

**Stefan Sokołowski**

# **JĘZYKI FORMALNE I METODY KOMPILACJI**

**wykład 13**

**Inst. Informatyki Stosowanej, ANS  
Elbląg, 2025/2026**

# **Automatyczna konstrukcja translatorów**

# Automatyczna konstrukcja translatorów

**GRAMATYKA**  $\longrightarrow$  **PARSER**

# Automatyczna konstrukcja translatorów

**GRAMATYKA**  $\longrightarrow$  **PARSER**

— to da się zrobić automatycznie

# Automatyczna konstrukcja translatorów

**GRAMATYKA**  $\longrightarrow$  **PARSER**

— to da się zrobić automatycznie

**GENEROWANIE PRZEKŁADU**

— z tym jest większy problem

# Automatyczna konstrukcja translatorów

**GRAMATYKA**  $\longrightarrow$  **PARSER**

— to da się zrobić automatycznie

**GENEROWANIE PRZEKŁADU**

— z tym jest większy problem, bo skąd system automatyczny ma wiedzieć, jakie znaczenie chcemy nadać składni?

# Automatyczna konstrukcja translatorów

**GRAMATYKA**  $\longrightarrow$  **PARSER**

— to da się zrobić automatycznie

**GENEROWANIE PRZEKŁADU**

— z tym jest większy problem, bo skąd system automatyczny ma wiedzieć, jakie znaczenie chcemy nadać składni?

Nadajemy znaczenie każdej **produkcji** z gramatyki osobno

# Automatyczna konstrukcja translatorów

## GRAMATYKA $\longrightarrow$ PARSER

— to da się zrobić automatycznie

## GENEROWANIE PRZEKŁADU

— z tym jest większy problem, bo skąd system automatyczny ma wiedzieć, jakie znaczenie chcemy nadać składni?

Nadajemy znaczenie każdej **produkcji** z gramatyki osobno — piszemy, co ma się dzieć, kiedy parser dokonuje odpowiadającej jej **redukcji**.

# Automatyczna konstrukcja translatorów

**GRAMATYKA**  $\longrightarrow$  **PARSER**

— to da się zrobić automatycznie

**GENEROWANIE PRZEKŁADU**

— z tym jest większy problem, bo skąd system automatyczny ma wiedzieć, jakie znaczenie chcemy nadać składni?

Nadajemy znaczenie każdej **produkcji** z gramatyki osobno — piszemy, co ma się dziać, kiedy parser dokonuje odpowiadającej jej **redukcji**.

**narzędzie: GRAMATYKA *ATRYBUTOWA***

# Gramatyki atrybutowe

*Gramatyka atrybutowa* — gramatyka bezkontekstowa wzbogacona o

- *atrybuty* symboli w drzewie wyvodu (terminali i nieterminali)

# Gramatyki atrybutowe

*Gramatyka atrybutowa* — gramatyka bezkontekstowa wzbogacona o

- *atrybuty* symboli w drzewie wyvodu (terminali i nieterminali); atrybuty mogą mieć wartości

# Gramatyki atrybutowe

***Gramatyka atrybutowa*** — gramatyka bezkontekstowa wzbogacona o

- ***atrybuty*** symboli w drzewie wyvodu (terminali i nieterminali); atrybuty mogą mieć wartości;
- ***funkcje obliczające*** wartości atrybutów.

# Gramatyki atrybutowe

**Gramatyka atrybutowa** — gramatyka bezkontekstowa wzbogacona o

- **atrybuty** symboli w drzewie wyvodu (terminali i nieterminali); atrybuty mogą mieć wartości;
- **funkcje obliczające** wartości atrybutów.



atrybut **syntetyzowany**:  
z dzieci do rodziców

# Gramatyki atrybutowe

**Gramatyka atrybutowa** — gramatyka bezkontekstowa wzbogacona o

- **atrybuty** symboli w drzewie wyvodu (terminali i nieterminali); atrybuty mogą mieć wartości;
- **funkcje obliczające** wartości atrybutów.



atrybut **syntetyzowany**:  
z dzieci do rodziców



atrybut **dziedziczony**:  
z rodziców do dzieci

# Gramatyki atrybutowe

**Przykład:** język  $\{a^n b^n c^n \mid n \geq 0\}$

# Gramatyki atrybutowe

**Przykład:** język  $\{a^n b^n c^n \mid n \geq 0\}$  —

nie daje się opisać gramatyką bezkontekstową

# Gramatyki atrybutowe

**Przykład:** język  $\{a^n b^n c^n \mid n \geq 0\}$  —

nie daje się opisać gramatyką bezkontekstową;  
daje się opisać gramatyką atrybutową.

# Gramatyki atrybutowe

**Przykład:** język  $\{a^n b^n c^n \mid n \geq 0\}$  —

nie daje się opisać gramatyką bezkontekstową;  
daje się opisać gramatyką atrybutową.

```
⟨słowo⟩ ::= ⟨część_a⟩⟨część_b⟩⟨część_c⟩  
⟨część_a⟩ ::= λ  
           | ⟨część_a⟩a  
⟨część_b⟩ ::= λ  
           | ⟨część_b⟩b  
⟨część_c⟩ ::= λ  
           | ⟨część_c⟩c
```

# Gramatyki atrybutowe

**Przykład:** język  $\{a^n b^n c^n \mid n \geq 0\}$  —

nie daje się opisać gramatyką bezkontekstową;  
daje się opisać gramatyką atrybutową.

```

⟨słowo⟩ ::= ⟨część_a⟩⟨część_b⟩⟨część_c⟩
⟨część_a⟩ ::= λ
              | ⟨część_a⟩a
⟨część_b⟩ ::= λ
              | ⟨część_b⟩b
⟨część_c⟩ ::= λ
              | ⟨część_c⟩c

```

W zwykłej gramatyce bezkontekstowej nie ma jak sprawdzić, czy długości tych trzech części są równe.

# Gramatyki atrybutowe

**Przykład:** język  $\{a^n b^n c^n \mid n \geq 0\}$  —

nie daje się opisać gramatyką bezkontekstową;  
daje się opisać gramatyką atrybutową.

```

⟨słowo⟩ ::= ⟨część_a⟩⟨część_b⟩⟨część_c⟩
⟨część_a⟩ ::= λ
              | ⟨część_a⟩a
⟨część_b⟩ ::= λ
              | ⟨część_b⟩b
⟨część_c⟩ ::= λ
              | ⟨część_c⟩c
  
```

W zwykłej gramatyce bezkontekstowej nie ma jak sprawdzić, czy długości tych trzech części są równe.

Należy wzbogacić gramatykę o:

- dla każdego nieterminalu  $\langle \text{część\_x} \rangle$  w drzewie wywodu — atrybut liczący, ile jest pod nim terminali  $x$

# Gramatyki atrybutowe

**Przykład:** język  $\{a^n b^n c^n \mid n \geq 0\}$  —

nie daje się opisać gramatyką bezkontekstową;  
daje się opisać gramatyką atrybutową.

```

⟨słowo⟩ ::= ⟨część_a⟩⟨część_b⟩⟨część_c⟩
⟨część_a⟩ ::= λ
              | ⟨część_a⟩a
⟨część_b⟩ ::= λ
              | ⟨część_b⟩b
⟨część_c⟩ ::= λ
              | ⟨część_c⟩c

```

W zwykłej gramatyce bezkontekstowej nie ma jak sprawdzić, czy długości tych trzech części są równe.

Należy wzbogacić gramatykę o:

- dla każdego nieterminalu  $\langle \text{część}_x \rangle$  w drzewie wyvodu — atrybut liczący, ile jest pod nim terminali  $x$
- w korzeniu — predykat sprawdzający równość tych atrybutów

# Gramatyki atrybutowe

Do nieterminali dodajemy atrybut **syntetyzowany** (od dzieci do rodziców)  
*ile*  
liczący litery w poddrzewie poniżej tego nieterminalu.

# Gramatyki atrybutowe

Do nieterminali dodajemy atrybut **syntetyzowany** (od dzieci do rodziców)  
*ile*

liczący litery w poddrzewie poniżej tego nieterminalu.

Wzbogacenia produkcji gramatycznych:

$\langle \text{część\_a} \rangle \rightarrow \lambda$

# Gramatyki atrybutowe

Do nieterminali dodajemy atrybut **syntetyzowany** (od dzieci do rodziców)  
*ile*

liczący litery w poddrzewie poniżej tego nieterminalu.

Wzbogacenia produkcji gramatycznych:

$\langle \text{część\_a} \rangle \rightarrow \lambda$

$\langle \text{część\_a} \rangle$

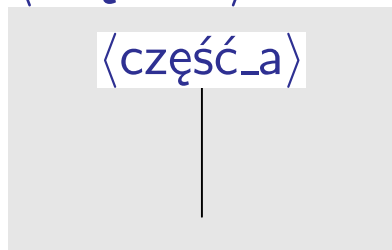


# Gramatyki atrybutowe

Do nieterminali dodajemy atrybut **syntetyzowany** (od dzieci do rodziców)  
*ile*

liczący litery w poddrzewie poniżej tego nieterminalu.  
 Wzbogacenia produkcji gramatycznych:

$\langle \text{część\_a} \rangle \rightarrow \lambda$

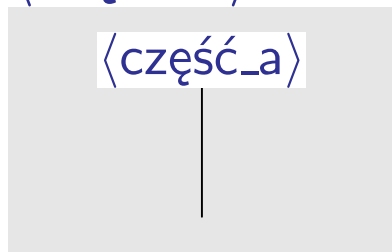


$\langle \text{część\_a} \rangle . \textit{ile} \leftarrow 0$

# Gramatyki atrybutowe

Do nieterminali dodajemy atrybut **syntetyzowany** (od dzieci do rodziców)  
*ile*

liczący litery w poddrzewie poniżej tego nieterminalu.  
Wzbogacenia produkcji gramatycznych:

$$\langle \text{część\_a} \rangle \rightarrow \lambda$$


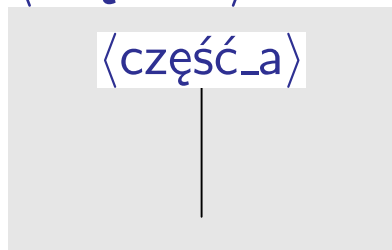
$$\langle \text{część\_a} \rangle . \textit{ile} \leftarrow 0$$

$$\langle \text{część\_a} \rangle \rightarrow \langle \text{część\_a} \rangle \textit{a}$$

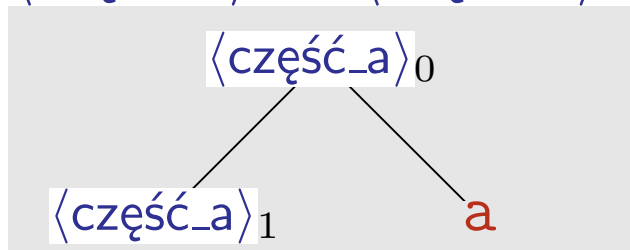
# Gramatyki atrybutowe

Do nieterminali dodajemy atrybut **syntetyzowany** (od dzieci do rodziców)  
*ile*

liczący litery w poddrzewie poniżej tego nieterminalu.  
Wzbogacenia produkcji gramatycznych:

$$\langle \text{część\_a} \rangle \rightarrow \lambda$$


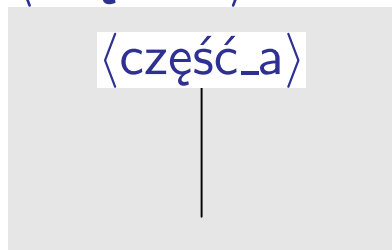
$$\langle \text{część\_a} \rangle. \textit{ile} \leftarrow 0$$

$$\langle \text{część\_a} \rangle \rightarrow \langle \text{część\_a} \rangle a$$


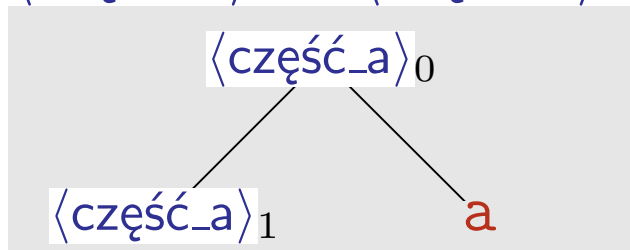
# Gramatyki atrybutowe

Do nieterminali dodajemy atrybut **syntetyzowany** (od dzieci do rodziców)  
*ile*

liczący litery w poddrzewie poniżej tego nieterminalu.  
Wzbogacenia produkcji gramatycznych:

$$\langle \text{część\_a} \rangle \rightarrow \lambda$$


$$\langle \text{część\_a} \rangle.ile \leftarrow 0$$

$$\langle \text{część\_a} \rangle \rightarrow \langle \text{część\_a} \rangle a$$


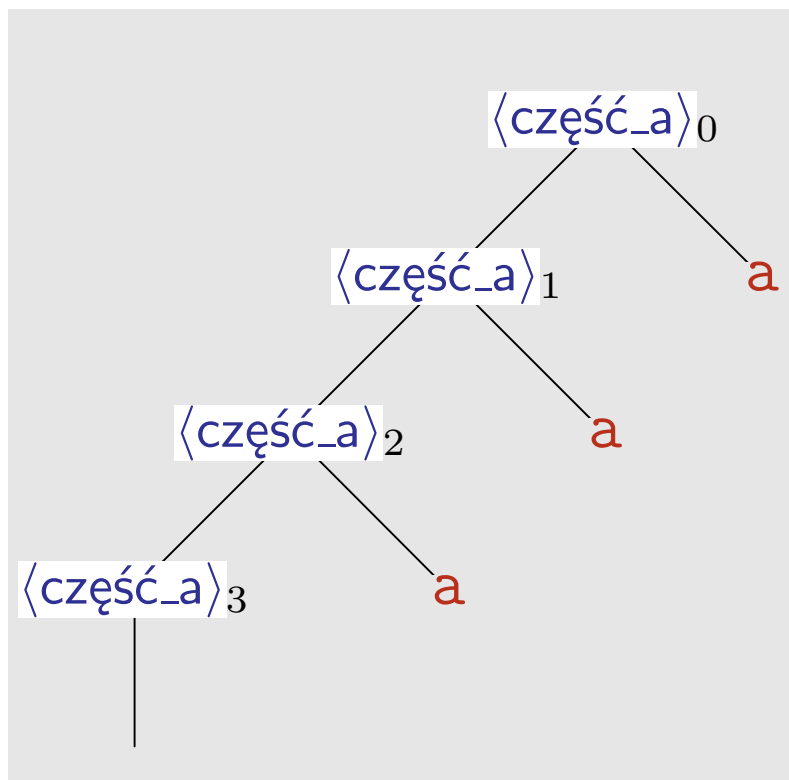
$$\langle \text{część\_a} \rangle_0.ile \leftarrow \langle \text{część\_a} \rangle_1.ile + 1$$

# Gramatyki atrybutowe

Funkcje obliczające powodują propagowanie wartości atrybutu po drzewie wyvodu

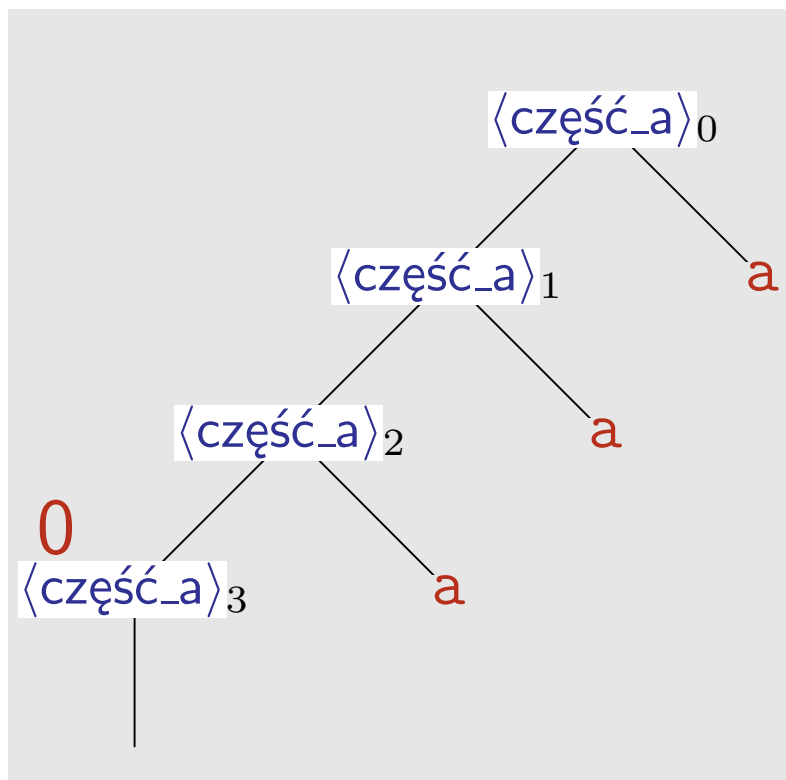
# Gramatyki atrybutowe

Funkcje obliczające powodują propagowanie wartości atrybutu po drzewie wywodu:



# Gramatyki atrybutowe

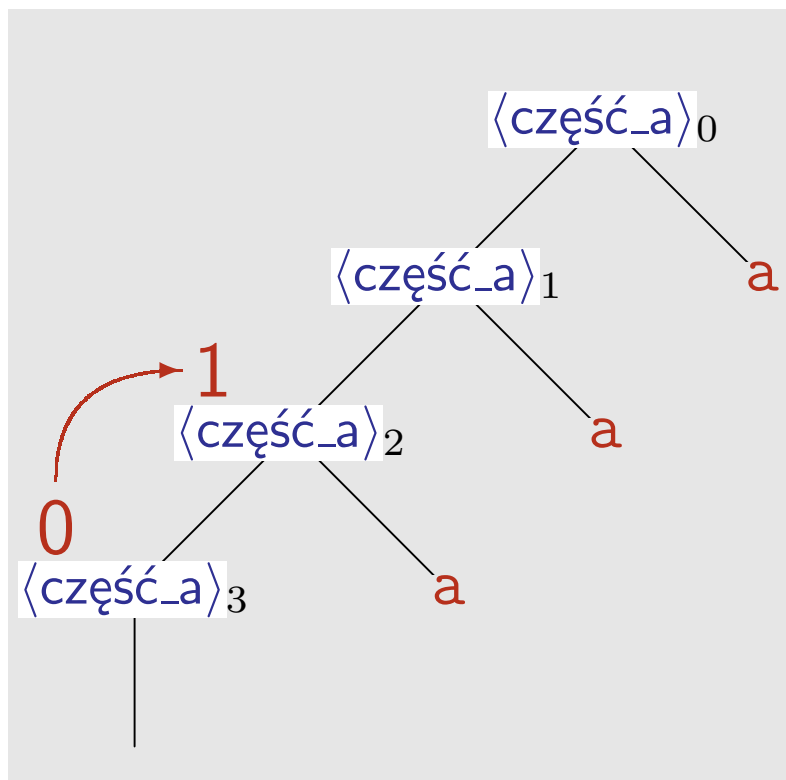
Funkcje obliczające powodują propagowanie wartości atrybutu po drzewie wyvodu:



$$\langle \text{część\_a} \rangle . \textit{ile} \leftarrow 0$$

# Gramatyki atrybutowe

Funkcje obliczające powodują propagowanie wartości atrybutu po drzewie wyvodu:

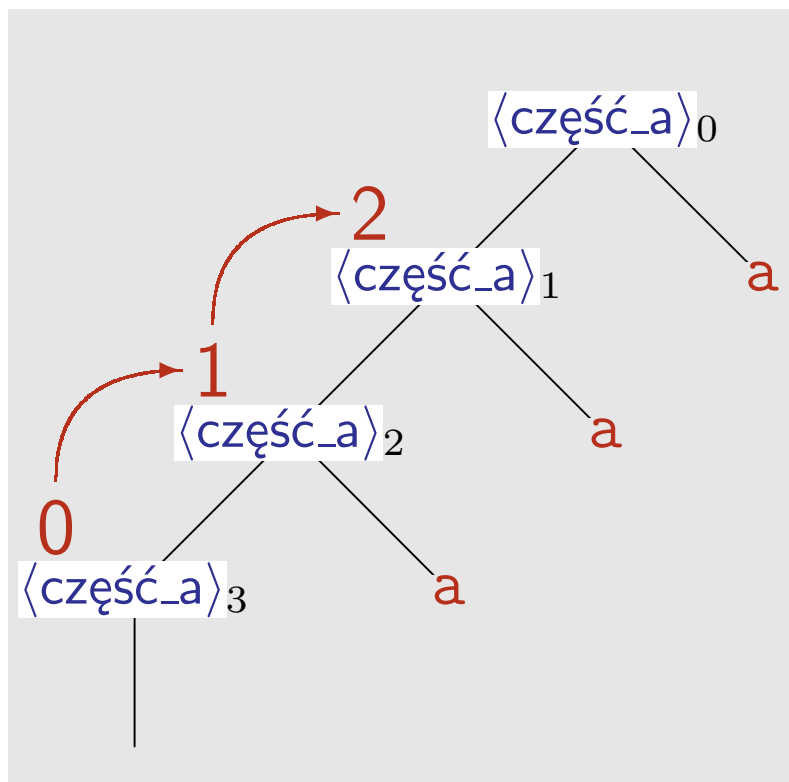


$$\langle \text{część\_a} \rangle_0.\textit{ile} \leftarrow \langle \text{część\_a} \rangle_1.\textit{ile} + 1$$

$$\langle \text{część\_a} \rangle.\textit{ile} \leftarrow 0$$

# Gramatyki atrybutowe

Funkcje obliczające powodują propagowanie wartości atrybutu po drzewie wyvodu:

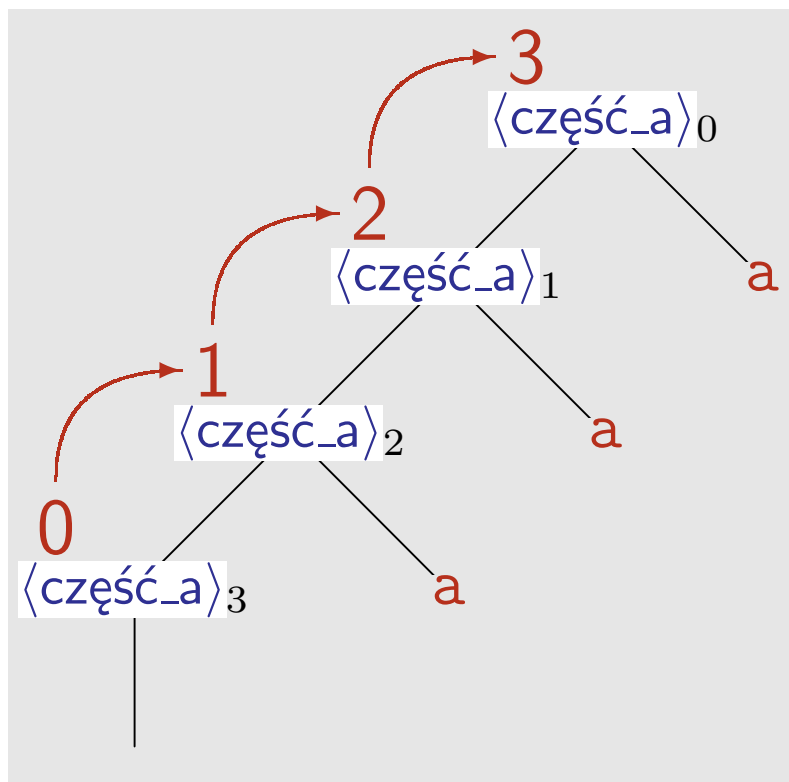


$$\langle \text{część\_a} \rangle_0.\textit{ile} \leftarrow \langle \text{część\_a} \rangle_1.\textit{ile} + 1$$

$$\langle \text{część\_a} \rangle.\textit{ile} \leftarrow 0$$

# Gramatyki atrybutowe

Funkcje obliczające powodują propagowanie wartości atrybutu po drzewie wywodu:



$$\langle \text{część\_a} \rangle_0.\textit{ile} \leftarrow \langle \text{część\_a} \rangle_1.\textit{ile} + 1$$

$$\langle \text{część\_a} \rangle.\textit{ile} \leftarrow 0$$

# Gramatyki atrybutowe

W korzeniu predykat sprawdzający, czy długości równe

# Gramatyki atrybutowe

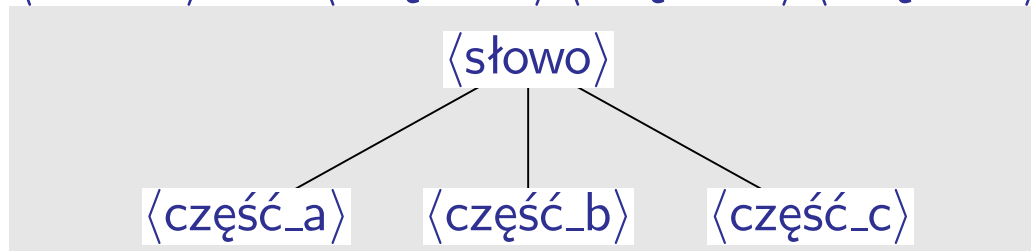
W korzeniu predykat sprawdzający, czy długości równe:

$$\langle \text{słowo} \rangle \rightarrow \langle \text{część\_a} \rangle \langle \text{część\_b} \rangle \langle \text{część\_c} \rangle$$

# Gramatyki atrybutowe

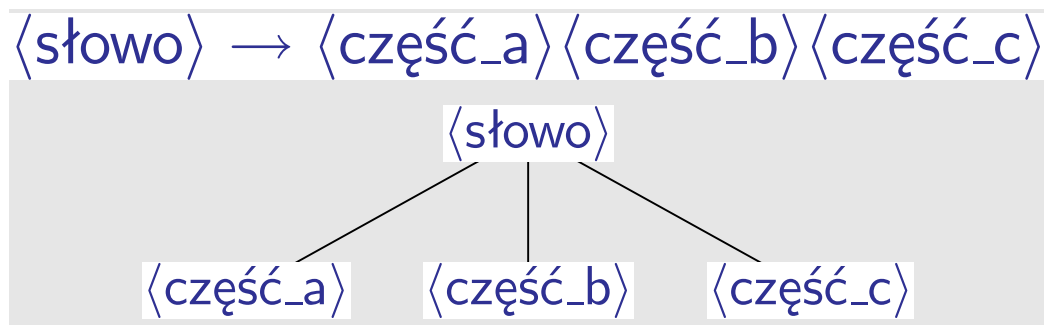
W korzeniu predykat sprawdzający, czy długości równe:

$\langle \text{słowo} \rangle \rightarrow \langle \text{część\_a} \rangle \langle \text{część\_b} \rangle \langle \text{część\_c} \rangle$



# Gramatyki atrybutowe

W korzeniu predykat sprawdzający, czy długości równe:



Czy  $\langle \text{część\_a} \rangle.\text{ile} = \langle \text{część\_b} \rangle.\text{ile}$   
 $= \langle \text{część\_c} \rangle.\text{ile} ?$

# Kompilator kompilatorów

# Kompilator kompilatorów

P  
R  
E  
K  
U  
R  
S  
O  
R

**YACC**

*Yet Another  
Compiler Compiler*

# Kompilator kompilatorów

P  
R  
E  
K  
U  
R  
S  
O  
R

**YACC**  
*Yet Another  
Compiler Compiler*

**BISON**  
[www.gnu.org/software/bison](http://www.gnu.org/software/bison)

B  
I  
S  
O  
N

# Kompilator kompilatorów

P  
R  
E  
K  
U  
R  
S  
O  
R

## YACC

*Yet Another  
Compiler Compiler*

## BISON

[www.gnu.org/software/bison](http://www.gnu.org/software/bison)  
stosuje atrybuty **syntetyzowane**  
(od liści do korzenia)

B  
I  
S  
O  
N

# Kompilator kompilatorów

P  
R  
E  
K  
U  
R  
S  
O  
R

**YACC**

*Yet Another  
Compiler Compiler*



**BISON**

[www.gnu.org/software/bison](http://www.gnu.org/software/bison)  
stosuje atrybuty **syntetyzowane**  
(od liści do korzenia)

B  
I  
S  
O  
N

# Kompilator kompilatorów

P  
R  
E  
K  
U  
R  
S  
O  
R**YACC***Yet Another  
Compiler Compiler***BISON**

[www.gnu.org/software/bison](http://www.gnu.org/software/bison)  
stosuje atrybuty **syntetyzowane**  
(od liści do korzenia)

B  
I  
S  
O  
N

# Kompilator kompilatorów

P  
R  
E  
K  
U  
R  
S  
O  
R**YACC***Yet Another  
Compiler Compiler***BISON**

[www.gnu.org/software/bison](http://www.gnu.org/software/bison)  
stosuje atrybuty **syntetyzowane**  
(od liści do korzenia)

B  
I  
S  
O  
N

# Działanie BISONa — rozpoznawanie $a^n b^n c^n$

```
slowo:  czesc_a czesc_b czesc_c '.'
```

```
    { if ($1 == $2 && $1 == $3) printf("\n  NALEZY.\n\n");
```

```
      else printf("\n  NIE NALEZY.\n\n");
```

```
      exit(1);
```

```
    } ;
```

```
czesc_a:  /* puste */  { $$ = 0; }
```

```
        |  czesc_a 'a'  { $$ = $1+1; } ;
```

```
czesc_b:  /* puste */  { $$ = 0; }
```

```
        |  czesc_b 'b'  { $$ = $1+1; } ;
```

```
czesc_c:  /* puste */  { $$ = 0; }
```

```
        |  czesc_c 'c'  { $$ = $1+1; } ;
```

# Działanie BISONa — rozpoznawanie $a^n b^n c^n$

```
slovo:  czesc_a czesc_b czesc_c '.'
```

```
{ if ($1 == $2 && $1 == $3) printf("\n  NALEZY.\n\n");
  else printf("\n  NIE NALEZY.\n\n");
  exit(1);
} ;
```

```
czesc_a:  /* puste */ { $$ = 0; }
          | czesc_a 'a' { $$ = $1+1; } ;

czesc_b:  /* puste */ { $$ = 0; }
          | czesc_b 'b' { $$ = $1+1; } ;

czesc_c:  /* puste */ { $$ = 0; }
          | czesc_c 'c' { $$ = $1+1; } ;
```

— gramatyka

# Działanie BISONa — rozpoznawanie $a^n b^n c^n$

```
slovo:  czesc_a czesc_b czesc_c '.'
```

```
{ if ($1 == $2 && $1 == $3) printf("\n  NALEZY.\n\n");
  else printf("\n  NIE NALEZY.\n\n");
  exit(1);
} ;
```

```
czesc_a:  /* puste */
          |  czesc_a 'a'
```

```
czesc_b:  /* puste */
          |  czesc_b 'b'
```

```
czesc_c:  /* puste */
          |  czesc_c 'c'
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

 — gramatyka

 — działania na atrybutach

# Działanie BISONa — rozpoznawanie $a^n b^n c^n$

```
slovo:  czesc_a czesc_b czesc_c '.'
```

```
{ if ($1 == $2 && $1 == $3) printf("\n  NALEZY.\n\n");
  else printf("\n  NIE NALEZY.\n\n");
  exit(1);
} ;
```

```
czesc_a:  /* puste */
          |  czesc_a 'a'
```

```
czesc_b:  /* puste */
          |  czesc_b 'b'
```

```
czesc_c:  /* puste */
          |  czesc_c 'c'
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

```
{ $$ = 0; }
{ $$ = $1+1; } ;
```

 — gramatyka

 — działania na atrybutach

# Działanie BISONa — przetwarzanie

plik z gramatyką aabbcc.y :

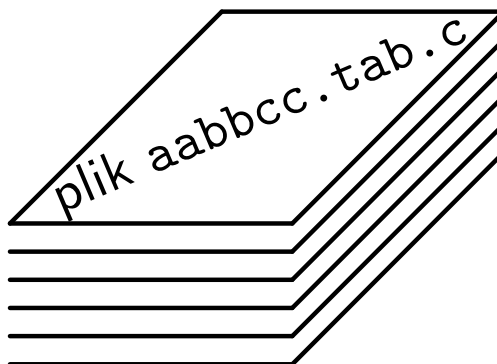
```
słowo: czesc_a czesc_b czesc_c '.' { ... }  
czesc_a: /* puste */ { ... }  
...
```

# Działanie BISONa — przetwarzanie

plik z gramatyką aabbcc.y :

```
slovo: czesc_a czesc_b czesc_c '.' { ... }  
czesc_a: /* puste */ { ... }  
...
```

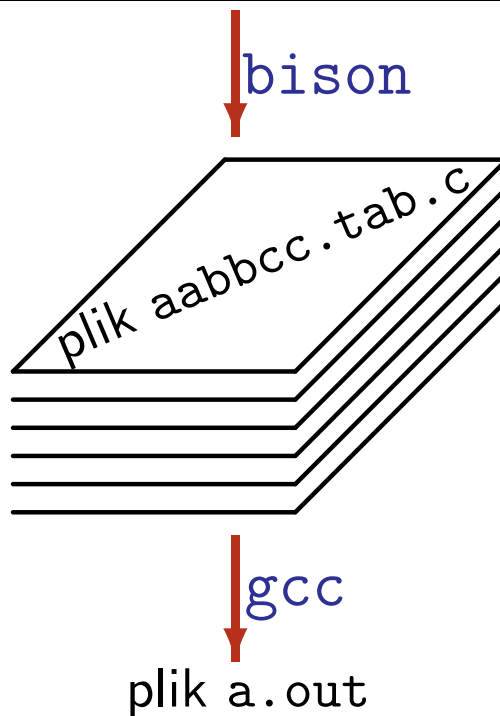
bison



# Działanie BISONa — przetwarzanie

plik z gramatyką aabbcc.y :

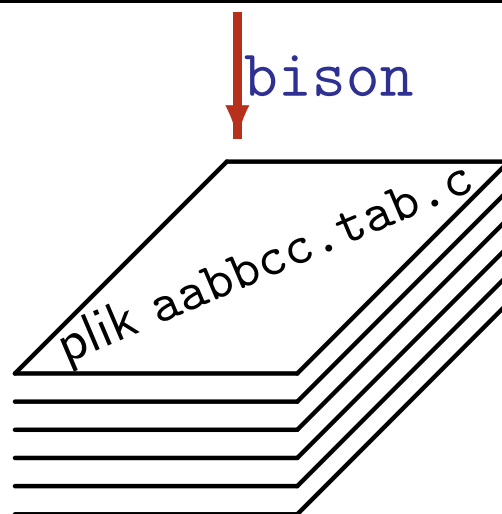
```
slowo: czesc_a czesc_b czesc_c '.' { ... }  
czesc_a: /* puste */ { ... }  
...
```



# Działanie BISONa — przetwarzanie

plik z gramatyką aabbcc.y :

```
slowo: czesc_a czesc_b czesc_c '.' { ... }  
czesc_a: /* puste */ { ... }  
...
```



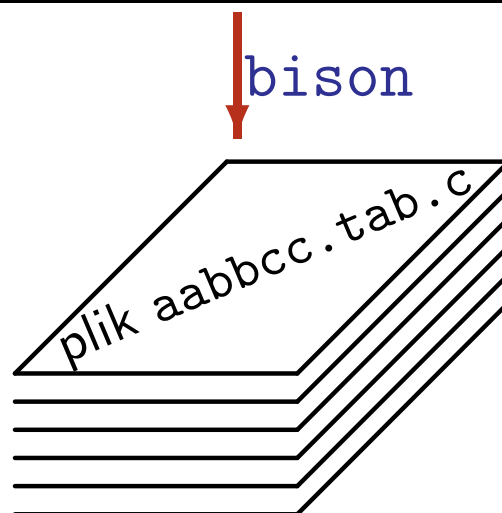
plik a.out

wejście

# Działanie BISONa — przetwarzanie

plik z gramatyką aabbcc.y :

```
slowo: czesc_a czesc_b czesc_c '.' { ... }  
czesc_a: /* puste */ { ... }  
...
```



wejście

wyjście

# Przykład: kalkulator

$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$

$\langle \text{komendy} \rangle ::=$   
                                   (puste)  
                                    $\langle \text{komendy} \rangle \langle \text{komenda} \rangle$

$\langle \text{komenda} \rangle ::=$   
                                    $\langle \text{zmienna} \rangle = \langle \text{wyrażenie} \rangle ;$   
                                    $\langle \text{wyrażenie} \rangle ;$

$\langle \text{wyrażenie} \rangle ::=$   
                                    $\langle \text{składnik} \rangle \mid - \langle \text{składnik} \rangle$   
                                    $\langle \text{wyrażenie} \rangle + \langle \text{składnik} \rangle \mid \langle \text{wyrażenie} \rangle - \langle \text{składnik} \rangle$

$\langle \text{składnik} \rangle ::=$   
                                    $\langle \text{potega} \rangle \mid \langle \text{składnik} \rangle * \langle \text{potega} \rangle$   
                                    $\langle \text{składnik} \rangle / \langle \text{potega} \rangle$

$\langle \text{potega} \rangle ::= \langle \text{podstawa} \rangle \mid \langle \text{podstawa} \rangle ^ \langle \text{potega} \rangle$

$\langle \text{podstawa} \rangle ::= \langle \text{liczba} \rangle \mid \langle \text{zmienna} \rangle \mid ( \langle \text{wyrażenie} \rangle )$

$\langle \text{liczba} \rangle ::= \dots$

$\langle \text{zmienna} \rangle ::= \dots$

# Przykład: kalkulator

$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$

$\langle \text{komendy} \rangle ::=$   
                                   (puste)  
                                    $\langle \text{komendy} \rangle \langle \text{komenda} \rangle$

$\langle \text{komenda} \rangle ::=$   
                                    $\langle \text{zmienna} \rangle = \langle \text{wyrażenie} \rangle ;$   
                                    $\langle \text{wyrażenie} \rangle ;$

$\langle \text{wyrażenie} \rangle ::=$   
                                    $\langle \text{składnik} \rangle \mid - \langle \text{składnik} \rangle$   
                                    $\langle \text{wyrażenie} \rangle + \langle \text{składnik} \rangle \mid \langle \text{wyrażenie} \rangle - \langle \text{składnik} \rangle$

$\langle \text{składnik} \rangle ::=$   
                                    $\langle \text{potega} \rangle \mid \langle \text{składnik} \rangle * \langle \text{potega} \rangle$   
                                    $\langle \text{składnik} \rangle / \langle \text{potega} \rangle$

$\langle \text{potega} \rangle ::= \langle \text{podstawa} \rangle \mid \langle \text{podstawa} \rangle ^ \langle \text{potega} \rangle$

$\langle \text{podstawa} \rangle ::= \langle \text{liczba} \rangle \mid \langle \text{zmienna} \rangle \mid ( \langle \text{wyrażenie} \rangle )$

$\langle \text{liczba} \rangle ::= \dots$

$\langle \text{zmienna} \rangle ::= \dots$

←  $\left\{ \begin{array}{l} \$\$ = \text{numer zmiennej:} \\ a == 0 \\ b == 1 \\ \dots \end{array} \right\}$

# Przykład: kalkulator

$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$

$\langle \text{komendy} \rangle ::=$   
                                   (puste)  
                                    $\langle \text{komendy} \rangle \langle \text{komenda} \rangle$

$\langle \text{komenda} \rangle ::= \langle \text{zmienna} \rangle = \langle \text{wyrażenie} \rangle ; \quad \leftarrow \{ \text{pamięć}[\$1] = \$3; \}$   
                                    $\langle \text{wyrażenie} \rangle ;$

$\langle \text{wyrażenie} \rangle ::= \langle \text{składnik} \rangle \mid - \langle \text{składnik} \rangle$   
                                    $\langle \text{wyrażenie} \rangle + \langle \text{składnik} \rangle \mid \langle \text{wyrażenie} \rangle - \langle \text{składnik} \rangle$

$\langle \text{składnik} \rangle ::= \langle \text{potega} \rangle \mid \langle \text{składnik} \rangle * \langle \text{potega} \rangle$   
                                    $\langle \text{składnik} \rangle / \langle \text{potega} \rangle$

$\langle \text{potega} \rangle ::= \langle \text{podstawa} \rangle \mid \langle \text{podstawa} \rangle ^ \langle \text{potega} \rangle$

$\langle \text{podstawa} \rangle ::= \langle \text{liczba} \rangle \mid \langle \text{zmienna} \rangle \mid ( \langle \text{wyrażenie} \rangle )$

$\langle \text{liczba} \rangle ::= \dots$

$\langle \text{zmienna} \rangle ::= \dots$

$\left\{ \begin{array}{l} \$\$ = \text{numer zmiennej:} \\ a == 0 \\ b == 1 \\ \dots \end{array} \right\}$

$\langle \text{liczba} \rangle ::= \dots$   
 $\langle \text{zmienna} \rangle ::= \dots$

← {
   
 $\$ \$ = \text{numer zmiennej:}$ 
  
 $a == 0$ 
  
 $b == 1$ 
  
 $\dots$ 
 }

# Przykład: kalkulator

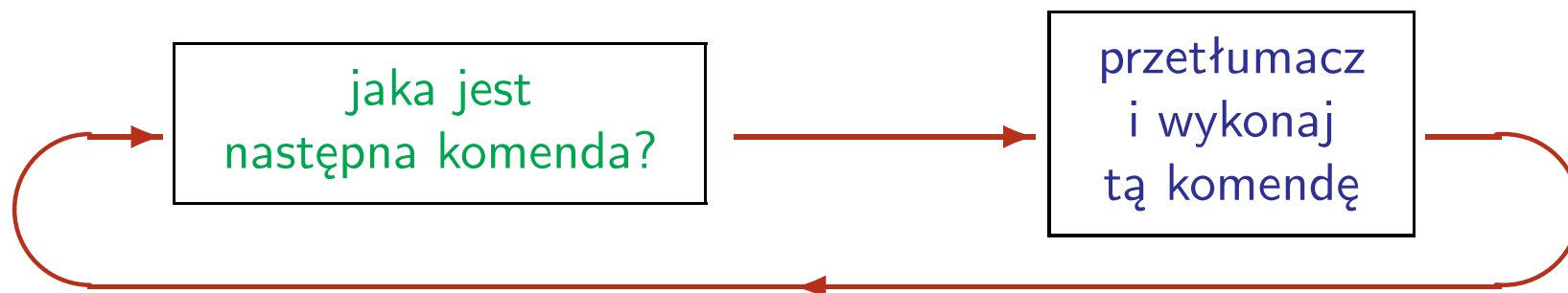
$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$   
 $\langle \text{komendy} \rangle ::=$   
     |  $\langle \text{komendy} \rangle \langle \text{komenda} \rangle$   
 $\langle \text{komenda} \rangle ::= \langle \text{zmienna} \rangle = \langle \text{wyrażenie} \rangle ; \quad \leftarrow \{ \text{pamięć}[\$1] = \$3; \}$   
     |  $\langle \text{wyrażenie} \rangle ;$   
 $\langle \text{wyrażenie} \rangle ::= \langle \text{składnik} \rangle \mid - \langle \text{składnik} \rangle$   
     |  $\langle \text{wyrażenie} \rangle + \langle \text{składnik} \rangle \mid \langle \text{wyrażenie} \rangle - \langle \text{składnik} \rangle$   
 $\langle \text{składnik} \rangle ::= \langle \text{potega} \rangle \mid \langle \text{składnik} \rangle * \langle \text{potega} \rangle$   
     |  $\langle \text{składnik} \rangle / \langle \text{potega} \rangle$   
 $\langle \text{potega} \rangle ::= \langle \text{podstawa} \rangle \mid \langle \text{podstawa} \rangle ^ \langle \text{potega} \rangle \quad \leftarrow \{ \$\$ = \text{pamięć}[\$1]; \}$   
 $\langle \text{podstawa} \rangle ::= \langle \text{liczba} \rangle \mid \langle \text{zmienna} \rangle \mid ( \langle \text{wyrażenie} \rangle )$   
 $\langle \text{liczba} \rangle ::= \dots$   
 $\langle \text{zmienna} \rangle ::= \dots \quad \leftarrow \left\{ \begin{array}{l} \$\$ = \text{numer zmiennej:} \\ a == 0 \\ b == 1 \\ \dots \end{array} \right\}$

# Interpretacja a kompilacja

**Interpreter** — superprogram wykonujący poszczególne komendy programu jedna za drugą

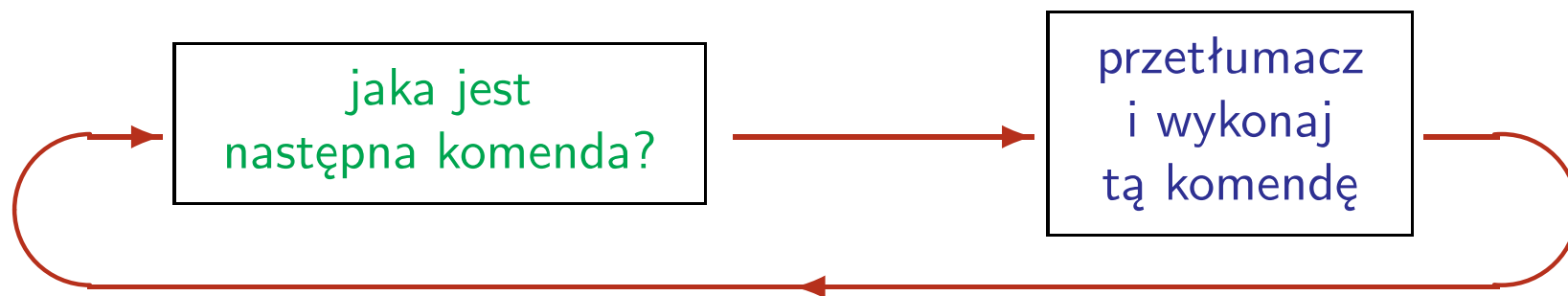
# Interpretacja a kompilacja

**Interpreter** — superprogram wykonujący poszczególne komendy programu jedna za drugą:



# Interpretacja a kompilacja

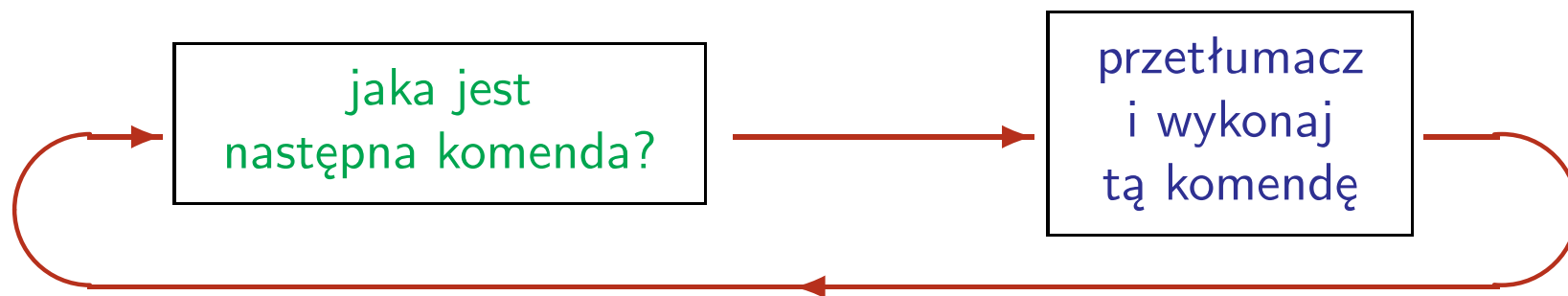
**Interpreter** — superprogram wykonujący poszczególne komendy programu jedna za drugą:



**Skutek:** wynik obliczenia.

# Interpretacja a kompilacja

**Interpreter** — superprogram wykonujący poszczególne komendy programu jedna za drugą:

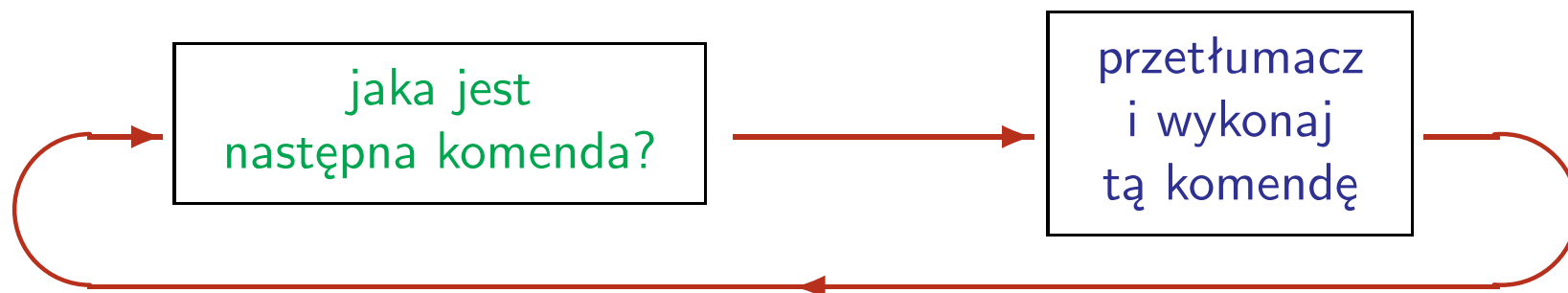


**Skutek:** wynik obliczenia.

**Kompilator** — superprogram tłumaczący cały program na język wewnętrzny, przygotowujący go do wykonania

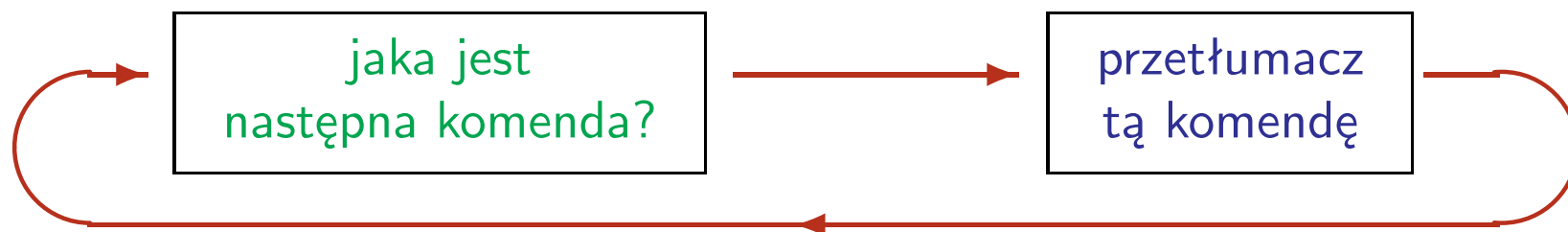
# Interpretacja a kompilacja

**Interpreter** — superprogram wykonujący poszczególne komendy programu jedna za drugą:



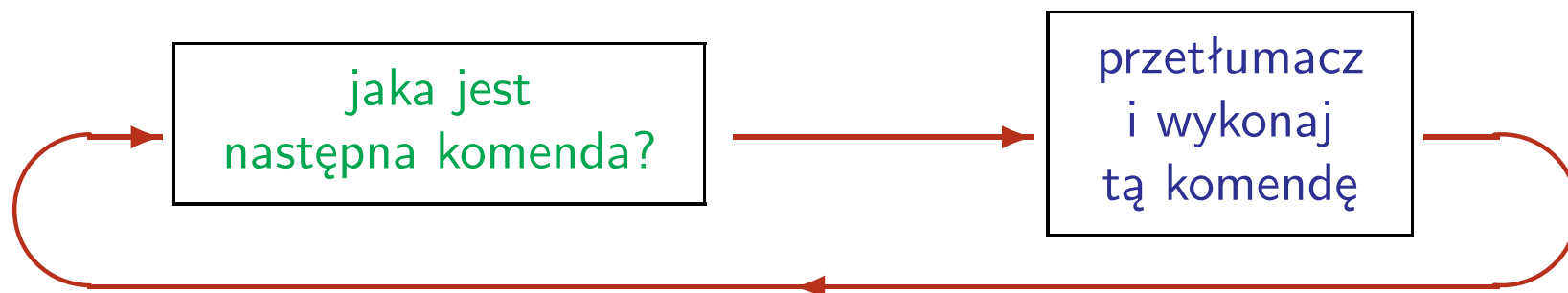
**Skutek:** wynik obliczenia.

**Kompilator** — superprogram tłumaczący cały program na język wewnętrzny, przygotowujący go do wykonania:



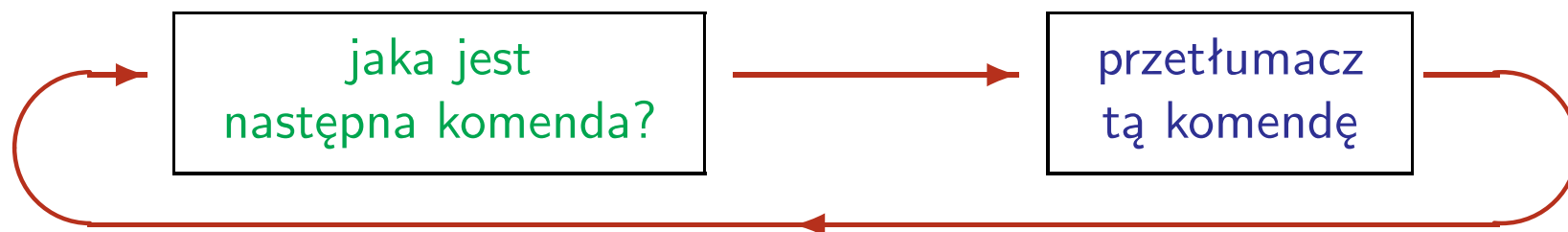
# Interpretacja a kompilacja

**Interpreter** — superprogram wykonujący poszczególne komendy programu jedna za drugą:



**Skutek:** wynik obliczenia.

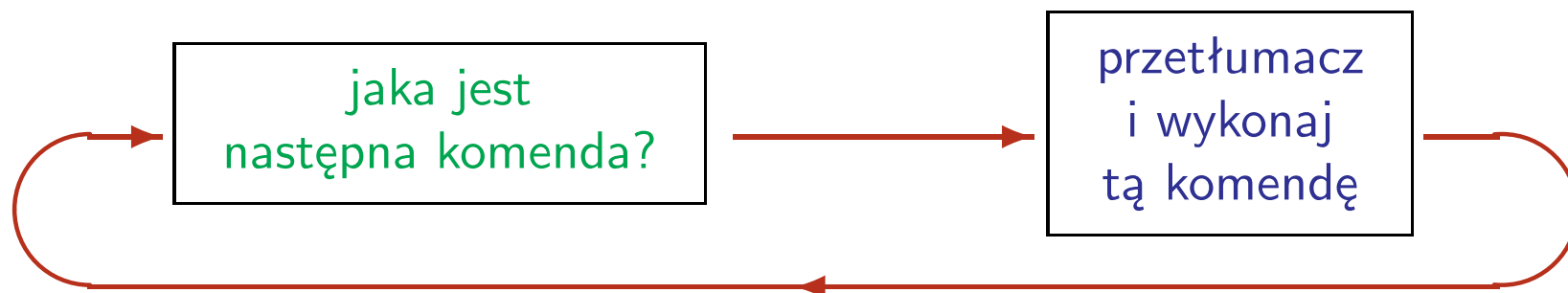
**Kompilator** — superprogram tłumaczący cały program na język wewnętrzny, przygotowujący go do wykonania:



**Skutek:** program w języku wewnętrznym gotowy do wykonania:

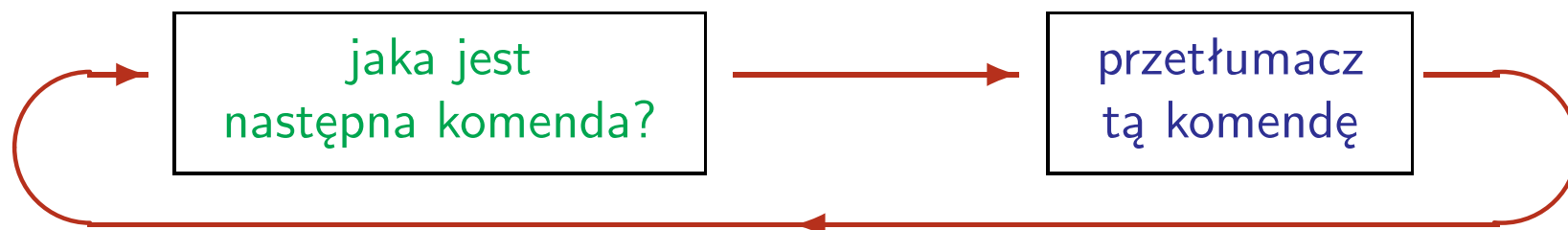
# Interpretacja a kompilacja

**Interpreter** — superprogram wykonujący poszczególne komendy programu jedna za drugą:



**Skutek:** wynik obliczenia.

**Kompilator** — superprogram tłumaczący cały program na język wewnętrzny, przygotowujący go do wykonania:

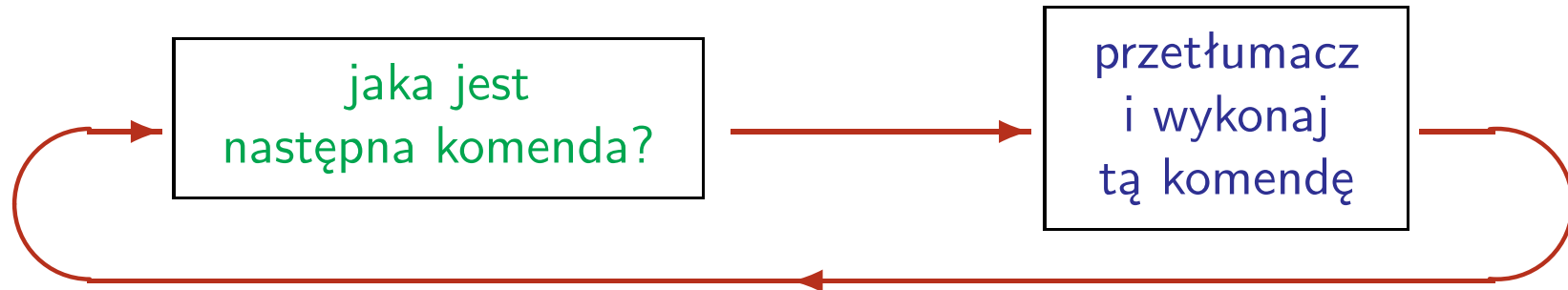


**Skutek:** program w języku wewnętrznym gotowy do wykonania:

Komendy wewnątrz pętli interpreter od nowa tłumaczy za każdym obrotem pętli

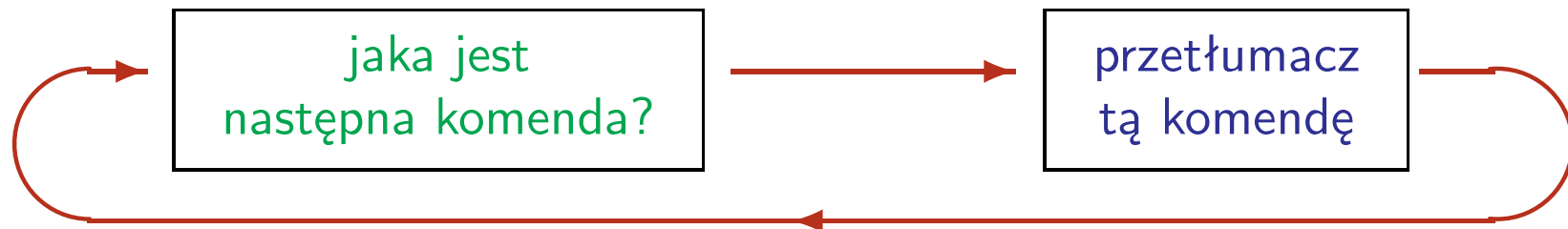
# Interpretacja a kompilacja

**Interpreter** — superprogram wykonujący poszczególne komendy programu jedna za drugą:



**Skutek:** wynik obliczenia.

**Kompilator** — superprogram tłumaczący cały program na język wewnętrzny, przygotowujący go do wykonania:



**Skutek:** program w języku wewnętrznym gotowy do wykonania:

Komendy wewnątrz pętli **interpreter** od nowa tłumaczy za każdym obrotem pętli; a **kompilator** tłumaczy raz, a potem przekład tylko wielokrotnie je wykonuje.

# Działanie BISONa — kompilacja

**Przykład:** „języczek”

$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$

$\langle \text{komendy} \rangle ::=$  (puste)  
 $\quad | \quad \langle \text{komendy} \rangle \langle \text{komenda} \rangle$

$\langle \text{komenda} \rangle ::=$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{liczba} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{liczba} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{liczba} \rangle ;$   
 $\quad ( \langle \text{zmienna} \rangle ) \{ \langle \text{komendy} \rangle \}$  (pętla)

$\langle \text{liczba} \rangle ::= \dots$

$\langle \text{zmienna} \rangle ::= \dots$

# Działanie BISONa — kompilacja

**Przykład:** „języczek”

$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$

$\langle \text{komendy} \rangle ::=$  (puste)  
 $\quad \mid \langle \text{komendy} \rangle \langle \text{komenda} \rangle$

$\langle \text{komenda} \rangle ::=$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{liczba} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{liczba} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{liczba} \rangle ;$   
 $\quad ( \langle \text{zmienna} \rangle ) \{ \langle \text{komendy} \rangle \}$

$\langle \text{liczba} \rangle ::= \dots$

$\langle \text{zmienna} \rangle ::= \dots$

(pętla)

↑  
 wyjaśnione dalej

# Działanie BISONa — kompilacja

Pętla:  $(x) \{ \dots \}$  — dopóki  $x \neq 0$ , wykonuj  $\dots$

# Działanie BISONa — kompilacja

Pętla:  $(x) \{ \dots \}$  — dopóki  $x \neq 0$ , wykonuj  $\dots$

## Przykłady programów:

```
n=100; s=0;  
(n) {  
    s=s+n; n=n-1;  
}  
.
```

# Działanie BISONa — kompilacja

Pętla:  $(x) \{ \dots \}$  — dopóki  $x \neq 0$ , wykonuj  $\dots$

## Przykłady programów:

```
n=100; s=0;
```

```
(n) {  
    s=s+n; n=n-1;  
}
```

```
.
```

← dopóki  $n \neq 0$

# Działanie BISONa — kompilacja

Pętla:  $(x) \{ \dots \}$  — dopóki  $x \neq 0$ , wykonuj  $\dots$

## Przykłady programów:

```
n=100; s=0;
```

```
(n) {  
    s=s+n; n=n-1;  
}
```

```
.
```

← dopóki  $n \neq 0$

→ oznacza

```
n=100; s=0;
```

```
while (n!=0) {  
    s=s+n; n=n-1;  
}
```

# Działanie BISONa — kompilacja

Pętla:  $(x) \{ \dots \}$  — dopóki  $x \neq 0$ , wykonuj  $\dots$

## Przykłady programów:

```
n=100; s=0;
```

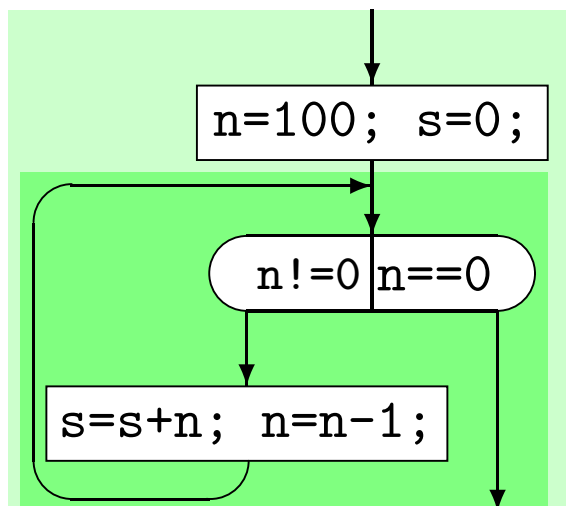
```
(n) {  
    s=s+n; n=n-1;  
}
```

dopóki  $n \neq 0$

oznacza

```
n=100; s=0;
```

```
while (n!=0) {  
    s=s+n; n=n-1;  
}
```



# Działanie BISONa — kompilacja

Pętla:  $(x) \{ \dots \}$  — dopóki  $x \neq 0$ , wykonuj  $\dots$

## Przykłady programów:

```
n=100; s=0;
```

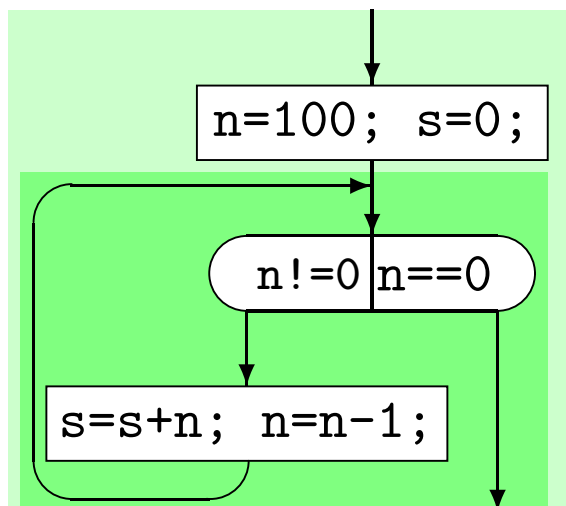
```
(n) {  
    s=s+n; n=n-1;  
}
```

← dopóki  $n \neq 0$

→ oznacza

```
n=100; s=0;
```

```
while (n!=0) {  
    s=s+n; n=n-1;  
}
```



— wylicza  $s = \sum_{n=1}^{100} n$

# Działanie BISONa — kompilacja

```
p=2; w=10; s=1;  
(w) {  
  r=p; t=0;  
  (r) { t=t+s; r=r-1; }  
  s=t; w=w-1;  
}  
.
```

# Działanie BISONa — kompilacja

```
p=2; w=10; s=1;  
(w) {  
  r=p; t=0;  
  (r) { t=t+s; r=r-1; }  
  s=t; w=w-1;  
}  
.
```

# Działanie BISONa — kompilacja

```
p=2; w=10; s=1;  
(w) {  
  r=p; t=0;  
  (r) { t=t+s; r=r-1; }  
  s=t; w=w-1;  
}  
.
```

pętla  
wewnętrzna

# Działanie BISONa — kompilacja

```
p=2; w=10; s=1;  
(w) {  
  r=p; t=0;  
  (r) { t=t+s; r=r-1; }  
  s=t; w=w-1;  
}
```



```
p=2; w=10; s=1;  
while (w!=0) {  
  r=p; t=0;  
  while (r!=0) {  
    t=t+s; r=r-1;  
  }  
  s=t; w=w-1;  
}
```

pętla  
wewnętrzna

# Działanie BISONa — kompilacja

```
p=2; w=10; s=1;
```

```
(w) {
```

```
  r=p; t=0;
```

```
  (r) { t=t+s; r=r-1; }
```

```
  s=t; w=w-1;
```

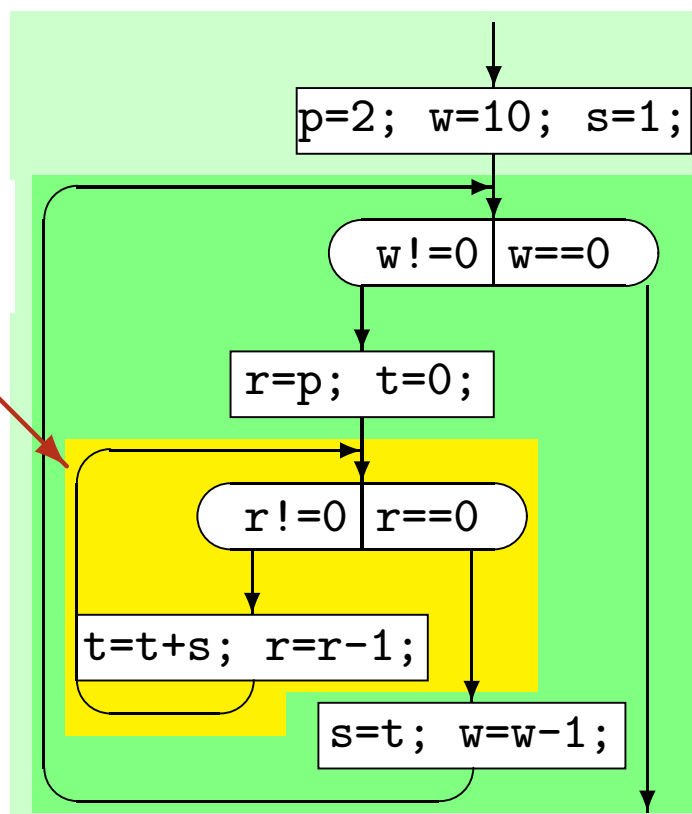
```
}
```

```
.
```

oznacza

```
p=2; w=10; s=1;
while (w!=0) {
  r=p; t=0;
  while (r!=0) {
    t=t+s; r=r-1;
  }
  s=t; w=w-1;
}
```

pętla  
wewnętrzna



# Działanie BISONa — kompilacja

```
p=2; w=10; s=1;
```

```
(w) {
```

```
  r=p; t=0;
```

```
  (r) { t=t+s; r=r-1; }
```

```
  s=t; w=w-1;
```

```
}
```

```
.
```

oznacza

```
p=2; w=10; s=1;
```

```
while (w!=0) {
```

```
  r=p; t=0;
```

```
  while (r!=0) {
```

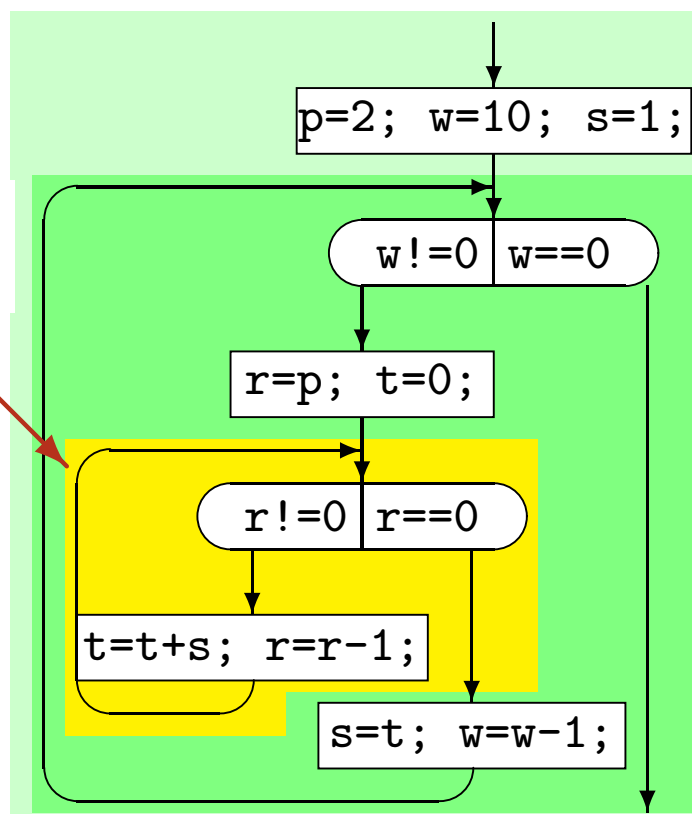
```
    t=t+s; r=r-1;
```

```
  }
```

```
  s=t; w=w-1;
```

```
}
```

pętla  
wewnętrzna



— wylicza  $s = 2^{10}$

# Działanie BISONa — kompilacja

Jak przetwarzać?

# Działanie BISONa — kompilacja

Jak przetwarzać?

kompil.y

Gramatyka języczka

+

fcje na atrybutach

# Działanie BISONa — kompilacja

## Jak przetwarzać?

kompil.y

Gramatyka języczka

+

fcje na atrybutach

bison  
→

kompil.tab.c

program w C

# Działanie BISONa — kompilacja

## Jak przetwarzać?



# Działanie BISONa — kompilacja

## Jak przetwarzać?



prog1  $\xrightarrow{\text{a.out}}$  przeklad

# Działanie BISONa — kompilacja

## Jak przetwarzać?



prog1  $\xrightarrow{\text{a.out}}$  przeklad

prog2  $\xrightarrow{\text{a.out}}$  przeklad

# Działanie BISONa — kompilacja

## Jak przetwarzać?



**prog1**  $\xrightarrow{\text{a.out}}$  **przeklad**

**prog2**  $\xrightarrow{\text{a.out}}$  **przeklad**

Te **przeklady** to już gotowe programy w języku „wewnętrzny”, gotowe do wykonania przy pomocy symulatora języka „wewnętrznego”.

# Działanie BISONa — kompilacja

## Jak przetwarzać?



prog1  $\xrightarrow{\text{a.out}}$  przeklad

prog2  $\xrightarrow{\text{a.out}}$  przeklad

Te **przeklady** to już gotowe programy w języku „wewnętrzny”, gotowe do wykonania przy pomocy symulatora języka „wewnętrznego”.

**Jak napisać funkcje na atrybutach?**

# Działanie BISONa — kompilacja

**Przykład:** „języczek”

$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$

$\langle \text{komendy} \rangle ::=$   
                   |  $\langle \text{komendy} \rangle \langle \text{komenda} \rangle$

$\langle \text{komenda} \rangle ::=$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{zmienna} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{zmienna} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{liczba} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{liczba} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{liczba} \rangle ;$   
                   |  $( \langle \text{zmienna} \rangle ) \{ \langle \text{komendy} \rangle \}$

$\langle \text{liczba} \rangle ::= \dots$

$\langle \text{zmienna} \rangle ::= \dots$

# Działanie BISONa — kompilacja

**Przykład:** „języczek”

$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$

$\langle \text{komendy} \rangle ::=$   
 $\quad | \quad \langle \text{komendy} \rangle \langle \text{komenda} \rangle$

$\langle \text{komenda} \rangle ::=$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{zmienna} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{liczba} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{liczba} \rangle ;$   
 $\quad \langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{liczba} \rangle ;$   
 $\quad ( \langle \text{zmienna} \rangle ) \{ \langle \text{komendy} \rangle \}$

$\langle \text{liczba} \rangle ::= \dots$

$\langle \text{zmienna} \rangle ::= \dots$

Do produkcji gramatycznych należy dopisać:

- atrybuty

# Działanie BISONa — kompilacja

**Przykład:** „języczek”

$\langle \text{program} \rangle ::= \langle \text{komendy} \rangle .$

$\langle \text{komendy} \rangle ::=$   
                   |  $\langle \text{komendy} \rangle \langle \text{komenda} \rangle$

$\langle \text{komenda} \rangle ::=$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{zmienna} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{zmienna} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{liczba} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle + \langle \text{liczba} \rangle ;$   
                   |  $\langle \text{zmienna} \rangle = \langle \text{zmienna} \rangle - \langle \text{liczba} \rangle ;$   
                   |  $( \langle \text{zmienna} \rangle ) \{ \langle \text{komendy} \rangle \}$

$\langle \text{liczba} \rangle ::= \dots$

$\langle \text{zmienna} \rangle ::= \dots$

Do produkcji gramatycznych należy dopisać:

- atrybuty,
- funkcje działające na tych atrybutach.

# Działanie BISONa — kompilacja

Co wyrażają atrybuty?

# Działanie BISONa — kompilacja

Co wyrażają atrybuty? {

- dla cyfry i liczby: wartość;

# Działanie BISONa — kompilacja

- Co wyrażają atrybuty?
- dla cyfry i liczby: wartość;
  - dla zmiennej: adres zmiennej;

# Działanie BISONa — kompilacja

- Co wyrażają atrybuty?
- dla cyfry i liczby: wartość;
  - dla zmiennej: adres zmiennej;
  - dla komendy: liczba komórek przekładu.

# Działanie BISONa — kompilacja

Co wyrażają atrybuty? {

- dla cyfry i liczby: wartość;
- dla zmiennej: adres zmiennej;
- dla komendy: liczba komórek przekładu.

## Przykłady:

komenda: zmienna '=' zmienna ';' ;

# Działanie BISONa — kompilacja

Co wyrażają atrybuty? {

- dla cyfry i liczby: wartość;
- dla zmiennej: adres zmiennej;
- dla komendy: liczba komórek przekładu.

## Przykłady:

komenda: zmienna '=' zmienna ';' ;

spacja  
↓  
{napisać „loada” , \$3  
napisać „store” , \$1  
\$\$ = 2; }

# Działanie BISONa — kompilacja

Co wyrażają atrybuty? {

- dla cyfry i liczby: wartość;
- dla zmiennej: adres zmiennej;
- dla komendy: liczba komórek przekładu.

## Przykłady:

komenda: zmienna '=' zmienna ';' ;

spacja  
↓  
{napisać „loada” , \$3  
napisać „store” , \$1  
\$\$ = 2; }

| zmienna '=' zmienna '+' zmienna ';' ;

# Działanie BISONa — kompilacja

Co wyrażają atrybuty? {

- dla cyfry i liczby: wartość;
- dla zmiennej: adres zmiennej;
- dla komendy: liczba komórek przekładu.

## Przykłady:

komenda: zmienna '=' zmienna ';' ;

spacja  
↓  
{napisać „loada ” , \$3  
napisać „store ” , \$1  
\$\$ = 2; }

| zmienna '=' zmienna '+' zmienna ';' ; {napisać „loada ” , \$3  
napisać „add ” , \$5  
napisać „store ” , \$1  
\$\$ = 3; }

# Działanie BISONa — kompilacja

Co wyrażają atrybuty? {

- dla cyfry i liczby: wartość;
- dla zmiennej: adres zmiennej;
- dla komendy: liczba komórek przekładu.

## Przykłady:

komenda: zmienna '=' zmienna ';' ;

spacja  
↓  
{napisać „loada ” , \$3  
napisać „store ” , \$1  
\$\$ = 2; }

| zmienna '=' zmienna '+' zmienna ';' ; {napisać „loada ” , \$3  
napisać „add ” , \$5  
napisać „store ” , \$1  
\$\$ = 3; }

| zmienna '=' zmienna '+' liczba ';' ;

# Działanie BISONa — kompilacja

Co wyrażają atrybuty? {

- dla cyfry i liczby: wartość;
- dla zmiennej: adres zmiennej;
- dla komendy: liczba komórek przekładu.

## Przykłady:

komenda: zmienna '=' zmienna ';' ;

{napisać „loada” , \$3  
napisać „store” , \$1  
\$\$ = 2; }

spacja



| zmienna '=' zmienna '+' zmienna ';' ;

{napisać „loada” , \$3  
napisać „add” , \$5  
napisać „store” , \$1  
\$\$ = 3; }

| zmienna '=' zmienna '+' liczba ';' ;

{napisać „loadn” , \$5  
napisać „add” , \$3  
napisać „store” , \$1  
\$\$ = 3; }

# Działanie BISONa — kompilacja

Co wyrażają atrybuty? {

- dla cyfry i liczby: wartość;
- dla zmiennej: adres zmiennej;
- dla komendy: liczba komórek przekładu.

## Przykłady:

komenda: zmienna '=' zmienna ';' ;

{napisać „loada ” , \$3  
napisać „store ” , \$1  
\$\$ = 2; }

spacja



| zmienna '=' zmienna '+' zmienna ';' ;

{napisać „loada ” , \$3  
napisać „add ” , \$5  
napisać „store ” , \$1  
\$\$ = 3; }

| zmienna '=' zmienna '+' liczba ';' ;

{napisać „loadn ” , \$5  
napisać „add ” , \$3  
napisać „store ” , \$1  
\$\$ = 3; }