

Stefan Sokołowski

JĘZYKI FORMALNE I METODY KOMPILACJI

wykład 13

Inst. Informatyki Stosowanej, ANS
Elbląg, 2025/2026

Wykład 13, str. 1

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); atrybuty mogą mieć wartości;
- **funkcje obliczające** wartości atrybutów.



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 **<część_x>** 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

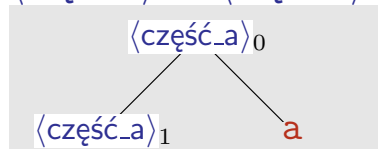
liczący litery w poddrzewie poniżej tego nieterminalu.
Wzbogacenia produkcji gramatycznych:

$\langle \text{część_a} \rangle \rightarrow \lambda$



$$\langle \text{część_a} \rangle.ile \leftarrow 0$$

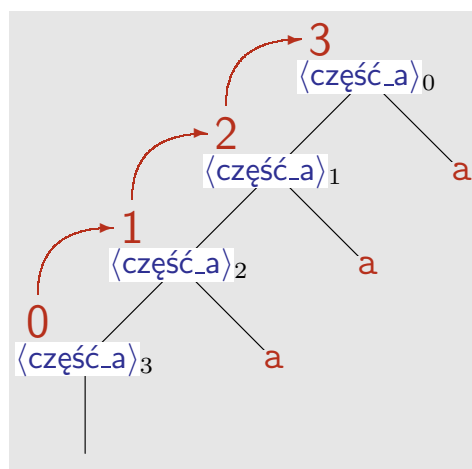
$\langle \text{część_a} \rangle \rightarrow \langle \text{część_a} \rangle a$



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

Gramatyki atrybutowe

Funkcje obliczające powodują propagowanie wartości atrybutu po drzewie wywodu:

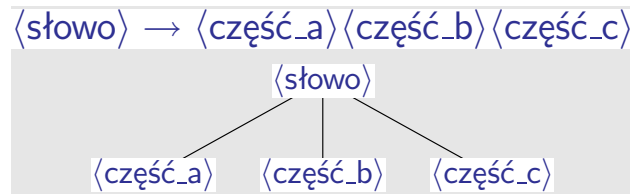


$$\langle \text{część_a} \rangle_0.ile \leftarrow \langle \text{część_a} \rangle_1.ile + 1$$

$$\langle \text{część_a} \rangle.ile \leftarrow 0$$

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

P
R
E
K
U
R
S
O
R

YACC
 Yet Another
 Compiler Compiler



BISON

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



B
I
S
O
N

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

```
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'
```

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

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

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

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

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

— gramatyka

— działania na atrybutach

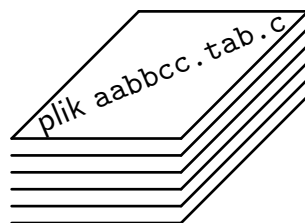
iis.ans-elblag.pl/~stefan/Dydaktyka/JezForm/Wyklady/13-bizony/1-aabbcc.y

Działanie BISONa — przetwarzanie

plik z gramatyką aabbcc.y :

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

bison



gcc

plik a.out

wejscie

wyjście

Przykład: kalkulator

```

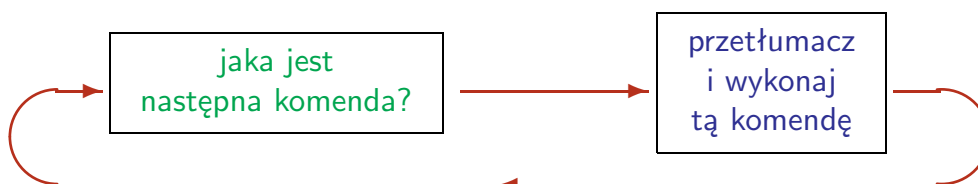
<program> ::= <komendy> .
<komendy> ::= (puste)
              | <komendy> <komenda>
<komenda> ::= <zmienna> = <wyrażenie> ; ← { pamięć[$1] = $3; }
              | <wyrażenie> ;
<wyrażenie> ::= <składnik> | - <składnik>
              | <wyrażenie> + <składnik> | <wyrażenie> - <składnik>
<składnik> ::= <potega> | <składnik> * <potega>
              | <składnik> / <potega>
<potega> ::= <podstawa> | <podstawa> ^ <potega> ← { $$ = pamięć[$1]; }
<podstawa> ::= <liczba> | <zmienna> | ( <wyrażenie> )
<liczba> ::= ...
<zmienna> ::= ... ← { $$ = numer zmiennej:
                      a == 0
                      b == 1
                      ...
                    }

```

iis.ans-elblag.pl/~stefan/Dydaktyka/JezForm/Wyklady/13-bizony/3-pjo.y

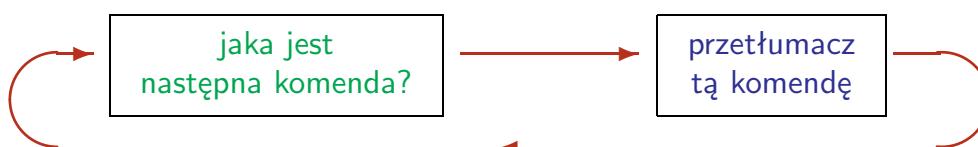
Interpretacja a kompilacja

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”

```

<program> ::= <komendy> .
<komendy> ::=
    | <komendy> <komenda>
<komenda> ::= <zmienna> = <zmienna> ;
               <zmienna> = <zmienna> + <zmienna> ;
               <zmienna> = <zmienna> - <zmienna> ;
               <zmienna> = <liczba> ;
               <zmienna> = <zmienna> + <liczba> ;
               <zmienna> = <zmienna> - <liczba> ;
               ( <zmienna> ) { <komendy> }
<liczba> ::= ...
<zmienna> ::= ...
  
```

(puste)

(pętla)

wyjaśnione dalej

Działanie BISONa — kompilacja

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

Przykłady programów:

```

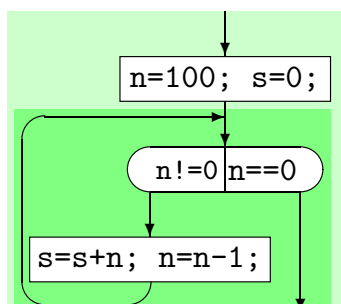
n=100; s=0;
(n) {
    s=s+n; n=n-1;
}
.
  
```

dopóki $n \neq 0$

oznacza

```

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



— wylicza $s = \sum_{n=1}^{100} n$

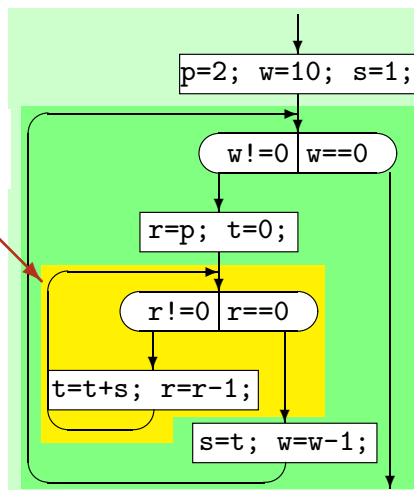
Działanie BISONa — kompilacja

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

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?



prog1 $\xrightarrow{a.out}$ przekład

prog2 $\xrightarrow{a.out}$ przekład

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

Jak napisać funkcje na atrybutach?

Działanie BISONa — kompilacja

Przykład: „języczek”

```

<program> ::= <komendy> .
<komendy> ::=
|
<komenda> <komendy>
<komenda> ::=
<zmienna> = <zmienna> ;
<zmienna> = <zmienna> + <zmienna> ;
<zmienna> = <zmienna> - <zmienna> ;
<zmienna> = <liczba> ;
<zmienna> = <zmienna> + <liczba> ;
<zmienna> = <zmienna> - <liczba> ;
( <zmienna> ) { <komendy> }

<liczba> ::= ...
<zmienna> ::= ...

```

Do produkcji gramatycznych należy dopisać:

- atrybuty,
- funkcje działające na tych atrybutach.

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