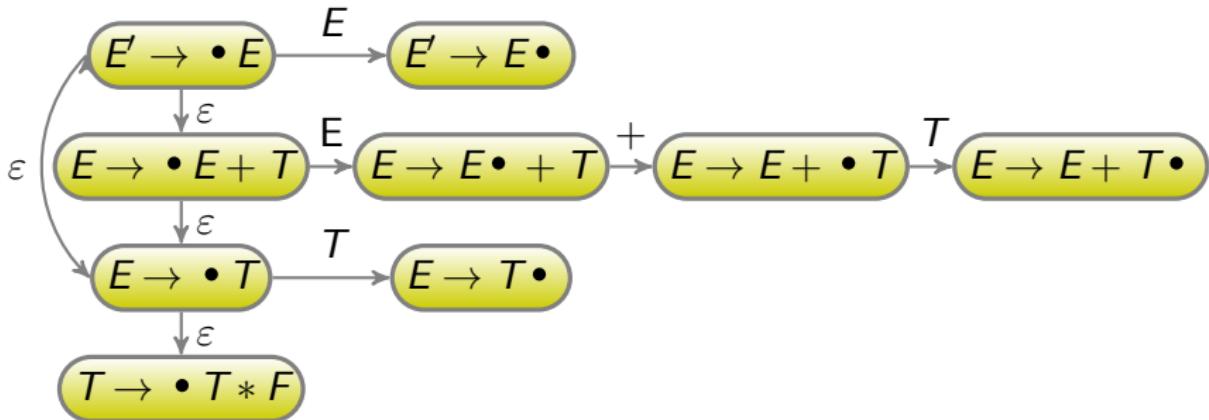
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



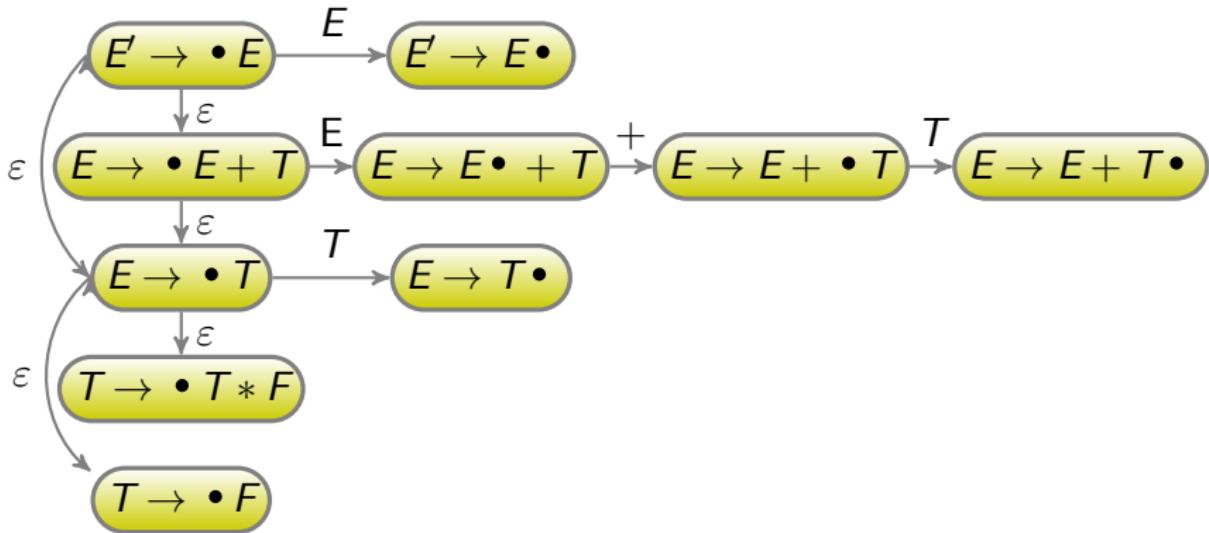
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



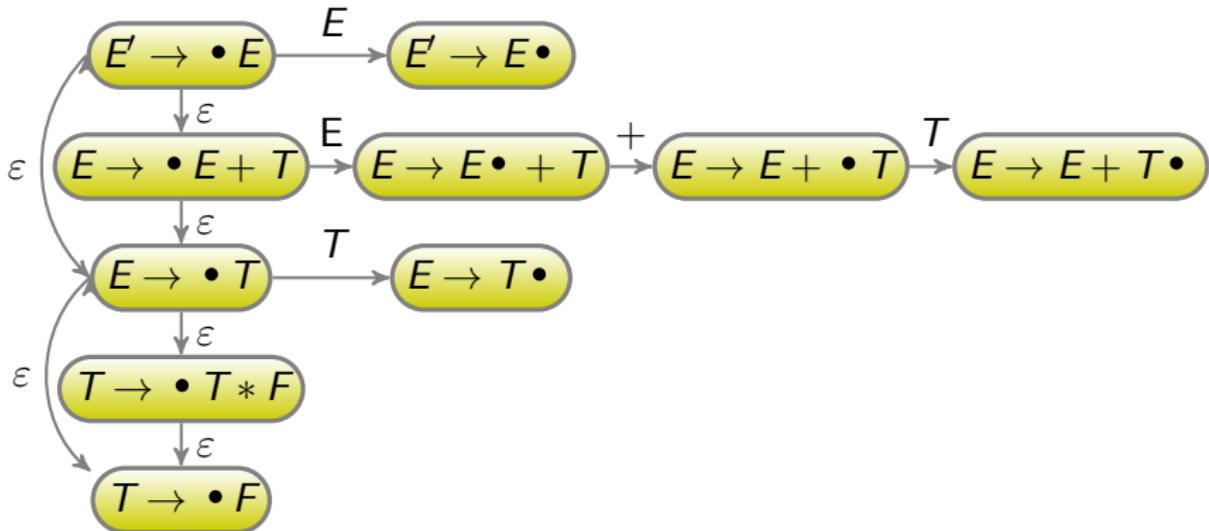
3: Każdą sytuację  $A \rightarrow \alpha • B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B • \beta$  przejściem etykietowanym  $B$



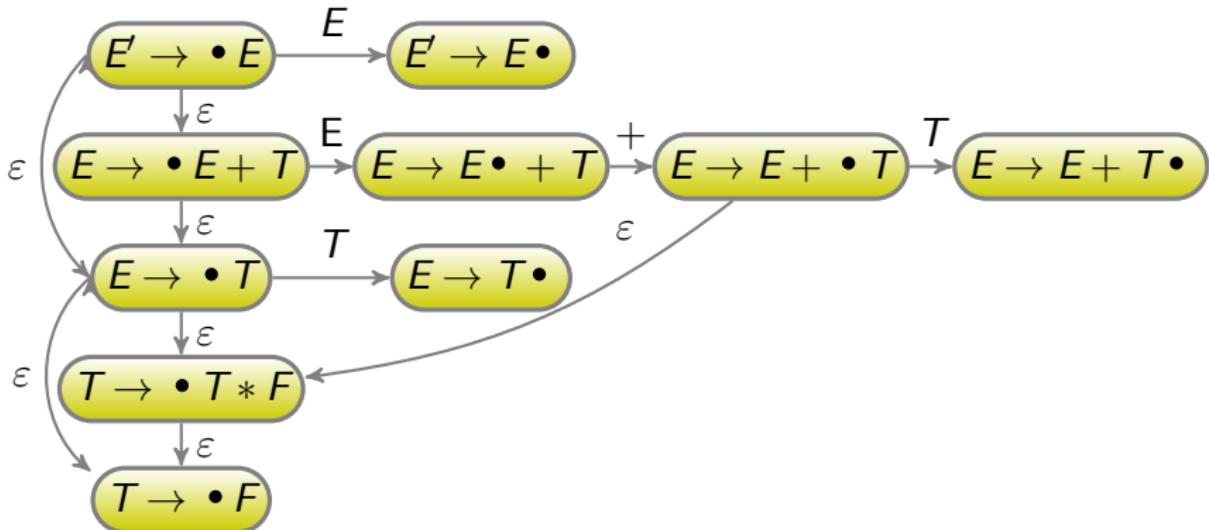
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



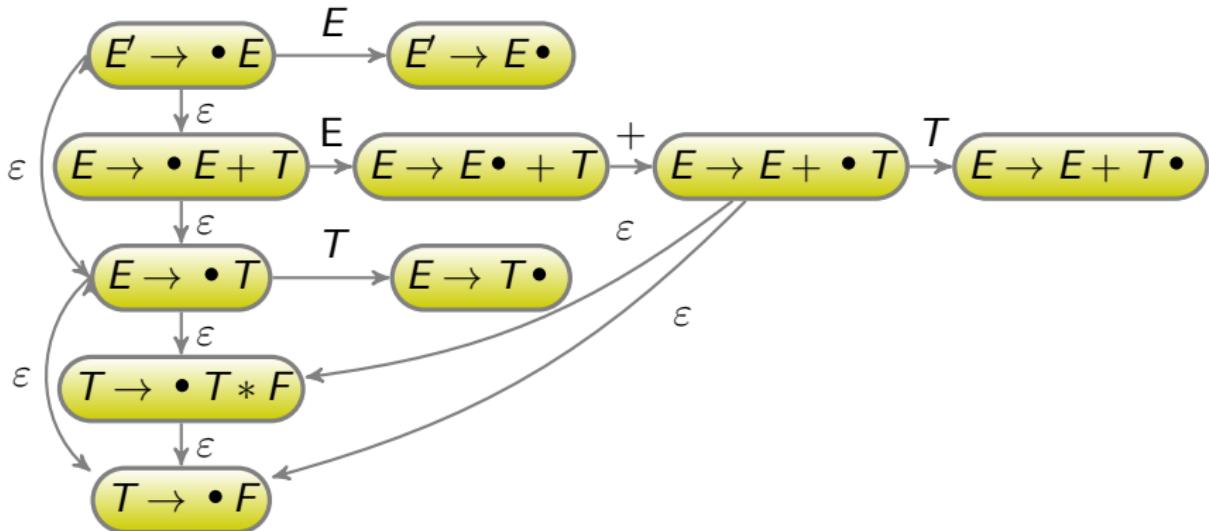
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\varepsilon$



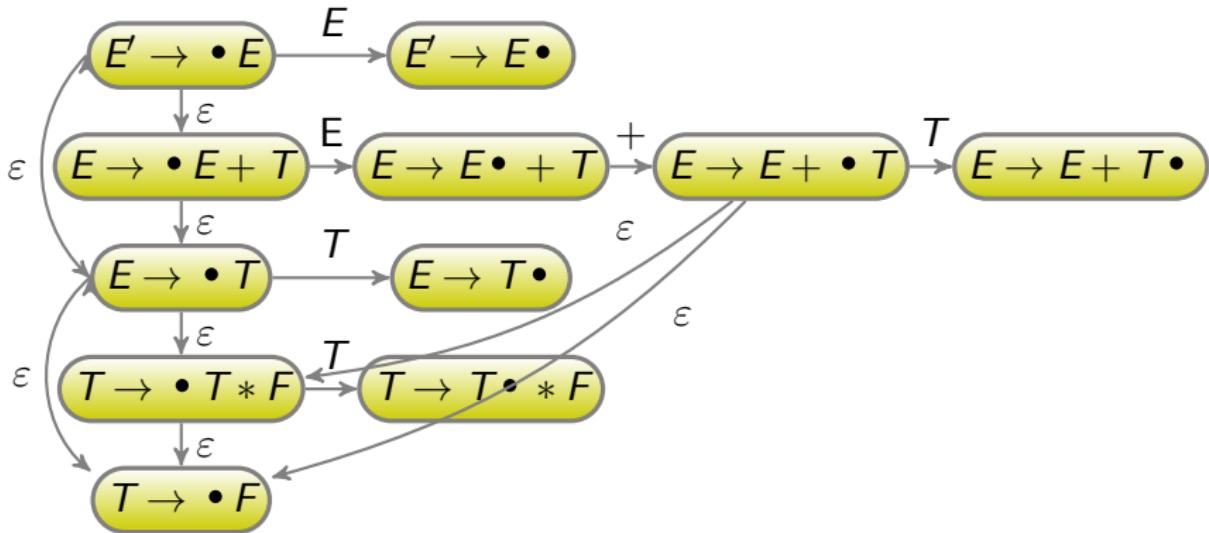
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



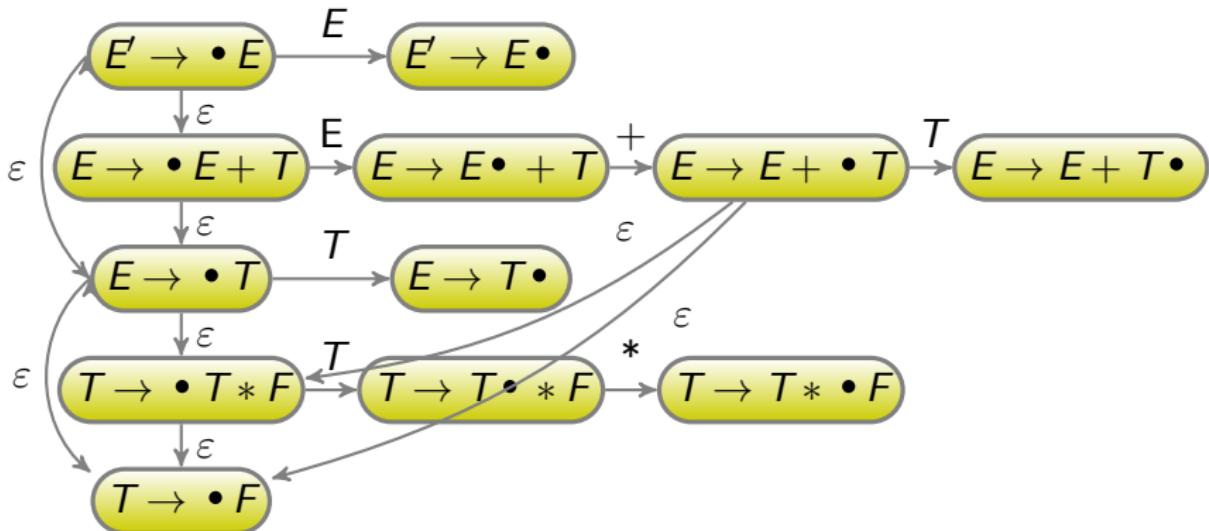
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



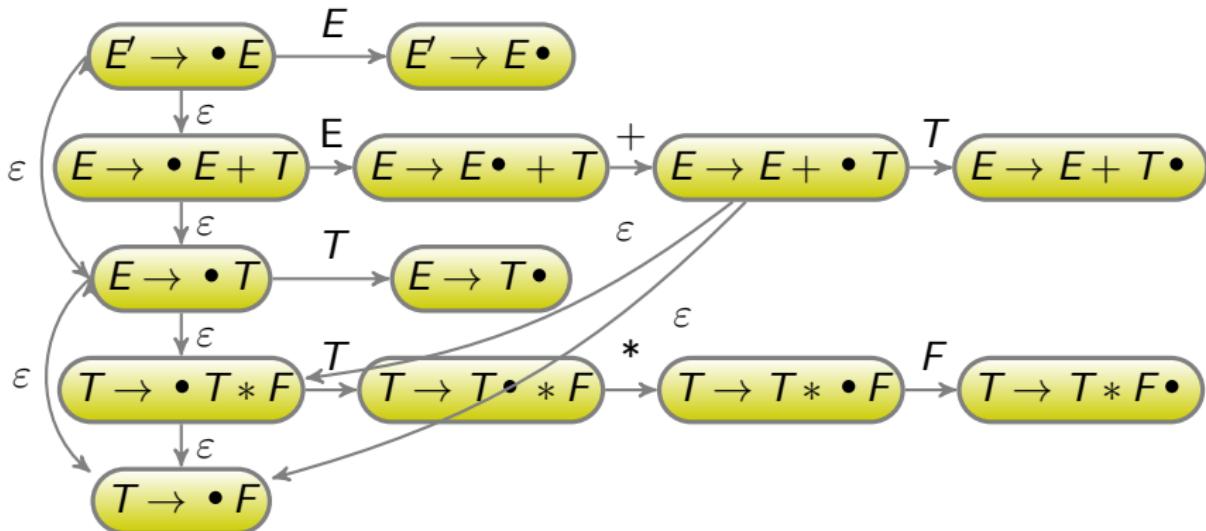
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



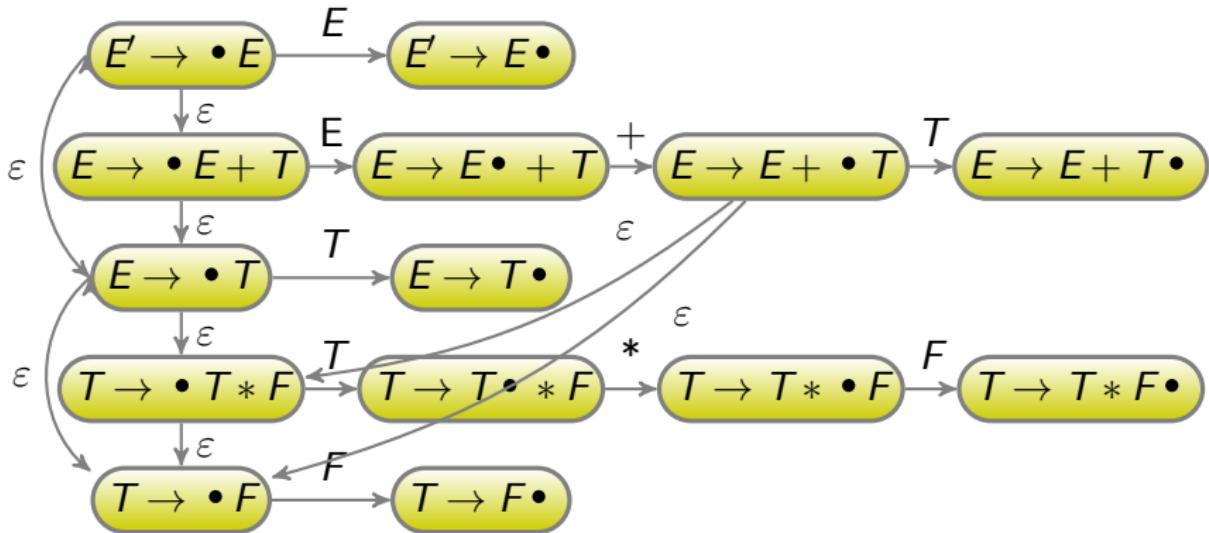
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



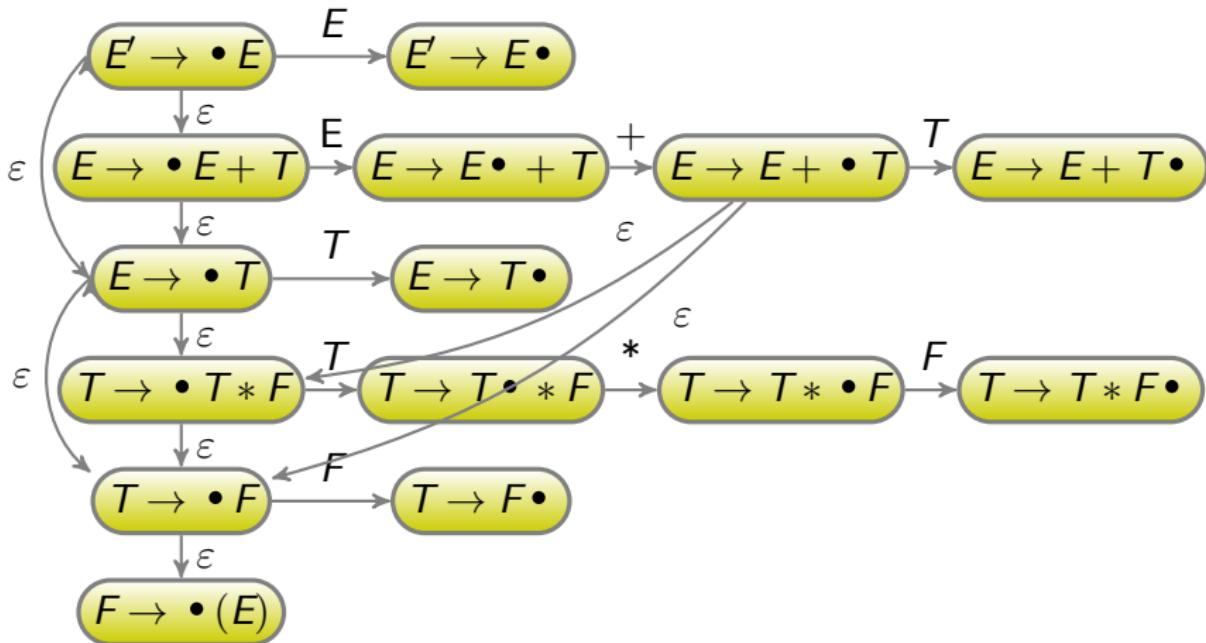
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



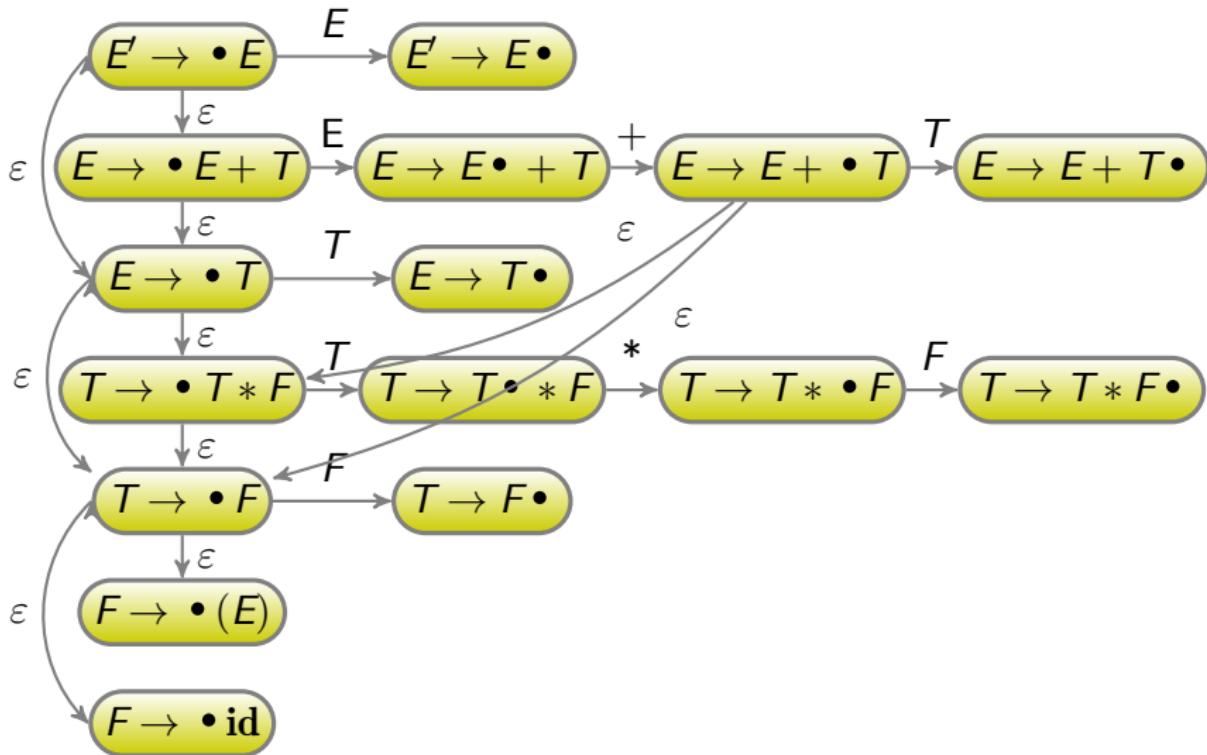
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



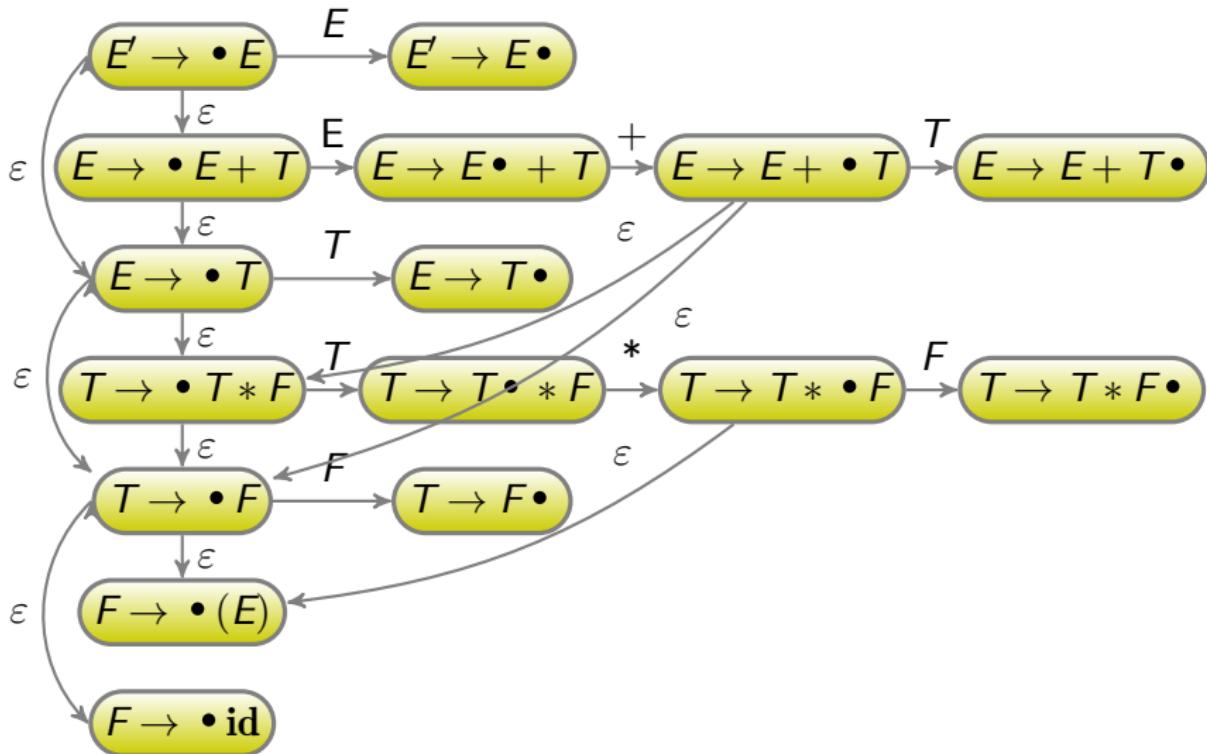
## Wypełnianie tablicy analizatora SLR



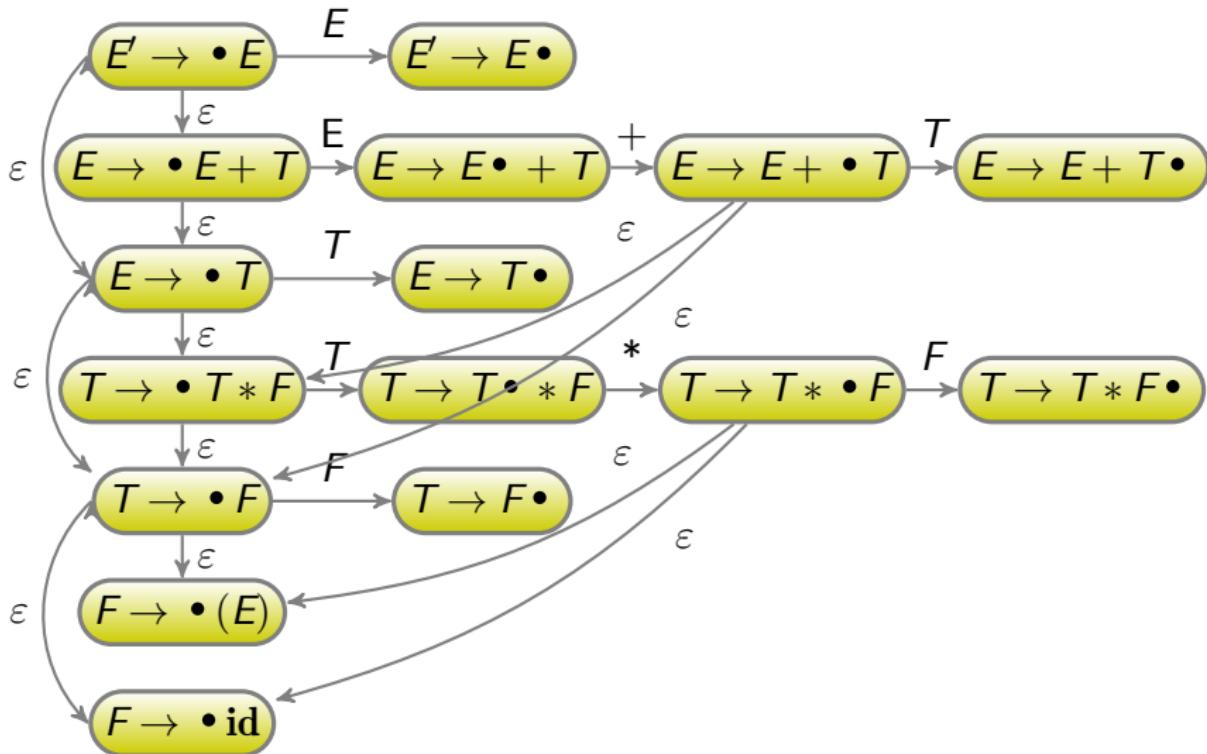
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\varepsilon$



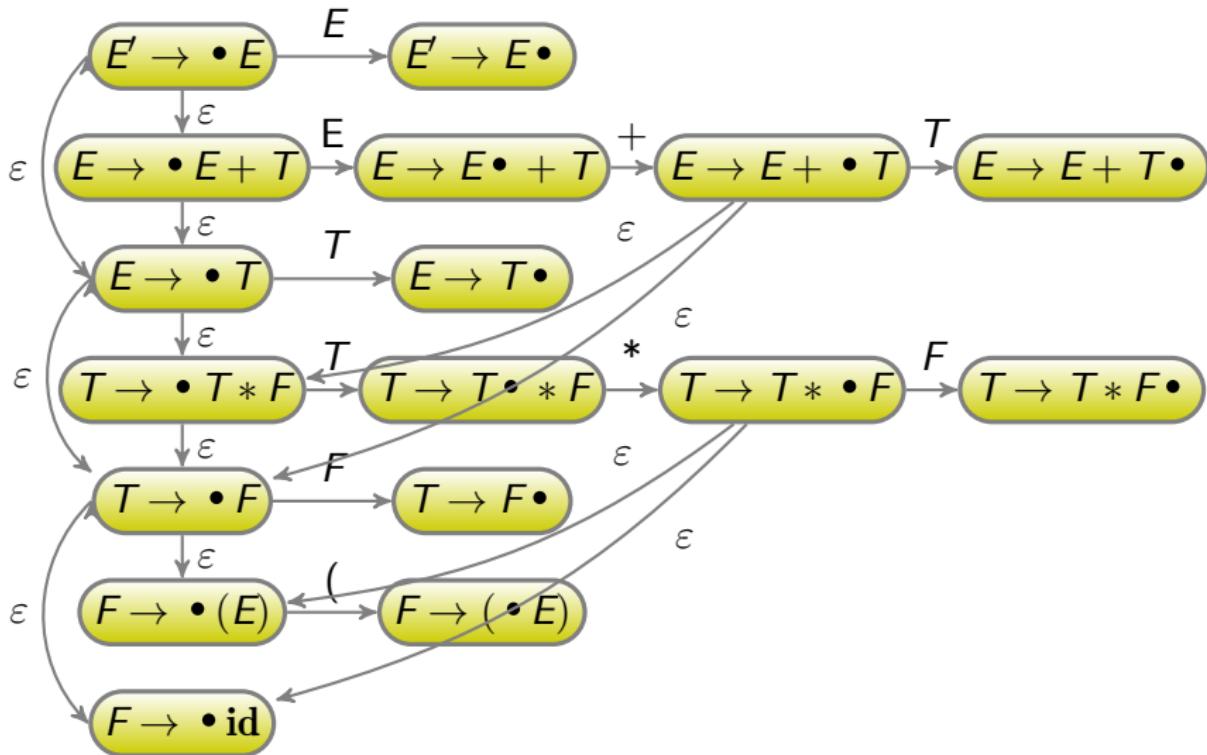
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



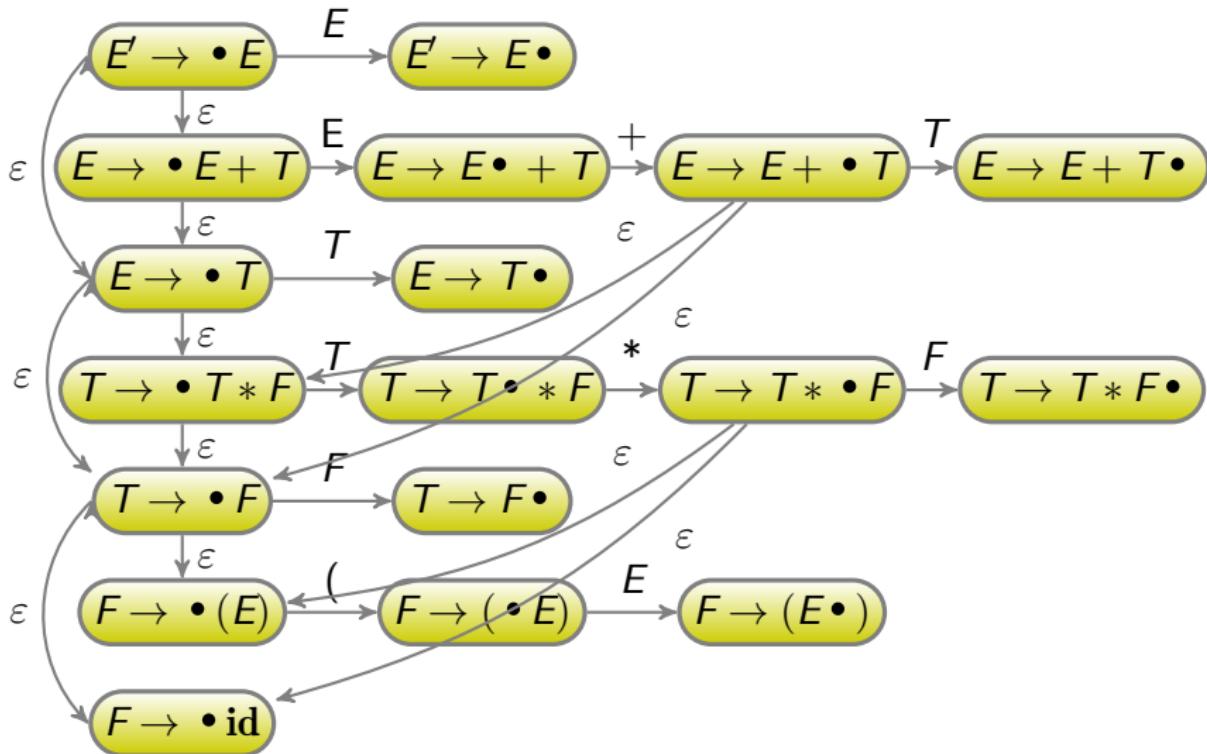
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



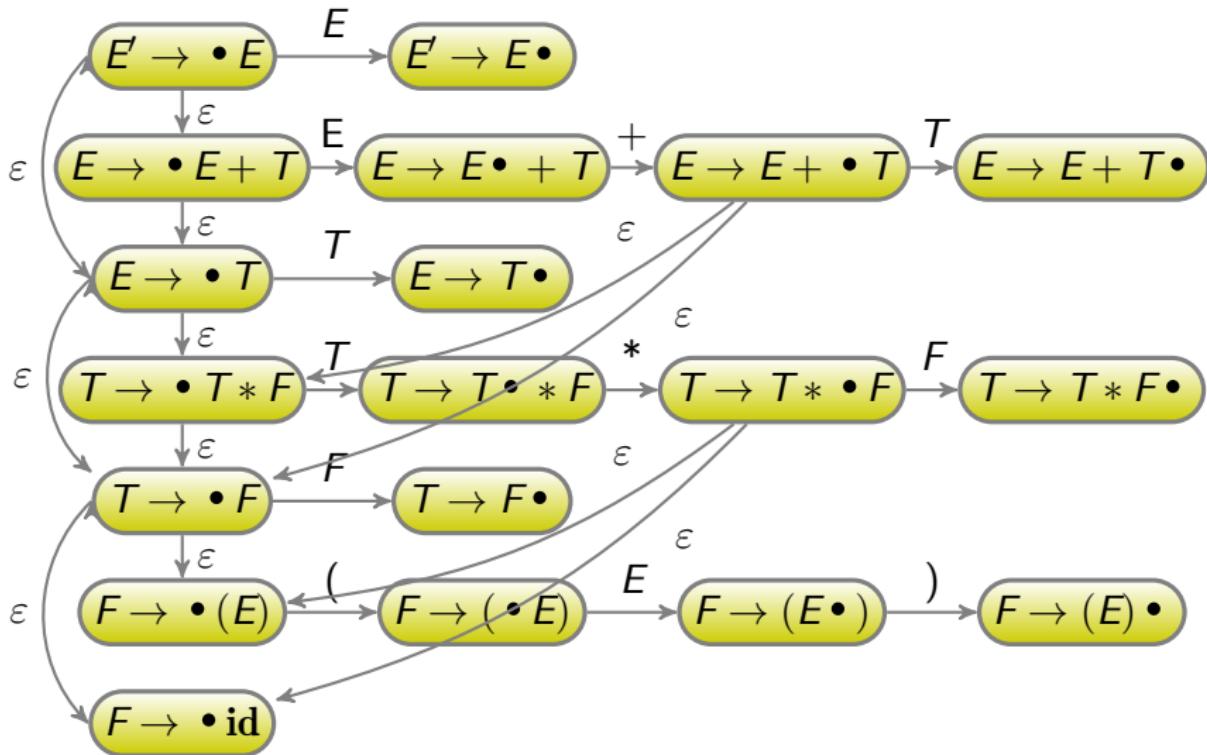
4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



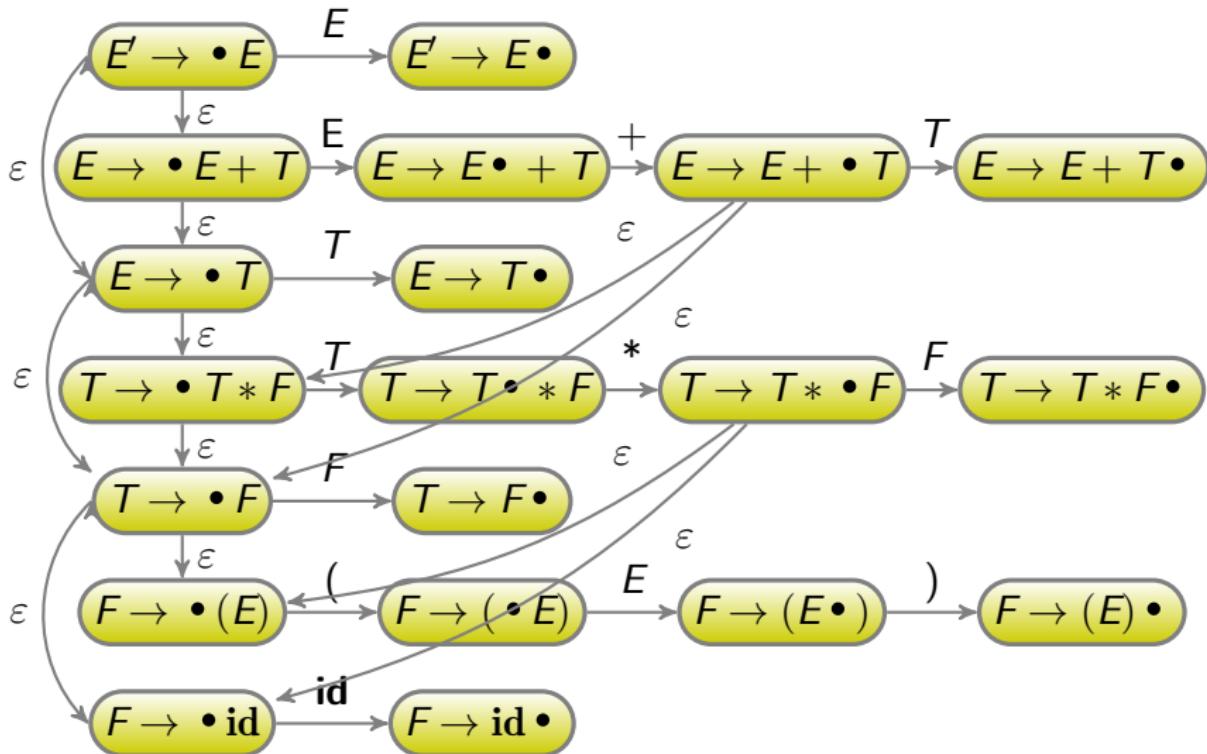
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



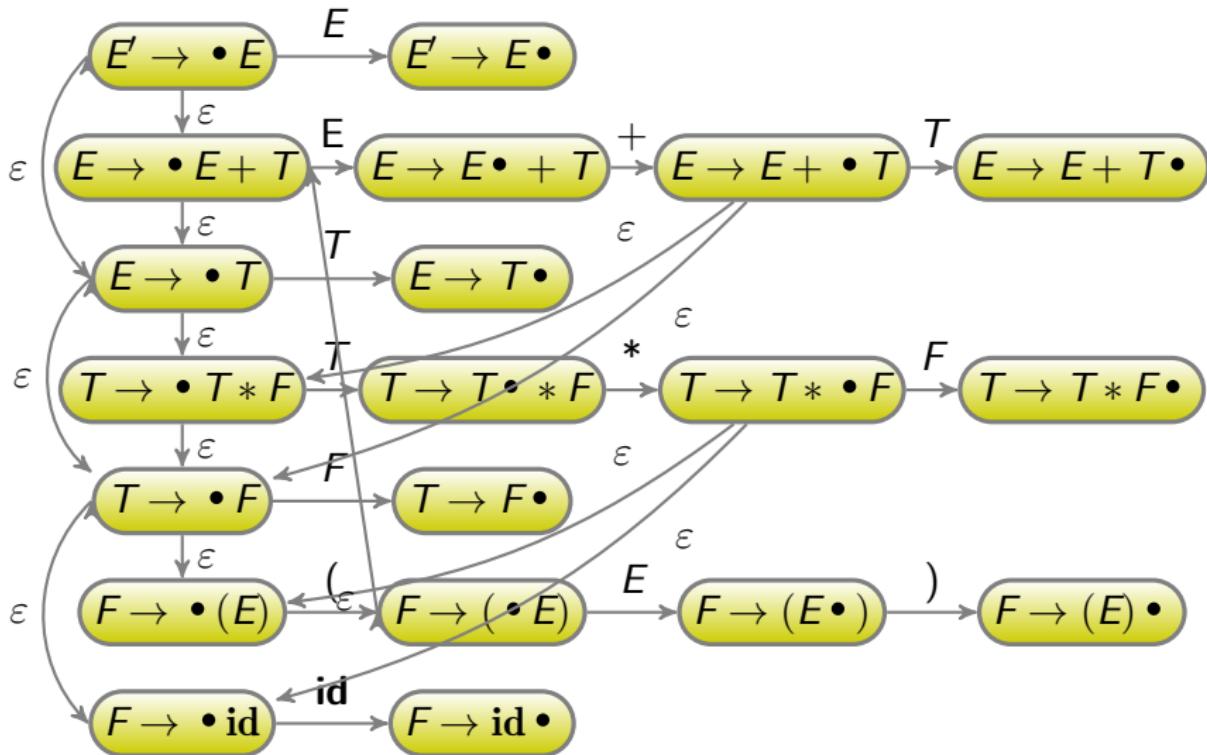
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



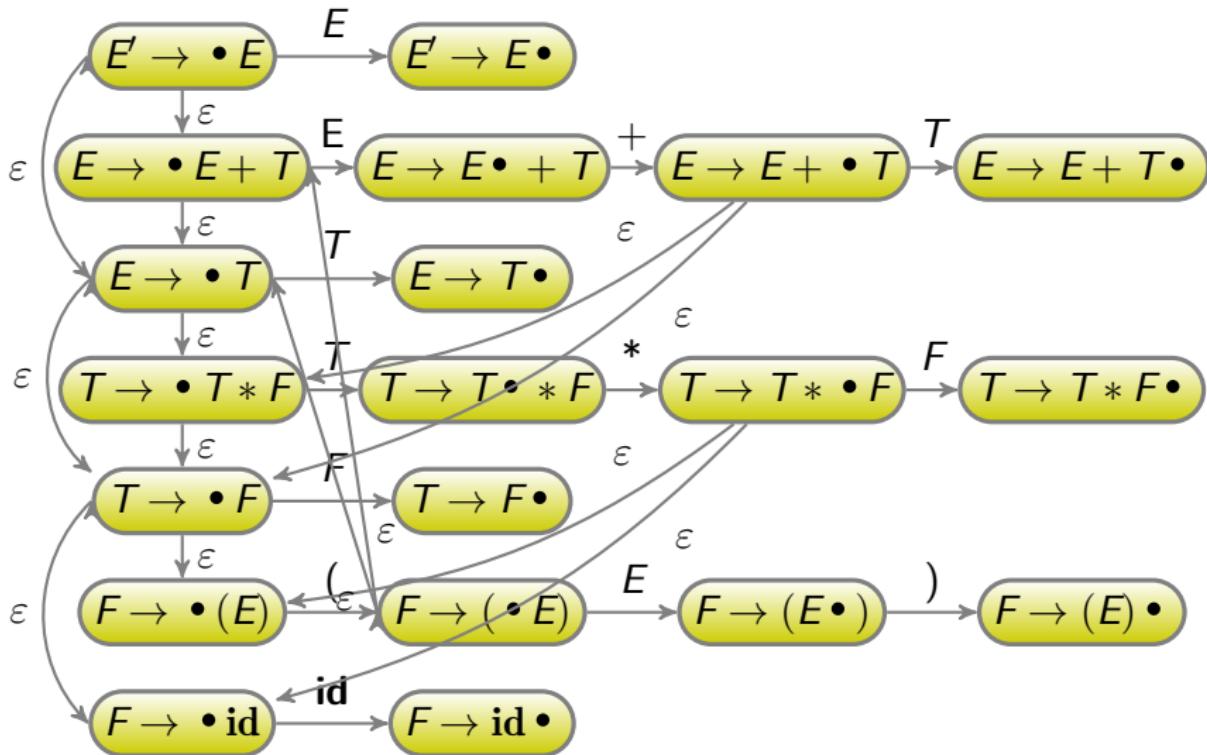
3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$



3: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$ ,  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $A \rightarrow \alpha B \bullet \beta$  przejściem etykietowanym  $B$

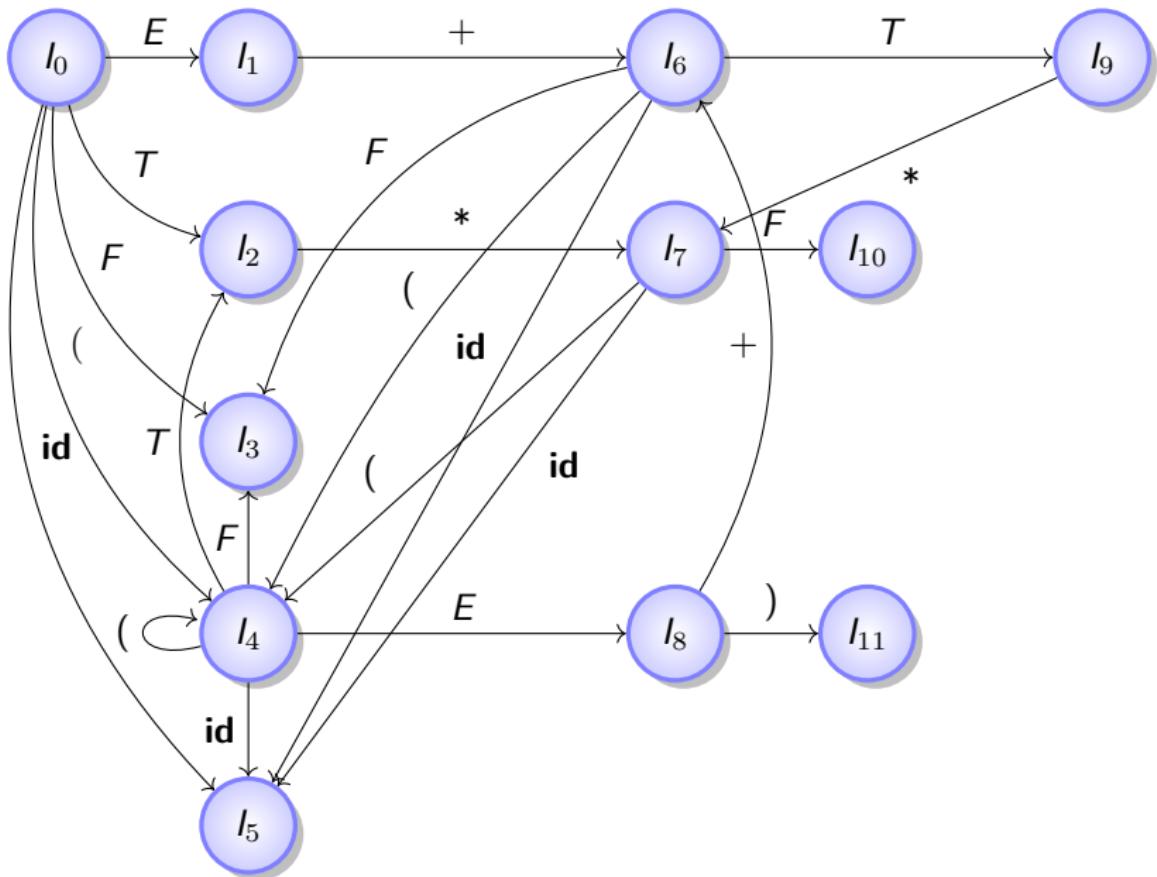


4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$



4: Każdą sytuację  $A \rightarrow \alpha \bullet B\beta$  połącz z sytuacją  $B \rightarrow \bullet \gamma$ ,  $\gamma \in (\Sigma \cup V)^*$  przejściem etykietowanym  $\epsilon$

# Wypełnianie tablicy analizatora SLR





# Wypełnianie tablicy analizator SLR

$I_0$

$$\begin{aligned}E' &\rightarrow \bullet E \\E &\rightarrow \bullet E + T \\E &\rightarrow \bullet T \\T &\rightarrow \bullet T * F \\T &\rightarrow \bullet F \\F &\rightarrow \bullet (E) \\F &\rightarrow \bullet \text{id}\end{aligned}$$

$I_1$

$$\begin{aligned}E' &\rightarrow E \bullet \\E &\rightarrow E \bullet + T\end{aligned}$$

$I_2$

$$\begin{aligned}E &\rightarrow T \bullet \\T &\rightarrow T \bullet * F\end{aligned}$$

$I_3$

$$T \rightarrow F \bullet$$

$I_4$

$$\begin{aligned}F &\rightarrow (\bullet E) \\E &\rightarrow \bullet E + T \\E &\rightarrow \bullet T \\T &\rightarrow \bullet T * F \\T &\rightarrow \bullet F \\F &\rightarrow \bullet (E) \\F &\rightarrow \bullet \text{id}\end{aligned}$$

$I_6$

$$\begin{aligned}E &\rightarrow E + \bullet T \\T &\rightarrow \bullet T * F \\T &\rightarrow \bullet F \\F &\rightarrow \bullet (E) \\F &\rightarrow \bullet \text{id}\end{aligned}$$

$I_9$

$$\begin{aligned}E &\rightarrow E + T \bullet \\T &\rightarrow T \bullet * F\end{aligned}$$

$I_{10}$

$$T \rightarrow T * F \bullet$$

$I_7$

$$\begin{aligned}T &\rightarrow T * \bullet F \\F &\rightarrow \bullet (E) \\F &\rightarrow \bullet \text{id}\end{aligned}$$

$I_{11}$

$$F \rightarrow (E) \bullet$$

$I_5$

$$F \rightarrow \text{id} \bullet$$

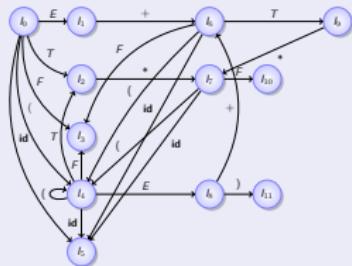
$I_8$

$$\begin{aligned}F &\rightarrow (E \bullet) \\E &\rightarrow E \bullet + T\end{aligned}$$

## tablica analizatora SLR

stan	id	+	*	()	\$	działanie			przejście		
						E	T	F	E	T	F
0											
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											

## Diagram przejść stanów



$I_0$

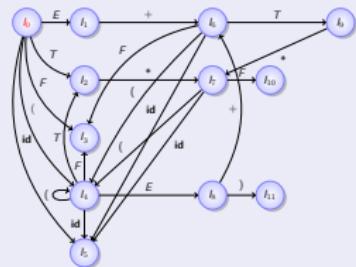
0.  $E' \rightarrow \bullet E$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**  $\text{działanie}[i, a] = s$  j

## tablica analizatora SLR

stan	id	+	*	()	\$	działanie			przejście		
						E	T	F	E	T	F
0											
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											

## Diagram przejść stanów



$I_0$

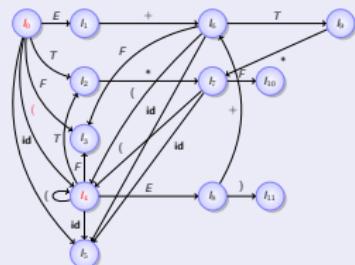
0.  $E' \rightarrow \bullet E$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	+	*	(	)	\$	przejście		
							E	T	F
0						s4			
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_0$

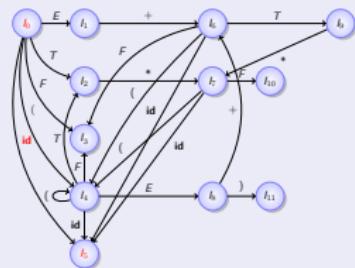
0.  $E' \rightarrow \bullet E$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	działanie					przejście			
	id	+	*	(	)	\$	$E$	$T$	$F$
0	s5			s4					
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_0$

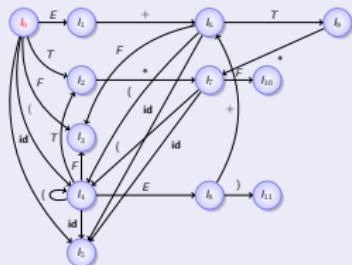
0.  $E' \rightarrow \bullet E$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4					
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_0$

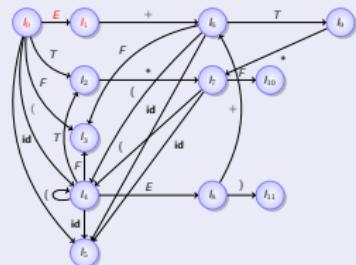
0.  $E' \rightarrow \bullet E$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(I_i, A) = I_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście <i>E</i>	<i>T</i>	<i>F</i>
		+	*	(	)	\$			
0	s5		s4				1		
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_0$

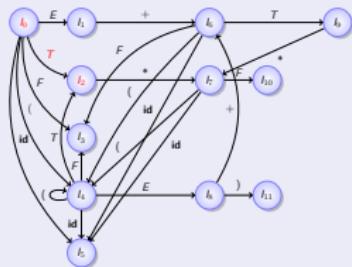
0.  $E' \rightarrow \bullet E$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet \text{id}$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_0$

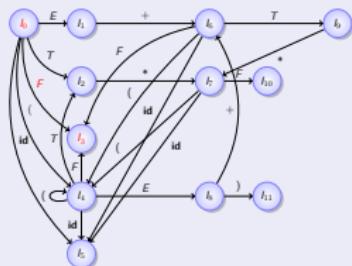
0.  $E' \rightarrow \bullet E$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_0$

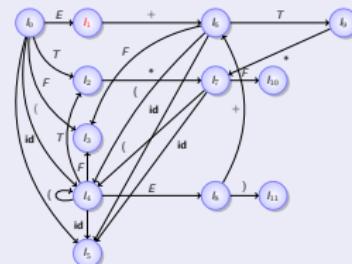
0.  $E' \rightarrow \bullet E$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(I_i, A) = I_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_1$

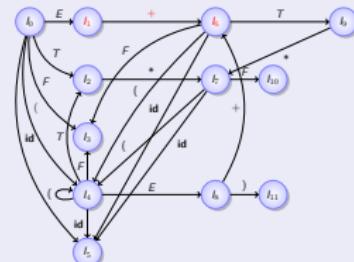
0.  $E' \rightarrow E \bullet$
1.  $E \rightarrow E \bullet + T$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6							
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_1$

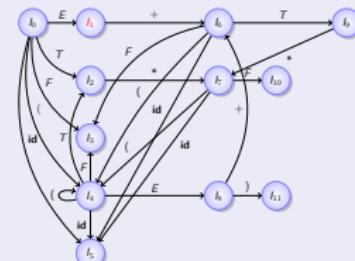
0.  $E' \rightarrow E \bullet$
1.  $E \rightarrow E \bullet + T$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_1$

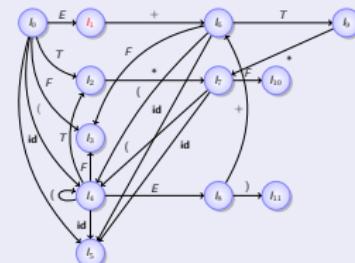
0.  $E' \rightarrow E \bullet$
1.  $E \rightarrow E \bullet + T$

if  $S' \rightarrow S \bullet$  jest w  $I_i$  then działanie[i, \$] = akc

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_1$

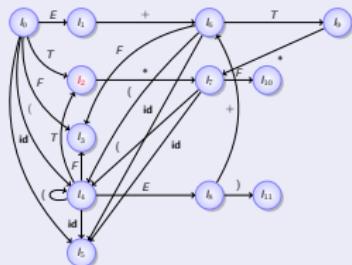
0.  $E' \rightarrow E \bullet$
1.  $E \rightarrow E \bullet + T$

if  $S' \rightarrow S \bullet$  jest w  $I_i$  then działanie[i, \$] = akc

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_2$

2.  $E \rightarrow T\bullet$
3.  $T \rightarrow T\bullet *F$

$nast(E)$

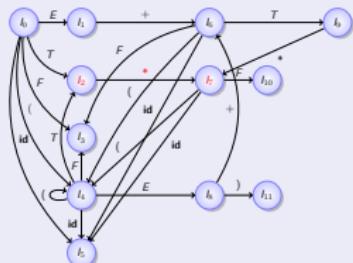
$+), \$$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $l_i$  i  $\delta(l_i, a) = l_j$  then  $działanie[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$		E	T
0	s5			s4			1	2	3
1	s6					akc			
2			s7						
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_2$

2.  $E \rightarrow T^\bullet$
3.  $T \rightarrow T^\bullet * F$

$nast(E)$

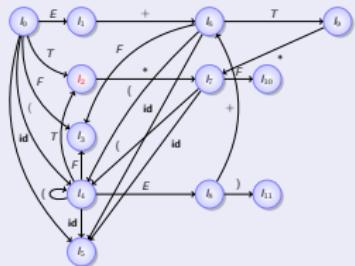
$+), \$$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $l_i$  i  $\delta(l_i, a) = l_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2				s7					
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_2$

2.  $E \rightarrow T \bullet$
3.  $T \rightarrow T \bullet * F$

$nast(E)$

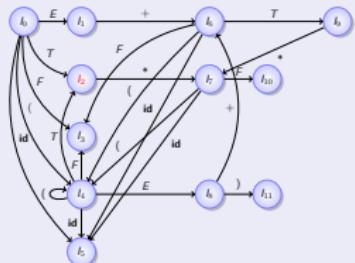
$, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i$ ,  $A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6							
2		r2	s7						
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_2$

- 2.  $E \rightarrow T \bullet$
- 3.  $T \rightarrow T \bullet *F$

$nast(E)$

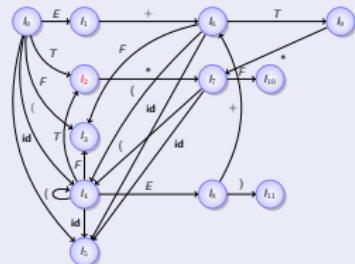
$+, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i$ ,  $A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6							akc
2		r2	s7			r2			
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_2$

- 2.  $E \rightarrow T \bullet$
- 3.  $T \rightarrow T \bullet * F$

$nast(E)$

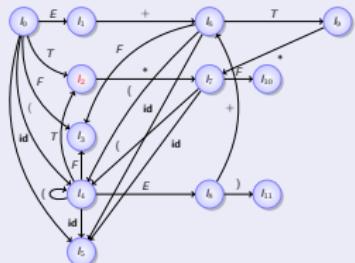
$+), \$$

if  $A \rightarrow \alpha \bullet \in I_i$ ,  $A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2		r2	s7		r2	r2			
3									
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_2$

- 2.  $E \rightarrow T \bullet$
- 3.  $T \rightarrow T \bullet * F$

$nast(E)$

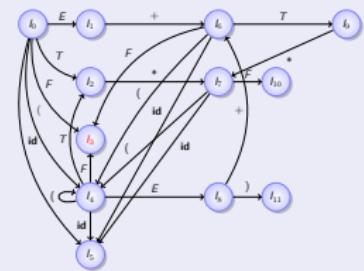
$+), \$$

if  $A \rightarrow \alpha \bullet \in I_i$ ,  $A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3			r4						
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_3$

4.  $T \rightarrow F \bullet$

$nast(T)$

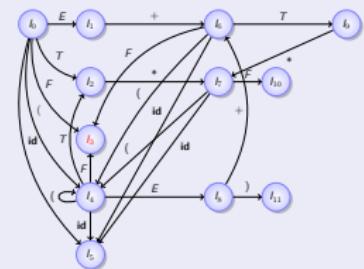
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r_m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4		r4						
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_3$

4.  $T \rightarrow F \bullet$

$nast(T)$

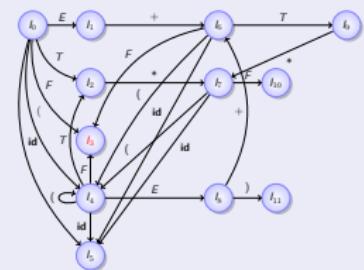
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r\ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4		r4					
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_3$

4.  $T \rightarrow F \bullet$

$nast(T)$

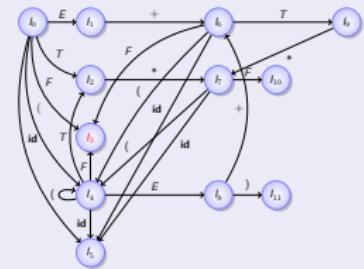
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r_m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_3$

4.  $T \rightarrow F \bullet$

$nast(T)$

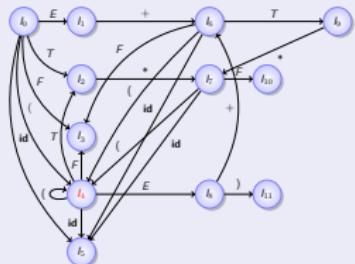
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r_m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4									
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_4$

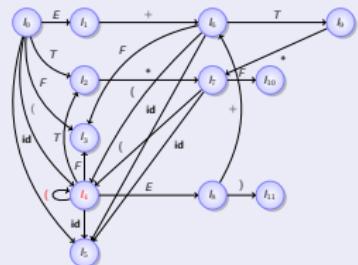
5.  $F \rightarrow (\bullet E)$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4				s4					
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_4$

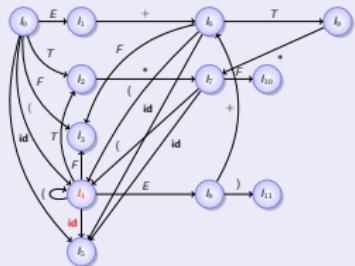
5.  $F \rightarrow (\bullet E)$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4					
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_4$

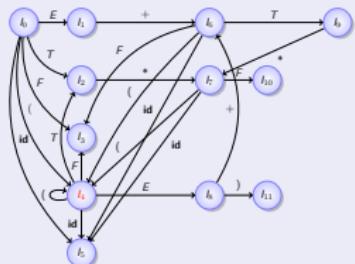
5.  $F \rightarrow (\bullet E)$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4					
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_4$

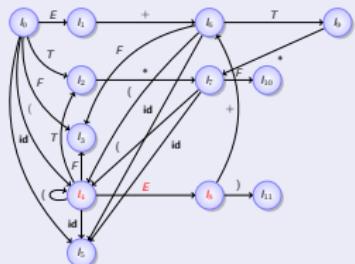
5.  $F \rightarrow (\bullet E)$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8		
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_4$

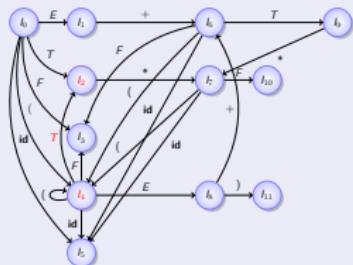
5.  $F \rightarrow (\bullet E)$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_4$

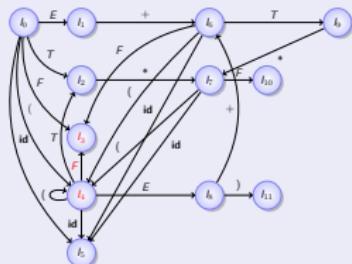
5.  $F \rightarrow (\bullet E)$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_4$

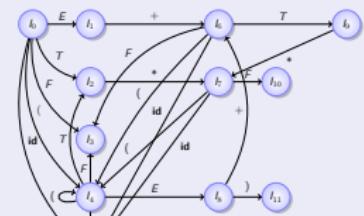
5.  $F \rightarrow (\bullet E)$
1.  $E \rightarrow \bullet E + T$
2.  $E \rightarrow \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5									
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_5$

6.  $F \rightarrow \text{id} \bullet$

$nast(F)$

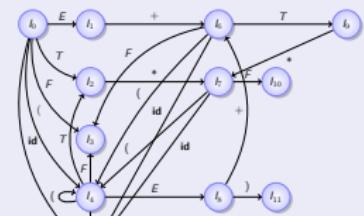
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $\text{działanie}[i, a] = r_m, m \text{ jest nr } A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5			r6						
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_5$

6.  $F \rightarrow \text{id} \bullet$

$nast(F)$

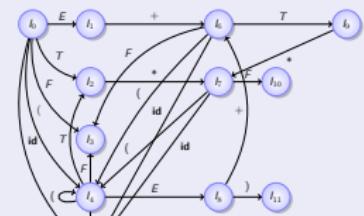
\* , + , ) , \$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $\text{działanie}[i, a] = r_m, m \text{ jest nr } A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6							
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_5$

6.  $F \rightarrow \text{id} \bullet$

$nast(F)$

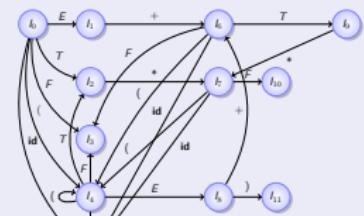
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $\text{działanie}[i, a] = r_m, m \text{ jest nr } A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6		r6					
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_5$

6.  $F \rightarrow \text{id} \bullet$

$nast(F)$

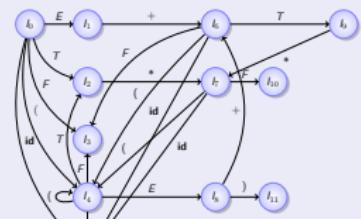
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $\text{działanie}[i, a] = r_m, m \text{ jest nr } A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6		r6	r6				
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_5$

6.  $F \rightarrow \text{id} \bullet$

$nast(F)$

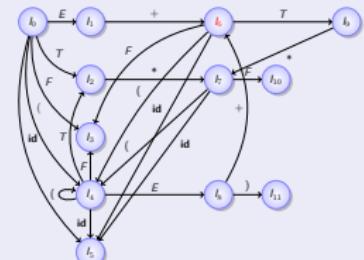
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $\text{działanie}[i, a] = r \ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6									
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_6$

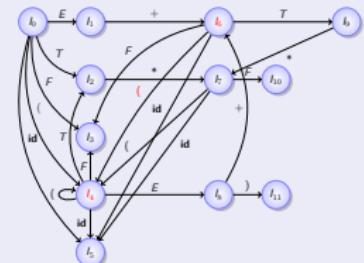
1.  $E \rightarrow E + \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6				s4					
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_6$

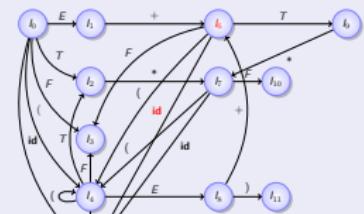
1.  $E \rightarrow E + \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4					
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_6$

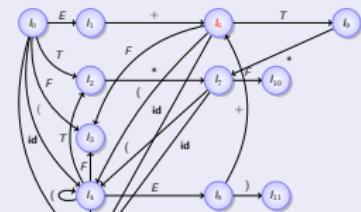
1.  $E \rightarrow E + \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4					
7									
8									
9									
10									
11									

## Diagram przejść stanów



$I_6$

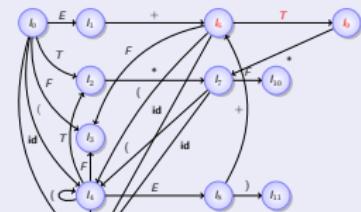
1.  $E \rightarrow E + \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do  $\text{przejście}[i, A] = j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4					9
7									
8									
9									
10									
11									

## Diagram przejść stanów



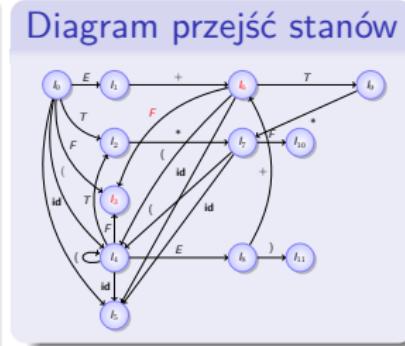
$I_6$

1.  $E \rightarrow E + \bullet T$
3.  $T \rightarrow \bullet T * F$
4.  $T \rightarrow \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do  $\text{przejście}[i, A] = j$



## tablica analizatora SLR



l<sub>6</sub>

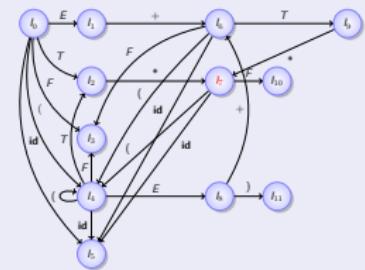
1.  $E \rightarrow E + \bullet T$
  3.  $T \rightarrow \bullet T * F$
  4.  $T \rightarrow \bullet F$
  5.  $F \rightarrow \bullet (E)$
  6.  $F \rightarrow \bullet \text{id}$

**for all**  $A$  takie że  $\delta(I_i, A) = I_j$  **do** przejście $[i, A] = j$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6		r6	r6				
6	s5			s4				9	3
7									
8									
9									
10									
11									

## Diagram przejść stanów



$l_7$

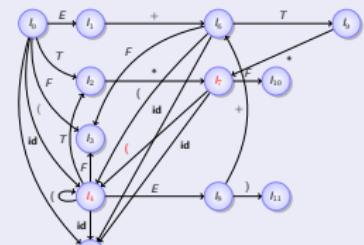
3.  $T \rightarrow T * \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $l_i$  i  $\delta(l_i, a) = l_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4				9	3
7				s4					
8									
9									
10									
11									

## Diagram przejść stanów



$I_7$

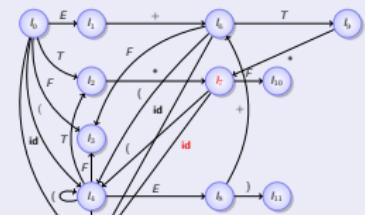
3.  $T \rightarrow T * \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4				9	3
7	s5			s4					
8									
9									
10									
11									

## Diagram przejść stanów



$l_7$

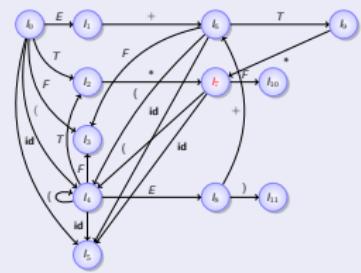
3.  $T \rightarrow T * \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $l_i$  i  $\delta(l_i, a) = l_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4				9	3
7	s5			s4					
8									
9									
10									
11									

## Diagram przejść stanów



$I_7$

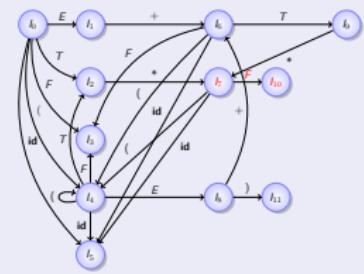
3.  $T \rightarrow T * \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(I_i, A) = I_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8									
9									
10									
11									

## Diagram przejść stanów



$I_7$

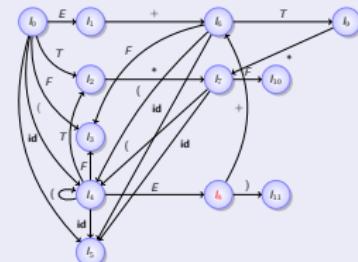
3.  $T \rightarrow T * \bullet F$
5.  $F \rightarrow \bullet (E)$
6.  $F \rightarrow \bullet id$

for all  $A$  takie że  $\delta(l_i, A) = l_j$  do przejście[i, A] = j

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6		r6	r6				
6	s5			s4			9	3	
7	s5			s4					10
8									
9									
10									
11									

## Diagram przejść stanów



$I_8$

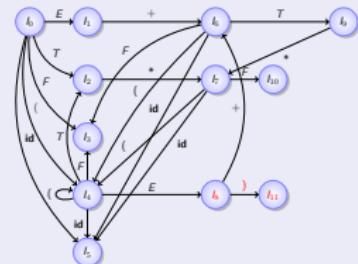
5.  $F \rightarrow (E \bullet)$
1.  $E \rightarrow E \bullet + T$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie				przejście		
		+	*	(	)	E	T	F
0	s5			s4		1	2	3
1	s6							akc
2	r2	s7		r2	r2			
3	r4	r4		r4	r4			
4	s5			s4		8	2	3
5	r6	r6		r6	r6			
6	s5			s4			9	3
7	s5			s4				10
8					s11			
9								
10								
11								

## Diagram przejść stanów



$l_8$

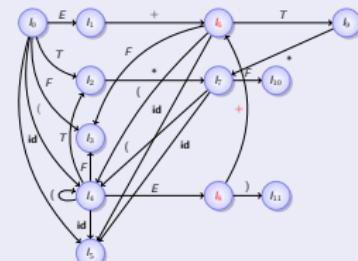
5.  $F \rightarrow (E \bullet)$
1.  $E \rightarrow E \bullet + T$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $l_i$  i  $\delta(l_i, a) = l_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7		r2	r2				
3	r4	r4		r4	r4				
4	s5			s4			8	2	3
5	r6	r6		r6	r6				
6	s5			s4			9	3	
7	s5			s4					10
8	s6				s11				
9									
10									
11									

## Diagram przejść stanów



$I_8$

5.  $F \rightarrow (E \bullet)$
1.  $E \rightarrow E \bullet + T$

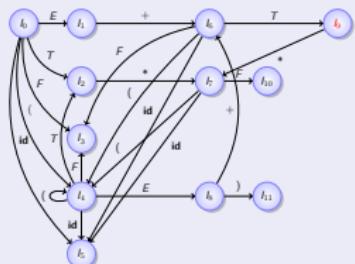
if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  then  $\text{działanie}[i, a] = s_j$



## Wypełnianie tablicy analizatora SLR



## tablica analizatora SLR



19

1.  $E \rightarrow E + T \bullet$
  3.  $T \rightarrow T \bullet * F$

*nast*( $E$ )

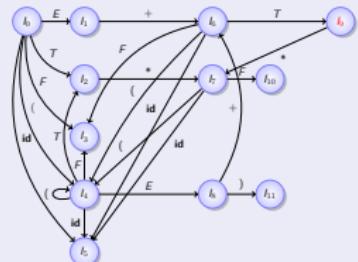
+,\$

**if**  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8	s6				s11				
9				s7					
10									
11									

## Diagram przejść stanów



$l_9$

1.  $E \rightarrow E + T \bullet$
3.  $T \rightarrow T \bullet *F$

$nast(E)$

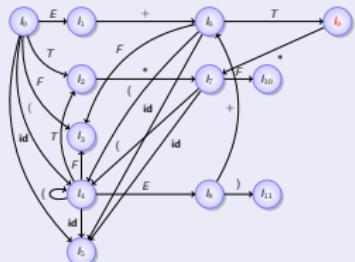
$+), \$$

if  $A \rightarrow \alpha \bullet a\beta$ ,  $a \in \Sigma$  jest w  $l_i$  i  $\delta(l_i, a) = l_j$  then  $\text{działanie}[i, a] = s_j$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8	s6				s11				
9				s7					
10									
11									

## Diagram przejść stanów



$l_9$

1.  $E \rightarrow E + T \bullet$
3.  $T \rightarrow T \bullet * F$

$nast(E)$

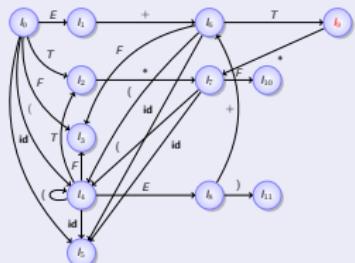
$+), \$$

if  $A \rightarrow \alpha \bullet \in I_i$ ,  $A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8	s6				s11				
9	r1	s7							
10									
11									

## Diagram przejść stanów



$l_9$

1.  $E \rightarrow E + T \bullet$
3.  $T \rightarrow T \bullet * F$

$nast(E)$

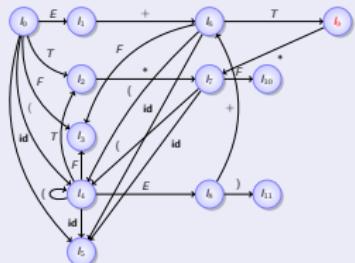
$+), \$$

if  $A \rightarrow \alpha \bullet \in I_i$ ,  $A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1				
10									
11									

## Diagram przejść stanów



$I_9$

1.  $E \rightarrow E + T \bullet$
3.  $T \rightarrow T \bullet * F$

$nast(E)$

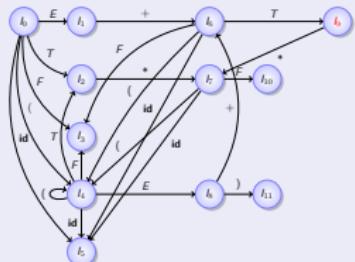
$, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i$ ,  $A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1		s6				akc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10									
11									

## Diagram przejść stanów



$I_9$

1.  $E \rightarrow E + T \bullet$
3.  $T \rightarrow T \bullet * F$

$nast(E)$

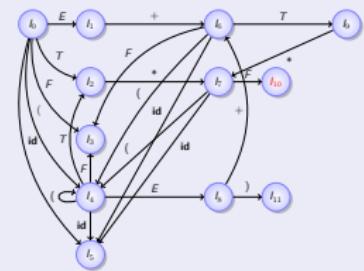
$+), \$$

if  $A \rightarrow \alpha \bullet \in I_i$ ,  $A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9		3
7	s5			s4					10
8	s6				s11				
9	r1	s7			r1	r1			
10									
11									

## Diagram przejść stanów



$l_{10}$

3.  $T \rightarrow T * F \bullet$

$nast(T)$

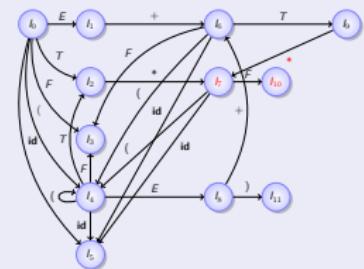
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r\ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9		3
7	s5			s4					10
8	s6				s11				
9	r1	s7			r1	r1			
10			r3						
11									

## Diagram przejść stanów



$l_{10}$

3.  $T \rightarrow T * F \bullet$

$nast(T)$

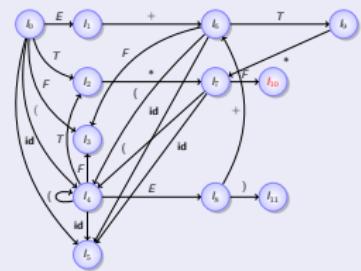
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r\ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9		3
7	s5			s4					10
8	s6				s11				
9	r1	s7			r1	r1			
10	r3	r3							
11									

## Diagram przejść stanów



$l_{10}$

3.  $T \rightarrow T * F \bullet$

$nast(T)$

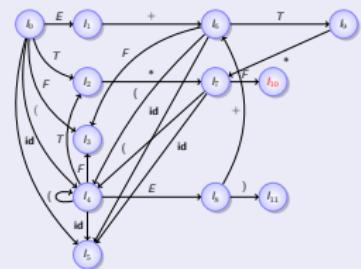
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r_m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie				przejście		
		+	*	(	)	E	T	F
0	s5			s4		1	2	3
1	s6							akc
2	r2	s7			r2	r2		
3	r4	r4			r4	r4		
4	s5			s4		8	2	3
5	r6	r6			r6	r6		
6	s5			s4			9	3
7	s5			s4				10
8	s6			s11				
9	r1	s7			r1	r1		
10	r3	r3		r3				
11								

## Diagram przejść stanów



$l_{10}$

3.  $T \rightarrow T * F \bullet$

$nast(T)$

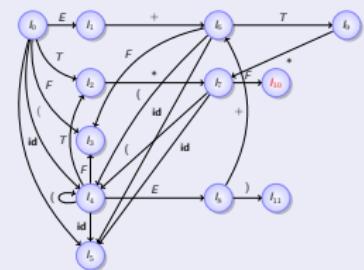
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r$  m, m jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8	s6			s11					
9	r1	s7			r1	r1			
10	r3	r3			r3	r3			
11									

## Diagram przejść stanów



$l_{10}$

3.  $T \rightarrow T * F \bullet$

$nast(T)$

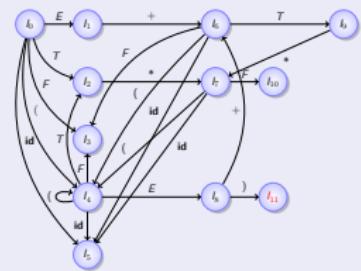
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r \ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8	s6			s11					
9	r1	s7			r1	r1			
10	r3	r3			r3	r3			
11									

## Diagram przejść stanów



$l_{11}$

5.  $F \rightarrow (E) \bullet$

$nast(F)$

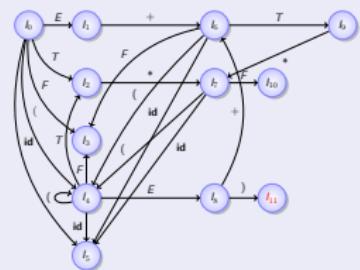
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r \ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8	s6			s11					
9	r1	s7			r1	r1			
10	r3	r3			r3	r3			
11				r5					

## Diagram przejść stanów



$l_{11}$

5.  $F \rightarrow (E) \bullet$

$nast(F)$

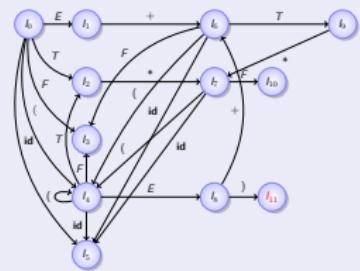
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r\ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie				przejście			
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8	s6			s11					
9	r1	s7			r1	r1			
10	r3	r3			r3	r3			
11	r5	r5							

## Diagram przejść stanów



$l_{11}$

5.  $F \rightarrow (E) \bullet$

$nast(F)$

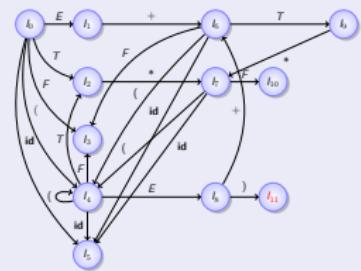
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r\ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie				przejście		
		+	*	(	)	E	T	F
0	s5			s4		1	2	3
1	s6							akc
2	r2	s7			r2	r2		
3	r4	r4			r4	r4		
4	s5			s4		8	2	3
5	r6	r6			r6	r6		
6	s5			s4			9	3
7	s5			s4				10
8	s6			s11				
9	r1	s7			r1	r1		
10	r3	r3			r3	r3		
11	r5	r5		r5				

## Diagram przejść stanów



$l_{11}$

5.  $F \rightarrow (E) \bullet$

$nast(F)$

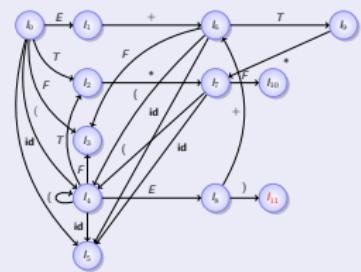
$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r \ m, m$  jest nr  $A \rightarrow \alpha$

## tablica analizatora SLR

stan	id	działanie					przejście		
		+	*	(	)	\$	E	T	F
0	s5			s4			1	2	3
1	s6					akc			
2	r2	s7			r2	r2			
3	r4	r4			r4	r4			
4	s5			s4			8	2	3
5	r6	r6			r6	r6			
6	s5			s4			9	3	
7	s5			s4					10
8	s6			s11					
9	r1	s7			r1	r1			
10	r3	r3			r3	r3			
11	r5	r5			r5	r5			

## Diagram przejść stanów



$l_{11}$

5.  $F \rightarrow (E) \bullet$

$nast(F)$

$*, +, ), \$$

if  $A \rightarrow \alpha \bullet \in I_i, A \neq S'$  then for all  $a \in nast(A)$  do  $działanie[i, a] = r \ m, m$  jest nr  $A \rightarrow \alpha$



Rozpatrzmy bardzo uproszczoną gramatykę wyrażeń:

$$E \rightarrow L < R$$

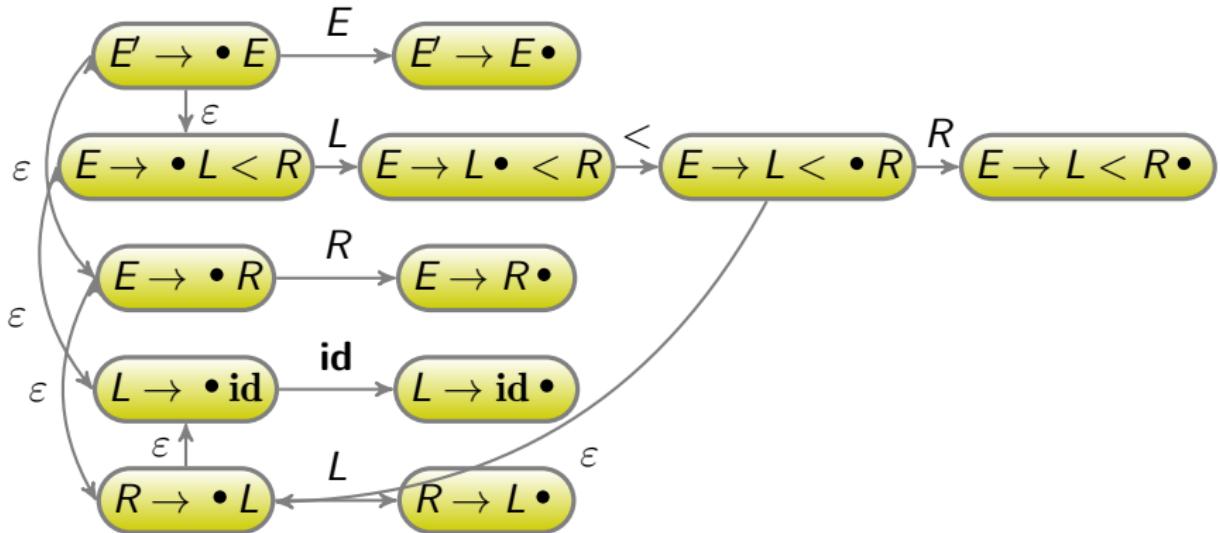
$$E \rightarrow R$$

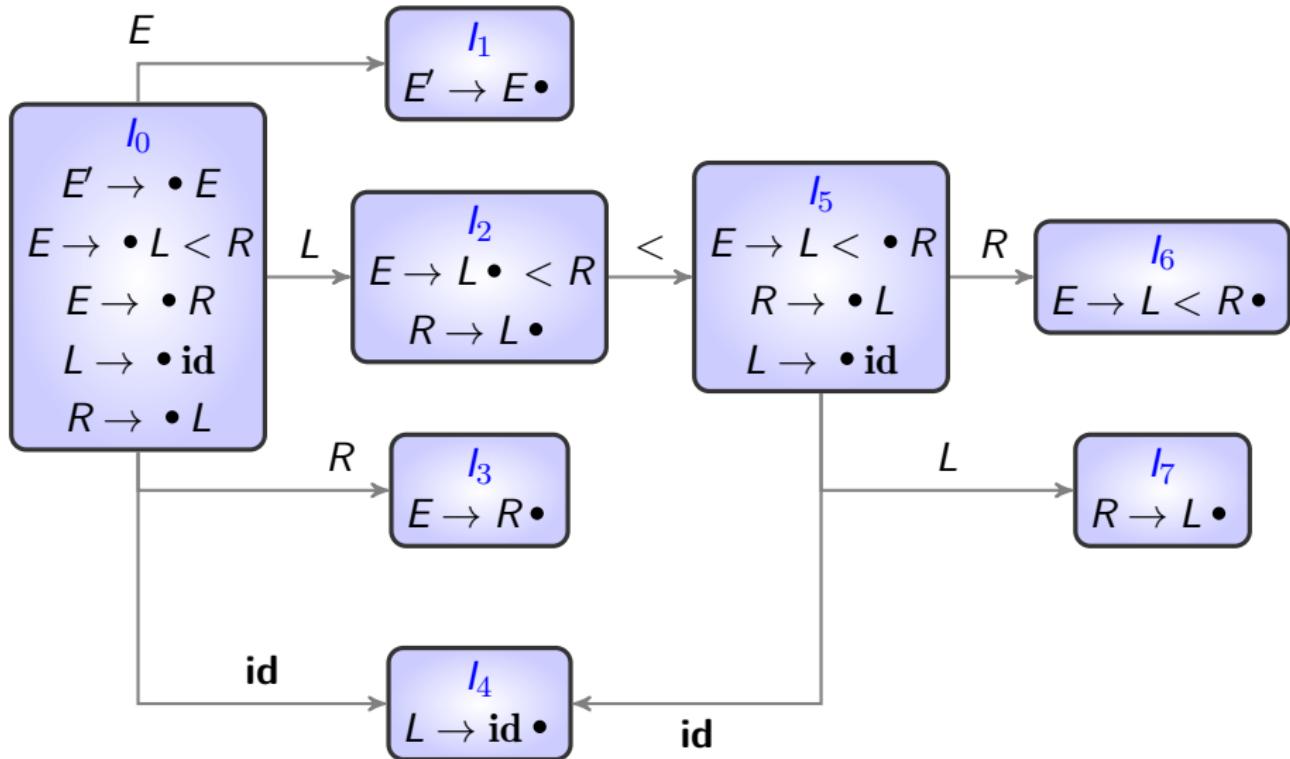
$$L \rightarrow \text{id}$$

$$R \rightarrow L$$

Gramatyka odpowiada wyrażeniom logicznym, gdzie porównywane są dwa wyrażenia arytmetyczne. Dla uproszczenia przyjęliśmy tylko jeden operator porównania. Wyrażenie może być też samym wyrażeniem arytmetycznym, którego też dla uproszczenia nie rozwijamy dalej, a tylko rozstawiliśmy z niego identyfikator. Jedyną cechą szczególną tej gramatyki jest to, że osobno rozwijamy lewy i prawy argument.

Zbudujmy dla tej gramatyki automat.







Stan  $I_2$  zawiera sytuacje:

$$E \rightarrow L \bullet < R$$

$$R \rightarrow L \bullet$$

Co to oznacza? Do działanie  $[2, <]$  możemy wstawić s5, ponieważ mamy sytuację  $E \rightarrow L \bullet < R$ . Jednocześnie druga sytuacja oznacza zwinięcie reguły i widać, że  $<$  jest w  $nast(L)$ , więc do tej samej komórki tablicy należało wstawić zwinięcie. Tymczasem nie istnieje taki ciąg produkcji, który doprowadzi do uzyskania ciągu symboli  $R <$ , więc zwinięcie w tym miejscu nie jest możliwe.



Rozwiązaniem może być uwzględnienie większego kontekstu w sytuacjach

Sytuację  $LR(1)$  nazywamy sytuację  $LR(0)$  z dołączonym symbolem końcowym lub symbolem końca wejścia  $\$$ . Np. dla reguły produkcji  $A \rightarrow XY$  i symbolu końcowego  $a$  mamy:

$$A \rightarrow \bullet XY, a$$

$$A \rightarrow X \bullet Y, a$$

$$A \rightarrow XY \bullet, a$$

Podobnie dla symbolu końca wejścia  $\$$ .

Z takich sytuacji tworzony jest automat.



## Tworzenie automatu dla tablicy analizy kanonicznej LR

- ① Z gramatyki  $G = (\Sigma, V, P, S)$  utwórz gramatykę  $G' = (\Sigma, V \cup \{S'\}, P \cup \{S' \rightarrow S\}, S')$
- ② Utwórz stany automatu z sytuacji  $LR(1)$   $G'$
- ③ Każdą sytuację  $[A \rightarrow \alpha \bullet B\beta, a]$ , gdzie  $\alpha, \beta \in (\Sigma \cup V)^*$ ,  $B \in (\Sigma \cup V)$  połącz z sytuacją  $[A \rightarrow \alpha B \bullet \beta, a]$  przejściem etykietowanym  $B$
- ④ Każdą sytuację  $[A \rightarrow \alpha \bullet B\beta, a]$  połącz z sytuacją  $[B \rightarrow \bullet \gamma, b]$ , gdzie  $\gamma \in (\Sigma \cup V)^*$ ,  $b \in \text{pierw}(\beta a)$  przejściem etykietowanym  $\varepsilon$
- ⑤ Determinizuj automat



## Algorytm wypełniania kanonicznej tablicy analizatora LR

- 1: Utwórz automat deterministyczny ze stanami  $I_0, I_1, \dots, I_n$
- 2: **if**  $[A \rightarrow \alpha \bullet a\beta, b]$  dla  $a \in \Sigma$ ,  $b \in (\Sigma \cup \{\$\})$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**
- 3:     działanie $[i, a] = s\ j$
- 4: **end if**
- 5: **if**  $[A \rightarrow \alpha \bullet, a]$  jest w  $I_i$ ,  $A \neq S'$  **then**
- 6:     działanie $[i, a] = r\ m$ , gdzie  $m$  jest numerem produkcji  $A \rightarrow \alpha$
- 7: **end if**
- 8: **if**  $S' \rightarrow S \bullet$  jest w  $I_i$  **then**
- 9:     działanie $[i, \$] = \text{akc}$
- 10: **end if**
- 11: **for all**  $A$  takie że  $\delta(I_i, A) = I_j$  **do**
- 12:     przejście $[i, A] = j$
- 13: **end for**

Tak więc w stosunku do algorytmu dla SLR praktycznie zmienia się nieznacznie zwinięcie reguły.



## Algorytm wypełniania kanonicznej tablicy analizatora LR

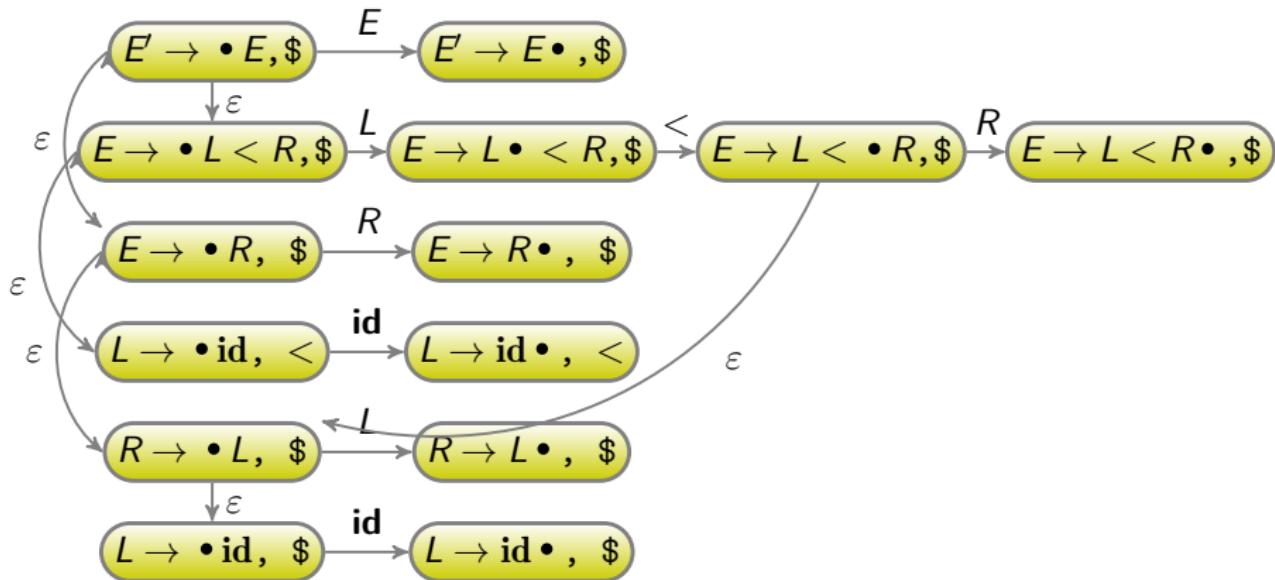
- 1: Utwórz automat deterministyczny ze stanami  $I_0, I_1, \dots, I_n$
- 2: **if**  $[A \rightarrow \alpha \bullet a\beta, b]$  dla  $a \in \Sigma$ ,  $b \in (\Sigma \cup \{\$\})$  jest w  $I_i$  i  $\delta(I_i, a) = I_j$  **then**
- 3:     działanie $[i, a] = s\ j$
- 4: **end if**
- 5: **if**  $[A \rightarrow \alpha \bullet, a]$  jest w  $I_i$ ,  $A \neq S'$  **then**
- 6:     działanie $[i, a] = r\ m$ , gdzie  $m$  jest numerem produkcji  $A \rightarrow \alpha$
- 7: **end if**
- 8: **if**  $S' \rightarrow S \bullet$  jest w  $I_i$  **then**
- 9:     działanie $[i, \$] = \text{akc}$
- 10: **end if**
- 11: **for all**  $A$  takie że  $\delta(I_i, A) = I_j$  **do**
- 12:     przejście $[i, A] = j$
- 13: **end for**

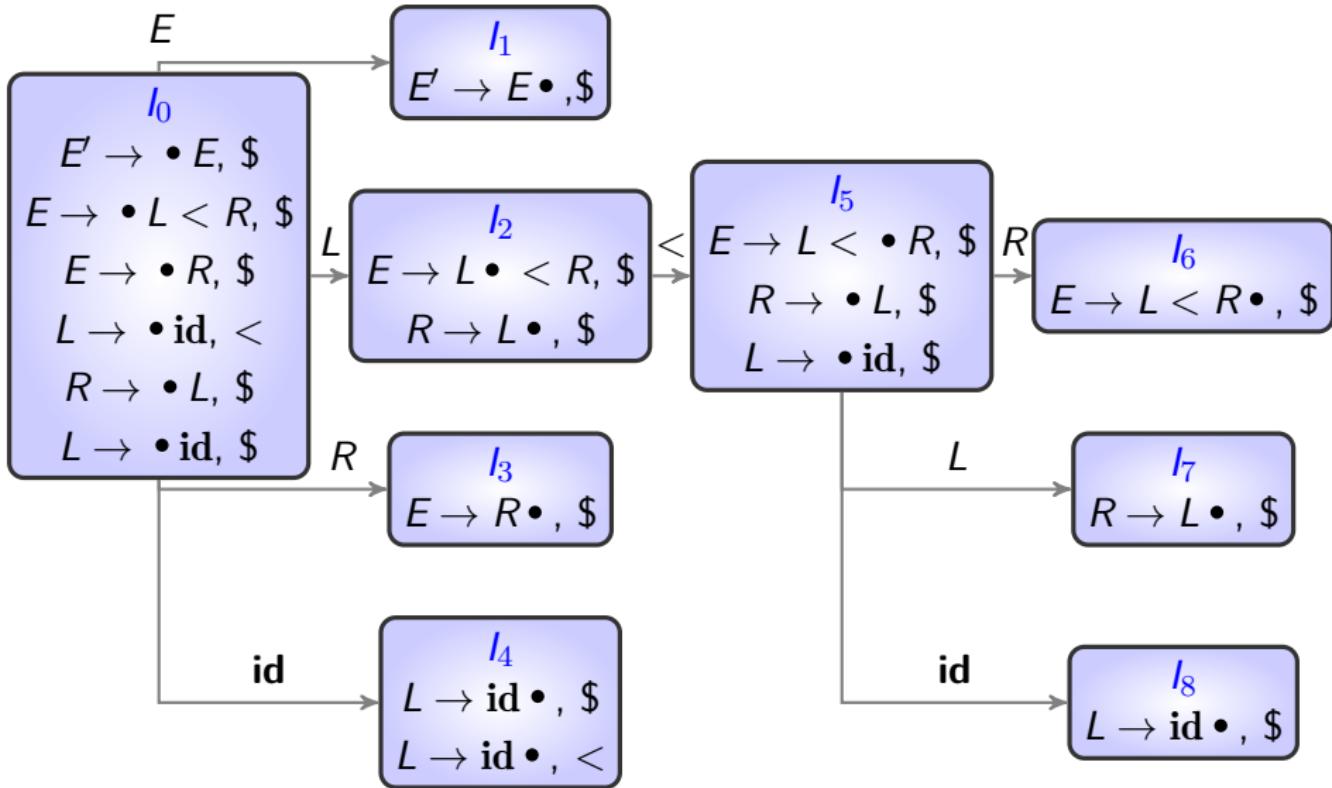
Tak więc w stosunku do algorytmu dla SLR praktycznie zmienia się nieznacznie zwinięcie reguły.



Rozpatrzmy jeszcze raz gramatykę wyrażeń:

1.  $E \rightarrow L < R$
2.  $E \rightarrow R$
3.  $L \rightarrow \mathbf{id}$
4.  $R \rightarrow L$







# Kanoniczna analiza LR — przykład

Tablica kanoniczna analizatora LR

stan	działanie			przejście		
	<	<b>id</b>	\$	<i>L</i>	<i>R</i>	<i>E</i>
0	s4			2	3	1
1		akc				
2	s5		r4			
3			r2			
4	r3		r3			
5	s8			7	6	
6			r1			
7			r4			
8			r3			

Jak widać, w stanie 2 mamy albo przesunięcie, jeśli na wejściu jest symbol <, oraz zwinięcie, jeśli mamy koniec wejścia.

Niestety analiza kanoniczna  $LR(1)$  produkuje bardzo duże tablice, nawet rząd wielkości większe niż w przypadku  $SLR(1)$ .

Rozmiar tablic można zmniejszyć. Analizator  $LALR(1)$  (*Look-Ahead LR*) ma dokładnie tyle samo stanów (tyle samo wierszy tablicy) co analizator  $SLR(1)$ . Rozpoznaje większą klasę języków niż analizator  $SLR$ , ale nieznacznie mniejszą niż kanoniczny analizator  $LR$ . Konfliktów pomiędzy przesunięciem a zwinięciem (*shift/reduce conflicts*) unika się przez dokładną analizę kontekstu podglądanego symbolu.

Są dwa podejścia do tworzenia tablic analizatora  $LALR(1)$ :

- ① podejście siłowe, w którym zaczynamy od stanów analizatora  $LR(1)$ , a następnie łączymy ze sobą zgodne stany (stany z tym samym jądrem)  
— ten sposób wymaga wiele pamięci
- ② podejście oszczędne, w którym zaczynamy od stanów z jadrami analizy  $SLR(1)$ , a potem dokładniej analizujemy kontekst w celu wyboru właściwej reguły — stosowane w praktyce



# Analiza LALR



Tworzenie tablicy w podejściu oszczędnym zaczyna się od utworzenia automatu SLR, czyli z użyciem sytuacji  $LR(0)$ , które stanowią *jądro* analizatora LALR: sytuacji początkowej  $[S' \rightarrow \bullet S, \$]$  i sytuacji, które mają kropkę w miejscu innym niż początek prawej strony. Drugi element sytuacji stanowią symbole podglądane. Wyróżniamy symbole generowane odruchowo i przepisywane. W algorytmie znajdowania symboli generowanych odruchowo i przepisywanych używany jest dodatkowy, specjalny symbol  $\#$  spoza alfabetu. Wejście stanowią jądro  $K$  zbioru  $I$  sytuacji  $LR(0)$  i symbol  $X$  gramatyki.



# Priorytet operatorów



W starszych podręcznikach konstrukcji kompilatorów znaczną część tekstu zajmują analizatory korzystające z **gramatyk operatorowych** (metoda **pierwszeństwa operatorów**). Ich użyteczność jest bardzo ograniczona. Można za to wykorzystać pierwszeństwo (priorytet) operatorów do unikania konfliktów w analizatorach LR.



Zapis dołączania dalszych części kompilatora (analizy semantycznej, generowania i optymalizacji kodu) do analizy składniowej odbywa się zwykle na jeden z dwóch sposobów. Są to:

- ① definicje sterowane składnią — bardzo ogólny sposób zapisu działań wykorzystujący tabelę dwukolumnową; w pierwszej kolumnie są reguły składniowe, w drugiej — powiązane reguły semantyczne
- ② schematy tłumaczenia — zapis, w którym reguły semantyczne umieszczane są w nawiasach klamrowych po prawej stronie składniowych reguł produkcji; mogą być umieszczane na końcu reguł lub rozbite na części umieszczane w dowolnym miejscu prawej strony produkcji; ten zapis stosowany jest w generatorze analizatorów składniowych **bison**

## definicje sterowane składnią

produkce	reguły semantyczne
$E \rightarrow C$	$E.val = C.val$
$E \rightarrow E + C$	$E.val = E_1.val + C.val$
$C \rightarrow F$	$C.val = F.val$
$C \rightarrow C * F$	$C.val = C_1.val * F.val$
$F \rightarrow (E)$	$F.val \rightarrow E.val$
$F \rightarrow \text{liczba}$	$F = \text{liczba}.val$

## schematy tłumaczenia

```
E: C { $$ = $1; };
E: E '+' C { $$ = $1 + $3; };
C: F { $$ = $1; };
C: C '*' F { $$ = $1 * $3
};
F: '(' E ')' { $$ = $2; };
F: liczba { $$ = $1; };
```



Każdej regule produkcji postaci  $A \rightarrow \alpha$  przypisujemy zbiór reguł semantycznych postaci  $b = f(c_1, c_2, \dots, c_k)$ , gdzie  $f$  jest funkcją, zaś  $b$  i  $c_i$  mogą mieć jedno z dwóch znaczeń:

- ①  $b$  jest atrybutem **syntetyzowanym** symbolu  $A$ , a  $c_1, c_2, \dots, c_k$  są atrybutami symboli z prawej strony reguły produkcji
- ②  $b$  jest atrybutem **dziedziczonym** jednego z symboli z prawej strony reguły produkcji, a  $c_1, c_2, \dots, c_k$  są atrybutami symboli z reguły produkcji

W obu przypadkach atrybut  $b$  zależy od symboli  $c_1, c_2, \dots, c_k$ . Takie reguły zapisuje się jako fragmenty programu. Drzewo wyprowadzenia z atrybutami nazywa się **drzewem dekorowanym**, a sam proces obliczania wartości atrybutów procesem **dekorowania drzewa**.

Definicję używającą tylko atrybutów syntetyzowanych nazywamy definicją **S-trybutowaną**.

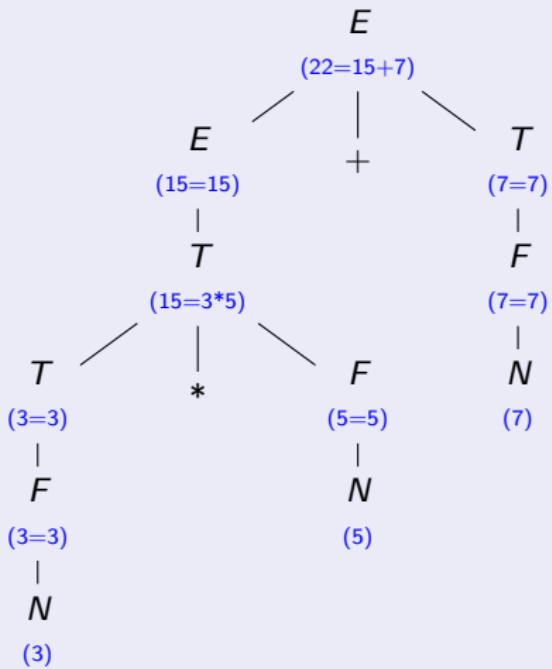
### gramatyka (bison)

E: T { \$\$ = \$1; }  
| E '+' T { \$\$ = \$1 + \$2; }  
;  
;

T: F { \$\$ = \$1; }  
| T '\*' F { \$\$ = \$1 \* \$2; }  
;

F: N { \$\$ = \$1; }  
;  
;

### drzewo wyprowadzenia



Definicję używającą tylko atrybutów syntetyzowanych nazywamy definicją **S-trybutowaną**.

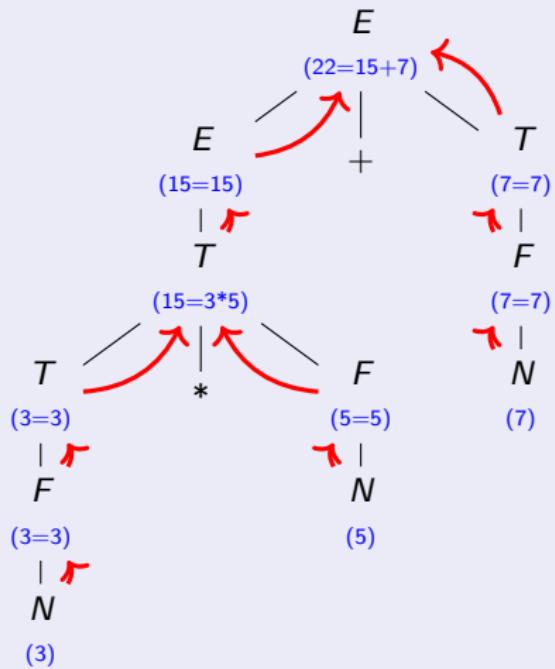
## gramatyka (bison)

E: T { \$\$ = \$1; }  
| E '+' T { \$\$ = \$1 + \$2; }  
;  
;

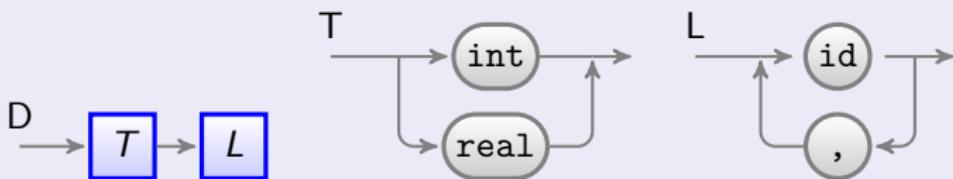
T: F { \$\$ = \$1; }  
| T '\*' F { \$\$ = \$1 \* \$2; }  
;  
;

F: N { \$\$ = \$1; }  
;  
;

## drzewo wyprowadzenia



## gramatyka



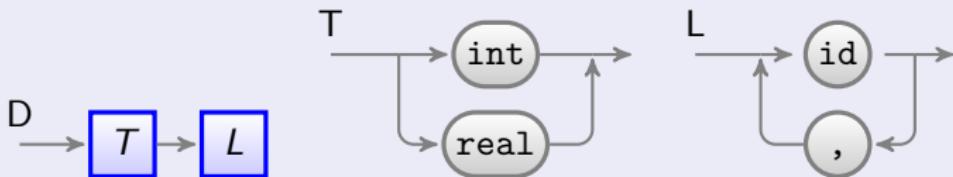
## zejścia rekurencyjne

```
procedure D;
if sym in [ int, real ] then
    t := T;
    L(t);
end if;
end D;
```

```
function T;
s := sym; sym := nextsym;
if s = int then
    return int;
elseif s = real then
    return real;
else
    error(...);
end T;
```

```
procedure L(t);
if sym = id then
    wstaw(value, t);
    sym := nextsym;
else
    error(...);
end if;
if sym = ',' then
    sym := nextsym; L(t);
end if;
end L;
```

## gramatyka



## zejścia rekurencyjne

```
procedure D;
if sym in [ int, real ] then
    t := T;
    L(t); atr. dziedziczony
end if;
end D;
```

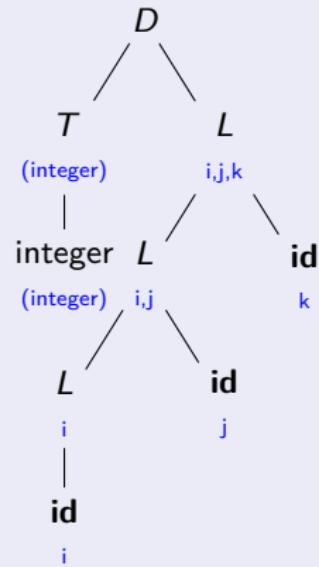
```
function T;
s := sym; sym := nextsym;
if s = int then
    return int;
elseif s = real then
    return real;
else
    error(...);
end T;
```

```
procedure L(t);
if sym = id then
    wstaw(value, t);
    sym := nextsym;
else
    error(...);
end if;
if sym = ',' then
    sym := nextsym; L(t);
end if;
end L;
```

## gramatyka

```
D: T { L.dz = T.s } L  
;  
T: int { T.s = int }  
| real { T.s = real }  
;  
L: id { id.s = L.dz }  
| { L1.dz = L.dz } L  
", "  
id { id.s = L.dz }  
;
```

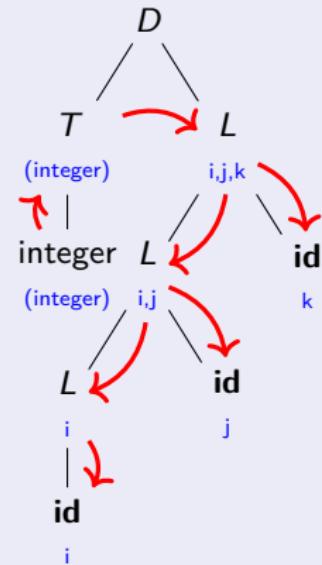
## drzewo wyprowadzenia



## gramatyka

```
D: T { L.dz = T.s } L  
;  
T: int { T.s = int }  
| real { T.s = real }  
;  
L: id { id.s = L.dz }  
| { L1.dz = L.dz } L  
", "  
id { id.s = L.dz }  
;
```

## drzewo wyprowadzenia





## Grafy zależności



Kolejność obliczania argumentów w drzewie wyprowadzenia nie jest oczywista. Dlatego tworzy się grafy zależności, w których strzałki prowadzą w kierunku kolejności obliczania atrybutów. Takie strzałki pokazano na poprzednich foliach.



## Definicja L-atrybutowana

Definicje S-atrybutowane dobrze pasują do analizy LR. Do analizy LL wykorzystuje się definicje L-atrybutowane.

Definicję nazywamy **L-atrybutowaną**, gdy każdy atrybut dziedziczony węzła  $X_j$  odpowiadającego symbolowi  $X_j$  prawej strony reguły produkcji  $A \rightarrow X_1, X_2, \dots, X_n$  zależy jedynie od:

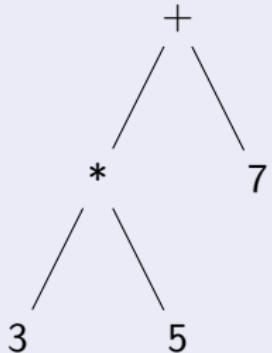
- ① atrybutów symboli  $X_1, X_2, \dots, X_{j-1}$  na lewo od  $X_j$  w tej regule produkcji
- ② atrybutów dziedziczonych  $A$

Definicje L-atrybutowane w schematach tłumaczenia mogą powodować konieczność wstawiania działań w innych miejscach niż na końcu.

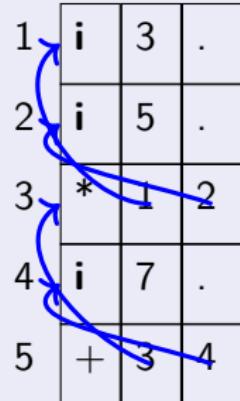
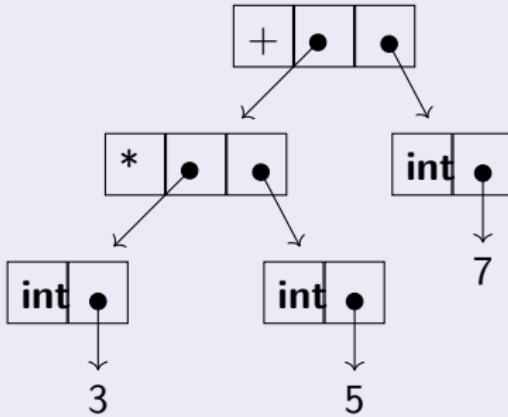
We wstępującej analizie składniowej wszystkie reguły semantyczne umieszczane są na końcu reguł składniowych. Gramatyki L-atrybutowane można przekształcić tak, by reguły semantyczne móc umieścić w całości na końcu.

- ① W każde miejsce oprócz końca reguły składniowej, gdzie wstawiona jest reguła semantyczna, wstawiana jest zmienna składniowa (znacznik).
- ② Dla każdego takiego znacznika tworzona jest reguła składniowa, w której znaczek przepisywany jest na  $\varepsilon$ .
- ③ Na końcu reguły składniowej każdego znacznika wstawiana jest żądana reguła semantyczna.

## abstrakcyjne drzewo składniowe



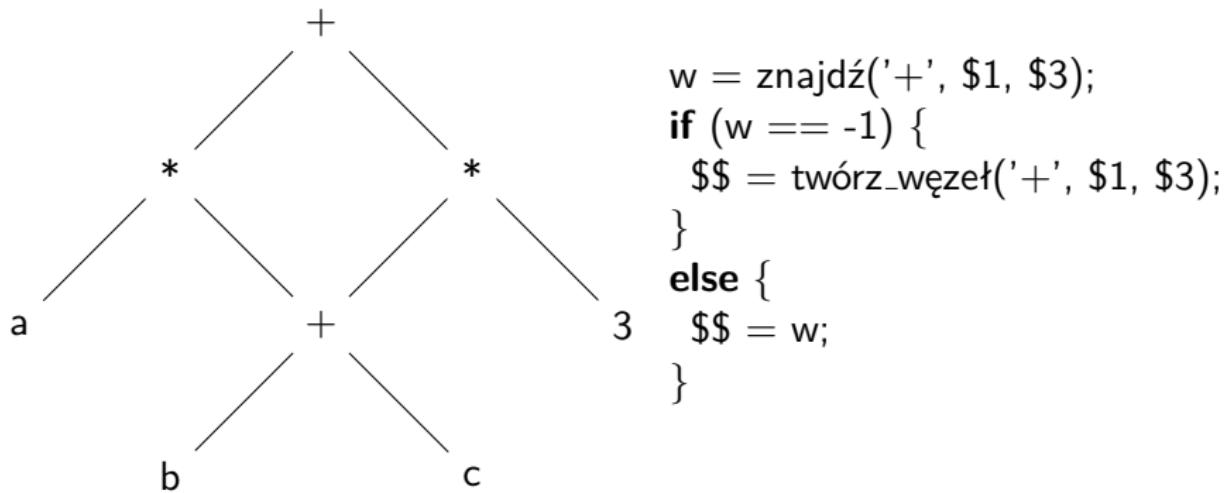
## realizacja drzewa



## Tworzenie drzewa

```
expr: number { $$ = twórz_węzeł('i', $1, nil); }
| expr '+' expr { $$ = twórz_węzeł('+', $1, $3); }
| expr '*' expr { $$ = twórz_węzeł('*', $1, $3); }
```

Wyrażenia mogą zawierać wspólne podwyrażenia. Ich wykrywanie jest częścią optymalizacji. Wówczas drzewo przestaje wystarczać, gdyż niektóre węzły mają więcej niż jednego rodzica.



Do znajdowania węzłów używamy tablicy rozproszonej.



Kompilator powinien sprawdzać reguły wykraczające poza gramatykę bezkontekstową. Na przykład:

- ① **Kontrola typów.** Kompilator powinien zgłosić błąd, jeśli typy argumentów, parametrów itp. są niewłaściwe.
- ② **Kontrola przepływu sterowania.** Kompilator powinien zgłosić błąd, jeśli instrukcja pojawi się w niewłaściwym kontekście, np. w przypadku braku instrukcji **return** w definicji funkcji zwracającej jakąś wartość.
- ③ **Kontrola niepowtarzalności.** Kompilator powinien zgłosić błąd, jeśli w tym samym zakresie widoczności deklarujemy zmienną o tej samej nazwie lub funkcję o tej samej nazwie i ew. sygnaturze (zależy od języka).
- ④ **Kontrola dopasowania nazw.** Np. w Adzie wiele konstrukcji wymaga podania tej samej nazwy na początku i na końcu.



# Kontrola typów

Kontrola typów może być realizowana **statycznie** poprzez sprawdzanie równoważności typów. Jest to procedura rekurencyjna, gdyż typy mogą być definiowane za pomocą innych typów, a takie konstrukcje mogą być dodatkowo nazywane. Niektóre typy dodatkowo mogą być uznawane za zgodne, przy czym zgodność może występować tylko w jednym kierunku.

Możliwa i stosowana jest też **dynamiczna** kontrola typów, której może towarzyszyć przekształcanie typów. Np. wartości całkowite mogą być w wyrażeniach zamieniane na wartości zmiennoprzecinkowe, jeśli w tym samym wyrażeniu występują wartości zmiennoprzecinkowe.



## typy proste

$W \rightarrow \text{liczba}$  {  $W.\text{typ} = \text{liczba}.\text{typ}$  }

## identyfikatory

$W \rightarrow \text{id}$  {  $W.\text{typ} = \text{znajd}\acute{\text{z}}(\text{id}).\text{typ}$  }

## typy złożone

$W \rightarrow W_1 [ W_2 ]$  { **if**  $W_2.\text{typ} == \text{s} \ \&\&$   
 $W_1.\text{typ} == \text{array}(\text{s},\text{t})$  **then**  $W.\text{typ} = \text{t}$   
**else**  $W.\text{typ} = \text{błąd}$  }

## instrukcje

$S \rightarrow \text{while } W \text{ do } S_1$  { **if**  $W.\text{typ} == \text{boolean} \ \&\&$   
 $S_1.\text{typ} == \text{void}$  **then**  $S.\text{type} = \text{void}$   
**else**  $W.\text{typ} = \text{błąd}$  }



Niektóre typy wartości mogą być niejawnie przekształcane przez kompilator. Na przykład w wielu językach liczby całkowite i rzeczywiste mogą być mieszane w wyrażeniach, chociaż ich reprezentacja jest zupełnie inna.

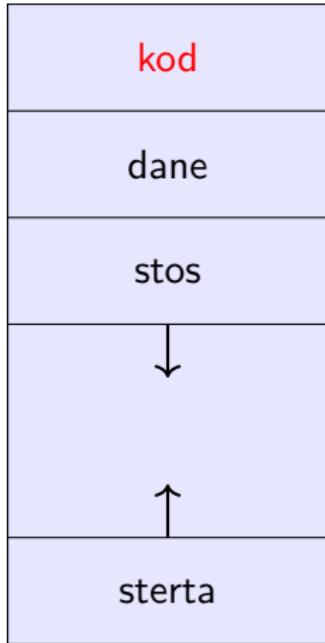
## oryginalny kod

```
double x;  
int i, j;  
x = 4 * i + 1.2 * j;
```

## kod po przekształceniu

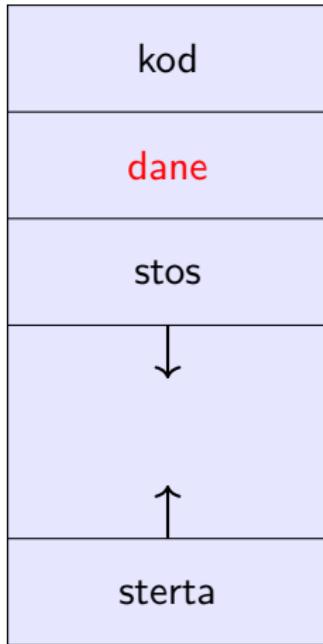
```
double x, temp1, temp2;  
int i, j;  
temp1 = int2double(4*i);  
temp2 = int2double(j);  
x = temp1 + 1.2 * temp2;
```

Kompilator powinien odpowiednio podzielić pamięć, którą program otrzyma od systemu operacyjnego w czasie wykonania.



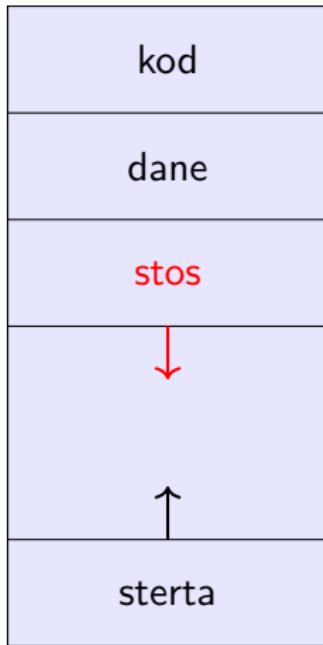
Rozmiar kodu jest znany w czasie komplikacji. Pamięć dla kodu może być przydzielona w pamięci, której zawartości nie można zmieniać.

Kompilator powinien odpowiednio podzielić pamięć, którą program otrzyma od systemu operacyjnego w czasie wykonania.



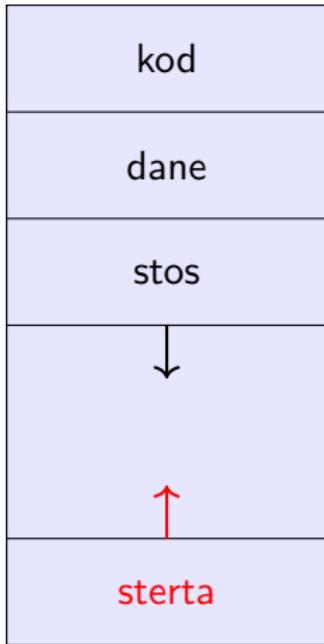
Rozmiar danych statycznych jest także znany w czasie komplikacji.

Kompilator powinien odpowiednio podzielić pamięć, którą program otrzyma od systemu operacyjnego w czasie wykonania.



Rozmiar stosu zmienia się w trakcie wykonywania programu. Na stosie umieszczane są rekordy aktywacji podprogramów i bloków, w tym dane tymczasowe. Stos rośnie w dół.

Kompilator powinien odpowiednio podzielić pamięć, którą program otrzyma od systemu operacyjnego w czasie wykonania.



Rozmiar sterty zmienia się w trakcie wykonywania programu. Sterta zawiera obiekty, dla których pamięć przydzielana jest dynamicznie. Dla niektórych języków programowania sterta może przechowywać rekordy aktywacji. Sterta rośnie w górę.



Dla różnych danych mogą być stosowane różne strategie rezerwacji pamięci.

- ① **Przydział statyczny** stosowany jest do wszystkich obiektów (w tym kodu programu) znanych w chwili kompilacji programu.
- ② **Przydział stosowy** stosowany jest do danych lokalnych i tymczasowych, których czas życia ograniczony jest do czasu życia podprogramu lub bloku w którym występują. Po zakończeniu podprogramu/bloku pamięć jest zwalniana, a jej zawartość może zostać nadpisana przez inne dane.
- ③ **Przydział stertowy** jest stosowany do obiektów, których rozmiar nie jest znany w chwili kompilacji programu lub których czas życia nie pokrywa się z czasem życia podprogramów. Istnieją różne algorytmy zarządzania stertą.

Wykonanie podprogramu (procedury lub funkcji) nie polega wyłącznie na skoku do innego fragmentu kodu z zapamiętaniem adresu powrotu. Sekwencje wywołania podprogramu i powrotu z podprogramu muszą zadbać o wiele zagadnień:

- ① przekazywanie parametrów
- ② zwracanie wartości przez funkcje
- ③ przydział pamięci i określanie adresów zmiennych lokalnych
- ④ określanie adresów zmiennych nielokalnych
- ⑤ zapamiętanie i przywracanie stanu procesora

Podział tych zadań pomiędzy kod wywołujący a wołany podprogram może być różny. Zależy od architektury komputera. Mniejszy kod uzyskuje się na ogół przez umieszczenie większej części zadań w podprogramie wywoływanym.

<b>zwracana wartość</b>
parametry aktualne
wiązanie sterowania
wiązanie dostępu
stan procesora
dane lokalne
dane tymczasowe

Pole przeznaczone na wynik funkcji. Zwracana wartość jest jednak zwykle przekazywana przez rejestr, co jest szybsze.



# Rekordy aktywacji



zwracana wartość
parametry aktualne
wiązanie sterowania
wiązanie dostępu
stan procesora
dane lokalne
dane tymczasowe

Pole przeznaczone na parametry aktualne. Przynajmniej niektóre typy wartości mogą być także wydajniej przekazywane w rejestrach.

zwracana wartość
parametry aktualne
<b>wiązanie sterowania</b>
wiązanie dostępu
stan procesora
dane lokalne
dane tymczasowe

Wiązanie sterowania wskazuje rekord aktywacji procedury wywołującej. Pole to nie jest obowiązkowe. W niektórych językach kompilator może obliczyć rozmiar całego rekordu aktywacji.

zwracana wartość
parametry aktualne
wiązanie sterowania
<b>wiązanie dostępu</b>
stan procesora
dane lokalne
dane tymczasowe

Wiązanie dostępu jest używane do odwoływania się do danych nielokalnych umieszczonych w rekordach aktywacji innych podprogramów. W niektórych językach, np. w Fortranie, nie jest potrzebne. Może być zastąpione przez tablice Display.

zwracana wartość
parametry aktualne
wiązanie sterowania
wiązanie dostępu
<b>stan procesora</b>
dane lokalne
dane tymczasowe

Stan procesora obejmuje zawartość rejestrów procesora, w tym licznika rozkazów i wskaźnika stosu. Dane te powinny być przywrócone po powrocie z podprogramu.

zwracana wartość
parametry aktualne
wiązanie sterowania
wiązanie dostępu
stan procesora
<b>dane lokalne</b>
dane tymczasowe

Dane lokalne zawierają dane deklarowane wewnątrz podprogramu. Dane mogą być przechowywane z uwzględnieniem wyrównania do pełnych słów itp.



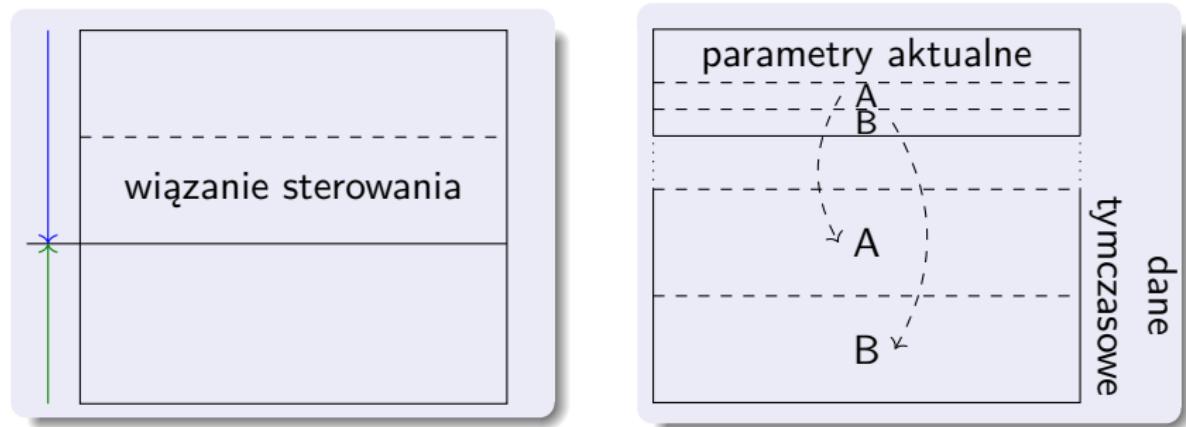
# Rekordy aktywacji

zwracana wartość
parametry aktualne
wiązanie sterowania
wiązanie dostępu
stan procesora
dane lokalne
dane tymczasowe

Stos może być wykorzystywany do przechowywania tymczasowych wyników obliczeń, np. przy zastosowaniu odwrotnej notacji polskiej.

Tworzenie rekordu aktywacji dzielone jest między kodem w miejscu wywołania, a wywoływaną procedurą.

- Im więcej wykonuje procedura wywoływana, tym bardziej zwarty kod.
- Podział zaraz za wiązaniem sterowania umożliwia m.in. przekazywanie jako parametrów aktualnych tablic zmiennej długości. W polu parametrów aktualnych przechowuje się tylko wskaźniki do tablic, których zawartość jest w danych tymczasowych.





# Zakres widoczności nazw

W programie mogą być deklarowane różne nazwy o różnym zasięgu.

```
program p;
var a, b, d: integer;

procedure q;
var a, b, c;
begin
end;

procedure x;
var a, c: integer;

procedure y;
var b, c: integer;
begin
    a := 0; b := 1; c := 2; d = 3
end;
begin
    y; a := 4; b := 5; c := 6
end;
begin
    x; a := 7; b := 8; c := 9
end.
```



# Zakres widoczności nazw

W programie mogą być deklarowane różne nazwy o różnym zasięgu.

```
program p;
var a, b, d: integer;

procedure q;
var a, b, c;
begin
end;

procedure x;
var a, c: integer;

procedure y;
var b, c: integer;
begin
    a := 0; b := 1; c := 2; d = 3
end;
begin
    y; a := 4; b := 5; c := 6
end;
begin
    x; a := 7; b := 8; c := 9 ← zmieniącą się jest zadeklarowana!
end.
```



# Dostęp do nazw nielokalnych

Dostęp do danych nielokalnych w przypadku podprogramów zagnieżdżonych realizuje się przez **wiązanie dostępu**. Wskazuje ono rekord aktywacji **ostatniego wywołania** podprogramu otaczającego dany podprogram. Jeśli procedura  $p$  z poziomu  $n_p$  wywołuje procedurę  $q$  z poziomu  $n_q$  (wiadomo, że  $|n_p - n_q| \leq 1$ ):

- $n_p < n_q$  ( $q$  jest procedurą bezpośrednio zagnieżdżoną w  $p$ ): wiązanie dostępu w  $q$  wskazuje na wiązanie dostępu w  $p$ .
- $n_p \geq n_q$  ( $p$  jest procedura zagnieżdżoną w  $q$  lub obie procedury są na tym samym poziomie, w szczególności  $p$  jest  $q$ ): wiązanie dostępu w  $q$  wskazuje na wiązanie dostępu procedury, w której zagnieżdżone jest  $p$ .

## poziom zagnieżdżenia

Dane globalne są na poziomie 0. Procedury niezagnieżdżone w innych są na poziomie 1. Procedura  $q$  bezpośrednio zagnieżdżona w procedurze  $p$  ma poziom zagnieżdżenia o jeden większy niż procedura  $p$ :  $n_q = n_p + 1$ . procedura

## struktura programu

 $p (a,b,d)$  $q (a,b,c)$  $x (a,c)$  $y (b,c)$ 

## wywołania 1

 $p \rightarrow x \rightarrow x \rightarrow y$ 

## wywołania 2

 $p \rightarrow x \rightarrow y \rightarrow x$ 

## rekordy aktywacji 1

p
wiązanie dostępu
a, b, d
x
wiązanie dostępu
a, c
x
wiązanie dostępu
a, c
y
wiązanie dostępu
b, c
x
wiązanie dostępu
a, c

## rekordy aktywacji 2

p
wiązanie dostępu
a, b, d
x
wiązanie dostępu
a, c
y
wiązanie dostępu
b, c
x
wiązanie dostępu
a, c



## Dostęp do nazw

Dostęp do zmiennych realizowany jest:

- **w przypadku nazw globalnych** — przez umieszczenie w kodzie bezpośrednio ich adresów (znanych w trakcie kompilacji);
- **w przypadku nazw zdefiniowanych w otaczających procedurach/funkcjach** — jeśli zmienna zadeklarowana w procedurze  $q$  na poziomie  $n_q$  ma zostać użyta w procedurze  $p$  na poziomie  $n_p$ ,  $n_p > n_q$ , to należy przejść po  $n_p - n_q$  wiązaniach dostępu i dalej dostęp jak do zmiennej lokalnej  $q$ ;
- **w przypadku nazw lokalnych** — adres jest przesunięciem na stosie względem wiązania sterowania.

### Przykład

Aby z procedury  $y$  (poziom 3) wywołanej z procedury  $x$  wywołanej z programu  $p$  (poziom 1) dostać się do zmiennej  $d$  zadeklarowanej w programie  $p$  należy przejść po dwóch wiązaniach dostępu (z  $y$  do  $x$  i z  $x$  do  $p$ ), a następnie dodać przesunięcie do zmiennej  $d$ .



# Dostęp do nazw nielokalnych

## struktura programu

$p(a,b,d)$

$q(a,b,c)$

$x(a,c)$

$y(b,c)$

## wywołania 1

$p \rightarrow x \rightarrow x \rightarrow y$

## wywołania 2

$p \rightarrow x \rightarrow y \rightarrow x$

## rekordy aktywacji 1

$p$
wiązanie dostępu
a, b, d
x
wiązanie dostępu
a, c
x
wiązanie dostępu
a, c
y
wiązanie dostępu
b, c

Aby z procedury  $y$  (poziom 3) wywołanej z procedury  $x$  wywołanej z programu  $p$  (poziom 1) dostać się do zmiennej  $d$  zadeklarowanej w programie  $p$  należy przejść po dwóch wiązaniach dostępu (z  $y$  do  $x$  i z  $x$  do  $p$ ), a następnie dodać przesunięcie do zmiennej  $d$ .



# Dostęp do nazw nielokalnych

## struktura programu

$p(a,b,d)$

$q(a,b,c)$

$x(a,c)$

$y(b,c)$

## wywołania 1

$p \rightarrow x \rightarrow x \rightarrow y$

## wywołania 2

$p \rightarrow x \rightarrow y \rightarrow x$

## rekordy aktywacji 2

$p$
wiązanie dostępu
$a, b, d$
$x$
wiązanie dostępu
$a, c$
$y$
wiązanie dostępu
$b, c$
$x$
wiązanie dostępu
$a, c$

Aby z procedury  $y$  (poziom 2) wywołanej z procedury  $x$  wywołanej z programu  $p$  (poziom 1) dostać się do zmiennej  $d$  zadeklarowanej w programie  $p$  należy przejść po dwóch wiązaniach dostępu (z  $y$  do  $x$  i z  $x$  do  $p$ ), a następnie dodać przesunięcie do zmiennej  $d$ .



## Tablice display



Zamiast chodzenia po wiązaniach dostępu można korzystać z tablic **display**. Tablice te zawierają wskaźniki do rekordów aktywacji ostatnich wywołań kolejnych otaczających podprogramów. Dla procedury  $p$  z poziomu  $n_p$  tablica *display* zawiera  $n_p - 1$  wskaźników.

Jeśli procedury nie są parametrami innych procedur, to można utrzymywać jedną tablicę *display* rosnącą w miarę wzrostu zagnieżdżenia wywoływanych procedur.

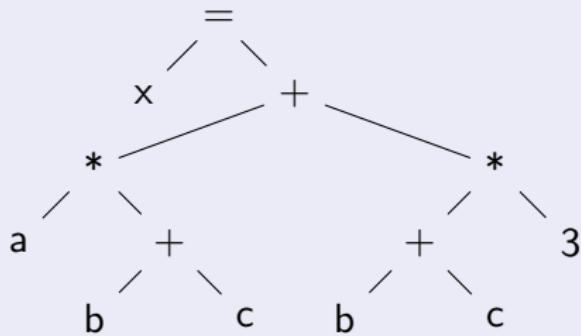


Kompilator na ogół nie tłumaczy bezpośrednio na kod wynikowy, a tłumaczy na kod pośredni. Kod pośredni jest niezależny od maszyny docelowej. Im później w kompilatorze pojawia się kod wynikowy, tym łatwiej jest zmienić maszynę docelową (stworzyć kompilator na nową maszynę na podstawie wcześniejszego kompilatora).

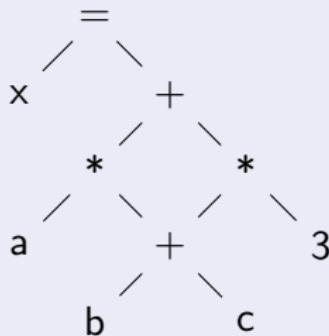
Jako reprezentację pośrednią wykorzystuje się:

- drzewa składniowe
- zapis przyrostkowy (odwrotna notacja polska)
- kod trójadresowy

## drzewo składniowe



## dag



## program

```
ASSIGNMENT: IDENT '=' EXPR { $$ = node("-", leaf($1), $3); }  
;  
EXPR: IDENT { $$ = leaf($1); }  
| NUMBER { $$ = leaf($1); }  
| EXPR '+' EXPR { $$ = node("+", $1, $3); }  
| EXPR '*' EXPR { $$ = node("*", $1, $3); }  
| '(' EXPR ')' { $$ = $2; } ;
```



## Zapis wrostkowy (tradycyjny)

$$x = a * (b + c) + (b + c) * 3$$

## zapis przyrostkowy (ONP)

$$x \ a \ b \ c \ + \ * \ b \ c \ + \ 3 \ * \ + \ =$$

## Tłumaczenie z zapisu tradycyjnego na ONP

ASSIGNMENT: IDENT { printf("%s",\$1); } '=' EXPR { printf(" ="); }

;

EXPR: IDENT { printf(" %s",\$1); }  
| NUMBER { printf(" %d",\$1); }  
| EXPR '+' EXPR { printf("+"); }  
| EXPR '\*' EXPR { printf("\*"); }  
| '(' EXPR ')' { } ;

Odwrotna notacja polska odpowiada obliczeniom na stosie. Najpierw odkładamy na stos argumenty, potem wykonujemy operację na elementach na szczytce stosu (na najwyższym elemencie, dwóch najwyższych, wyjątkowo na trzech najwyższych).



Kod trójadresowy upraszcza skomplikowane wyrażenia przez ograniczenie ich do co najwyżej trzech argumentów.

## katalog instrukcji trójadresowych

$x = y \ op \ z$  — przypisanie z operatorem dwuargumentowym

$x = op \ y$  — przypisanie z argumentem jednoargumentowym

$x = y$  — przypisanie

goto L — skok bezwarunkowy

if  $x \ oprel \ y$  goto L — skok warunkowy

param x — parametr wywołania podprogramu (przed call)

call p, n — wywołanie procedury z n parametrami

return y — zwrócenie wartości y w funkcji

$x = y[i]$ ,  $x[i] = y$  — przypisania indeksowane

$x = &y$ ,  $x = *y$ ,  $*x = y$  — przypisania adresowe i wskaźnikowe

$x = typ2typ \ y$  — przekształcenie typu (np. int2real).

W kompilatorze kod trójadresowy jest przechowywany w postaci czwórek, trójek lub trójek pośrednich.

czwórki

nr	op	arg1	arg2	wynik
[0]	+	b	c	$t_1$
[1]	*	a	$t_1$	$t_2$
[2]	*	$t_1$	3	$t_3$
[3]	+	$t_2$	$t_3$	$t_4$
[4]	=	$t_4$		x

trójki

nr	op	arg1	arg2
[0]	+	b	c
[1]	*	a	(0)
[2]	*	(0)	3
[3]	+	(1)	(2)
[4]	=	x	(3)

W trójkach zmienne tymczasowe  $t_i$  zostają zastąpione odwołaniami do numeru instrukcji, którego są wynikiem. Czwórki pozwalają na zamianę kolejności instrukcji bez potrzeby zmiany odwołań do ich numerów. Trójki pośrednie mają dodatkową tablicę kolejności instrukcji, która usuwa tę niedogodność zwykłych trójek.



Nazwy napotykane w czasie kompilacji przechowywane są wraz z opisującymi je informacjami w jednej lub większej liczbie tablicy symboli.

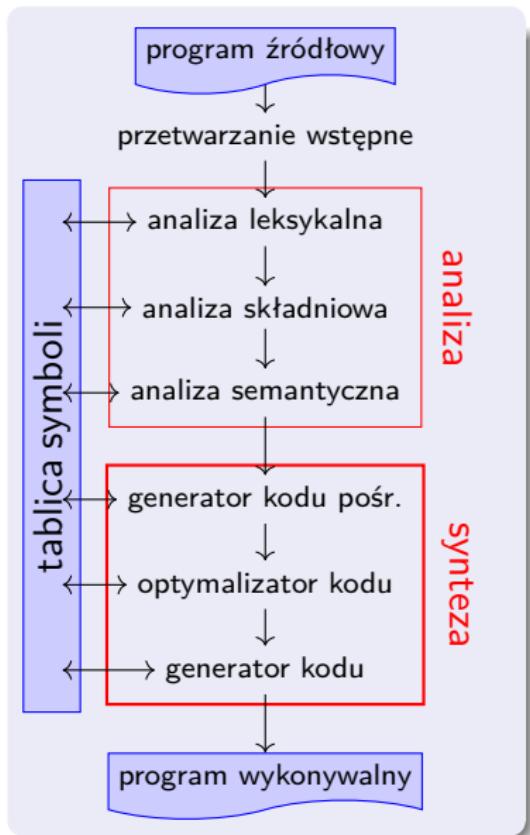
- Sam tekst symbolu (nazwy zmiennej długości) jest zazwyczaj przechowywany poza tablicą, a tablicy jest tylko odsyłacz.
- Tablica zawiera też информацию o typie symbolu, o rozmiarze i o adresie związanego z symbolem obiektu.
- Ze względu na szybkość dostępu tablica jest na ogół realizowana jako tablica mieszająca.
- Wpisy w tablicy symboli mogą być tworzone już przez analizator leksykalny. Są uzupełniane w dalszych fazach.



# Tablica symboli

Liczba tablic symboli zależy od języka.

- Dla języków z zagnieżdżaniem procedur dla procedur tworzone są oddzielne tablice. Każda tablica zawiera wskaźnik do procedury składniowo ją obejmującej. Po napotkaniu końca procedury tablica jest usuwana.
- Rekordy (struktury, klasy) mają wpisy w ogólnej tablicy symboli zawierające wskaźnik na tablicę symboli dla pól (o analogicznej strukturze). Jeśli tablica zawierająca wpis dla rekordu jest usuwana, usuwana jest także tablica nazw pól.
- Dla niektórych instrukcji i wyrażeń tworzone są etykiety tymczasowe. Ważne są tylko wewnątrz danych konstrukcji i mogą być usunięte natychmiast po ich ukończeniu. Można je przechowywać w dodatkowych tablicach.



## Synteza

**generator kodu pośr.**

**optymizator kodu**

analiza przepływu sterowania

analiza przepływu danych

przekształcenia

**generator kodu**



- ① Optymalizacja nie może zmieniać znaczenia programu.
- ② Przekształcenie musi przyspieszać kod. Może zmniejszać objętość generowanego kodu.
- ③ Optymalizacja musi być opłacałna.

Optymalizacja wykonywana przez kompilator nie jest jedyną możliwą. Także programista może wpływać na szybkość wykonywania programu nie tylko przez wybór odpowiednich algorytmów, ale także przez deklarowanie zmiennych, które powinny być przechowywane w rejestrach (w niektórych językach), przenoszenie kodu poza pętle, upraszczanie wyrażeń itp.



Kod wynikowy produkowany przez kompilator ma zwykle jedną z trzech postaci:

- ① **kod maszynowy z adresami bezwzględnymi** — taki kod musi być umieszczony pod określonym adresem w pamięci;
- ② **relokowalny kod maszynowy** — taki kod może być przemieszczany w pamięci (np. z użyciem tablicy adresów, które należy zmienić przy przemieszczeniu);
- ③ **kod w asemblerze** — instrukcje asemblera są w postaci symbolicznej, ale bliskiej maszyny, a adresy określone są za pomocą etykiet.

Przy generowaniu kodu wynikowego należy rozwiązać kilka problemów:

- **zarządzanie pamięcią** — generowany kod może odwoływać się do przyszłych instrukcji, których adresy są nieznane; może to powodować konieczność późniejszego poprawiania takich odwołań;
- **wybór rozkazów** — zwykle dane wyrażenie może być zrealizowane na różne sposoby z zastosowaniem różnych rozkazów; rozkazy różnią się czasem wykonania;
- **zarządzanie rejestrami** — krótszy i szybszy kod używa rejestrów procesora, ale użycie rejestrów podlega ograniczeniom.



# Kod wynikowy

Standardowo operacje  $x := y \ op \ z$  realizujemy za pomocą trójki rozkazów:

MOV R0,y

$op$  R0,z

MOV x,R0

Nie zawsze tak wygenerowany kod jest optymalny:

## trójadresowy

$a = b + c$

$e = a - f$

## rozkazy

MOV R0,B

ADD R0,C

MOV A,RO

MOV RO,A

SUB RO,F

MOV E,RO

## rozkazy poprawione

MOV R0,B

ADD R0,C

MOV A,RO

SUB RO,F

MOV E,RO



## blok bazowy

Blok bazowy jest ciągiem kolejnych instrukcji, do którego sterowanie wchodzi na początku i wychodzi na końcu, bez zatrzymywania ani możliwości rozgałęzienia przed końcem.

## pierwsza instrukcja bloku

- pierwsza instrukcja jest pierwszą instrukcją bloku
- każda instrukcja, która jest celem skoku jest pierwszą instrukcją bloku
- każda instrukcja następująca bezpośrednio po skoku lub po wywołaniu procedury jest pierwszą instrukcją bloku

Każdy blok składa się z pierwszej instrukcji bloku i instrukcji po nim następujących aż do pierwszej instrukcji następnego bloku wyłącznie lub do końca programu.

Dag (ang. *directed acyclic graph*) to skierowany graf acykliczny. Użyty do reprezentacji bloków bazowych pokazuje, jak wartość obliczana przez każdą instrukcję w bloku bazowym używana jest przez następne instrukcje w bloku.

### dag dla bloku bazowego

- ① Liście są etykietowane niepowtarzalnymi identyfikatorami, które są nazwami zmiennych lub stałych. Liście reprezentują wartości początkowe nazw. Indeksowane są liczbą 0 w celu odróżnienia od aktualnych wartości nazw. Na podstawie operatora określa się, czy potrzebna jest l-wartość czy r-wartość.
- ② Wierzchołki wewnętrzne są etykietowane symbolem operatora.
- ③ Wierzchołki wewnętrzne mogą również mieć etykietę będącą ciągiem identyfikatorów. Takie wierzchołki reprezentują wartości odpowiadające tym etykietom.



## Budowa daga dla bloków bazowych



Algorytm wykorzystuje funkcję *wierzchołek*(*identyfikator*), która zwraca wierzchołek w dagu reprezentujący aktualną wartość identyfikatora. Początkowo jest niezdefiniowana dla wszystkich argumentów. Przyjmijmy instrukcje postaci:

- $x = y \ op \ z$
- $x = op \ y$
- $x = y$

Inne instrukcje obsługuje się analogicznie.



## algorytm

Powtarzaj kroki 1 do 3 po kolej i dla każdej instrukcji z bloku.

- ① Jeśli wartość  $wierzchołek(y)$  jest niezdefiniowana, stwórz liść o etykiecie  $y$  i niech  $wierzchołek(y)$  będzie tym wierzchołkiem. Zrób to samo dla  $z$ , o ile występuje w instrukcji.
- ② Gdy instrukcja zawiera  $z$ , sprawdź, czy istnieje wierzchołek  $n$  o etykiecie  $op$ , którego lewym dzieckiem jest  $wierzchołek(y)$ , a prawym —  $wierzchołek(z)$ . Jeśli nie, stwórz wierzchołek  $n$ . Gdy instrukcja jest postaci  $x = op y$ , sprawdź, czy istnieje wierzchołek  $n$  o etykiecie  $op$  i jedynym dziecku  $wierzchołek(y)$ . Jeśli nie, to stwórz taki wierzchołek. Dla instrukcji postaci  $x = y$  niech  $n$  będzie wartością  $wierzchołek(y)$ .
- ③ Usuń  $x$  z listy identyfikatorów dołączonych do  $wierzchołek(x)$ . Dodaj  $x$  do listy identyfikatorów dla wierzchołka  $n$  i nadaj  $wierzchołek(x)$  wartość  $n$ .



Dagi pozwalają uzyskać szereg cennych informacji:

- ① wykrywają wspólne podwyrażenia,
- ② pokazują, które identyfikatory są używane w danym bloku,
- ③ pokazują, które instrukcje obliczają wartości, które mogą być używane poza blokiem — są to instrukcje związane z wierzchołkami zwracanymi przez *wierzchołek(x)* dla jakiegoś identyfikatora *x*,
- ④ pozwalają usuwać zbędne przypisania  $x = y$  — jeśli wierzchołek ma więcej niż jeden identyfikator na liście, sprawdza się, czy i które z tych identyfikatorów są używane poza blokiem.



# Grafy przepływu

Graf przepływu modeluje przepływ sterowania w programie.

- Węzłami są bloki, krawędzie pokazują przepływ sterowania.
- Węzłem początkowym jest blok zawierający pierwszą instrukcję programu.
- W grafie istnieje krawędź z bloku  $B_1$  do bloku  $B_2$ , jeżeli  $B_2$  może wystąpić bezpośrednio po  $B_1$  w pewnym ciągu wykonania. Innymi słowy, blok  $B_2$  jest *następnikiem*  $B_1$ , gdy:
  - ▶ istnieje skok od ostatniej instrukcji  $B_1$  do pierwszej instrukcji  $B_2$  lub
  - ▶  $B_2$  jest bezpośrednio po  $B_1$  w porządku programu i  $B_1$  nie kończy się bezwarunkowym skokiem.

## reprezentacja węzłów grafu przepływu

Węzeł grafu przepływu jest rekordem zawierającym:

- ① liczbę czwórek w bloku,
- ② wskaźnik do pierwszej instrukcji bloku,
- ③ listę poprzedników bloku,
- ④ listę następców bloku.



# Analiza przepływu danych

Analiza przepływu danych może np. pomóc w zwijaniu stałych czy eliminacji kodu martwego oraz do poprawy przydziału rejestrów. Przepływ może następować wprzód lub wstecz, a więc:

$$\text{Out}[s] = f_s(\text{In}[s]) \quad \text{lub} \quad \text{In}[s] = f_s(\text{Out}[s])$$

gdzie:

- $s$  to instrukcja
- $\text{In}[s]$  to informacje o przepływie na początku  $s$
- $\text{Out}[s]$  to informacje o przepływie na końcu  $s$
- $f_s$  to funkcja przekazania informacji

Wewnątrz bloku, dla sąsiednich instrukcji  $s_i$  i  $s_{i+1}$ :

$$\text{Out}[s_i] = \text{In}[s_{i+1}]$$



# Analiza przepływu danych

Funkcja przekazania informacji  $f_B$  bloku  $B$  o instrukcjach  $s_1, \dots, s_n$  to:

$$f_B = f_{s_1} \circ \dots \circ f_{s_n}, \quad \text{In}[B] = \text{In}[s_1], \quad \text{Out}[B] = \text{Out}[s_n]$$

i mamy jedną z możliwości w zależności od kierunku przepływu:

$$\text{Out}[B] = f_B(\text{In}[B]) \quad \text{lub} \quad \text{In}[B] = f_B(\text{Out}[B])$$

Dla przepływu wprzód, dla bloków  $P$  poprzedzających blok  $B$ :

$$\text{In}[B] = \bigcup_{P \prec B} \text{Out}[P]$$

Dla przepływu wstecz, dla bloków  $S$  następujących po bloku  $B$ :

$$\text{Out}[B] = \bigcup_{S \succ B} \text{In}[S]$$



## definicja osiągająca

Mówimy, że definicja  $d$  zmiennej  $x$  osiąga punkt  $p$ , jeżeli istnieje ścieżka z punktu bezpośrednio następującego po  $d$  wolna od dowolnej innej definicji zmiennej  $x$ .

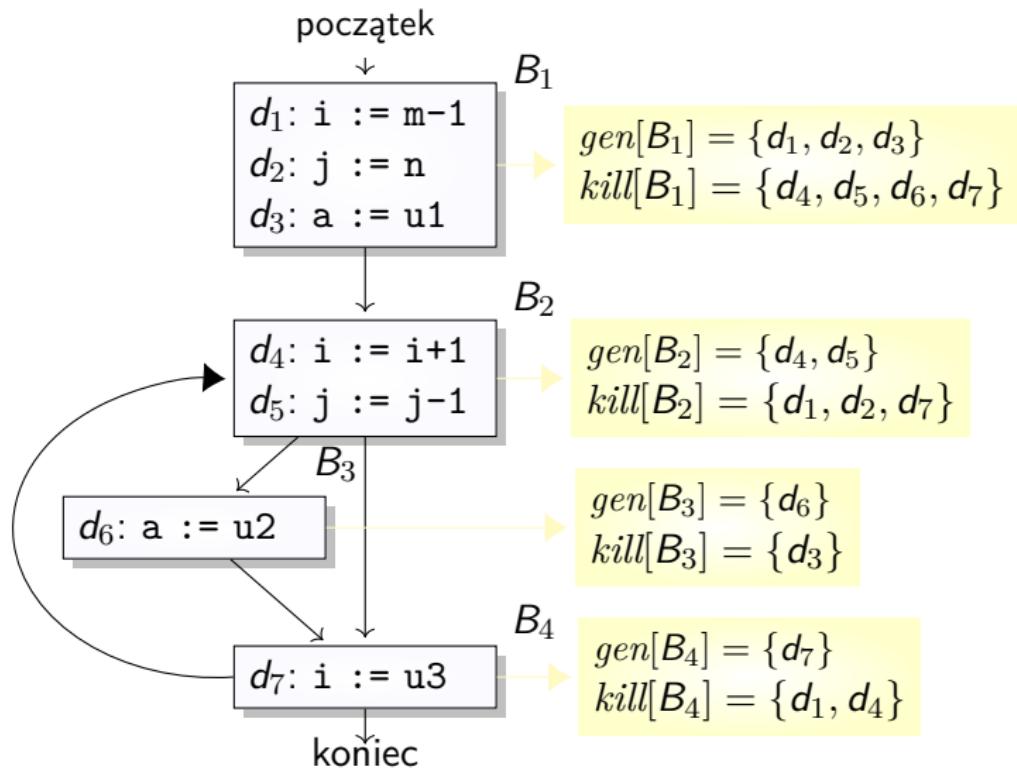
Definicją zmiennej  $x$  jest instrukcja, która *może* przypisywać wartość zmiennej  $x$ . Nie zawsze można ustalić, czy rzeczywiście to robi. Parametry procedur, dostęp do zmiennych indeksowanych czy adresowanie pośrednie czynią taką analizę bardzo trudną. Wówczas w przetwarzaniu podejmujemy taką decyzję, która na pewno nie zmieni znaczenia programu po optymalizacji.

Funkcja przekazania ma najczęściej postać:

$$f_s(y) = \text{gen}[s] \cup (y[s] - \text{kill}[s])$$

gdzie  $y$  jest równe In lub Out w zależności od kierunku przepływu informacji,  $\text{gen}[s]$  to informacje tworzone przez  $s$ , a  $\text{kill}[s]$  informacje niszczone przez  $s$ .

Przykład:





# Analiza przepływu danych



Równania przepływu danych rozwiązuje się rekurencyjnie stosując wektory bitów jako reprezentację zbiorów In i Out.

## wykrywanie użycia zmiennych bez wartości początkowej

Dodaje się fikcyjne definicje dla każdej zmiennej  $x$  na wejściu grafu. Jeżeli taka definicja osiąga punkt  $p$ , w którym zmienność  $x$  może być użyta, może to być przypadek użycia zmiennej przed ustaleniem jej wartości początkowej.

## zwijanie stałych

Jeżeli użycie zmiennej  $x$  jest osiągalne tylko przez jedną definicję i ta definicja przypisuje zmiennej  $x$  stałą wartość, możemy w tym punkcie zastąpić zmienną  $x$  tą stałą. Do analizy potrzebujemy tylko informacji, czy jest to stała, do czego wystarcza 1 bit. Dodatkowo jeśli stała występuje w instrukcjach sterujących, np.:

**if debug then**

wtedy możemy dokonać eliminacji martwego kodu.



## Definicja nazwy

**Definicją** nazwy w instrukcjach trójadresowych jest instrukcja, w której tej nazwie nadawana jest wartość.

## Użycie nazwy

**Użyciem** nazwy w instrukcjach trójadresowych jest instrukcja, w której wartość nazwy bierze udział w obliczeniach.

## Żywa nazwa

Nazwa w bloku bazowym jest **żywa** w danym punkcie, jeśli jej wartość jest używana za tym punktem w programie, być może w innym bloku bazowym.

Jeżeli zmienna, której nadajemy wartość w danej instrukcji, nie jest żywa, to tę instrukcję można usunąć bez wpływu na działanie programu.



Dla potrzeb badania żywotności przyjmujemy, że wywołanie podprogramu (procedury czy funkcji) zaczyna nowy blok bazowy ze względu na możliwe efekty uboczne. Przeglądamy instrukcje trójadresowe od tyłu bloku.

Dla instrukcji postaci  $x := y \ op \ z$

- ① Dołączamy do instrukcji informacje aktualnie znajdujące się w tablicy symboli dotyczące następnego użycia i żywotności  $x$ ,  $y$  i  $z$ .
- ② W tablicy symboli nadajemy zmiennej  $x$  atrybuty *martwa* i *nie ma dalszych użyci*.
- ③ W tablicy symboli zapisujemy, że  $y$  i  $z$  są żywe i ich następnym użyciem jest bieżąca instrukcja.

Zakładamy, że na końcu bloku wszystkie zmienne są żywe. Możemy też analizować żywotność zmiennych między blokami — wtedy na końcu bloku żywotne są zmienne żywe w dowolnym następcu danego bloku.



## Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

(4)  $b := d + f$

(5)  $e := a - c$

(6)  $b := d + c$



## Obliczanie żywotności zmiennych w bloku



Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

(4)  $b := d + f$

(5)  $e := a - c$

(6)  $b := d + c$

na końcu wszystkie zmienne są żywe



# Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

(4)  $b := d + f$

(5)  $e := a - c$

(6)  $b := d + c$       b jest martwa i nie ma dalszych użyc

na końcu wszystkie zmienne są żywe



Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

(4)  $b := d + f$

(5)  $e := a - c$

(6)  $b := d + c$

b jest martwa i nie ma dalszych użyć  
d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



# Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

(4)  $b := d + f$

(5)  $e := a - c$       e jest martwa i nie ma dalszych użyć

(6)  $b := d + c$       b jest martwa i nie ma dalszych użyć  
d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



# Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

(4)  $b := d + f$

(5)  $e := a - c$

e jest martwa i nie ma dalszych użyć  
a i c są żywe, użyte w (5)

(6)  $b := d + c$

b jest martwa i nie ma dalszych użyć  
d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



## Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

(4)  $b := d + f$       b jest martwa i nie ma dalszych użyć

(5)  $e := a - c$       e jest martwa i nie ma dalszych użyć  
a i c są żywe, użyte w (5)

(6)  $b := d + c$       b jest martwa i nie ma dalszych użyć  
d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

(4)  $b := d + f$

b jest martwa i nie ma dalszych użyć  
d i f są żywe, użyte w (4)

$$(5) \quad e := a - c$$

e jest martwa i nie ma dalszych użytków, a i c są żywe, użyte w (5)

(6)  $b := d + c$

b jest martwa i nie ma dalszych użyć  
d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



## Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$       e jest martwa i nie ma dalszych użyć

(4)  $b := d + f$       b jest martwa i nie ma dalszych użyć  
d i f są żywe, użyte w (4)

(5)  $e := a - c$       e jest martwa i nie ma dalszych użyć  
a i c są żywe, użyte w (5)

(6)  $b := d + c$       b jest martwa i nie ma dalszych użyć  
d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



## Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$

(3)  $e := a + f$

e jest martwa i nie ma dalszych użyć  
a i f są żywe, użyte w (3)

(4)  $b := d + f$

b jest martwa i nie ma dalszych użyć  
d i f są żywe, użyte w (4)

(5)  $e := a - c$

e jest martwa i nie ma dalszych użyć  
a i c są żywe, użyte w (5)

(6)  $b := d + c$

b jest martwa i nie ma dalszych użyć  
d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



# Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$        $d$  jest martwa i nie ma dalszych użyć

(3)  $e := a + f$        $e$  jest martwa i nie ma dalszych użyć  
 $a$  i  $f$  są żywe, użyte w (3)

(4)  $b := d + f$        $b$  jest martwa i nie ma dalszych użyć  
 $d$  i  $f$  są żywe, użyte w (4)

(5)  $e := a - c$        $e$  jest martwa i nie ma dalszych użyć  
 $a$  i  $c$  są żywe, użyte w (5)

(6)  $b := d + c$        $b$  jest martwa i nie ma dalszych użyć  
 $d$  i  $c$  są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



# Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$

(2)  $d := d - b$       d jest martwa i nie ma dalszych użyć  
                                d i b są żywe, użyte w (2)

(3)  $e := a + f$       e jest martwa i nie ma dalszych użyć  
                                a i f są żywe, użyte w (3)

(4)  $b := d + f$       b jest martwa i nie ma dalszych użyć  
                                d i f są żywe, użyte w (4)

(5)  $e := a - c$       e jest martwa i nie ma dalszych użyć  
                                a i c są żywe, użyte w (5)

(6)  $b := d + c$       b jest martwa i nie ma dalszych użyć  
                                d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



# Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

(1)  $a := b + c$       a jest martwa i nie ma dalszych użyć

(2)  $d := d - b$       d jest martwa i nie ma dalszych użyć  
d i b są żywe, użyte w (2)

(3)  $e := a + f$       e jest martwa i nie ma dalszych użyć  
a i f są żywe, użyte w (3)

(4)  $b := d + f$       b jest martwa i nie ma dalszych użyć  
d i f są żywe, użyte w (4)

(5)  $e := a - c$       e jest martwa i nie ma dalszych użyć  
a i c są żywe, użyte w (5)

(6)  $b := d + c$       b jest martwa i nie ma dalszych użyć  
d i c są żywe, użyte w (6)

na końcu wszystkie zmienne są żywe



# Obliczanie żywotności zmiennych w bloku

Przykład: obliczanie żywotności zmiennych w ciągu instrukcji trójadresowych:

- |                  |  |
|------------------|--|
| (1) $a := b + c$ | a jest martwa i nie ma dalszych użyć<br>b i c są żywe, użyte w (1) |
| (2) $d := d - b$ | d jest martwa i nie ma dalszych użyć<br>d i b są żywe, użyte w (2) |
| (3) $e := a + f$ | e jest martwa i nie ma dalszych użyć<br>a i f są żywe, użyte w (3) |
| (4) $b := d + f$ | b jest martwa i nie ma dalszych użyć<br>d i f są żywe, użyte w (4) |
| (5) $e := a - c$ | e jest martwa i nie ma dalszych użyć<br>a i c są żywe, użyte w (5) |
| (6) $b := d + c$ | b jest martwa i nie ma dalszych użyć<br>d i c są żywe, użyte w (6) |

na końcu wszystkie zmienne są żywe

Oznaczmy:

- $def[B]$  - zbiór zmiennych *definiowanych* w bloku  $B$ , czyli mających przypisanie wartości
- $use[B]$  - zbiór zmiennych *używanych* w bloku  $B$  przed definicją

Ustawiamy:

$$In[koniec] = \emptyset$$

Dla różnych bloków podstawowych:

$$In[B] = use[B] \cup (Out[B] - def[B])$$

$$Out[B] = \bigcup_{S \succ B} In[S]$$

Równania rozwiązywane są rekurencyjnie do czasu, aż wartości przestaną się zmieniać.



# Optymalizacja przez szparkę

Optymalizacja „przez szparkę” polega na analizie krótkich ciągów instrukcji i zastępując je bardziej wydajnymi. „Szparka” to małe okienko przesuwające się po programie. Procedura może wymagać kolejnych przebiegów.

Przekształcenia programów wykonywane w czasie optymalizacji przez szparkę:

- usuwanie niepotrzebnych rozkazów,
- optymalizacja przepływu sterowania,
- uproszczenia algebraiczne,
- wykorzystanie szczególnych rozkazów maszyny.



Jeśli w jednym bloku mamy ciąg instrukcji:

MOV R<sub>i</sub>, x

MOV x, R<sub>i</sub>

to drugą instrukcję możemy bezpiecznie usunąć, gdyż wartość w pamięci się nie zmieniła.

Blok, do którego nie prowadzi żadna krawędź w grafie przepływu, może zostać usunięty. W szczególności możemy nie tworzyć krawędzi do bloku w przypadku skoku warunkowego, jeśli warunek jest stałą znaną w trakcie kompilacji i jego wartość wskazuje na niemożność wykonania skoku. Druga możliwość to wartość warunku wymuszającego skok warunkowy, przez co następny w kolejności blok staje się nieosiągalny.



Automatycznie generowany kod może zawierać skoki do skoków, zarówno skoki bezwarunkowe do skoków bezwarunkowych, jak i skoki warunkowe do skoków bezwarunkowych i skoki bezwarunkowe do skoków warunkowych.

Jeśli mamy skok do etykiety  $L_1$ , pod którą jest skok do etykiety  $L_2$ , możemy skok do  $L_1$  zastąpić skokiem do  $L_2$  (a jeśli drugi skok był osiągalny tylko przez pierwszy, można go w ogóle wyeliminować).

Ciąg instrukcji:

```
if warunek goto L1
```

```
...
```

```
L1: goto L2
```

zastąpić ciągiem:

```
if warunek goto L2
```

```
...
```

```
L1: goto L2
```



Tożsamości algebraiczne postaci:

$$x = x + 0$$

lub

$$x = x * 1$$

pozwalają uprościć kod, chociaż występują w generowanym kodzie dość rzadko. Mogą pojawiać się w wyniku propagacji stałych.

Częstszym zabiegiem jest zamiana kosztownych instrukcji na tańsze. Na przykład podnoszenie do kwadratu może być zrealizowane jako mnożenie przez siebie, co jest dużo szybsze niż obliczanie drugiej potęgi wyrażenia. Podobnie dzielenie przez stałą można zastąpić mnożeniem przez odwrotność stałej.

Maszyna docelowa może mieć instrukcje, która pozwalają pewne operacje wykonać szybciej. Np. zamiast rozkazu:

ADD R0, 1

można użyć instrukcji:

INC R0



Optymalizacja może być:

- lokalna, gdy można ją wykonać rozpatrując tylko instrukcje wewnętrz jednego bloku,
- globalna, gdy wymaga rozpatrzenia więcej niż jednego bloku.

Rozważymy następujące optymalizacje:

- przekształcenia zachowujące funkcję,
  - ▶ podwyrażenia wspólne,
  - ▶ propagację kopii,
  - ▶ usuwanie kodu martwego,
- optymalizacje pętli:
  - ▶ przemieszczenie kodu,
  - ▶ zmienne indukcyjne oraz redukcja mocy,



# Program przykładowy

Optymalizacje będziemy rozważać na fragmencie następującej procedury:

```
void quicksort(const int m, const int n) {  
    int i, j, v, x;  
    if (m >= n) return;  
    i = m - 1; j = n; v = a[n];  
    while (1) {  
        do i = i + 1; while (a[i] < v);  
        do j = j - 1; while (a[j] > v);  
        if (i >= j) break;  
        x = a[i]; a[i] = a[j]; a[j] = x;  
    }  
    x = a[i]; a[i] = a[n]; a[n] = x;  
    quicksort(m, j); quicksort(i + 1, n);  
}
```



# Program przykładowy



Optymalizacje będziemy rozważać na fragmencie następującej procedury:

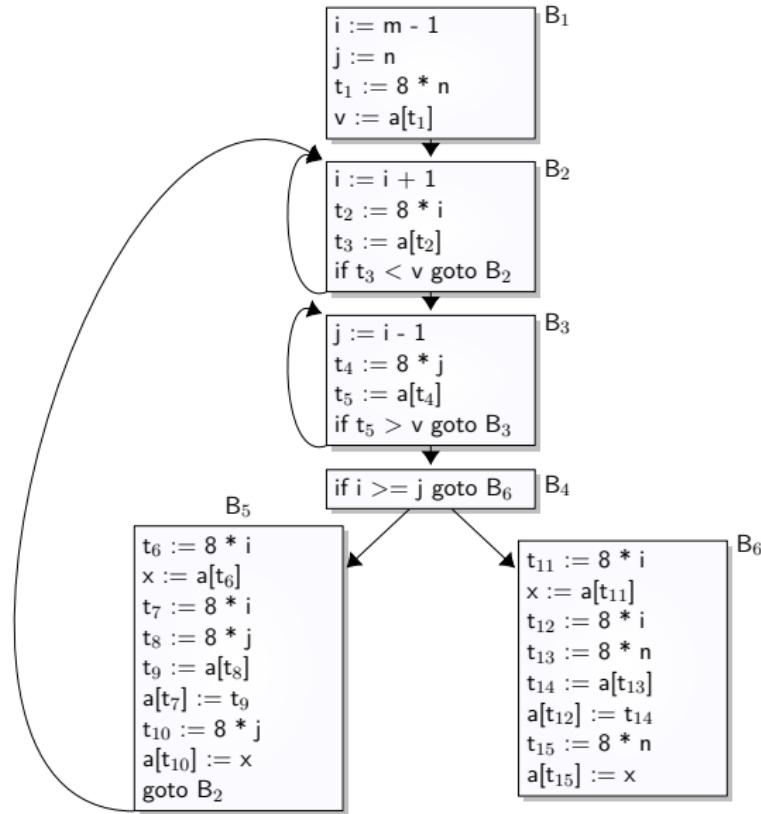
```
void quicksort(const int m, const int n) {  
    int i, j, v, x;  
    if (m >= n) return;  
    i = m - 1; j = n; v = a[n];  
    while (1) {  
        do i = i + 1; while (a[i] < v);  
        do j = j - 1; while (a[j] > v);  
        if (i >= j) break;  
        x = a[i]; a[i] = a[j]; a[j] = x;  
    }  
    x = a[i]; a[i] = a[n]; a[n] = x;  
    quicksort(m, j); quicksort(i + 1, n);  
}
```

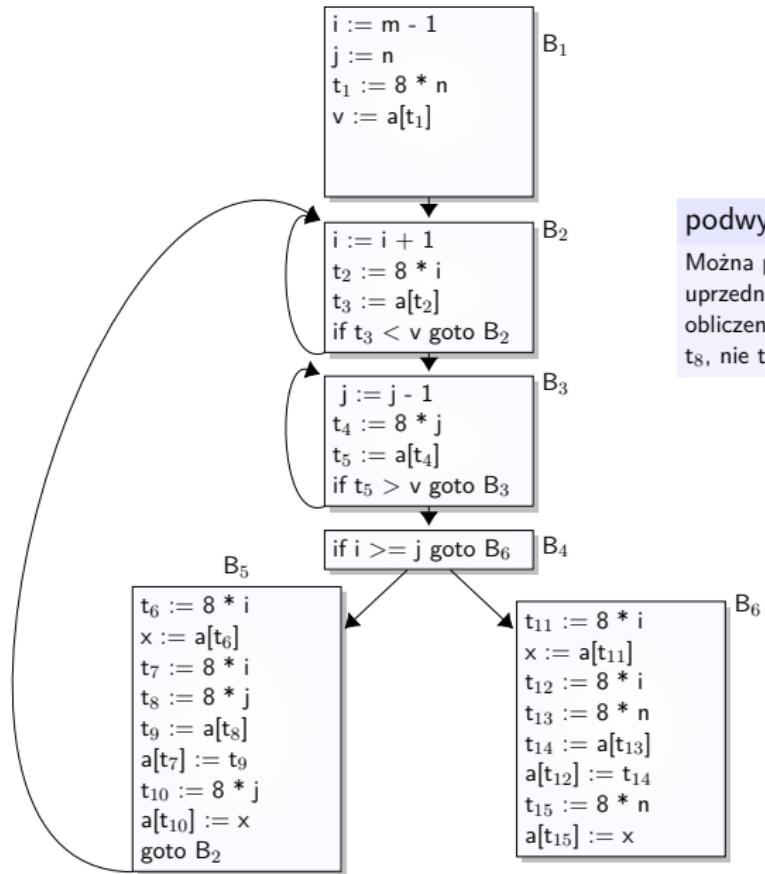


# Program przykładowy — kod trójadresowy

1	i = m - 1	16	t <sub>7</sub> := 8 * i
2	j := n	17	t <sub>8</sub> := 8 * j
3	t <sub>1</sub> := 8 * n	18	t <sub>9</sub> := a[t <sub>8</sub> ]
4	v := a[t <sub>1</sub> ]	19	a[t <sub>7</sub> ] = t <sub>9</sub>
5	i := i + 1	20	t <sub>10</sub> := 8 * j
6	t <sub>2</sub> := 8 * i	21	a[t <sub>10</sub> ] := x
7	t <sub>3</sub> := a[t <sub>2</sub> ]	22	goto 5
8	if t <sub>3</sub> < v goto 5	23	t <sub>11</sub> := 8 * i
9	j := j - 1	24	x := a[t <sub>11</sub> ]
10	t <sub>4</sub> := 8 * j	25	t <sub>12</sub> := 8 * i
11	t <sub>5</sub> := a[t <sub>4</sub> ]	26	t <sub>13</sub> := 8 * n
12	if t <sub>5</sub> > v goto 9	27	t <sub>14</sub> := a[t <sub>13</sub> ]
13	if i > j goto 23	28	a[t <sub>12</sub> ] := t <sub>14</sub>
14	t <sub>6</sub> := 8 * i	29	t <sub>15</sub> := 8 * n
15	x := a[t <sub>6</sub> ]	30	a[t <sub>15</sub> ] := x

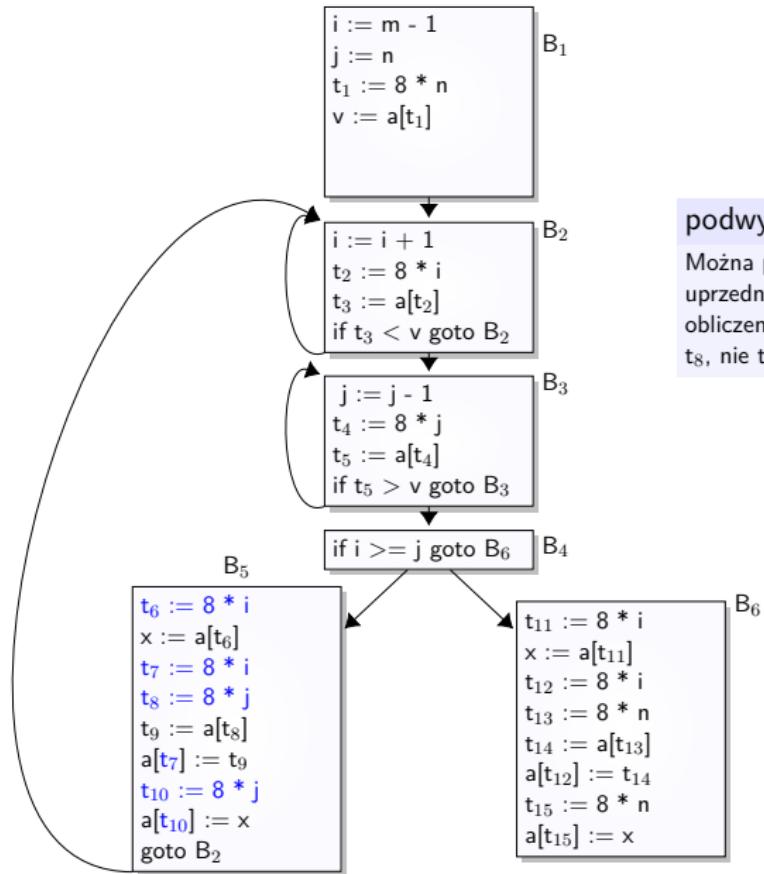
# Program przykładowy - graf przepływu





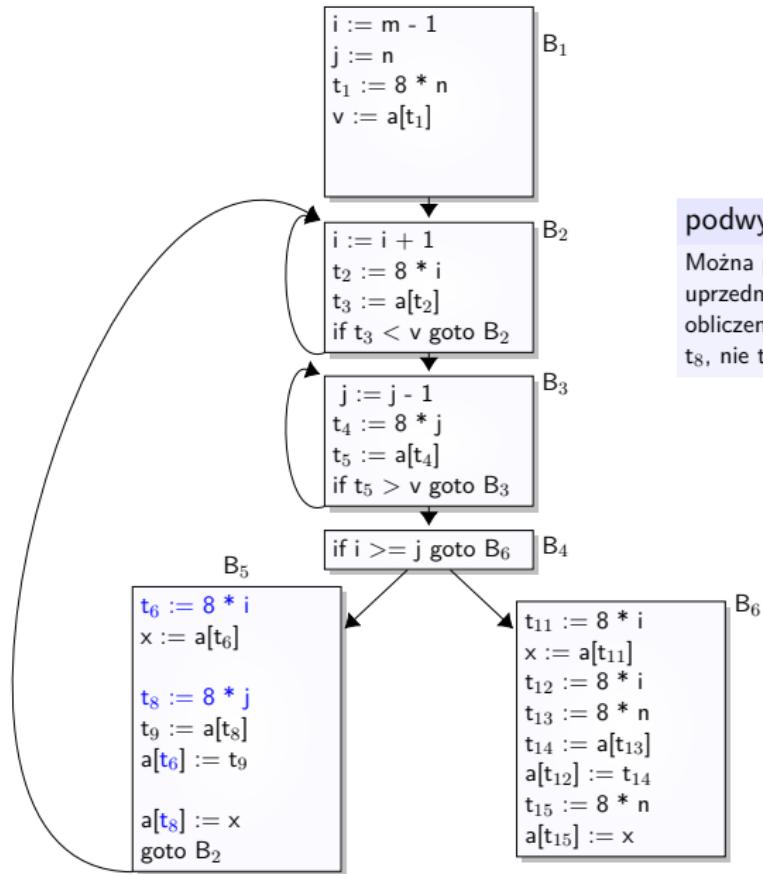
## podwyrażenia wspólne

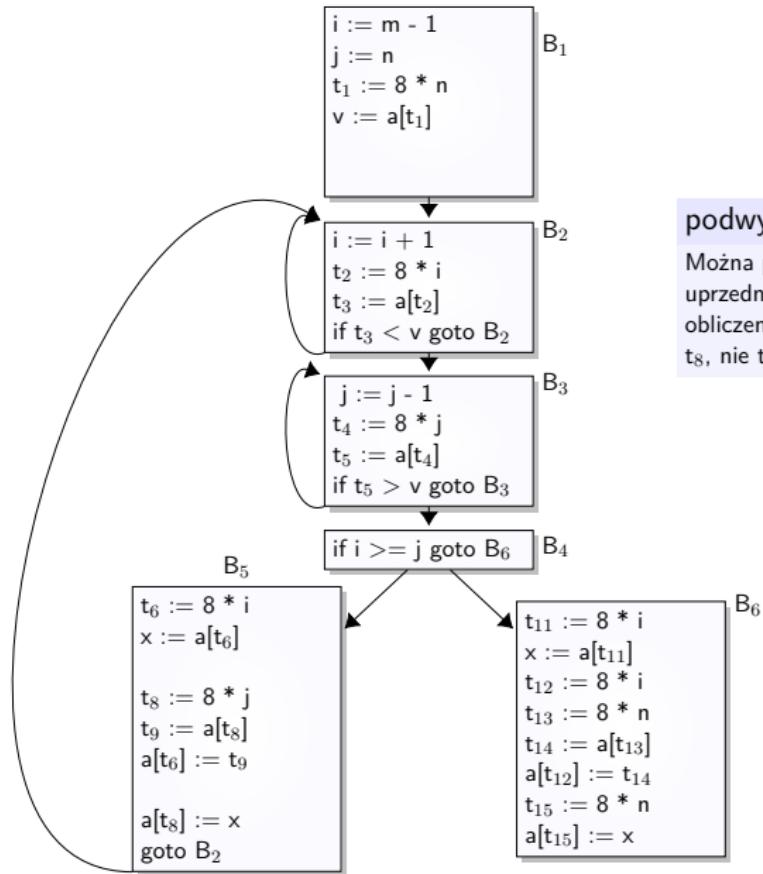
Można ponownie użyć obliczonych uprzednio wartości. Ponieważ wynik obliczenia  $8*i$  jest już w  $t_6$ , a  $8*j$  w  $t_8$ , nie trzeba liczyć ich ponownie



## podwyrażenia wspólne

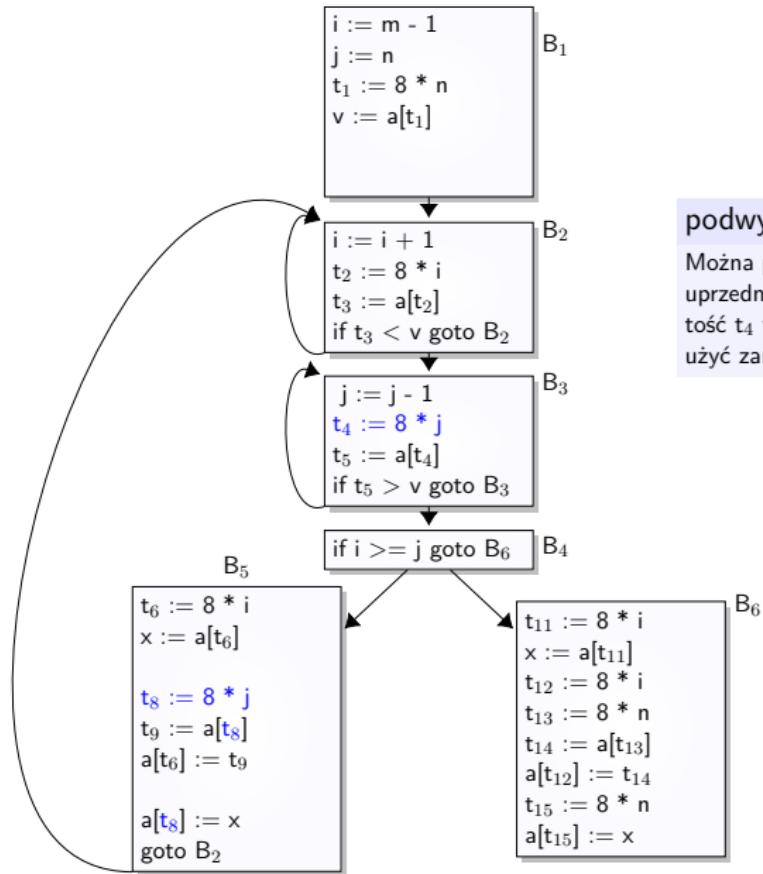
Można ponownie użyć obliczonych uprzednio wartości. Ponieważ wynik obliczenia  $8 \cdot i$  jest już w  $t_6$ , a  $8 \cdot j$  w  $t_8$ , nie trzeba liczyć ich ponownie





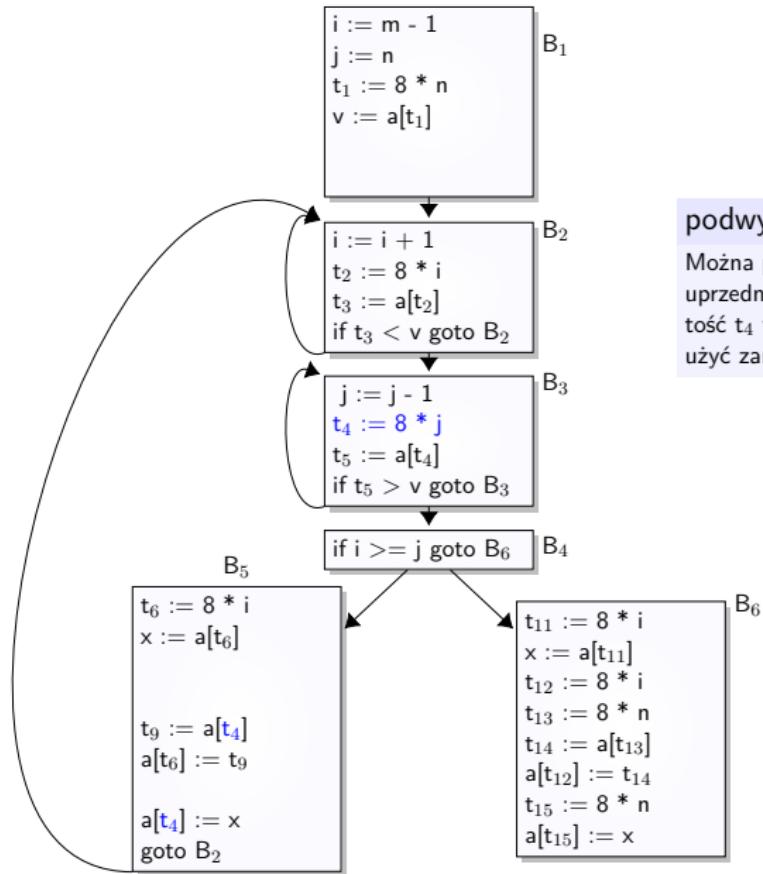
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Ponieważ wynik obliczenia  $8*i$  jest już w  $t_6$ , a  $8*j$  w  $t_8$ , nie trzeba liczyć ich ponownie



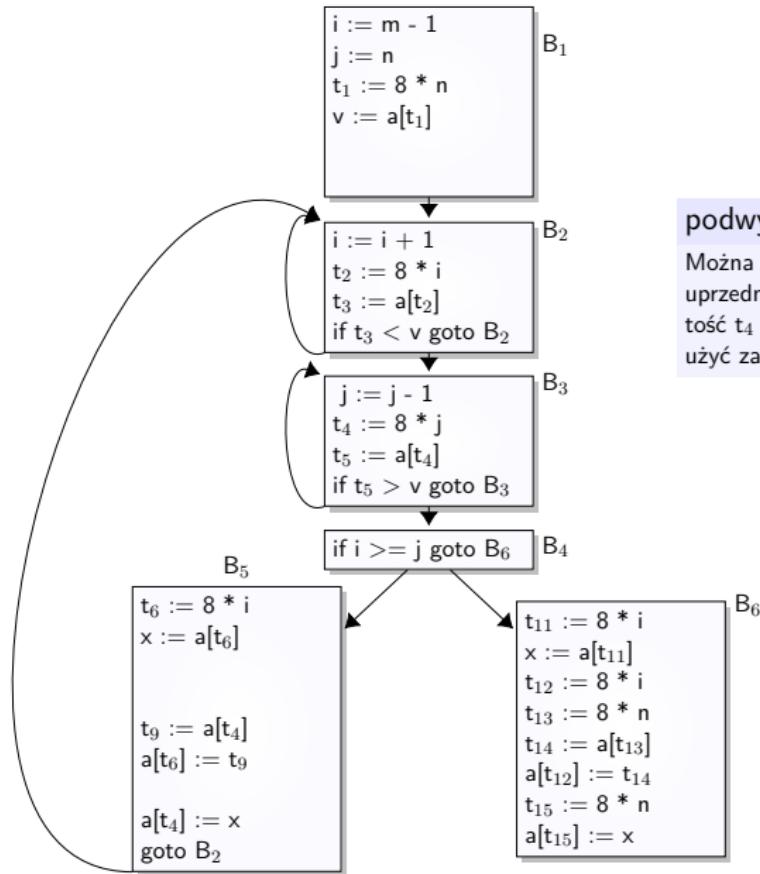
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Ponieważ wartość t<sub>4</sub> to też 8\*j, więc można t<sub>4</sub> użyć zamiast t<sub>8</sub>.



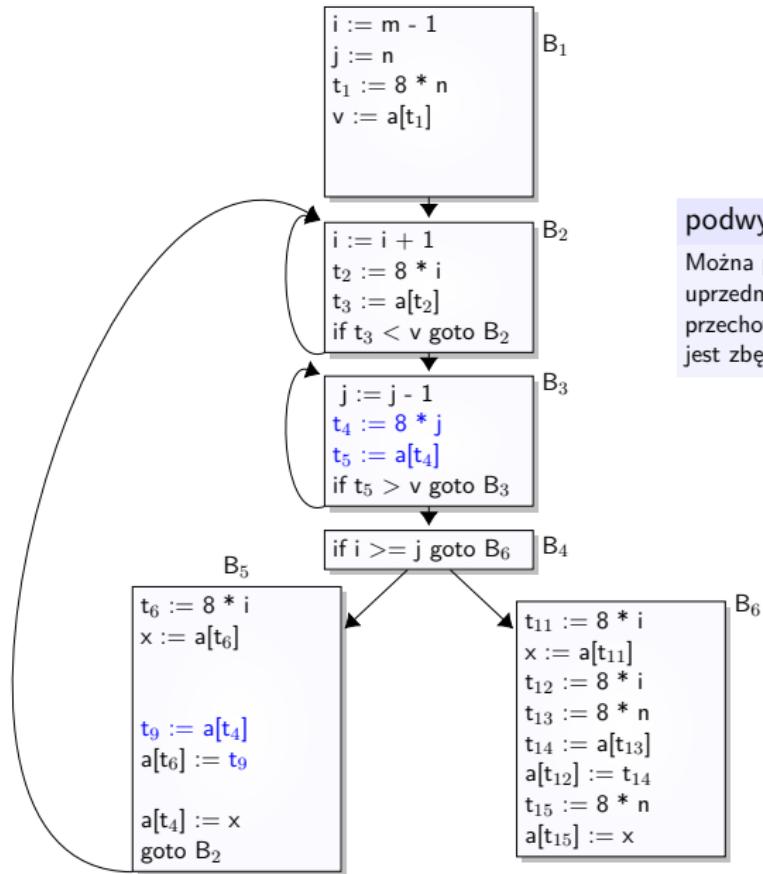
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Ponieważ wartość t<sub>4</sub> to też 8\*j, więc można t<sub>4</sub> użyć zamiast t<sub>8</sub>.



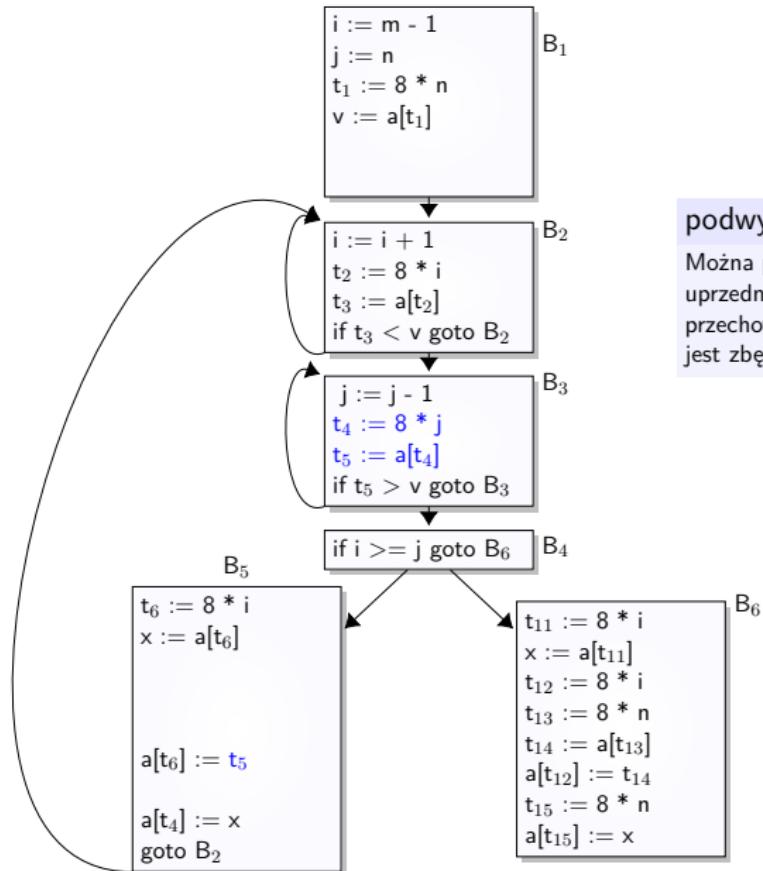
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Ponieważ wartość *t<sub>4</sub>* to też  $8 \cdot j$ , więc można *t<sub>4</sub>* użyć zamiast *t<sub>8</sub>*.



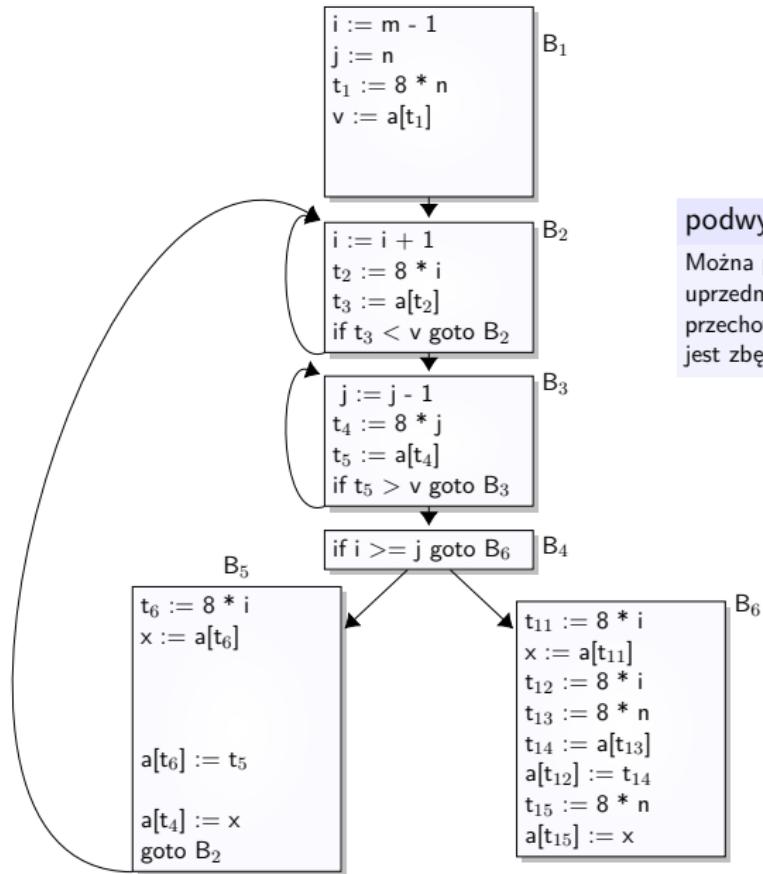
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Ponieważ  $t_5$  przechowuje wartość  $a[t_4]$ , więc  $t_9$  jest zbędne.



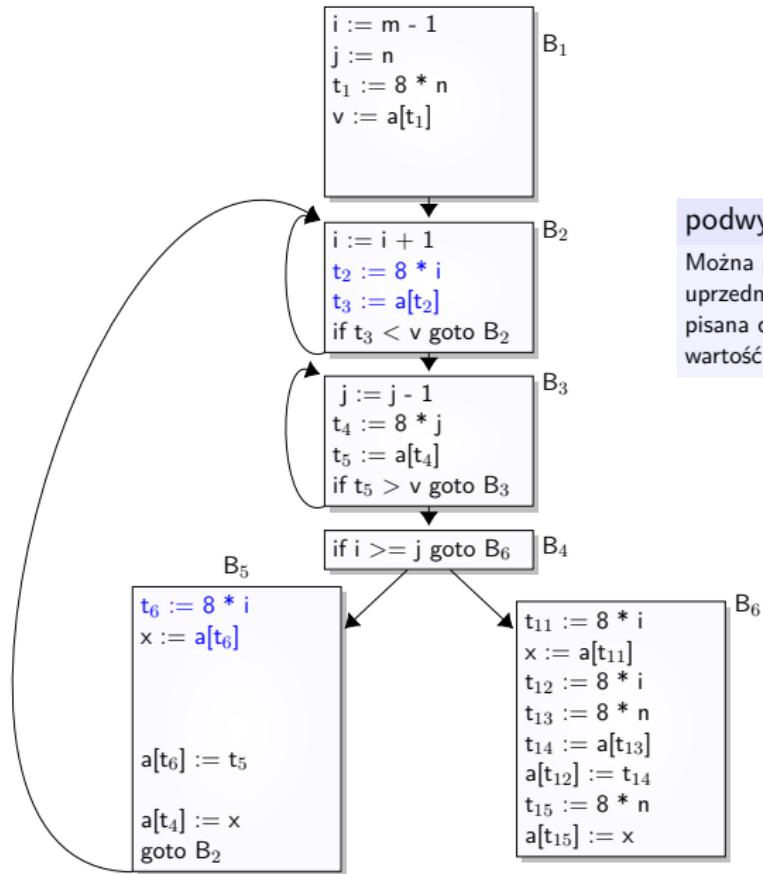
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Ponieważ  $t_5$  przechowuje wartość  $a[t_4]$ , więc  $t_9$  jest zbędne.



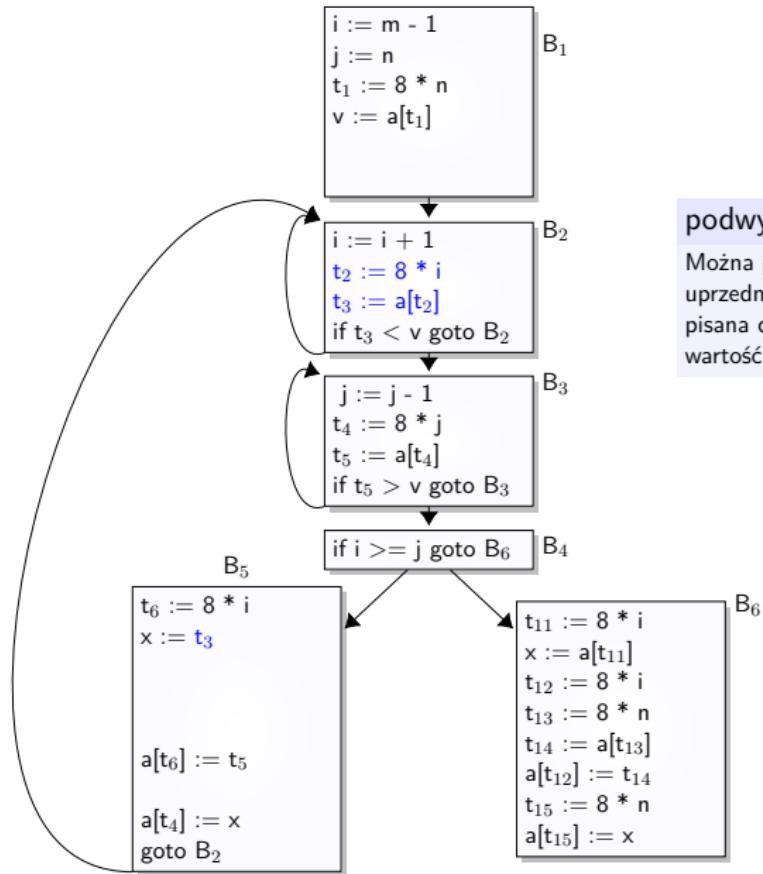
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Ponieważ  $t_5$  przechowuje wartość  $a[t_4]$ , więc  $t_9$  jest zbędne.



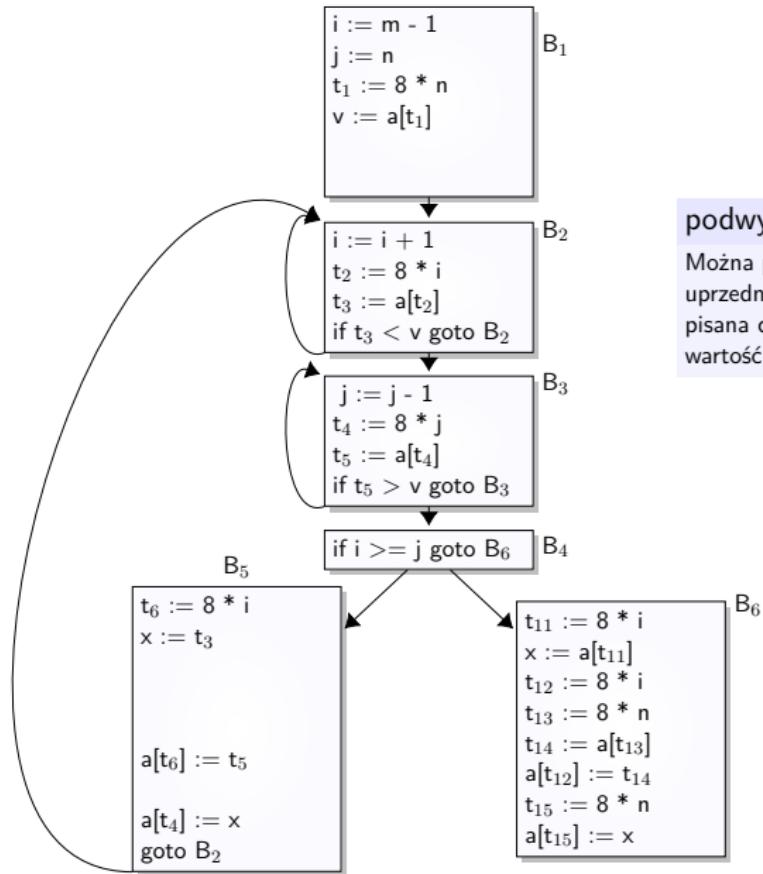
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartość przypisana do *x* w B<sub>5</sub> jest taka sama, jak wartość przypisana do *t<sub>3</sub>* w B<sub>2</sub>.



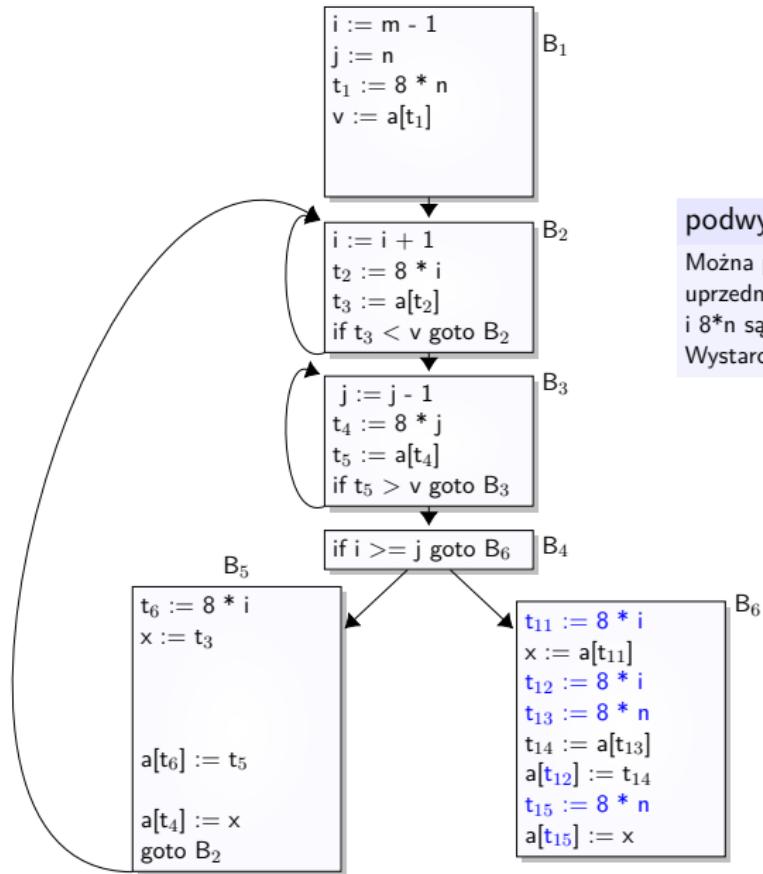
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartość przypisana do *x* w B<sub>5</sub> jest taka sama, jak wartość przypisana do *t<sub>3</sub>* w B<sub>2</sub>.



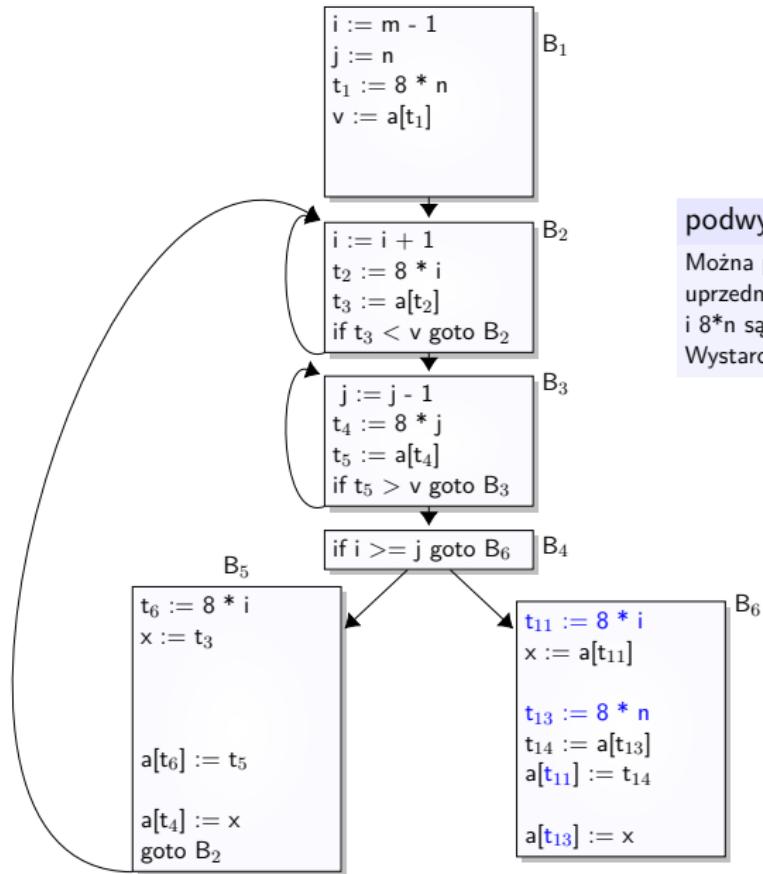
## podwyrażenia wspólne

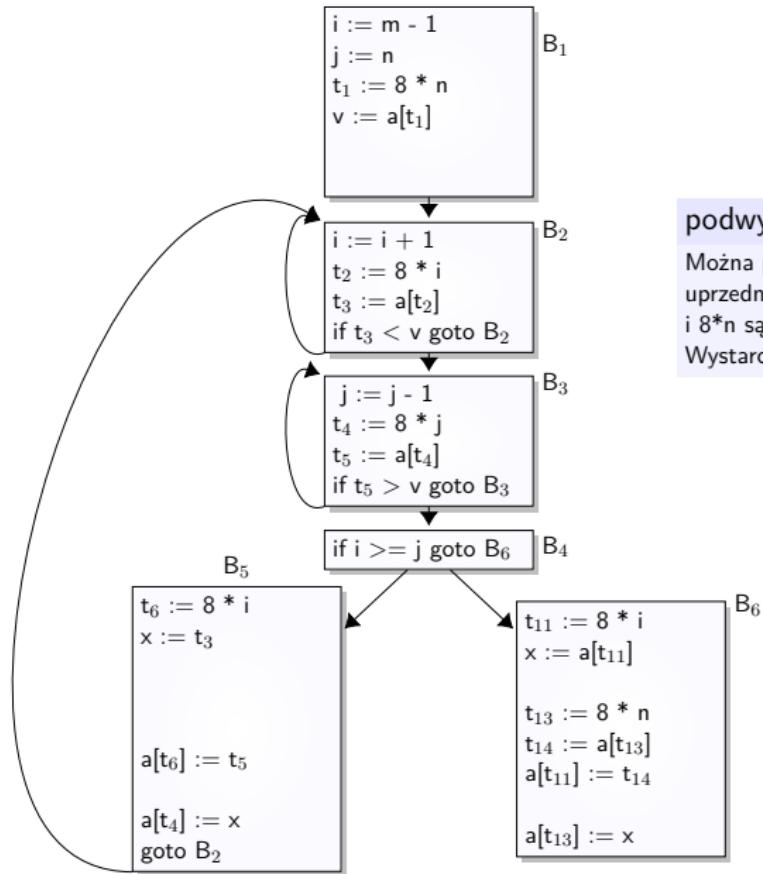
Można ponownie użyć obliczonych uprzednio wartości. Wartość przypisana do *x* w B<sub>5</sub> jest taka sama, jak wartość przypisana do *t<sub>3</sub>* w B<sub>2</sub>.

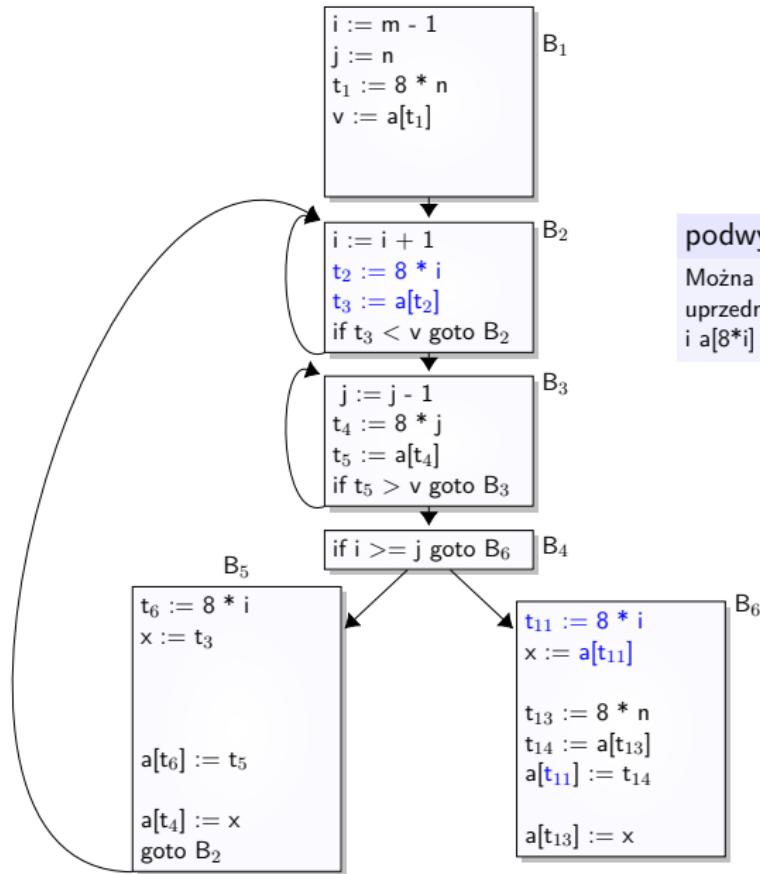


## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartości  $8*i$  i  $8*n$  są liczone w B6 dwukrotnie. Wystarczy raz.

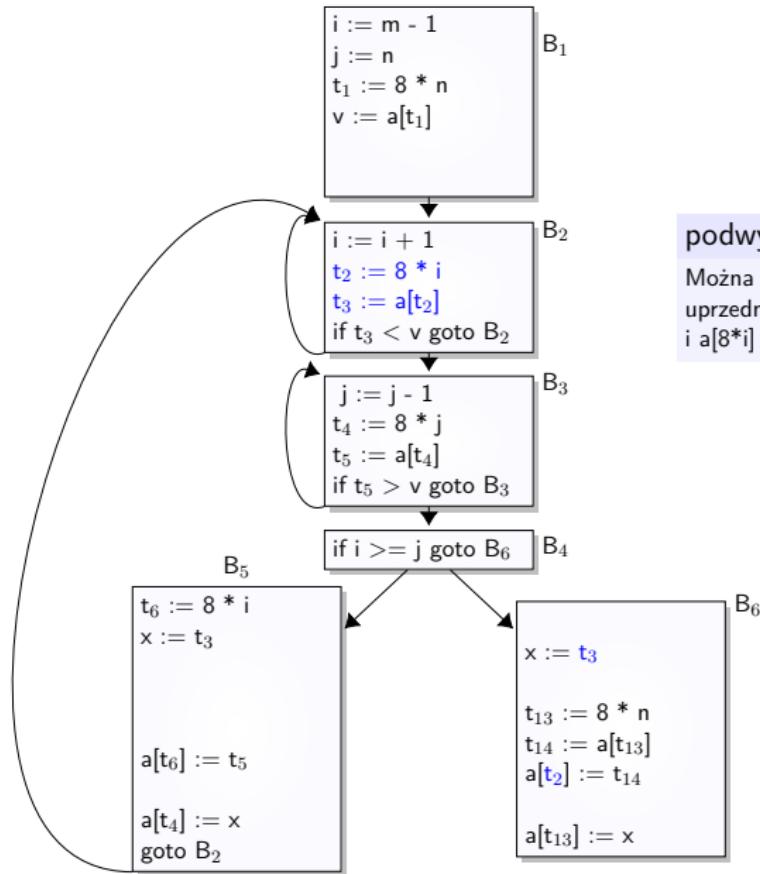






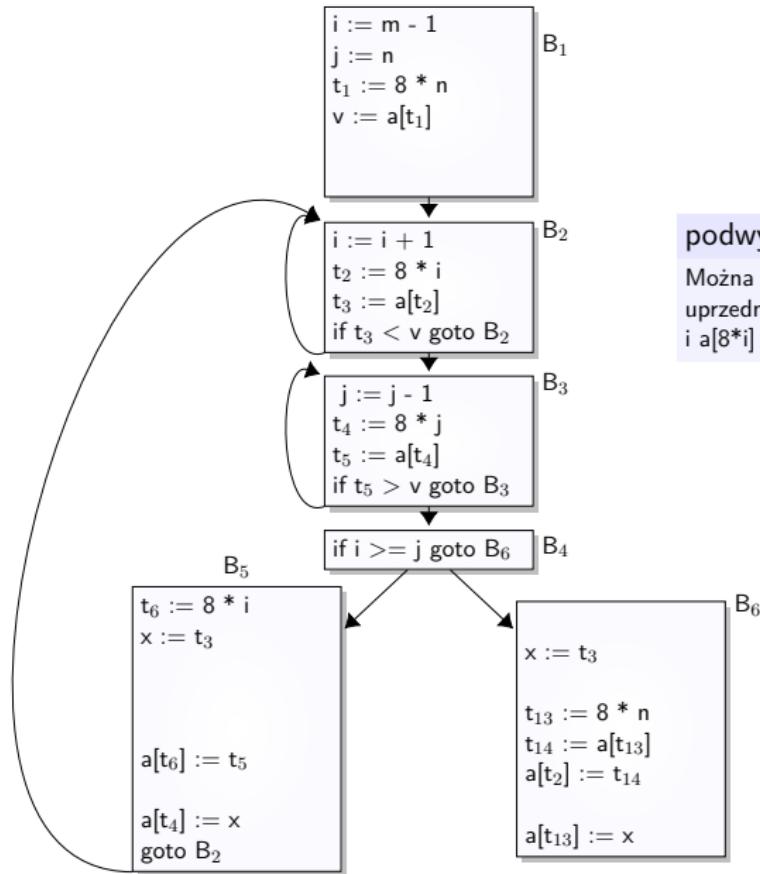
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartości  $8*i$  i  $a[8*i]$  (jako  $t_3$ ) są już liczone w B2.



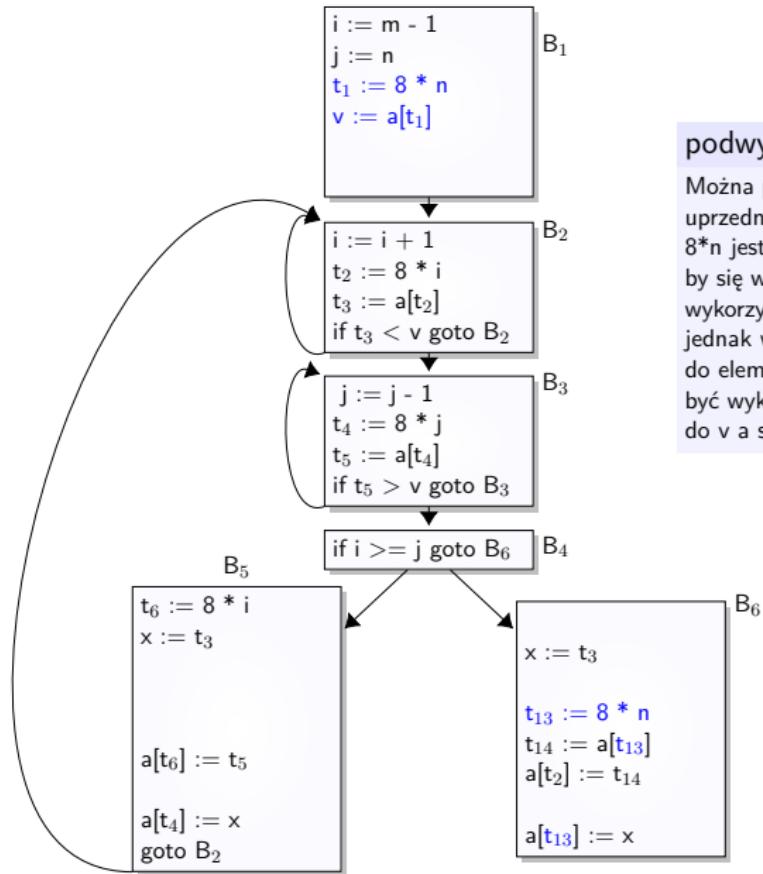
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartości  $8*i$  i  $a[8*i]$  (jako  $t_3$ ) są już liczone w B2.



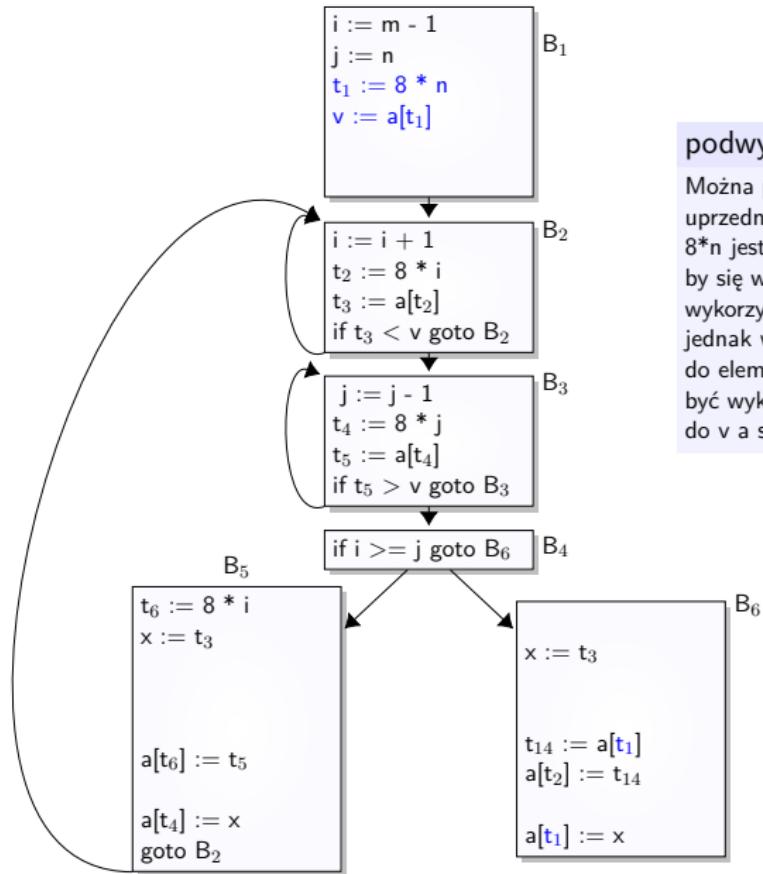
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartości 8\*i i a[8\*i] (jako t<sub>3</sub>) są już liczone w B<sub>2</sub>.



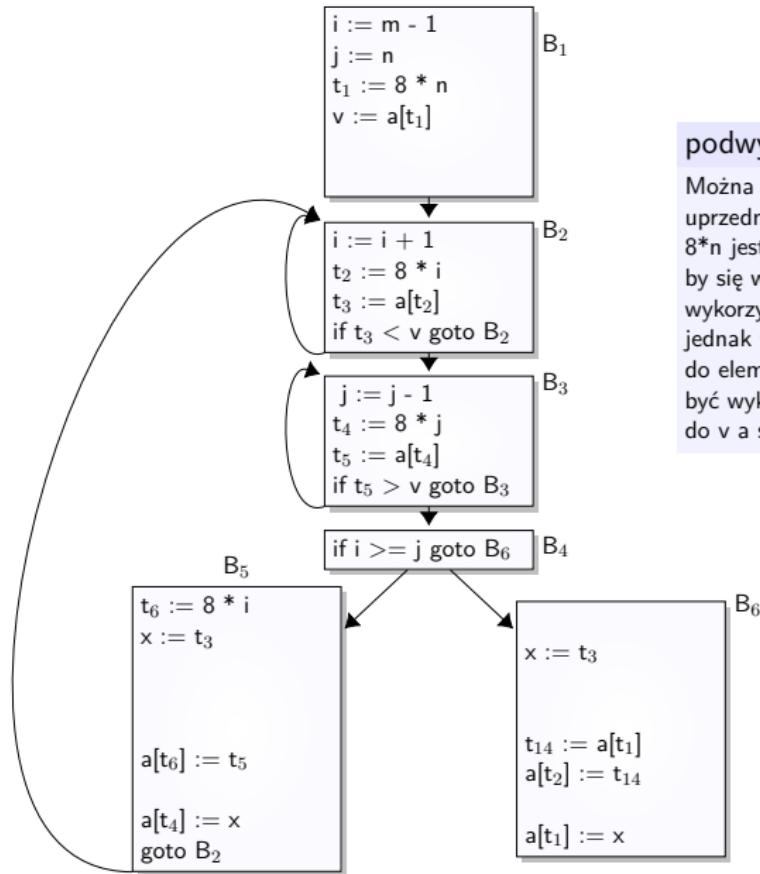
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartość  $8 * n$  jest liczona już w B1. Mogło by się wydawać, że w B5 można też wykorzystać wartość v jako  $a[t_1]$ , jednak w B5 następuje przypisanie do elementów tablicy a i B5 może być wykonany między przypisaniem do v a skorzystaniem z  $a[t_1]$ .



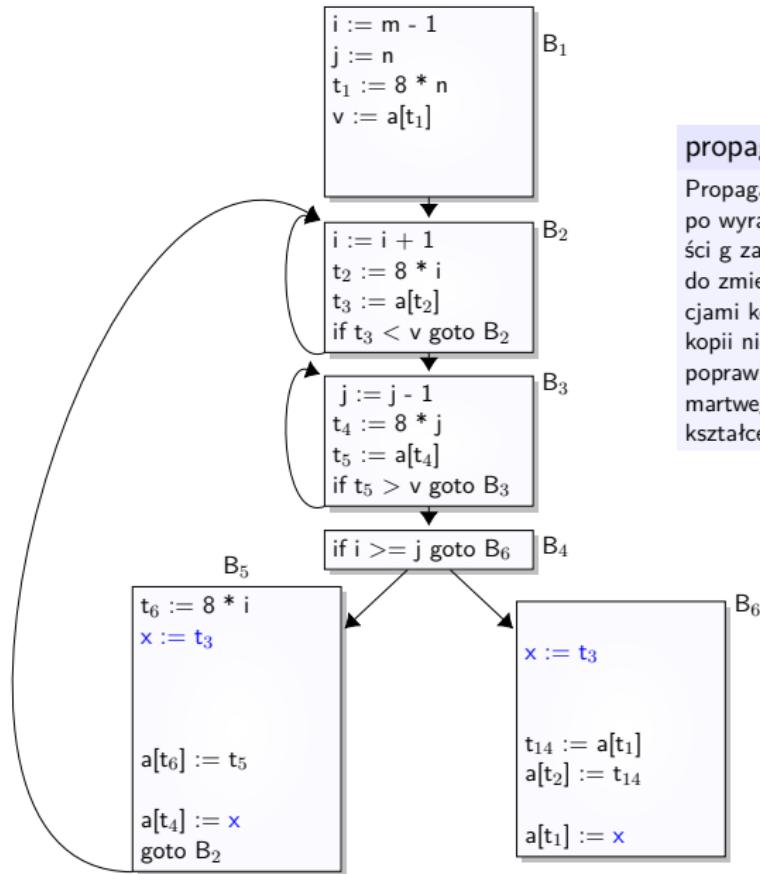
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartość  $8*n$  jest liczona już w B<sub>1</sub>. Mogło by się wydawać, że w B<sub>5</sub> można też wykorzystać wartość v jako a[t<sub>1</sub>], jednak w B<sub>5</sub> następuje przypisanie do elementów tablicy a i B<sub>5</sub> może być wykonany między przypisaniem do v a skorzystaniem z a[t<sub>1</sub>].



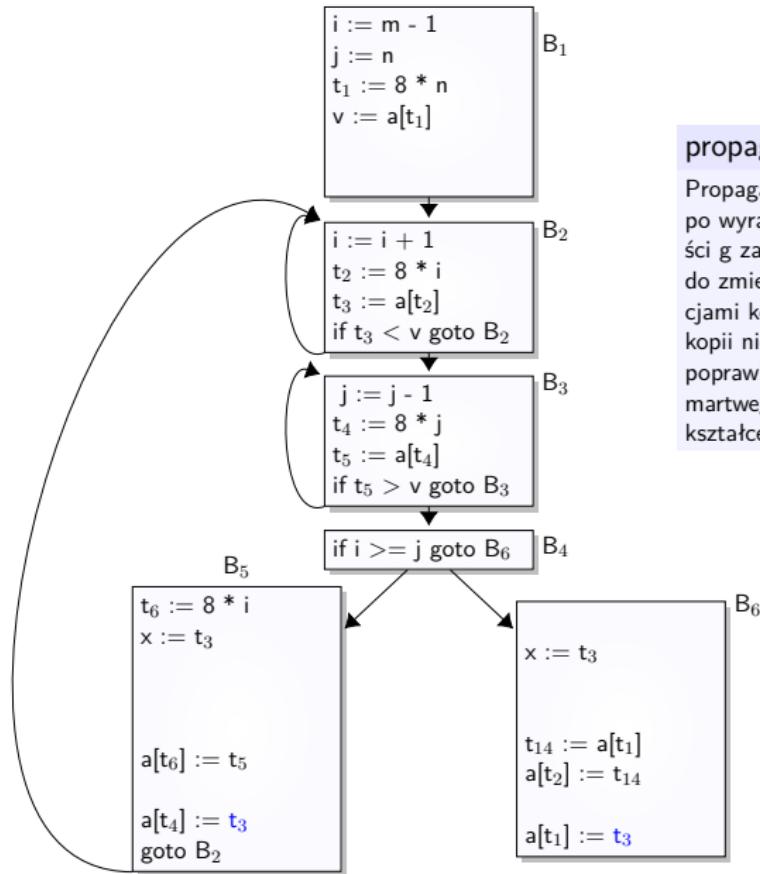
## podwyrażenia wspólne

Można ponownie użyć obliczonych uprzednio wartości. Wartość  $8 * n$  jest liczona już w B<sub>1</sub>. Mogło by się wydawać, że w B<sub>5</sub> można też wykorzystać wartość v jako a[t<sub>1</sub>], jednak w B<sub>5</sub> następuje przypisanie do elementów tablicy a i B<sub>5</sub> może być wykonany między przypisaniem do v a skorzystaniem z a[t<sub>1</sub>].



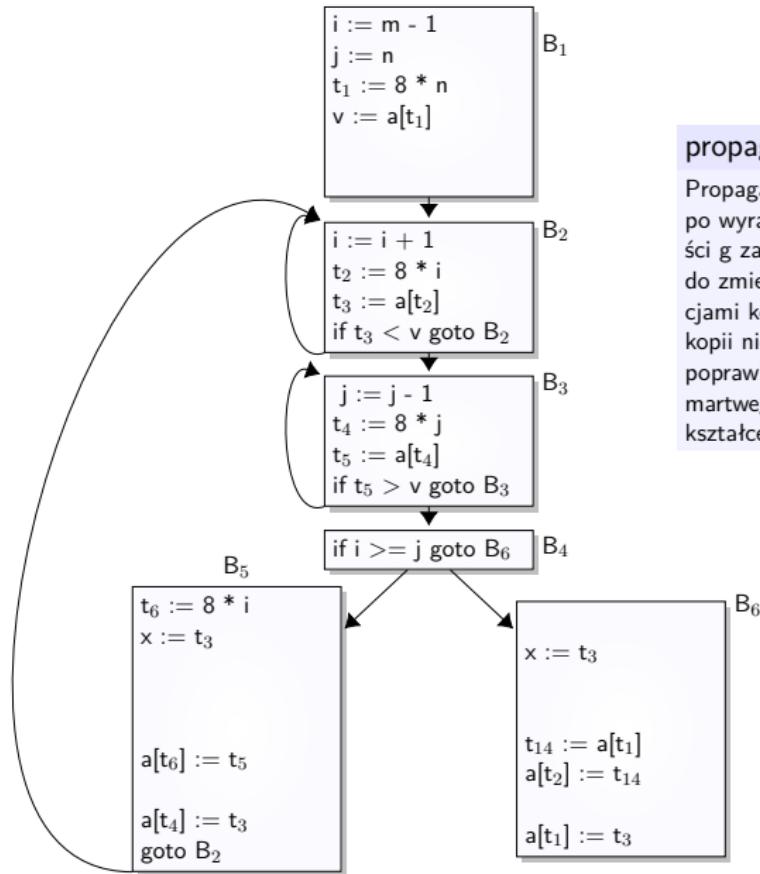
## propagacja kopii

Propagacja kopii polega na używaniu po wyrażeniach postaci  $f := g$  wartości  $g$  zamiast wartości  $f$ . Przypisania do zmiennej  $x$  w B<sub>5</sub> i B<sub>6</sub> są instrukcjami kopiowania. Samą propagacją kopii niczego tu bezpośrednio nie poprawia, ale umożliwia usunięcie martwego kodu w późniejszym przekształceniu.



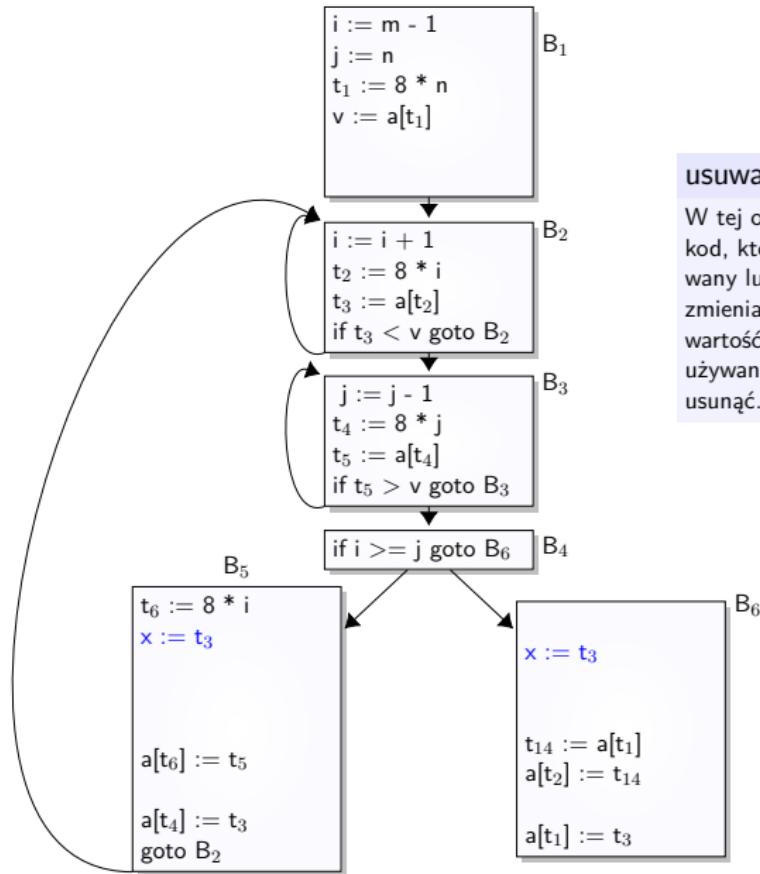
## propagacja kopii

Propagacja kopii polega na używaniu po wyrażeniach postaci  $f := g$  wartości  $g$  zamiast wartości  $f$ . Przypisania do zmiennej  $x$  w  $B_5$  i  $B_6$  są instrukcjami kopiowania. Samą propagacją kopii niczego tu bezpośrednio nie poprawia, ale umożliwia usunięcie martwego kodu w późniejszym przekształceniu.



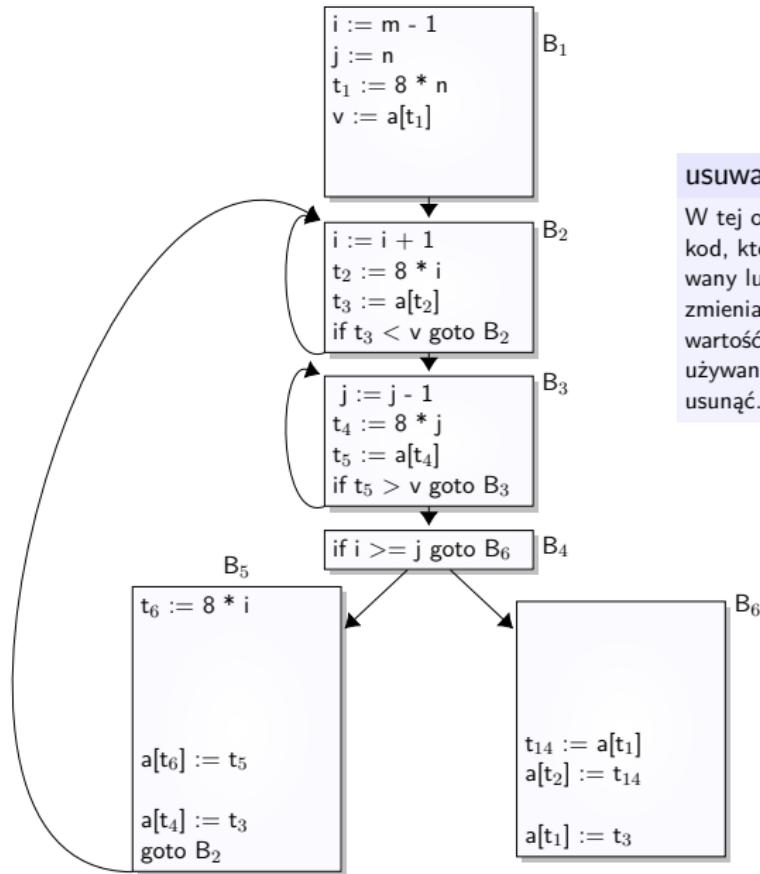
## propagacja kopii

Propagacja kopii polega na używaniu po wyrażeniach postaci  $f := g$  wartości  $g$  zamiast wartości  $f$ . Przypisania do zmiennej  $x$  w B<sub>5</sub> i B<sub>6</sub> są instrukcjami kopiowania. Samą propagacją kopii niczego tu bezpośrednio nie poprawia, ale umożliwia usunięcie martwego kodu w późniejszym przekształceniu.



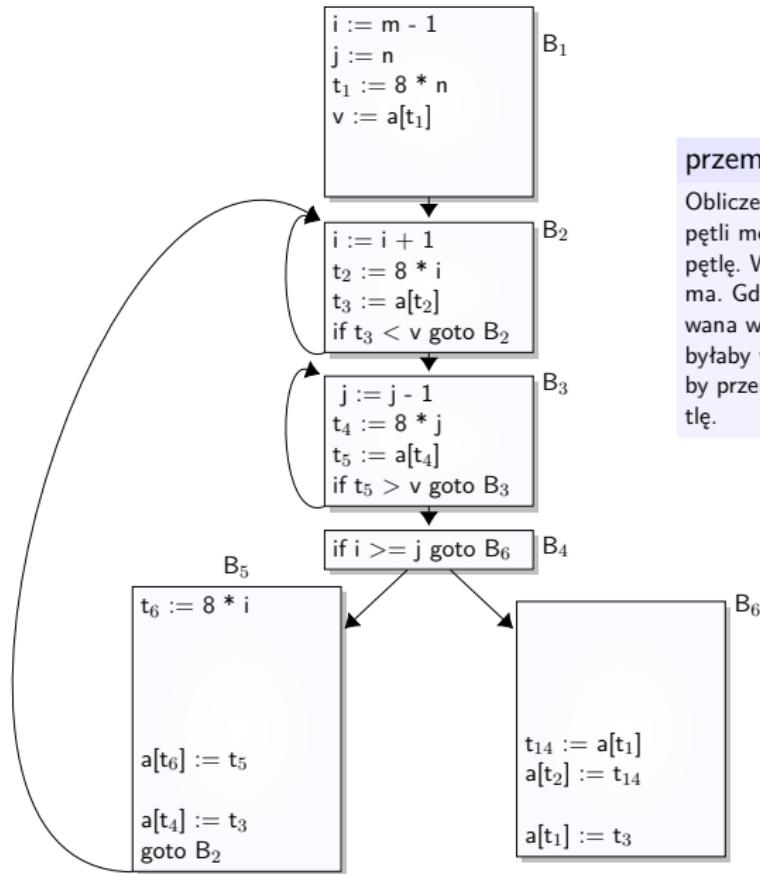
## usuwanie martwego kodu

W tej optymalizacji usuwany jest kod, który nie jest nigdy wykonywany lub którego wykonanie nie zmienia wyniku programu. Ponieważ wartość zmiennej  $x$  nie jest nigdzie używana, nadanie jej wartości można usunąć.



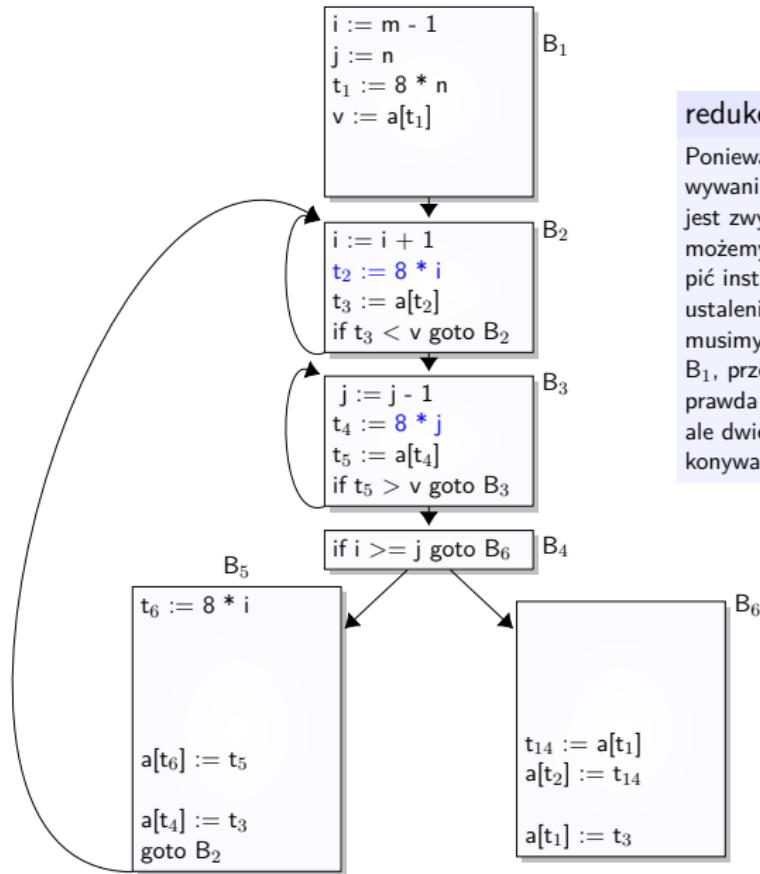
### usuwanie martwego kodu

W tej optymalizacji usuwany jest kod, który nie jest nigdy wykonywany lub którego wykonanie nie zmienia wyniku programu. Ponieważ wartość zmiennej x nie jest nigdzie używana, nadanie jej wartości można usunąć.



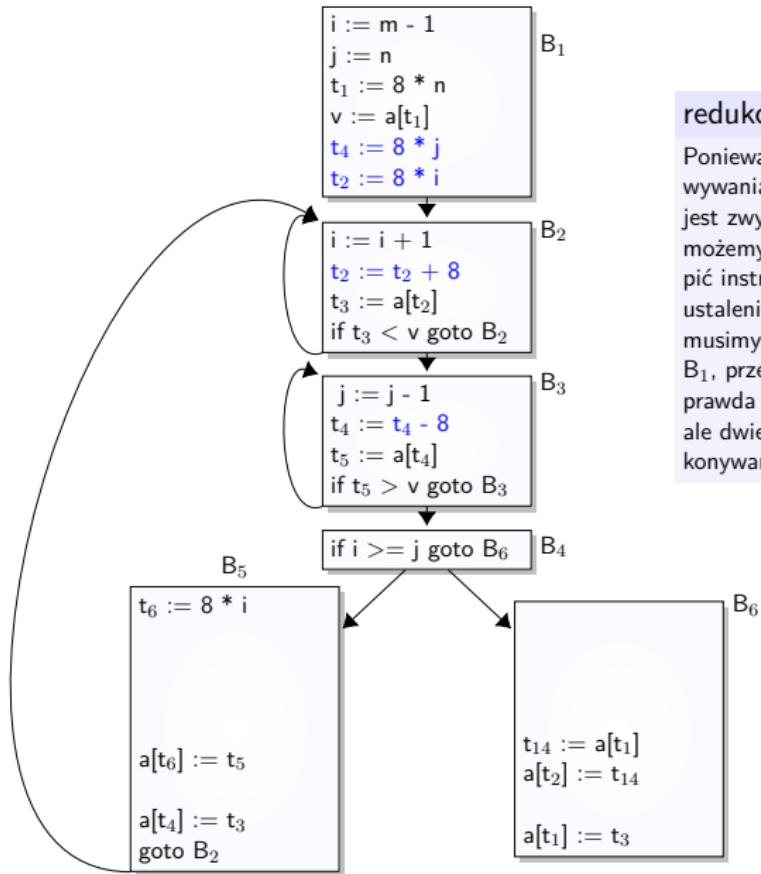
## przemieszczenie kodu

Obliczenia niezmiennicze względem pętli mogą być przemieszczone przed pętlą. W tym przykładzie takich nie ma. Gdyby np. w pętli była dodawana wartość  $x^2$ , a wartość  $x$  nie byłaby w niej zmieniana, można było by przenieść obliczenie  $x*x$  przed pętlę.



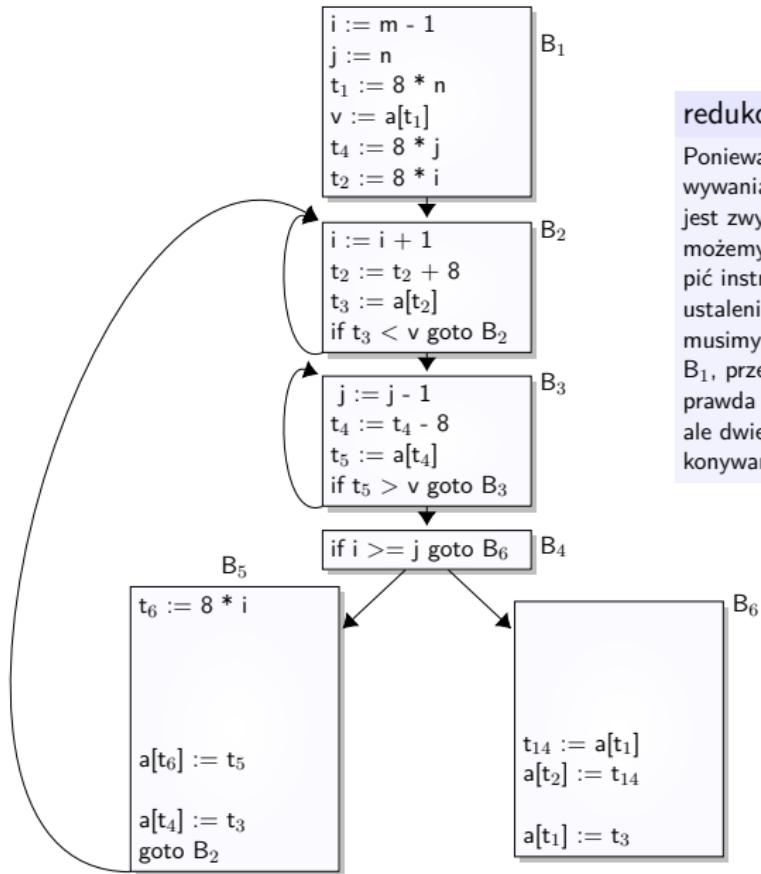
## redukacja mocy

Ponieważ *t<sub>4</sub>* służy nam do przechowywania wartości  $8 \cdot j$ , a dodawanie jest zwykle szybsze niż mnożenie, możemy instrukcję *t<sub>4</sub> := 8 \* j* zastąpić instrukcją *t<sub>4</sub> := t<sub>4</sub> + 8*. W celu ustalenia wartości początkowej *t<sub>4</sub>* musimy dodać instrukcję na końcu B<sub>1</sub>, przed pętlami. Podobnie z t<sub>2</sub>. Co prawda długość kodu się zwiększyła, ale dwie dodatkowe instrukcje są wykonywane tylko raz.



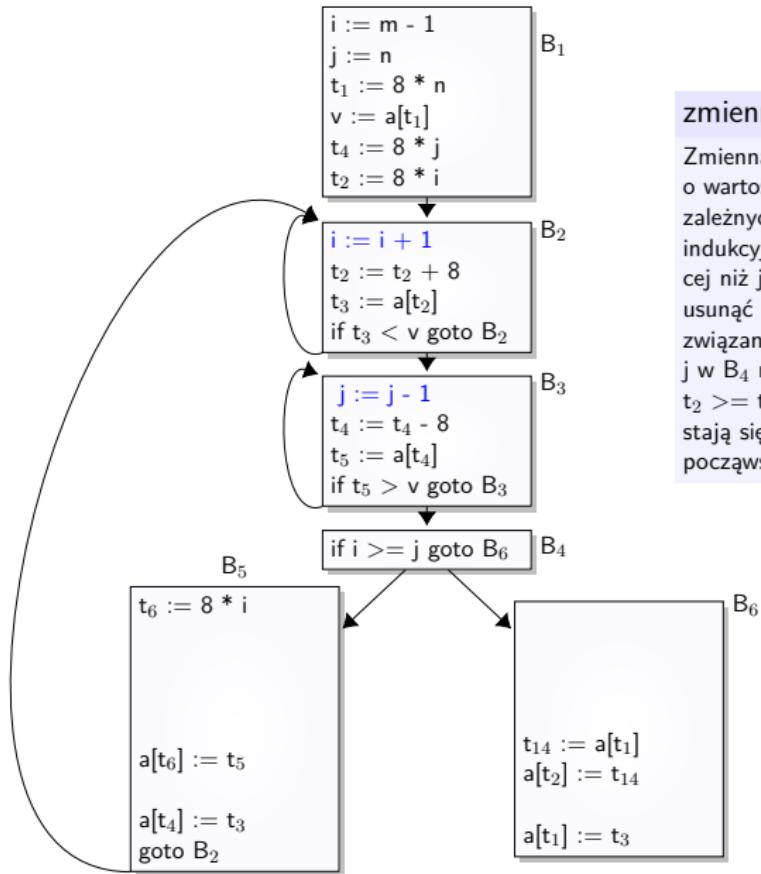
## redukacja mocy

Ponieważ t<sub>4</sub> służy nam do przechowywania wartości 8\*j, a dodawanie jest zwykle szybsze niż mnożenie, możemy instrukcję t<sub>4</sub> := 8\*j zastąpić instrukcją t<sub>4</sub> := t<sub>4</sub>+8. W celu ustalenia wartości początkowej t<sub>4</sub> musimy dodać instrukcję na końcu B<sub>1</sub>, przed pętlami. Podobnie z t<sub>2</sub>. Co prawda długość kodu się zwiększyła, ale dwie dodatkowe instrukcje są wykonywane tylko raz.



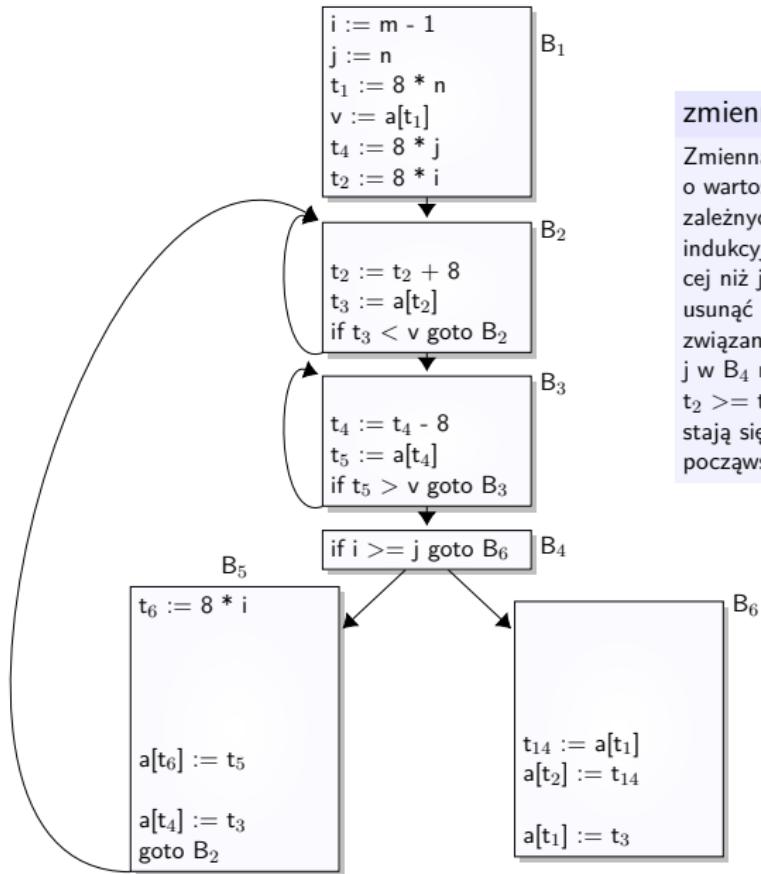
## redukacja mocy

Ponieważ *t<sub>4</sub>* służy nam do przechowywania wartości  $8 \cdot j$ , a dodawanie jest zwykle szybsze niż mnożenie, możemy instrukcję  $t_4 := 8 \cdot j$  zastąpić instrukcją  $t_4 := t_4 + 8$ . W celu ustalenia wartości początkowej  $t_4$  musimy dodać instrukcję na końcu B<sub>1</sub>, przed pętlami. Podobnie z  $t_2$ . Co prawda długość kodu się zwiększyła, ale dwie dodatkowe instrukcje są wykonywane tylko raz.



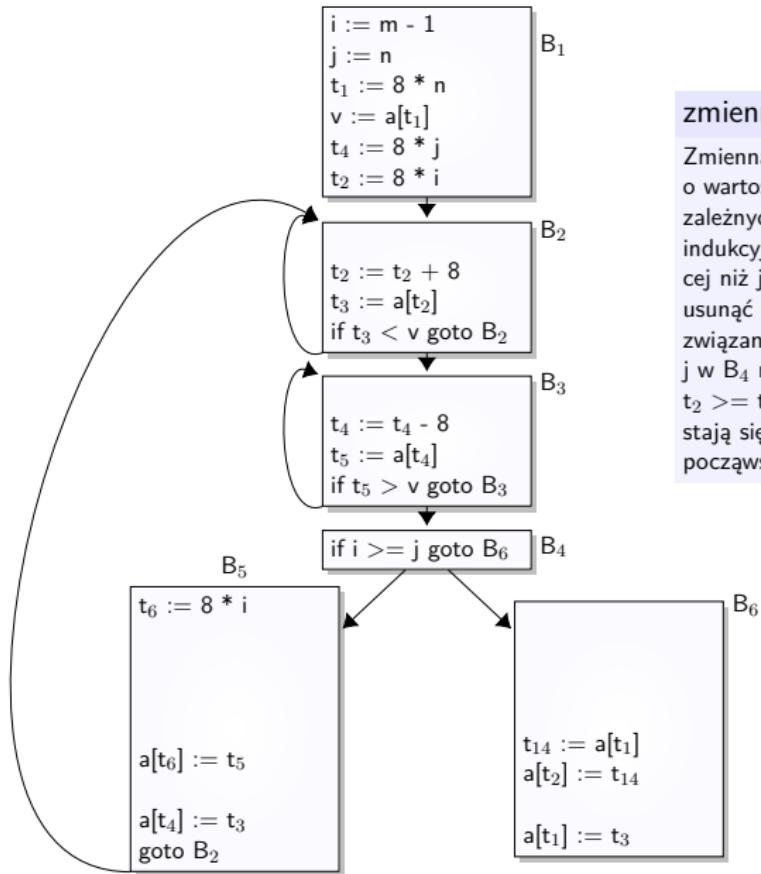
## zmienne indukcyjne

Zmienna sterujące pętlą oraz zmienne o wartościach bezpośrednio od niej zależnych nazywane są zmiennymi indukcyjnymi. Jeśli występuje więcej niż jedna, można takie zmienne usunąć poza jedną. Ponieważ  $t_2$  jest związana z  $i$ , a  $t_4$  z  $j$ , warunek  $i \geq j$  w  $B_4$  można zastąpić warunkiem  $t_2 \geq t_4$ . Wtedy zmienne  $i$  oraz  $j$  stają się zbędne i można je usunąć począwszy od  $B_2$ .



## zmienne indukcyjne

Zmienna sterujące pętlą oraz zmienne o wartościach bezpośrednio od niej zależnych nazywane są zmiennymi indukcyjnymi. Jeśli występuje więcej niż jedna, można takie zmienne usunąć poza jedną. Ponieważ  $t_2$  jest związana z  $i$ , a  $t_4$  z  $j$ , warunek  $i \geq j$  w  $B_4$  można zastąpić warunkiem  $t_2 \geq t_4$ . Wtedy zmienne  $i$  oraz  $j$  stają się zbędne i można je usunąć począwszy od  $B_2$ .



## zmienne indukcyjne

Zmienna sterujące pętlą oraz zmienne o wartościach bezpośrednio od niej zależnych nazywane są zmiennymi indukcyjnymi. Jeśli występuje więcej niż jedna, można takie zmienne usunąć poza jedną. Ponieważ  $t_2$  jest związana z  $i$ , a  $t_4$  z  $j$ , warunek  $i \geq j$  w  $B_4$  można zastąpić warunkiem  $t_2 \geq t_4$ . Wtedy zmienne  $i$  oraz  $j$  stają się zbędne i można je usunąć począwszy od  $B_2$ .