Metody Kompilacji
Wykład 7
Analiza Syntaktyczna

Parsowanie

Parsowanie jest to proces określenia jak ciąg terminali może być generowany przez gramatykę.

Parsowanie

- Dla każdej gramatyki bezkontekstowej istnieje parser, dla którego czas parsowania ciągu z *n* terminali zajmuje co najwyżej *O(n^3)* czasu.
- Ale czas ten jest zazwyczaj nie do przyjęcia w praktyce.
- Na szczęście, dla rzeczywistych języków programowania, możemy zaprojektować gramatykę, która może szybko parsować.

Parsowanie

Dla większości języków programowania zostały opracowane algorytmy parsowania o złożoności liniowej.

Parsowanie

Analizatory składni języka programowania prawie zawsze przeglądają ciąg wejściowy od lewej do prawej strony, biorąc pod uwagę tylko jeden terminal w danym momencie.

Parsowanie

- Większość metod parsowania należy do jednej z dwóch klas: metody zstępujące (*top-down*) i metody wstępujące (*bottom-up*).

- Terminy te odnoszą się do kolejności, w jakiej są budowane węzły w drzewie parsowania.

Parsowanie

- W parserach zstępujących (*top-down*), tworzenie drzewa rozpoczyna się od korzenia i polega na przechodzeniu do liści, natomiast w parserach wstępujących(*bottom-up*) budowa rozpoczyna się od liści i przebiega w kierunku korzenia.

Parsowanie

Popularność parserów zstępujących jest spowodowana tym, że łatwo mogą być zbudowane wydajne parsery przez mały nakład pracy programisty.

Parsowanie

Parser wstępujący może obsłużyć jednak więcej bardziej złożonych gramatyk i schematów translacji, czyli jego zakres stosowalności jest szerszy.

Parsowanie zstępujące

W parsowaniu zstępującym budowa drzewa syntaktycznego dokonuje się począwszy od korzenia, oznakowanego przez nieterminal startowy, i następnie wielokrotnie są wykonywane następujące dwa kroki.

Parsowanie zstępujące

1. W węźle *N*, oznaczonym nieterminalem *A*, wybierz jedną z produkcji, której lewa strona jest *A*, i utwórz dzieci dla *N* oznaczone symbolami prawej strony tej produkcji.
2. Znajdź następny węzeł, w którym poddrzewo ma być utworzone, zazwyczaj jest to skrajny lewy nieterminal drzewa bieżącego.

Parsowanie zstępujące

Aktualny symbol terminalny z ciągu wejściowego jest często nazywany symbolem bieżącym -- *lookahead.*

Początkowo symbolem bieżącym jest pierwszy terminal z lewej strony ciągu wejściowego.

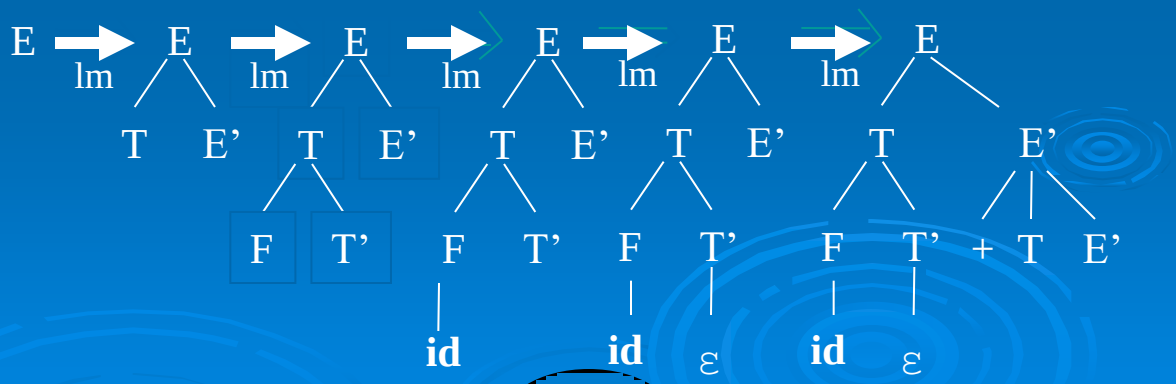
Parsowanie zstępujące

Parser próbuje utworzyć drzewo parsujące od korzenia w kierunku liści skanując wejście od lewej do prawej strony.

Przykład: dla wejścia id+id*id jest tworzone drzewo parsowania:

lm oznacza wyprowadzenie lewostronne, czyli zawsze jest wybierany skrajny lewy nieterminal

E -> TE’
E’ -> +TE’ | ε
T -> FT’
T’ -> *FT’ | ε
F -> (E) | **id**



Parsowanie zstępujące: tworzenie kodu parsera

1. Tworzymy jedną procedurę dla wszystkich symboli terminalnych
2. Tworzymy jedną procedurę dla każdego symbolu pomocniczego
3. Tworzymy kod dla każdej produkcji jako ciąg procedur zgodnie z kolejnością symboli w prawej stronie odpowiedniej produkcji.

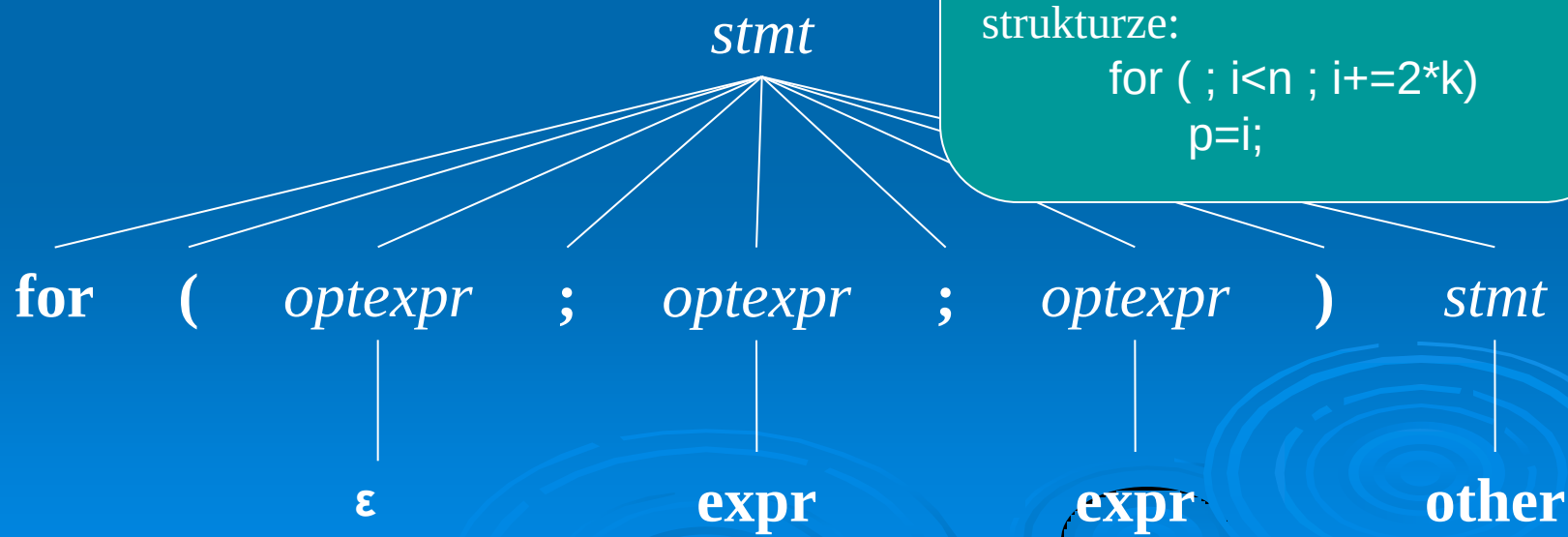
Parsowanie zstępujące: przykład

stmt → **expr ;**
| **if (expr)** *stmt*
| **for (** *optexpr* **;** *optexpr* **;** *optexpr* **)** *stmt*
| **other**

optexpr → ε
| **expr**

Drzewo parsowania dla produkcji *stmt*->**for (** *optexpr***;** *optexpr* **;** *optexpr* **)** *stmt* dla kodu o przykładowej strukturze:

```
for ( ; i<n ; i+=2*k)
    p=i;
```



Pseudokod parsera, który sprawdza czy ciąg wejściowy zawiera błędy syntaktyczne

stmt → **expr ;**
| **if (expr)** *stmt*
| **for (** *optexpr* **;** *optexpr* **;** *optexpr* **)** *stmt*
| **other**

```
void stmt() {
    switch ( lookahead ) {
        case expr:
          match(expr); match(';'); break;
        case if:
          match(if); match('(');
          match(expr); match(')');
          stmt(); break;
        case for:
          match(for); match('(');
          optexpr(); match(';');
          optexpr(); match(';');
          optexpr(); match(')');
          stmt(); break;
        case other:
          match(other); break;
        default:
          report("syntax error");
    }
}
```

Zastosowanie ε-produkcji:

optexpr → ε | **expr**

```
void optexpr() {
    if ( lookahead == expr ) match(expr);
}
```

```
void match(terminal t) {
    if ( lookahead == t )
        lookahead = nextTerminal;
    else
        report("syntax error");
}
```

Sprawdzenie pasowania terminala t do symbolu wejścia

Pseudokod:

stmt → **expr ;**
| **if (expr)** *stmt*
| **for (** *optexpr* **;** *optexpr* **;** *optexpr* **)** *stmt*
| **other**

```
void stmt() {
    switch ( lookahead ) {
        case expr:
          match(expr); match(';'); break;
        case if:
          match(if); match('(');
          match(expr); match(')');
          stmt(); break;
        case for:
          match(for); match('(');
          optexpr(); match(';');
          optexpr(); match(';');
          optexpr(); match(')');
          stmt(); break;
        case other:
          match(other); break;
        default:
          report("syntax error");
    }
}
```

Zastosowanie ε-produkcji:

optexpr → ε | **expr**

```
void optexpr() {
    if ( lookahead == expr ) match(expr);
}
```

Jeśli wartość lookahead równa się expr, to wywołaj funkcje match(expr)

```
void match(terminal t) {
    if ( lookahead == t )
        lookahead = nextTerminal;
    else
        report("syntax error");
}
```

Pseudokod:

stmt → **expr ;**
| **if (expr)** *stmt*
| **for (** *optexpr* **;** *optexpr* **;** *optexpr* **)** *stmt*
| **other**

```
void stmt() {
    switch ( lookahead ) {
        case expr:
          match(expr); match(';'); break;
        case if:
          match(if); match('(');
          match(expr); match(')');
          stmt(); break;
        case for:
          match(for); match('(');
          optexpr(); match(';');
          optexpr(); match(';');
          optexpr(); match(')');
          stmt(); break;
        case other:
          match(other); break;
        default:
          report("syntax error");
    }
}
```

Zastosowanie ε-produkcji:

optexpr → ε | **expr**

```
void optexpr() {
    if ( lookahead == expr ) match(expr);
}
```

Jeśli terminal t jest taki sam jak wartość lookahead, to wczytaj następny znak, inaczej jest błąd składni

```
void match(terminal t) {
    if ( lookahead == t )
        lookahead = nextTerminal;
    else
        report("syntax error");
}
```

Pseudokod:

Kolejność wywołania funkcji jest taka sama jak i kolejność symboli w prawej stronie odpowiedniej produkcji

stmt → **expr ;**
| **if (expr)** *stmt*
| **for (** *optexpr* **;** *optexpr* **;** *optexpr* **)** *stmt*
| **other**

```
void stmt() {
    switch ( lookahead ) {
        case expr:
            match(expr); match(';'); break;
        case if:
            match(if); match('(');
            match(expr); match(')');
            stmt(); break;
        case for:
            match(for); match('(');
            optexpr(); match(';');
            optexpr(); match(';');
            optexpr(); match(')');
            stmt(); break;
        case other:
            match(other); break;
        default:
            report("syntax error");
    }
}
```

Zastosowanie ε-produkcji:

optexpr → ε | **expr**

```
void optexpr() {
    if ( lookahead == expr ) match(expr);
}
```

```
void match(terminal t) {
    if ( lookahead == t )
        lookahead = nextTerminal;
    else
        report("syntax error");
}
```

expr ;

| **if (expr)** *stmt*

for (*optexpr* **;** *optexpr* **;** *optexpr* **)** *stmt*

stmt → **expr ;**
| **if (expr)** *stmt*
| **for (** *optexpr* **;** *optexpr* **;** *optexpr* **)** *stmt*
| **other**

other

```
void stmt() {
   switch ( lookahead ) {
      case expr:
        match(expr); match(';'); break;
      case if:
        match(if); match('(');
        match(expr); match(')');
        stmt(); break;
      case for:
        match(for); match('(');
        optexpr(); match(';');
        optexpr(); match(';');
        optexpr(); match(')');
        stmt(); break;
      case other:
        match(other); break;
      default:
        report("syntax error");
   }
}
```

Zastosowanie ε-produkcji:

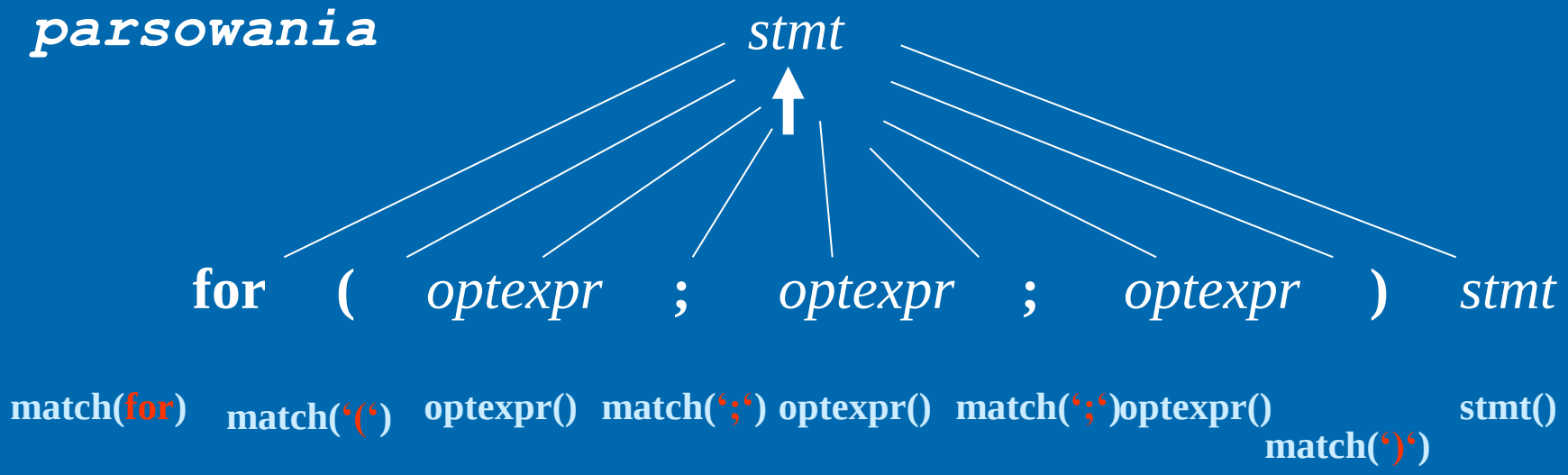
optexpr → ε | **expr**

```
void optexpr() {
   if ( lookahead == expr ) match(expr);
}
```

```
void match(terminal t) {
   if ( lookahead == t )
      lookahead = nextTerminal;
   else
      report("syntax error");
}
```

Parsowanie zstępujące: przykład

Drzewo parsowania



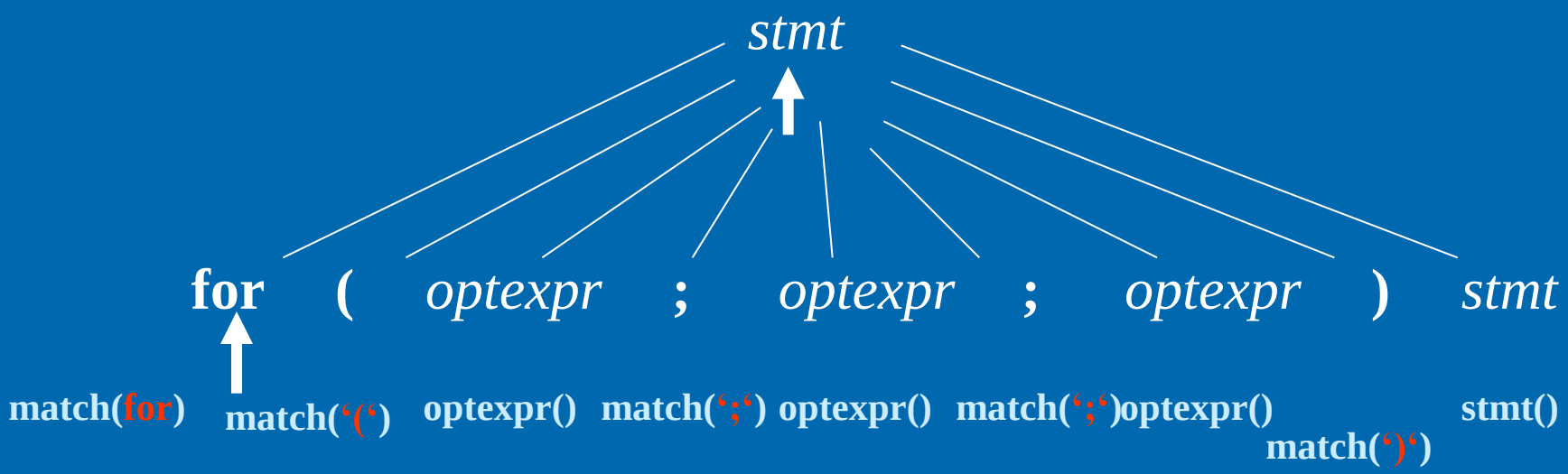
Wejście

for (; expr ; expr) other

lookahead

Parsowanie zstępujące: przykład

Krok 1:



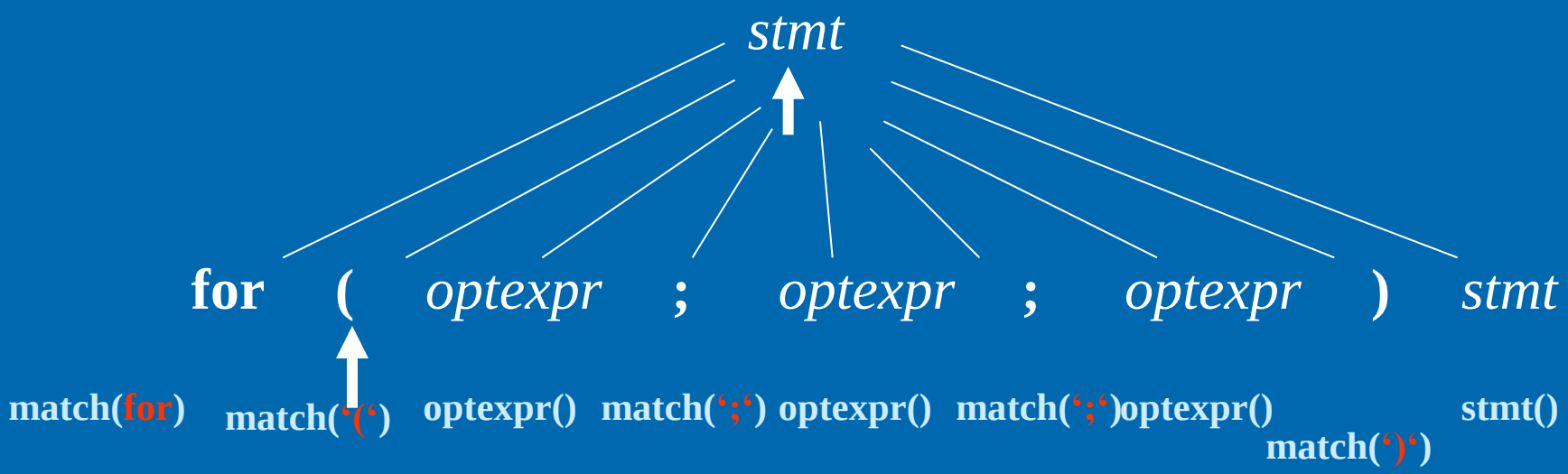
Wejście

for (; expr ; expr) other

lookahead

Parsowanie zstępujące: przykład

Krok 2:

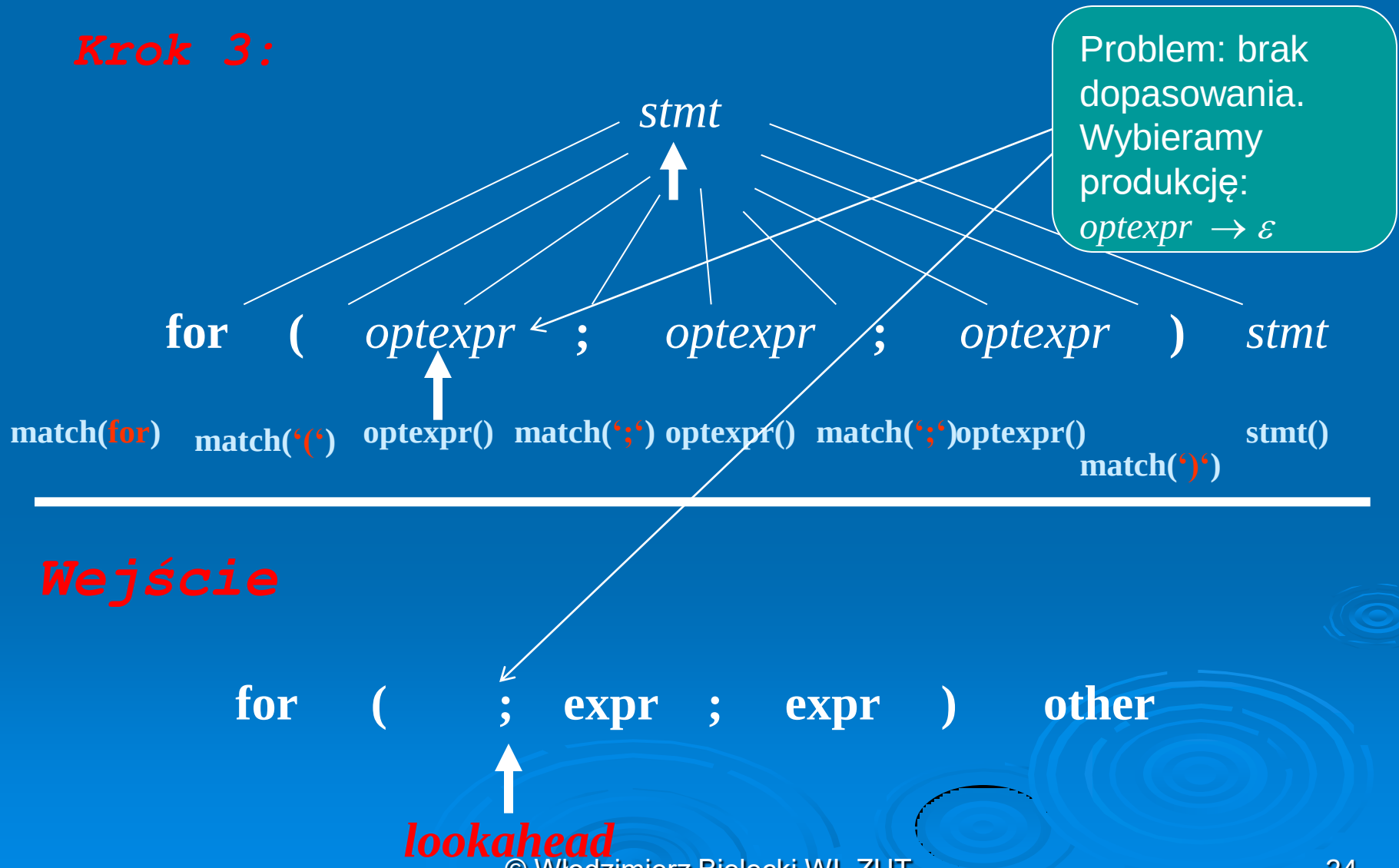


Wejście

for (; expr ; expr) other

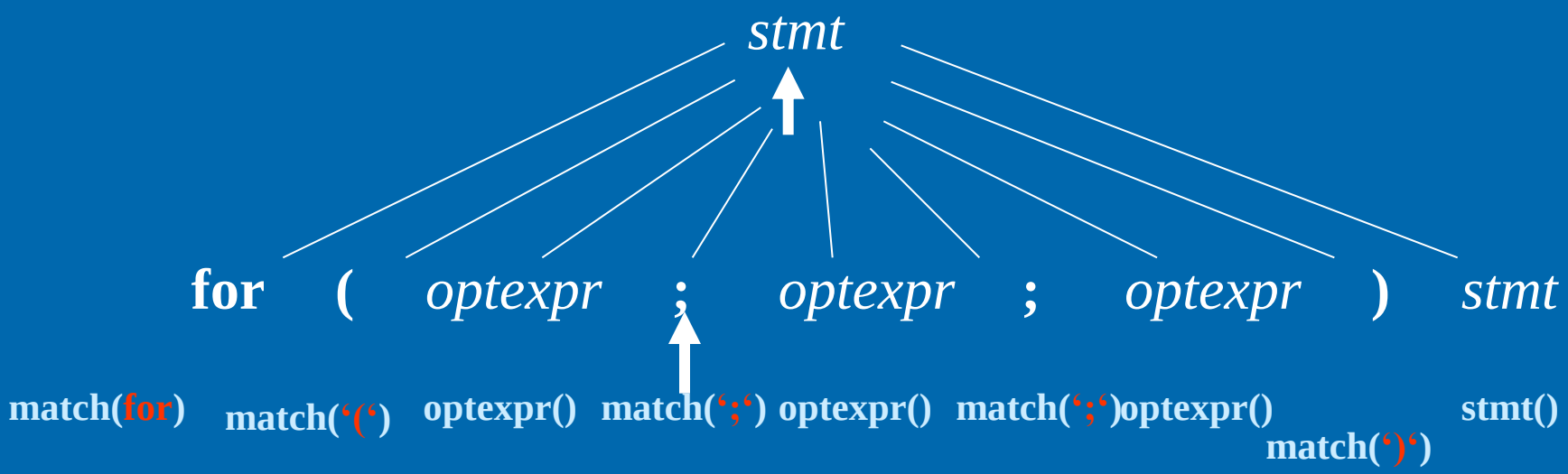
lookahead

Parsowanie zstępujące: przykład



Parsowanie zstępujące: przykład

Krok 4:



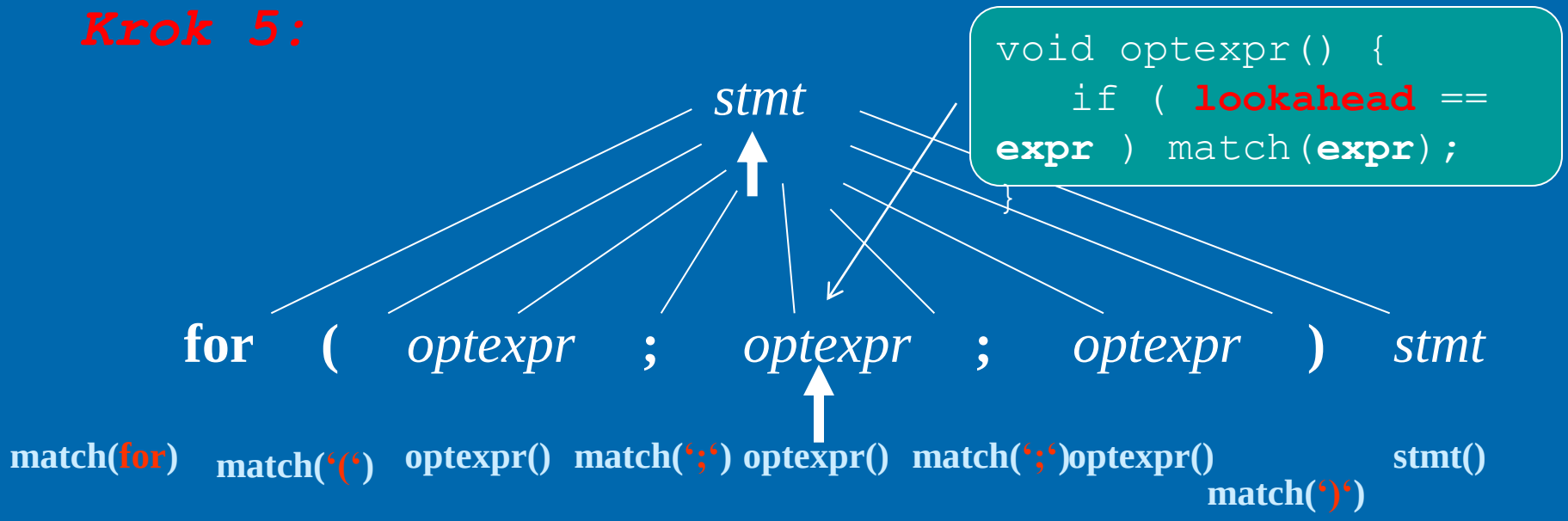
Wejście

for (; expr ; expr) other

lookahead

Parsowanie zstępujące: przykład

Krok 5:




```
void optexpr() {
    if ( lookahead ==
expr ) match(expr);
}
```

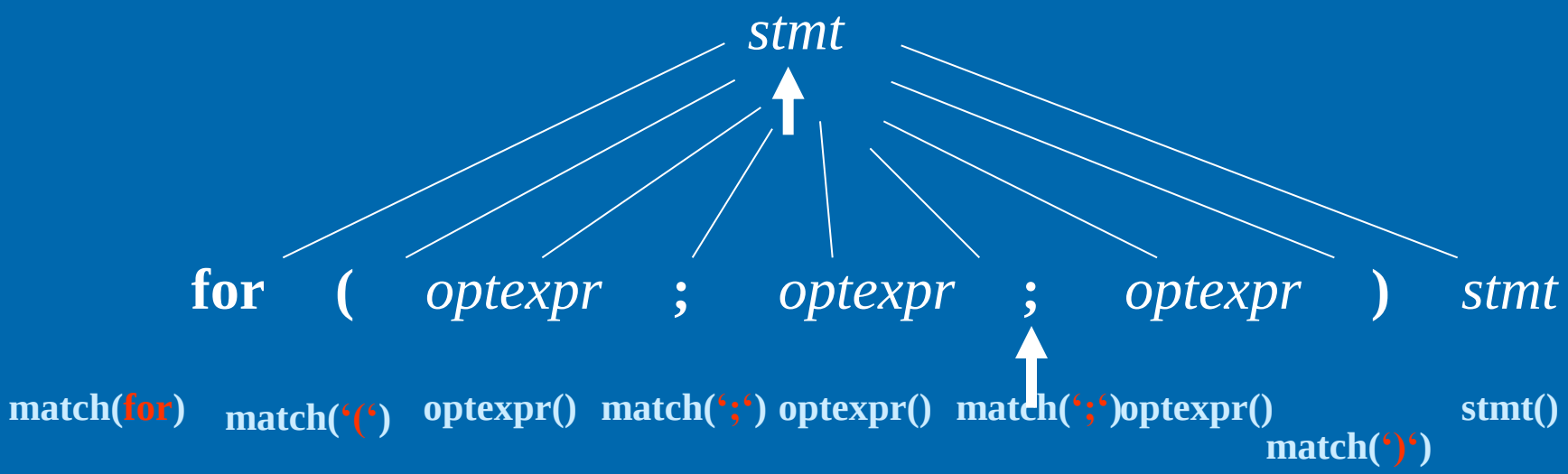
Wejście

for (; expr ; expr) other

lookahead

Parsowanie zstępujące: przykład

Krok 6:



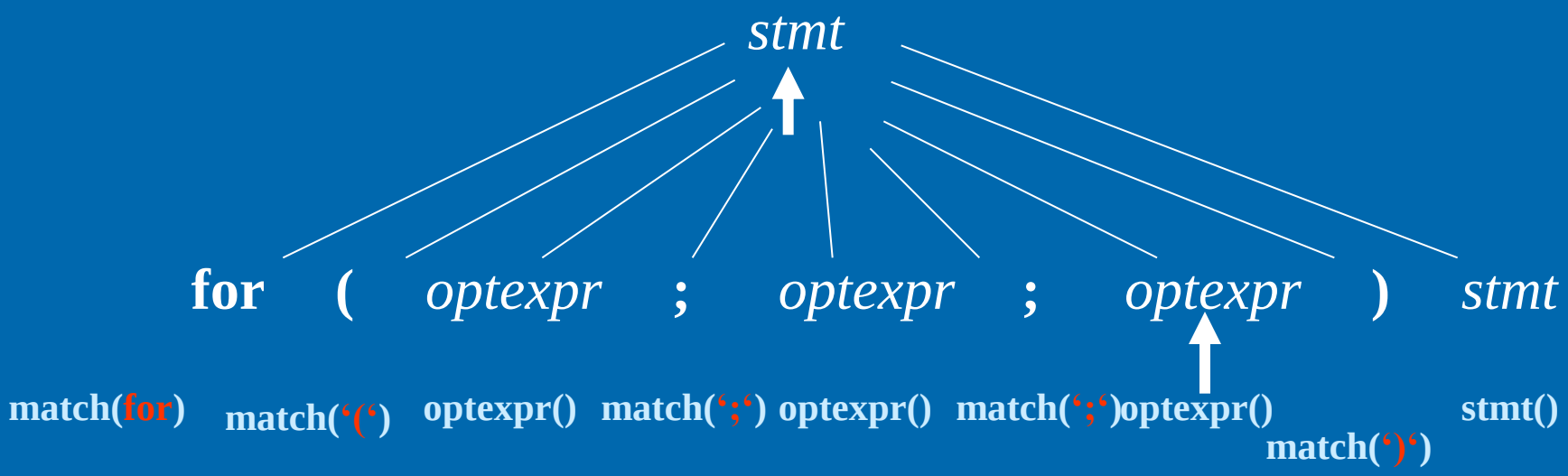
Wejście

for (; expr ; expr) other

lookahead

Parsowanie zstępujące: przykład

Krok 7:



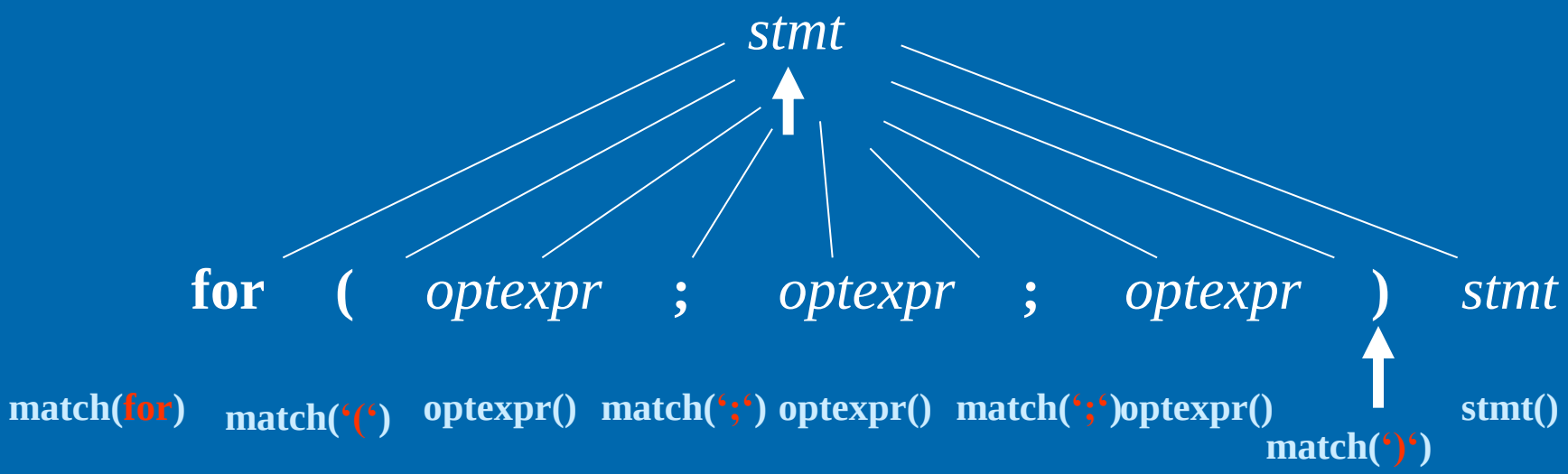
Wejście

for (; expr ; expr) other

lookahedd

Parsowanie zstępujące: przykład

Krok 7:



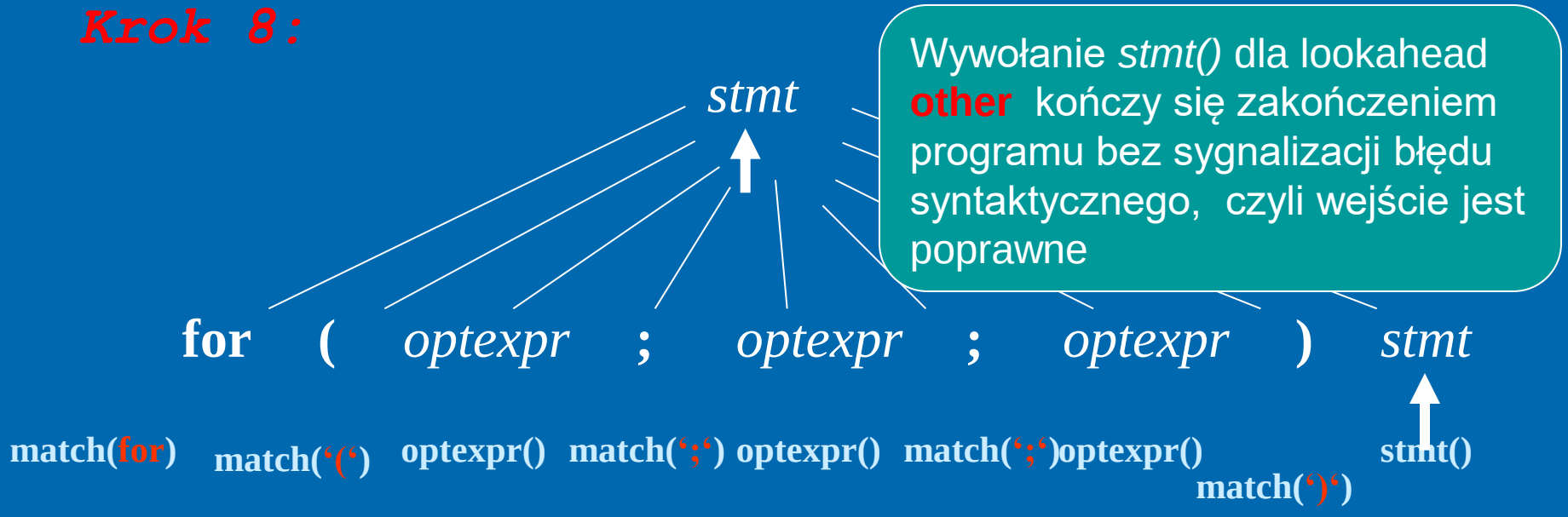
Wejście

for (; expr ; expr) other

lookahead

Parsowanie zstępujące: przykład

Krok 8:



Wywołanie *stmt()* dla lookahead **other** kończy się zakończeniem programu bez sygnalizacji błędu syntaktycznego, czyli wejście jest poprawne

Wejście

for (; expr ; expr) other

lookahead

Parsowanie zstępujące

Kroki parsowania:

ε-produkcje (produkcje, których prawa strona jest napisem pustym ε) wymagają specjalnego traktowania.

Używamy je jako domyślnych, gdy żadna inna produkcja nie może być użyta.

Parsowanie zstępujące

Kroki parsowania:

- Z nieterminalem *optexpr* i symbolem bieżącym „ ; „ ε-produkcja jest wykorzystywana, ponieważ nie ma takiej produkcji, której lewa strona jest *optexpr*, a prawą stroną jest symbol „ ; „ .

Parsowanie zstępujące

Kroki parsowania:

- Ogólnie rzecz biorąc, wybór produkcji dla nieterminala może być oparty na metodzie prób i błędów; to znaczy, możemy spróbować jakiejś produkcji, jeśli jest ona niewłaściwa, to możemy wrócić i spróbować innej produkcji i t.d.

Parsowanie zstępujące: przykład

type → *simple*
| **^ id**
| **array [** *simple* **] of** *type*
simple → **integer**
| **char**
| **num dotdot num**

Ile procedur należy utworzyć?

Parsowanie zstępujące:program

```
procedure match(t : token);
begin
    if lookahead = t then
        lookahead := nexttoken()
    else error()
end;

procedure type();
begin
    if lookahead in { 'integer', 'char', 'num' } then
        simple()
    else if lookahead = '^' then
        match('^'); match(id)
    else if lookahead = 'array' then
        match('array'); match('['); simple();
        match(']'); match('of'); type()
    else error()
end;
```

```
procedure simple();
begin
    if lookahead = 'integer' then
        match('integer')
    else if lookahead = 'char' then
        match('char')
    else if lookahead = 'num' then
        match('num');
        match('dotdot');
        match('num')
    else error()
end;
```

Parsowanie zstępujące:program

```
procedure match(t : token);
begin
    if lookahead = t then
        lookahead := nexttoken()
    else error()
end;

procedure type();
begin
    if lookahead in { 'integer', 'char', 'num' } then
        simple()
    else if lookahead = '^' then
        match('^'); match(id)
    else if lookahead = 'array' then
        match('array'); match('['); simple();
        match(']'); match('of'); type()
    else error()
end;
```

simple → **integer**
| **char**
| **num dotdot num**

```
procedure simple();
begin
    if lookahead = 'integer' then
        match('integer')
    else if lookahead = 'char' then
        match('char')
    else if lookahead = 'num' then
        match('num');
        match('dotdot');
        match('num')
    else error()
end;
```

Parsowanie zstępujące:program

```
procedure match(t : token);
begin
    if lookahead = t then
        lookahead := nexttoken()
    else error()
end;
```

type → *simple*
| **^ id**
| **array [** *simple* **] of** *type*

```
procedure type();
begin
    if lookahead in { 'integer', 'char', 'num' } then
        simple()
    else if lookahead = '^' then
        match('^'); match(id)
    else if lookahead = 'array' then
        match('array'); match('['); simple();
        match(']'); match('of'); type()
    else error()
end;
```

```
procedure simple();
begin
    if lookahead = 'integer' then
        match('integer')
    else if lookahead = 'char' then
        match('char')
    else if lookahead = 'num' then
        match('num');
        match('dotdot');
        match('num')
    else error()
end;
```

Parsowanie zstępujące: krok 1

type()

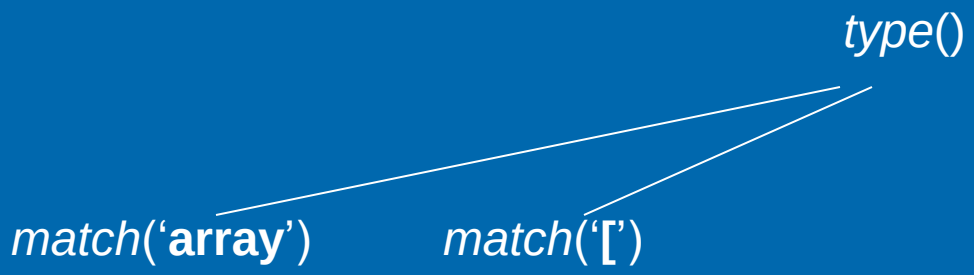
```
procedure type();
begin
   if lookahead in { 'integer', 'char', 'num' } then
      simple()
   else if lookahead = '^' then
      match('^'); match(id)
   else if lookahead = 'array' then
      match('array'); match('['); simple();
      match(']'); match('of'); type()
   else error()
end;
```

match('**array**')

Input: **array** **[** **num** **dotdot** **num** **]** **of** **integer**

lookahead

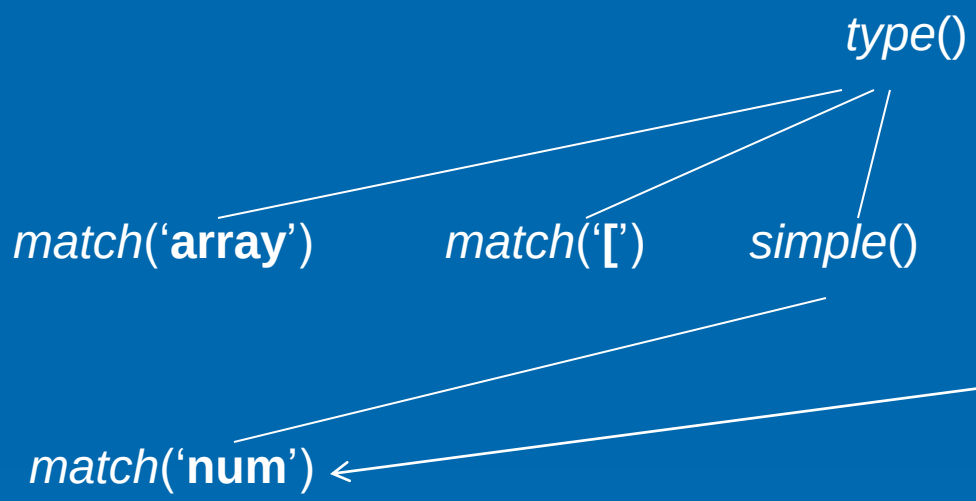
Parsowanie zstępujące: krok 2



Input: **array** **[** **num** **dotdot** **num** **]** **of** **integer**

lookahead

Parsowanie zstępujące: krok 3

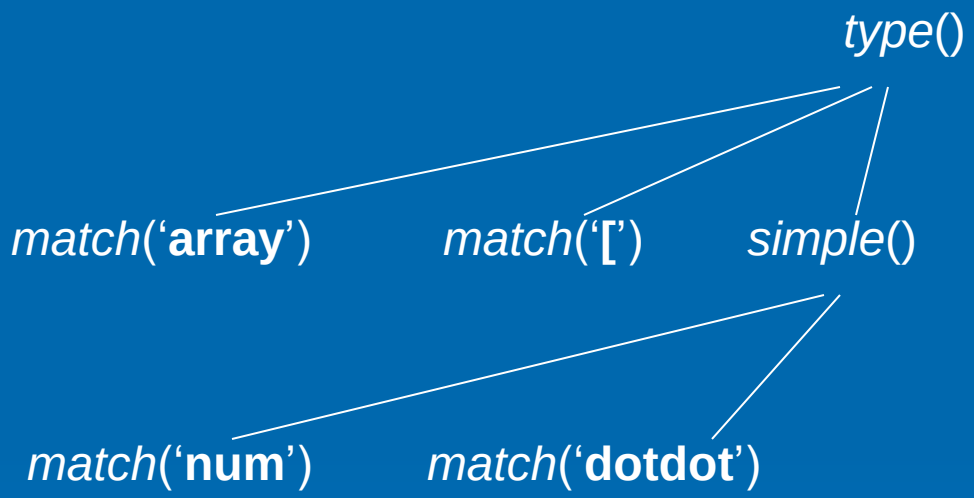



```
procedure simple();
begin
   if lookahead = ‘integer’ then
      match(‘integer’)
   else if lookahead = ‘char’ then
      match(‘char’)
   else if lookahead = ‘num’ then
      match(‘num’);
      match(‘dotdot’);
      match(‘num’)
   else error()
end;
```

Input: **array [num dotdot num] of integer**

lookahead

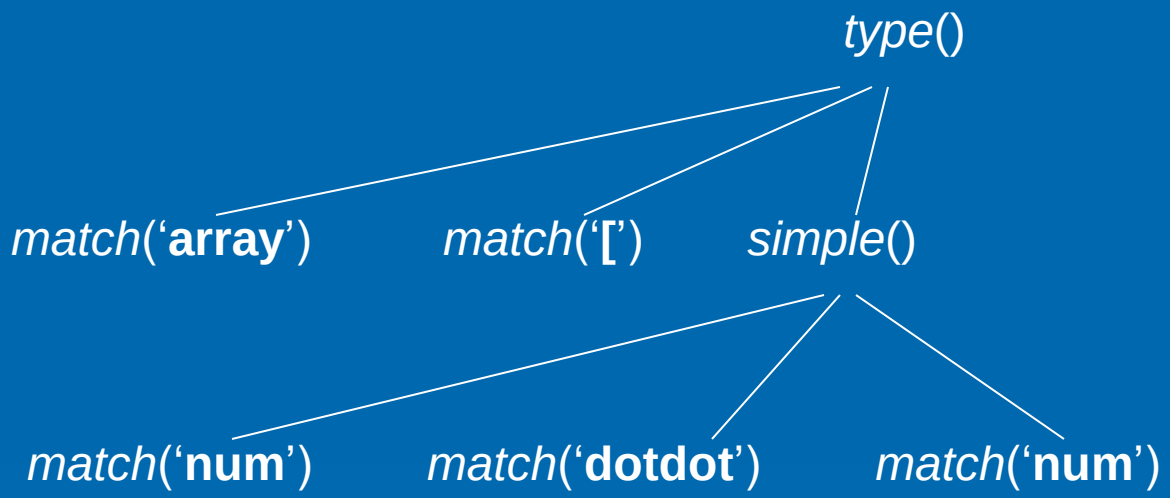
Parsowanie zstępujące: krok 4



Input: **array** **[** **num** **dotdot** **num** **]** **of** **integer**

lookahead

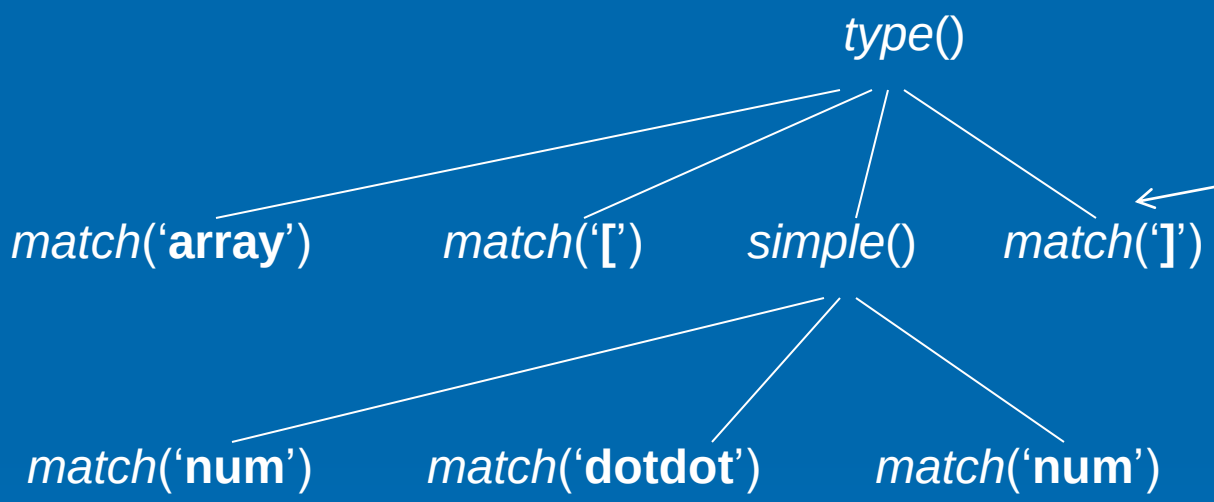
Parsowanie zstępujące: krok 5



Input: **array** **[** **num** **dotdot** **num** **]** **of** **integer**

lookahead

Parsowanie zstępujące: krok 6

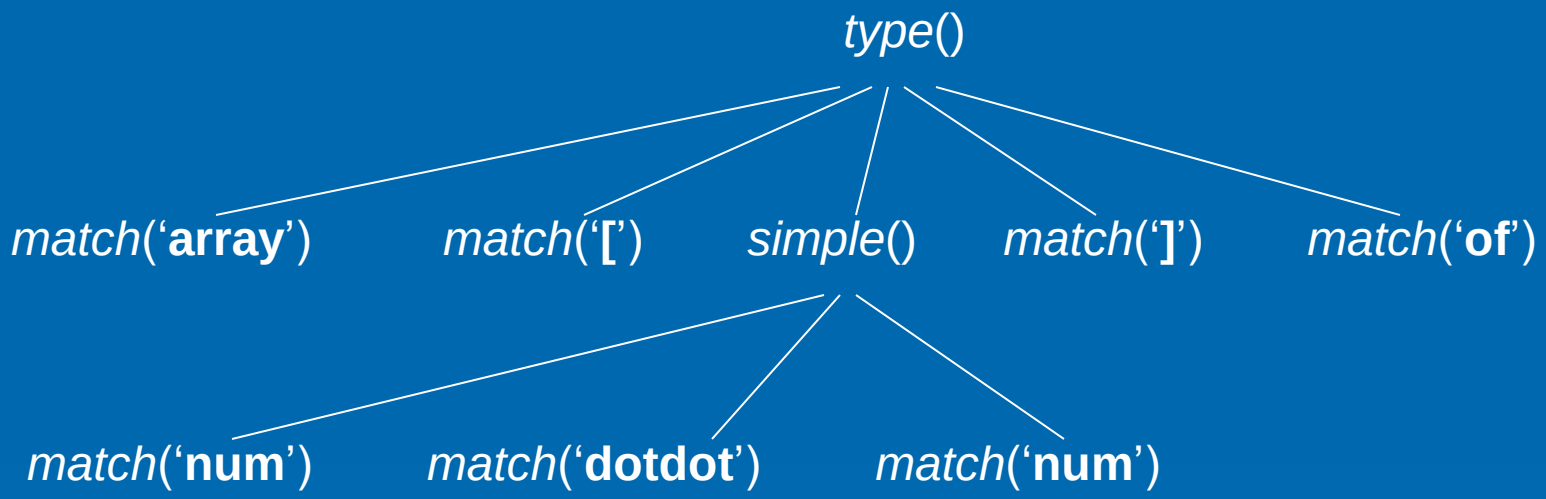



```
procedure type();
begin
.........
   else if lookahead = ‘array’ then
      … simple();
       match(‘]’); match(‘of’); type()
   else error()
end;
```

Input: **array [num dotdot num] of integer**

lookahead

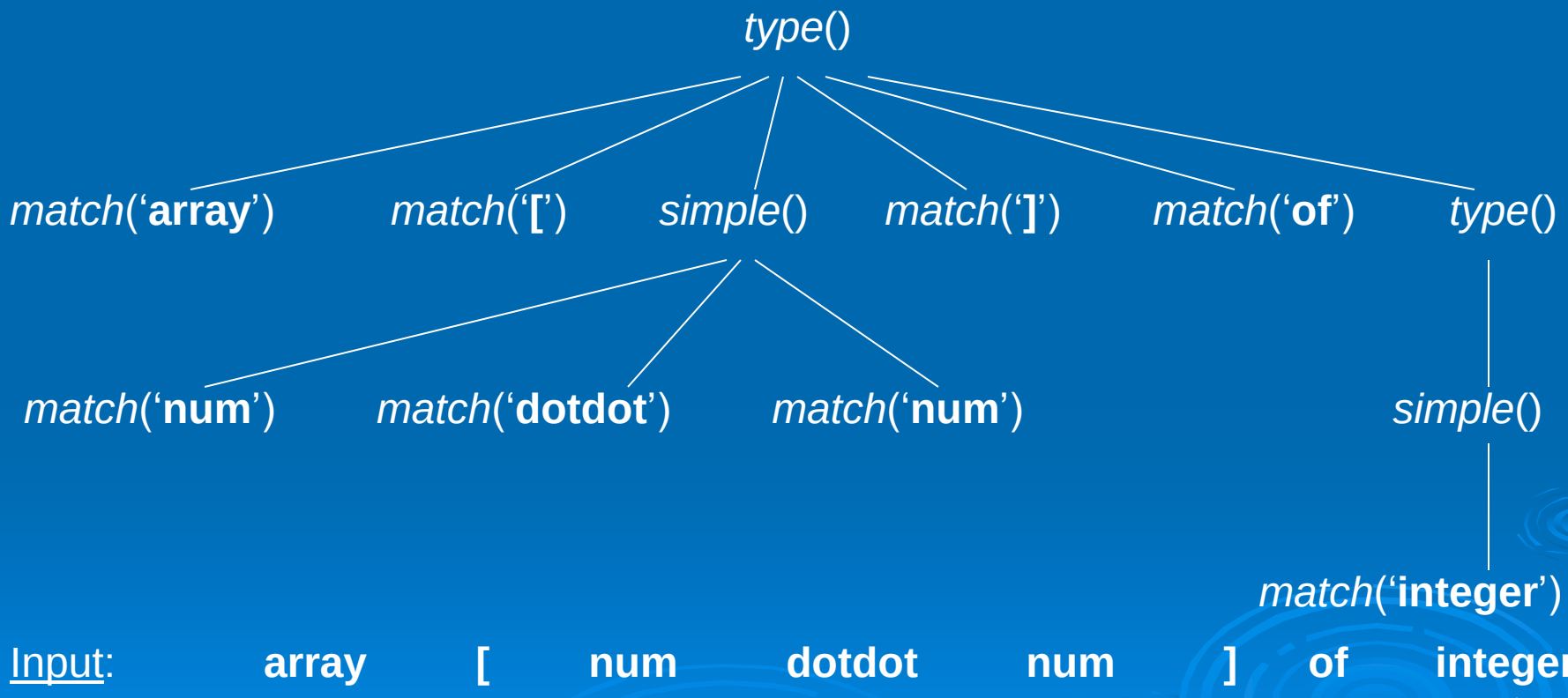
Parsowanie zstępujące: krok 7



Input: **array** **[** **num** **dotdot** **num** **]** **of** **integer**

lookahead

Parsowanie zstępujące: krok 8



Parsowanie zstępujące

Rekurencja lewostronna

- Dla parsera zstępującego możliwe jest zapętlenie.
- Problem pojawia się, gdy korzystamy z produkcji lewostronnie rekurencyjnych, takich jak: *expr* -> *expr* + *term*,

gdzie lewy skrajny symbol ciała produkcji jest taki sam jak symbol po lewej stronie produkcji.

Parsowanie zstępujące

Rekurencja lewostronna

Produkcja lewostronnie rekurencyjna może być wyeliminowana poprzez jej modyfikację.

- Rozważmy produkcje:

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

gdzie α i β są to ciągi terminali i nieterminali, które nie zaczynają się od *A*.

Parsowanie zstępujące

Rekurencja lewostronna

- Na przykład, dla produkcji

  expr -> expr + term | term

Nieterminal *A = expr*,

ciąg *α= +term*,

ciąg *β = term.*

Parsowanie zstępujące

Rekurencja lewostronna

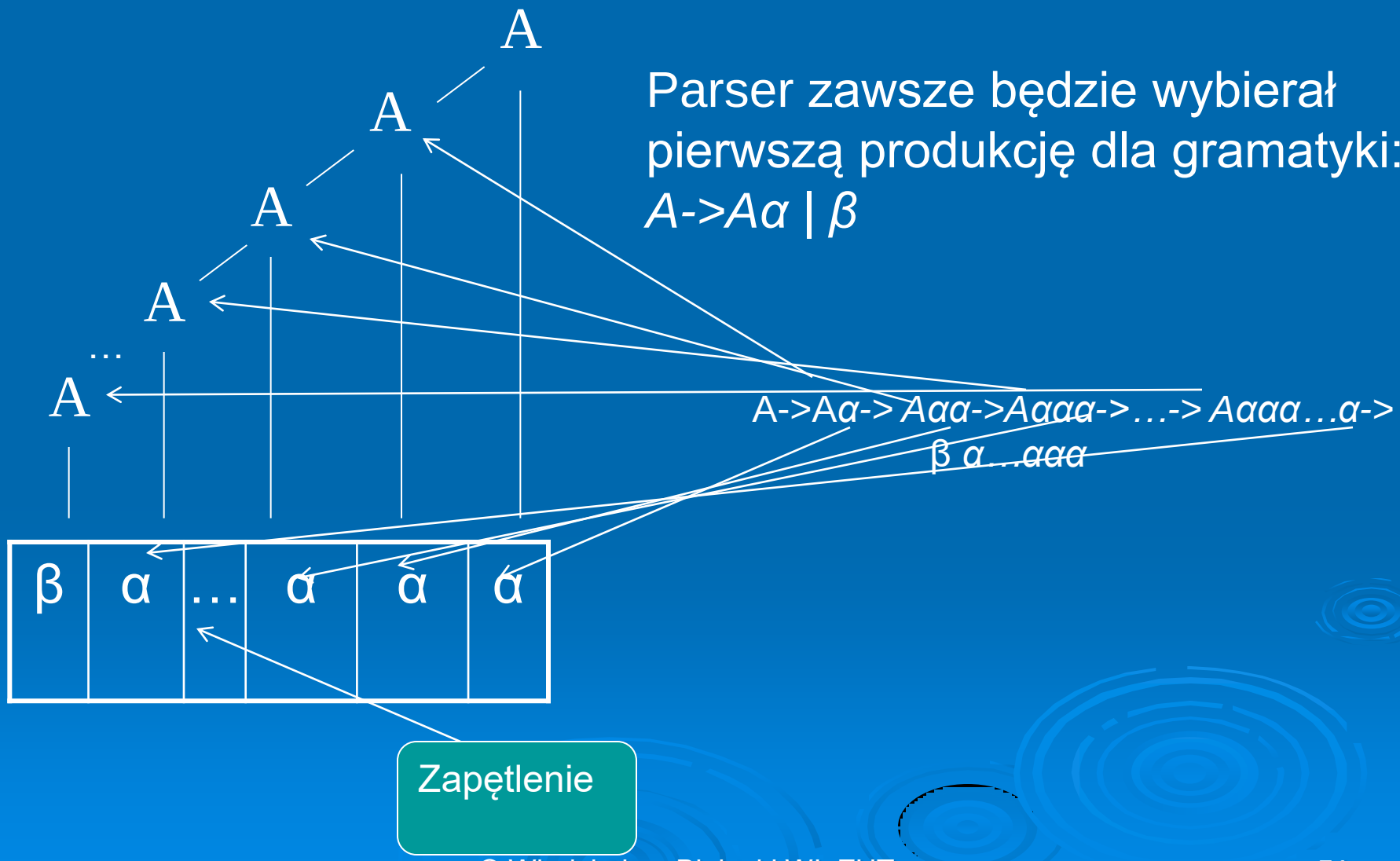
- Nieterminal *A* i jego produkcja są lewostronnie rekurencyjne.
- W ogólnym przypadku, gramatyka może być lewostronnie rekurencyjna, jeśli nieterminal *A* wyprowadza napis $A\alpha$ przez zastosowanie dwóch lub więcej produkcji bezpośrednich.

Parsowanie zstępujące

Rekurencja lewostronna

- Powtarzające się stosowanie tej produkcji tworzy sekwencję ciągu α po prawej stronie A.

Rekurencja lewostronna



Parser zawsze będzie wybierał pierwszą produkcję dla gramatyki:
A->Aα | β

Parsowanie zstępujące

Rekurencja lewostronna

- Jeśli w końcu symbol A zostanie zastąpiony przez β, to uzyskamy β, po którym następuje sekwencja zero lub więcej α.

- Problem: A *nigdy nie* zostanie zastąpione przez β ponieważ zawsze będzie wybrana pierwsza produkcja: $A \rightarrow A\alpha$.

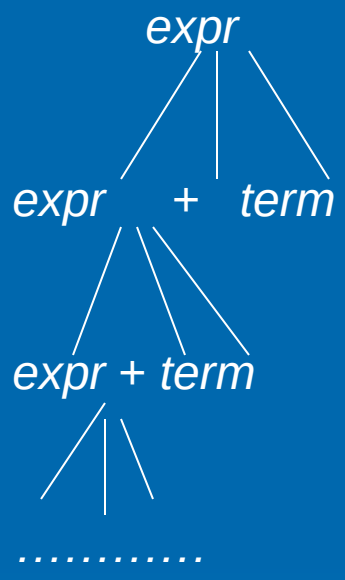
Przykład

Gramatyka:

expr -> expr + term | term

term->0,1…,9

Dla wejścia: 2+2, proces tworzenia drzewa parsowania jest nieskończony.



Parsowanie zstępujące

Rekurencja lewostronna

Problem można rozwiązać przez przepisanie produkcji dla A

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

w następujący sposób przy użyciu nowego nieterminala R:

$$A \rightarrow \beta R$$

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

Parsowanie zstępujące

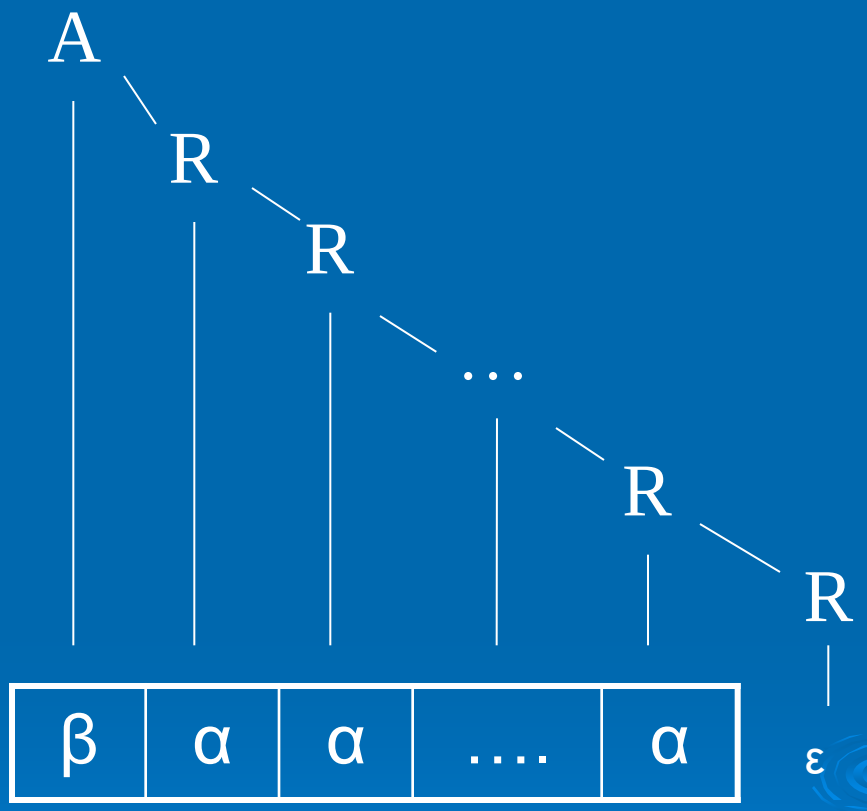
Rekurencja prawostronna

- Nieterminal R i jego produkcja $R->\alpha R$ są prawostronnie rekurencyjne.
- Produkcje prawostronnie rekurencyjne prowadzą do drzew, które rosną w dół i w prawo jak jest pokazane na rysunku na następnym slajdzie

Rekurencja prawostronna

A->βR
R->αR | ε

A->βR->βαR-> βααR->…
βαα…αR-> βαα…αε



Przykład

Gramatyka:

expr -> expr + term | term

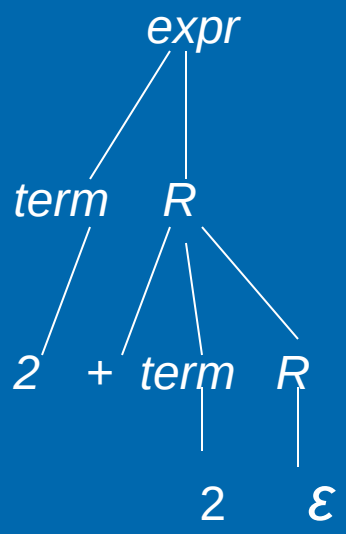
term->0,1…,9

Przekonwertowana do postaci

expr -> term R

R-> +term R | ε

term->0,1…,9



Liście drzewa tworzą zdanie wejściowe, więc parser kończy pracę

Dla wejścia: 2+2 drzewo parsowania wygląda jak wyżej.

Dziękuję za uwagę