

Kapitola VI.

Lexikální analýza

Lexikální analyzátor (Scanner)

Zdrojový program

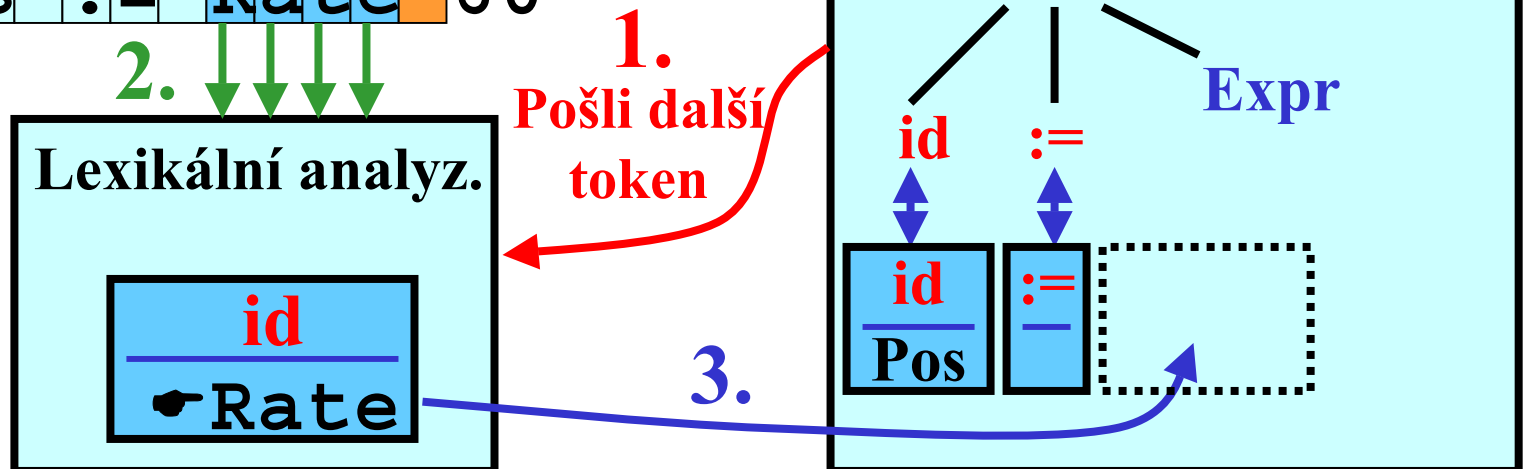
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Scanner: Činnost

Hlavní činnost

- rozpoznání a klasifikace lexémů
 - reprezentace lexému pomocí tokenu
-

Další činnosti

- odstranění komentářů a „prázdných míst“
- komunikace s tabulkou symbolů

Vztah k modelům pro regulární jazyky:

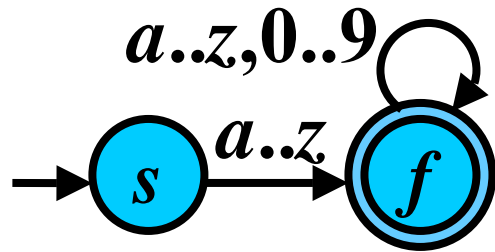
- **Regulární výrazy** specifikují lexémy
- Na **DKA** je založena implementace

Rozpoznávání lexémů pomocí DKA 1/2

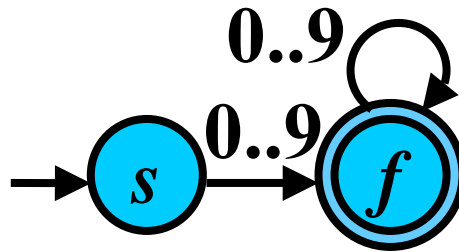
1) Rozpoznávání jednotlivých lexémů pomocí DKA:

Příklad:

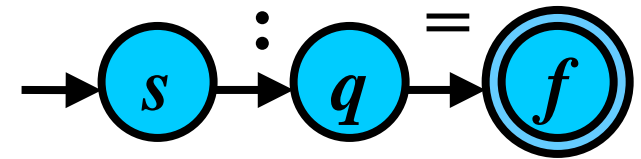
Identifikátor:



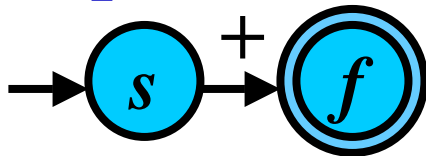
Celé číslo:



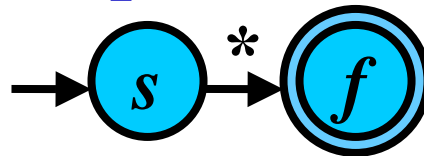
Přiřazení:



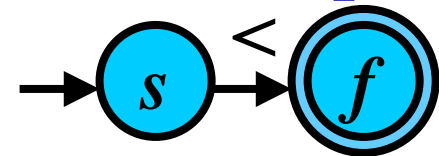
Operátor +:



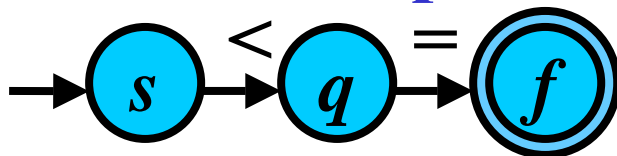
Operátor *:



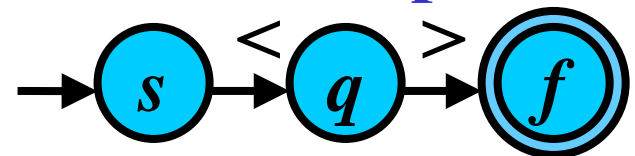
Relační op. <:



Relační op. <=:

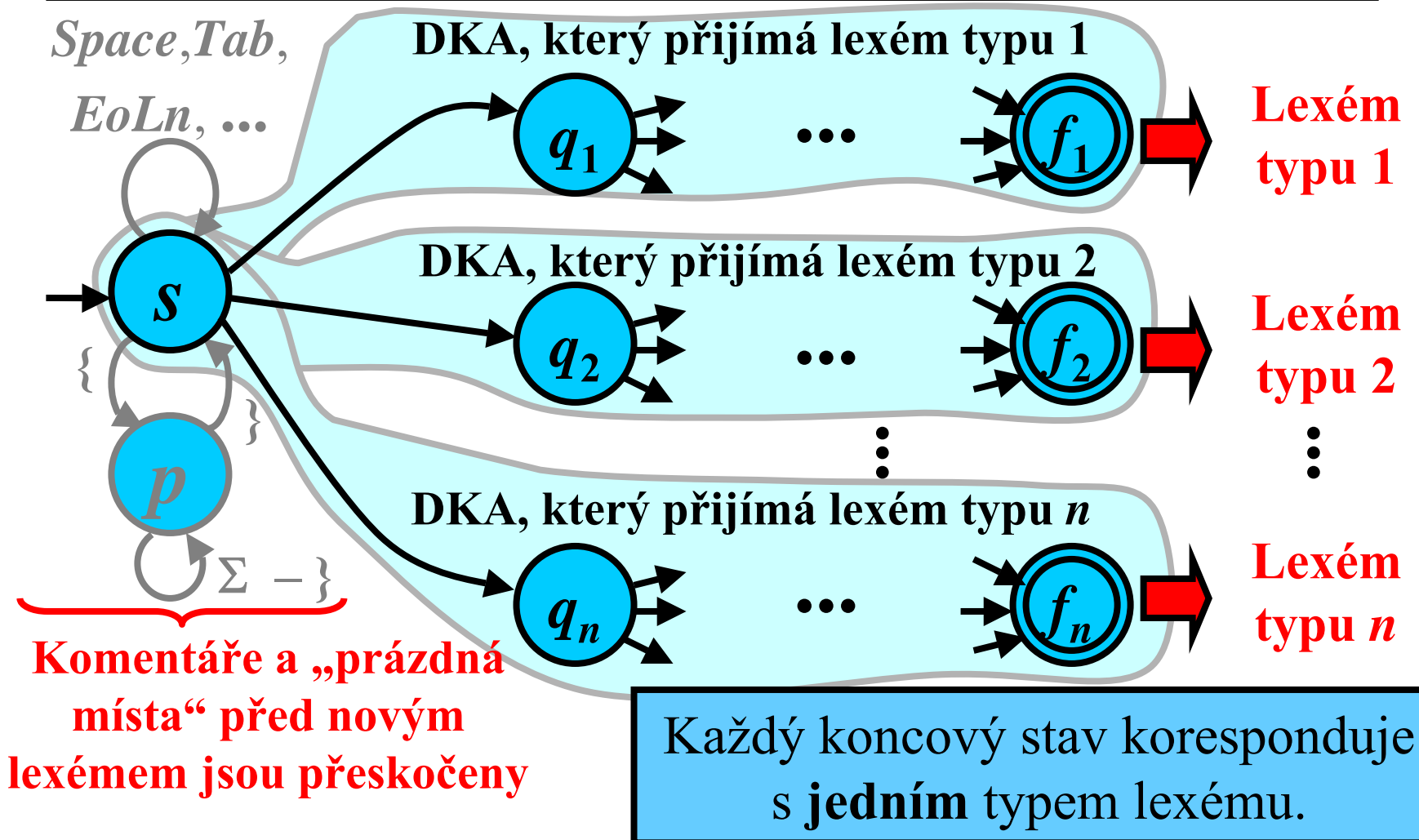


Relační op. <>:



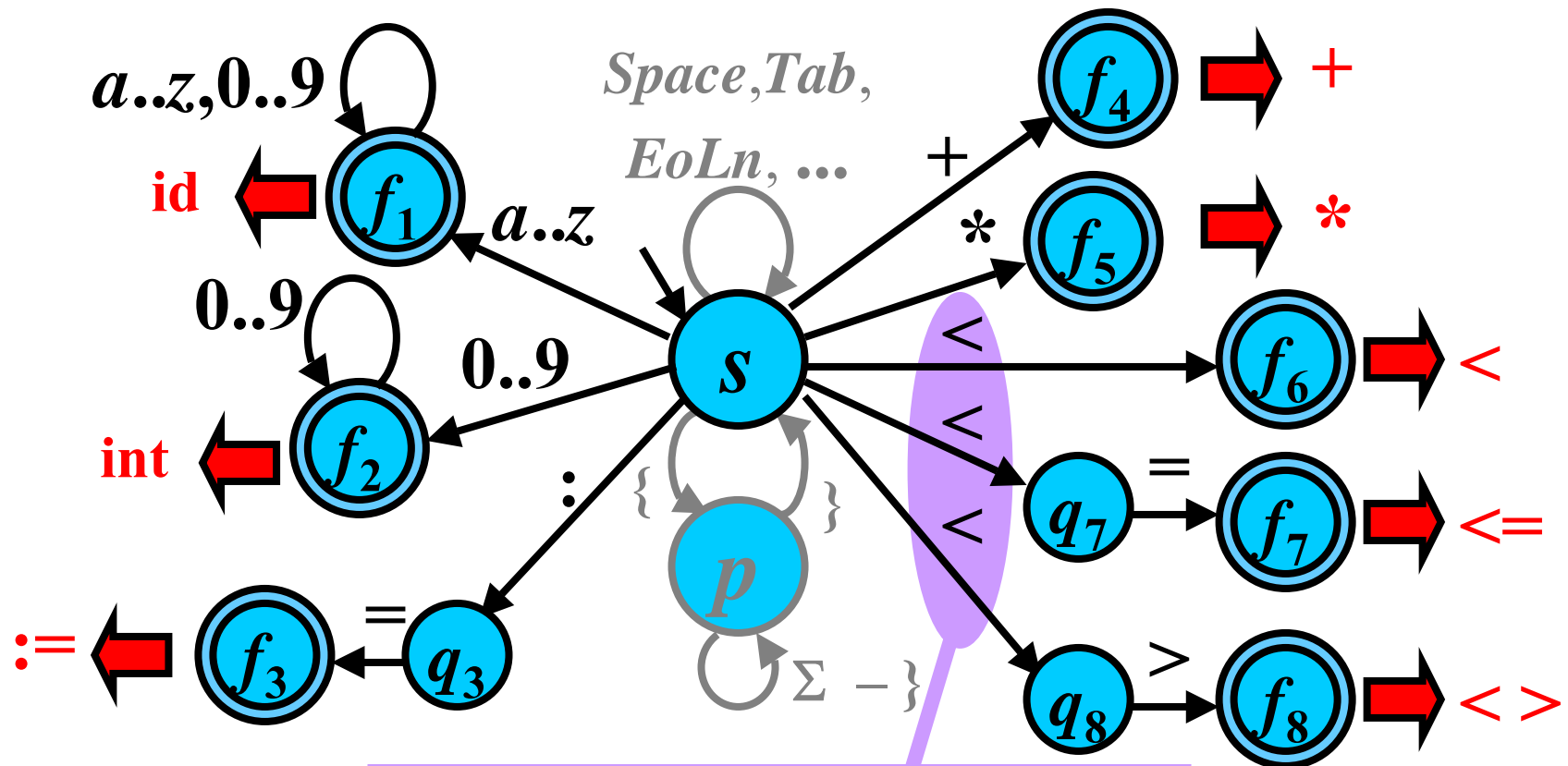
Rozpoznávání lexémů pomocí DKA 2/2

2) Konstrukce KA, který rozpoznává všechny lexémy:



DKA přijímací lexémy: Příklad 1/2

- KA, který přijímá následující lexémy:
identifikátory, celá čísla, :=, +, *, <, <=, <>

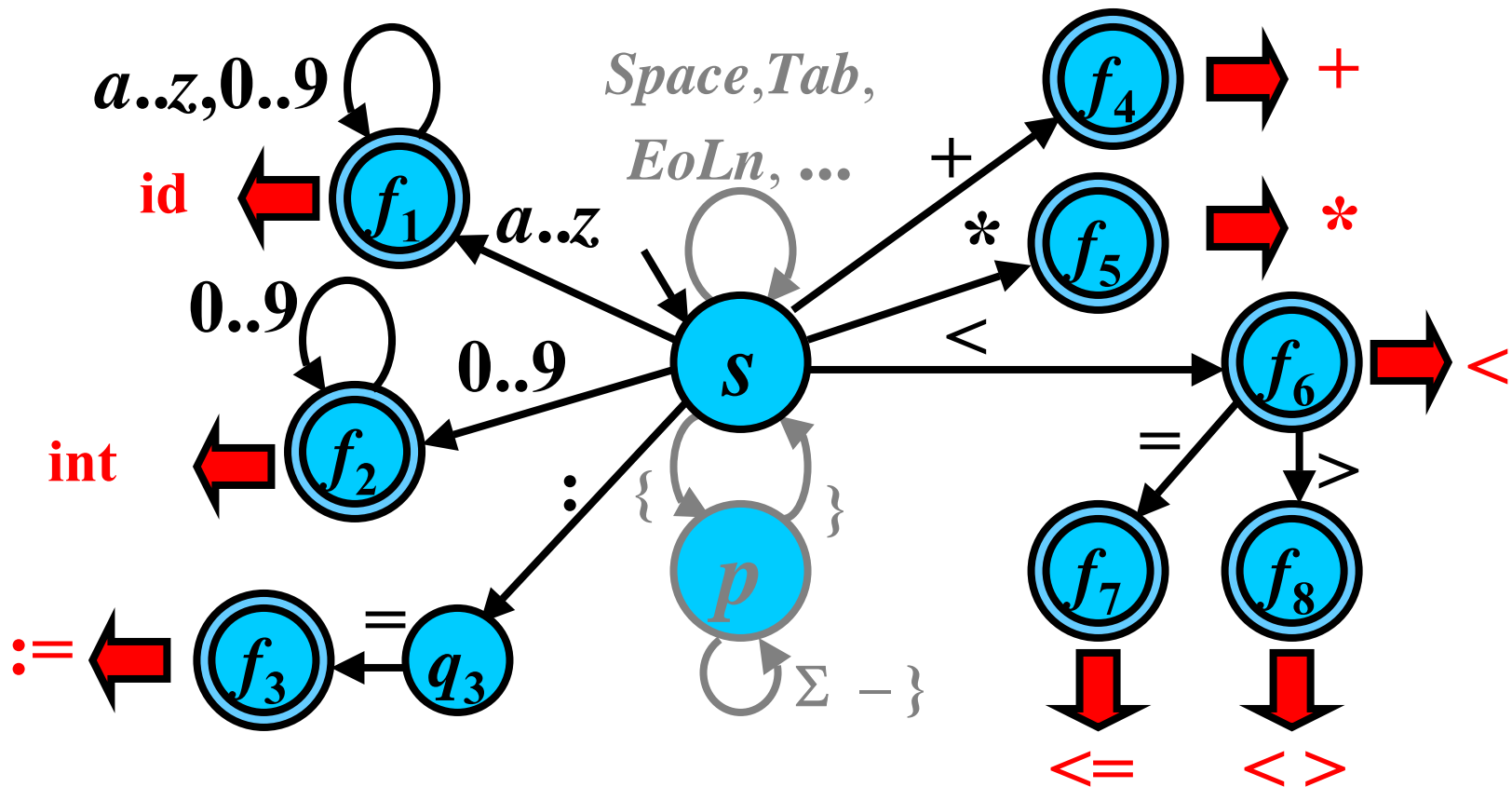


Převést tento KA na DKA.

DKA přijímací lexémy: Příklad 2/2

- Ekvivalentní DKA:

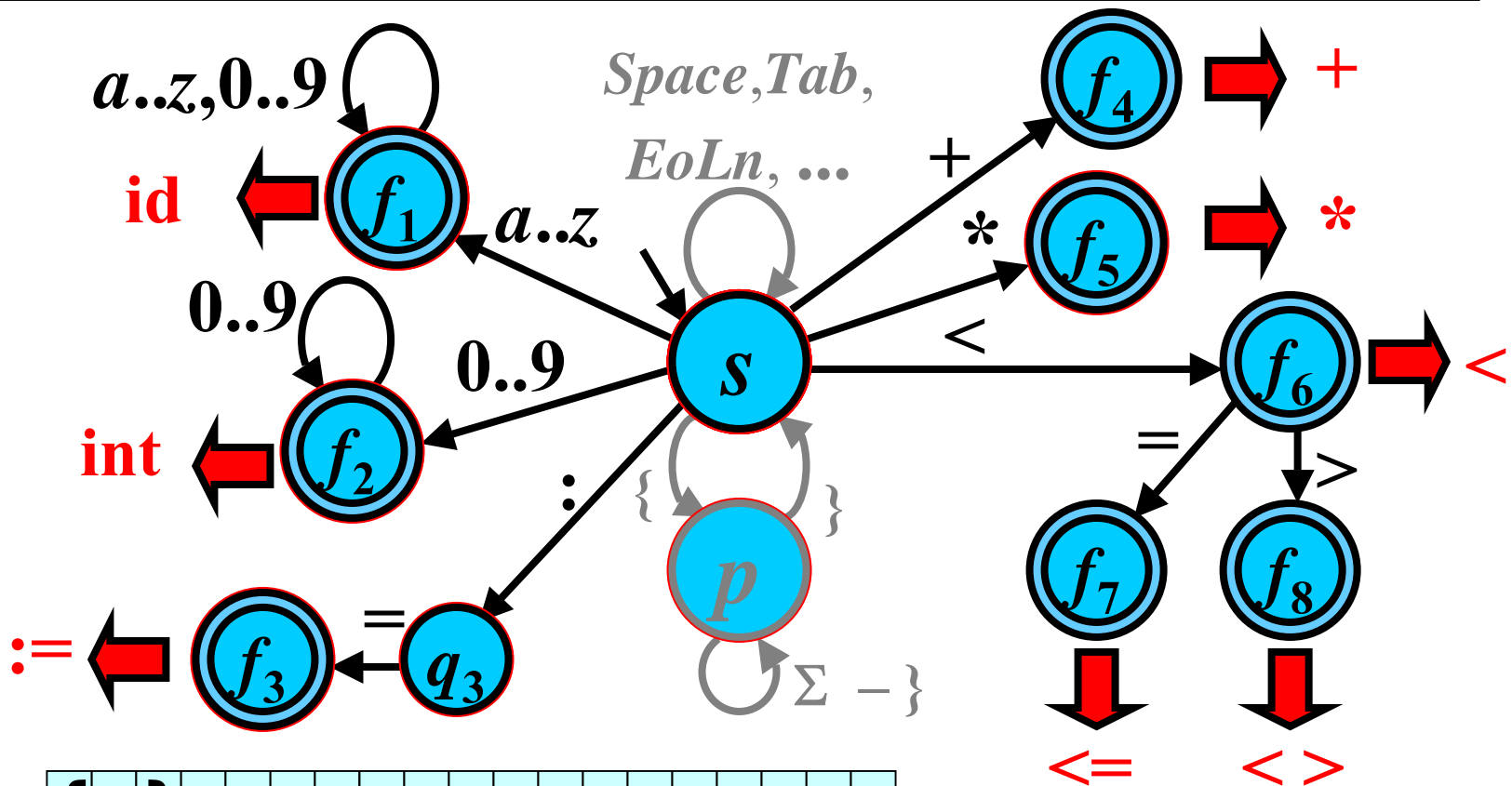
identifikátory, celá čísla, $:=$, $+$, $*$, $<$, $<=$, $<>$



Algoritmus: Určení typu lexému

- **Vstup:** DKA M rozpoznávající lexémy
 - **Výstup:** typ lexému
-
- **Metoda:**
 - while a je další znak ze zdrojového programu and M může udělat přechod se symbolem a do:
 - přečti a
 - udělej přechod v M pro symbol a
 - if M je v koncovém stavu then
 - urči typ lexému, který koresponduje danému koncovému stavu
 - else zahlas chybu (napiš zprávu atd.)

Určení typu lexému: Příklad



{1} Pos := Rate*60

id Pos	:=	id Rate	*	int 60
------------------	-----------	-------------------	----------	------------------

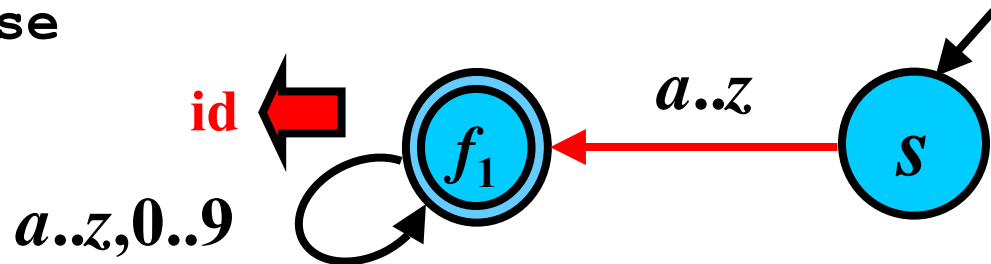
Konec
souboru

Implementace DKA 1/10

```

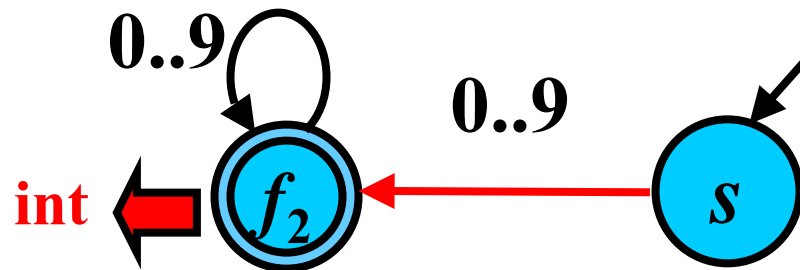
procedure get_Next-Token (var TOKEN: ....);
...
str := '';
state := S;
repeat
  symbol = getchar();
  case state of
    s : begin
      if symbol in ['a'..'z'] then
        begin
          state := f1;
          str := symbol;
        end else

```



Implementace DKA 2/10

```
case state of
  s : begin                                {počáteční stav}
    ...
    if symbol in ['0'..'9'] then
    begin
      state := f2;                       {celé číslo}
      str := symbol;
    end else
```



Implementace DKA 3/10

```

case state of
  s : begin                                {počáteční stav}

```

```

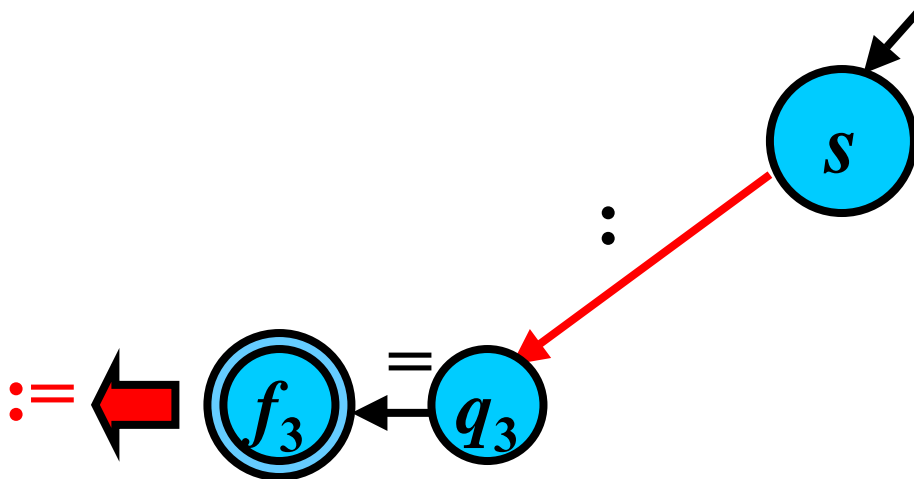
  ...

```

```

  if symbol = ':' then
    state := q3;                          {přiřazení}
  else

```



Implementace DKA 4/10

```

case state of
  s : begin                                {počáteční stav}

```

```

  ...

```

```

  if symbol = '+' then
  begin

```

```

    TOKEN := ADDITION;

```

```

    break;

```

```

  end else

```

```

  if symbol = '*' then
  begin

```

```

    TOKEN := MULTIPLICATION;

```

```

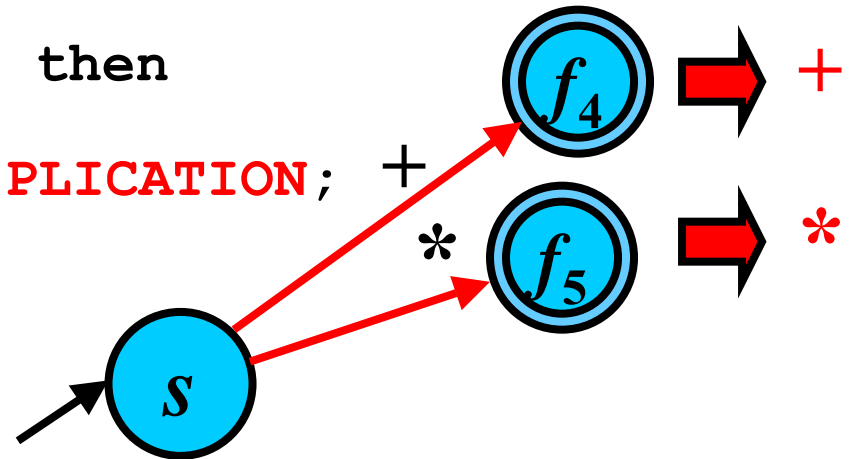
    break;

```

```

  end else

```



Implementace DKA 5/10

```

case state of
  s : begin                                {počáteční stav}

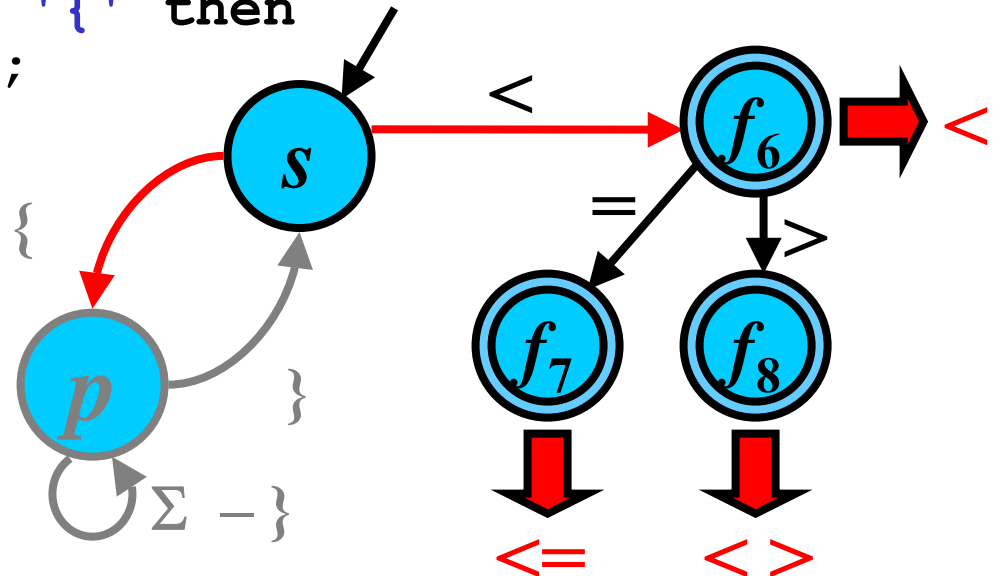
```

...

```

  if symbol = '<' then
    state := f6;
  else
    if symbol = '{' then
      state := p;
    end; {stav s}

```



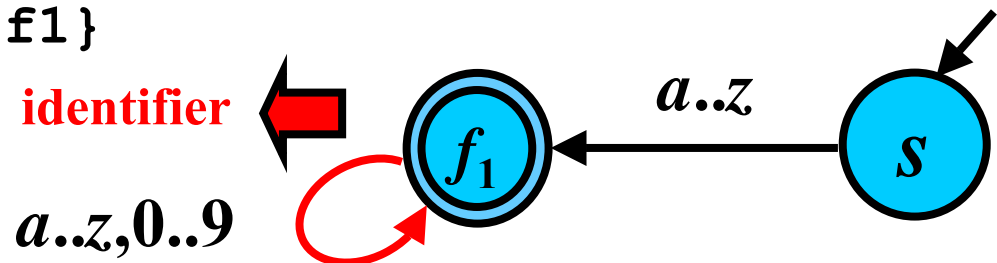
Implementace DKA 6/10

case **state** of

```

f1: begin
    if symbol in ['a'..'z', '0'..'9'] then
        str := str + symbol;
    else
        begin
            ungetchar(symbol);           {návrát znaku}
            if is_keyword(str) then {klíčové slovo}
                TOKEN := get_keyword(str);
            else
                TOKEN := IDENTIFIER;
            break;
        end;
    end;
end; {stav f1}

```



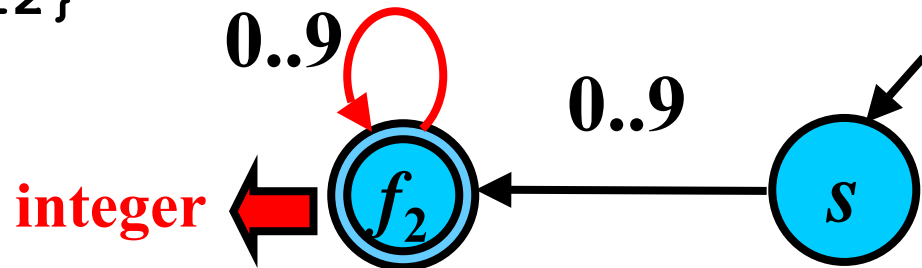
Implementace DKA 7/10

case **state** of

```

..
f2: begin                                {celé číslo}
    if symbol in ['0'..'9'] then
        str := str + symbol;
    else
    begin
        ungetchar(symbol);              {návrat znaku}
        TOKEN := INTEGER;
        {převod hodnoty str na integer}
        break;
    end;
end; {stav f2}

```



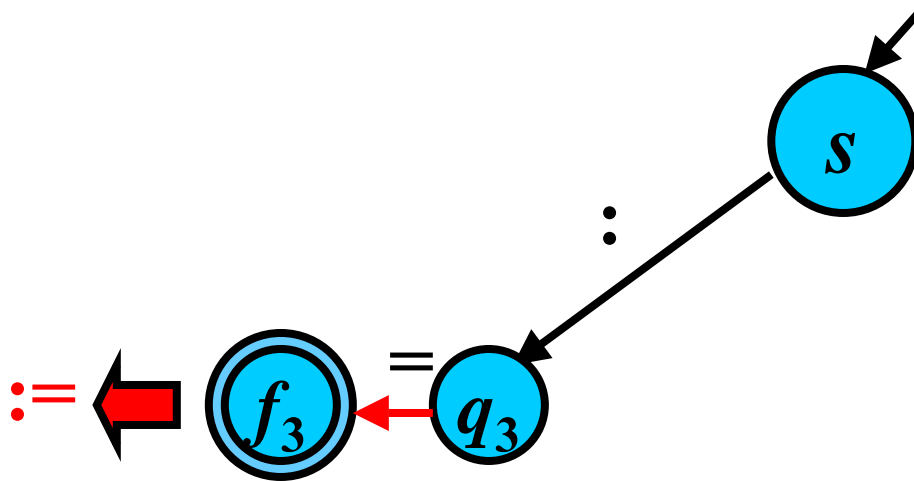
Implementace DKA 8/10

case **state** of

```

q3: begin
    if symbol = '=' then {přiřazení}
    begin
        TOKEN := ASSIGNMENT;
        break;
    end; {stav q3}

```



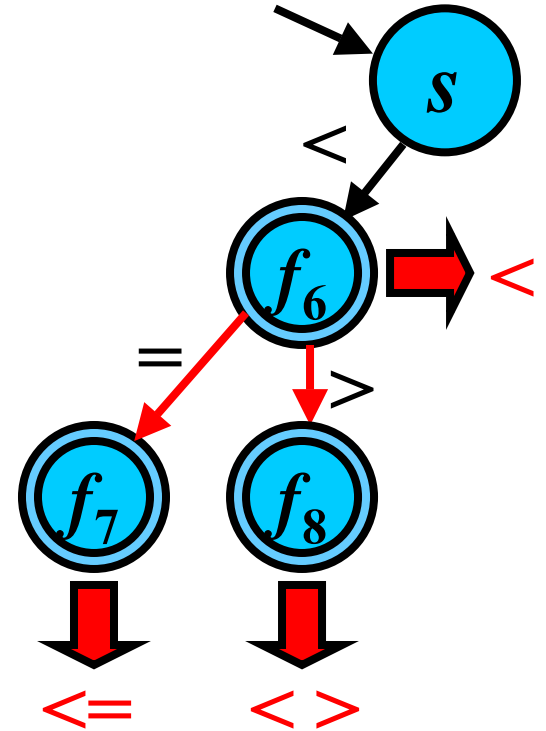
Implementace DKA 9/10

case **state** of

```

...
f6: begin
  if symbol = '=' then
  begin
    TOKEN := LEQ;      {<=}&
    break;
  end else
  if symbol = '>' then
  begin
    TOKEN := NEQ;      {<>}
    break;
  end else
    ungetchar(symbol); {návrát symbolu}
    TOKEN := LTN;      {<}
    break;
  end;
end; {stav f6}

```



Implementace DKA 10/10

case **state** of

```

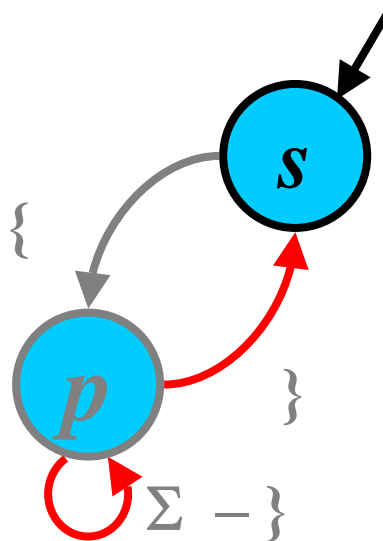
...
p : begin                                {komentář}
    if symbol = '}' then
        state := s;                 {počáteční stav}
    end; {stav p}

```

until EOF;

...

end;



Tokeny v praxi

- Tokeny reprezentují každý lexém ze zdrojového programu jednotně. Obecně tvar tokenu je:

[type, attribute]

-
- 1) Atributy tokenu mohou být rozdílné:

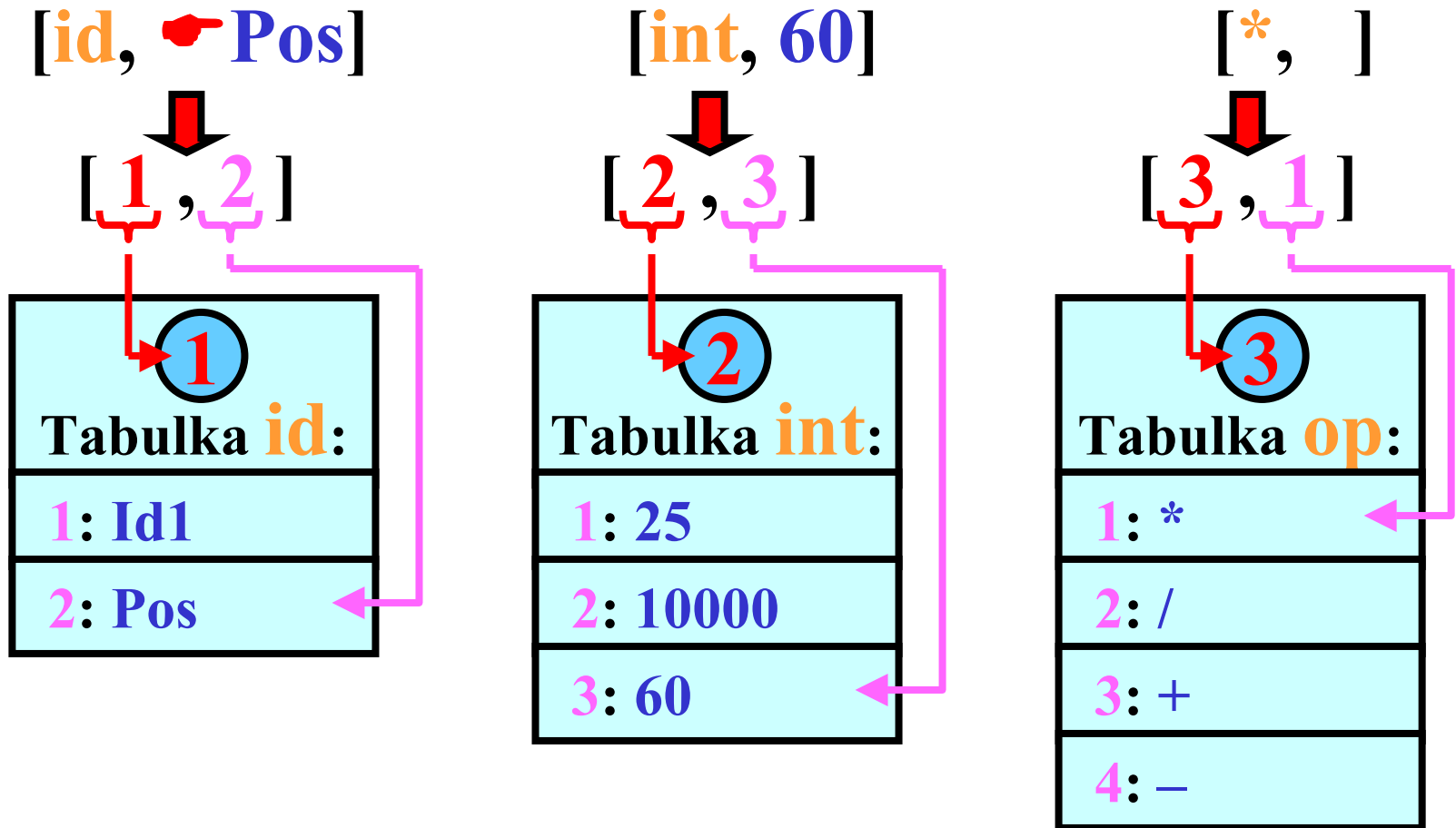
[id,  Pos],
[int,  60],
[*,  _]
 ukazatel celé číslo nic

-
- 2) Atributy tokenu mají stejný typ:

[1, 2], [2, 3], [3, 1]

Pozn.: V praxi se používají většinou rozdílné atributy.

Řešení stejného typu atributů



Uniformní forma tokenů: $[1, 2]; [2, 3]; [3, 1]$

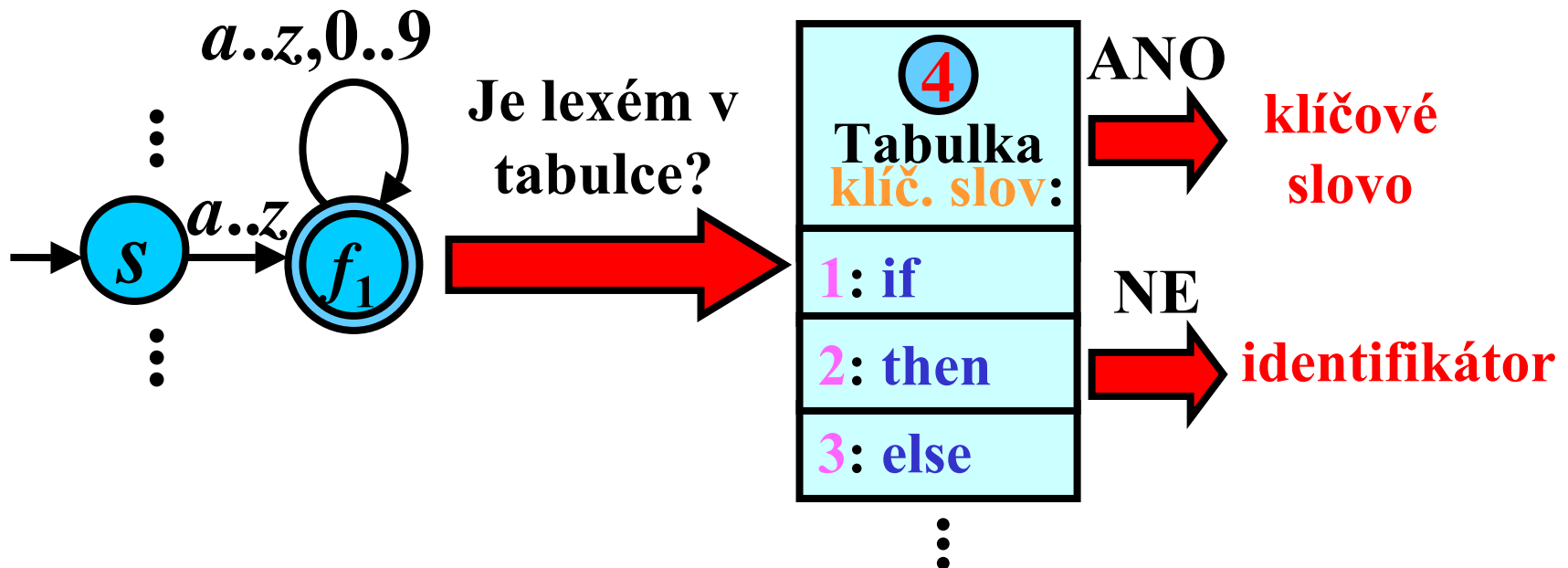
Homogenní struktura

Identifikátory × Klíčová slova

Otázka: Jak rozlišit identifikátory od klíčových slov?

if → **klíčové slovo** × **ifj** → **identifikátor**

Odpověď: Pomocí tabulky klíčových slov.
(Tokeny mají stejný typ)



Tabulka symbolů (tabulka identif.)

Praktický problém:

1) Krátký identifikátor:

- Prázdná místa v paměti (-)

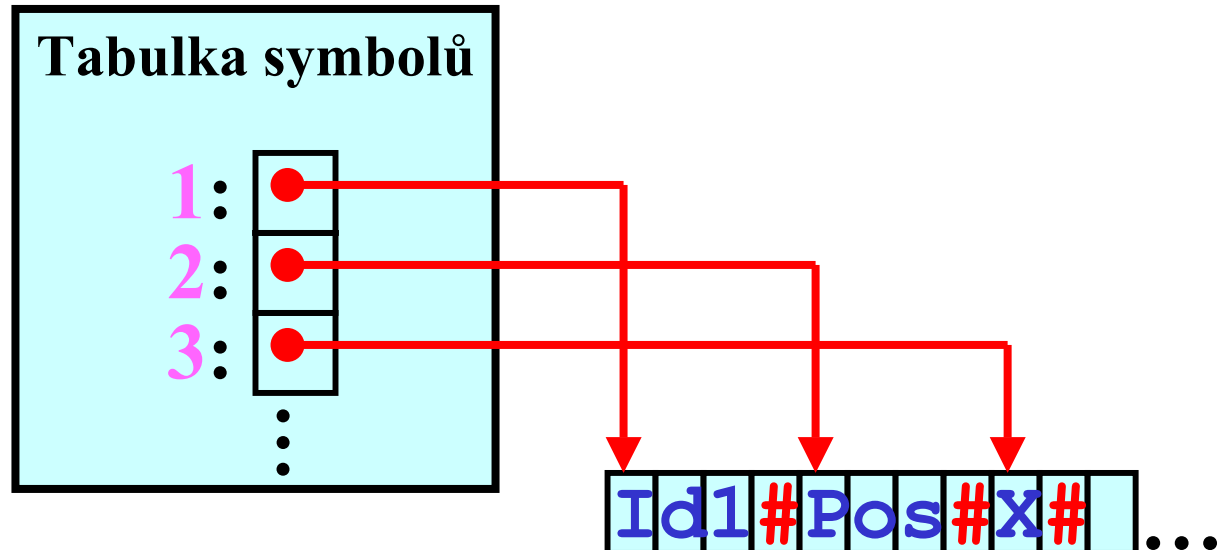
2) Dlouhý identifikátor:

- $\text{Length}(\text{Id}) \leq n$

Tabulka symbolů:

	1	2	3	4	5	...	n
1:	Id	1	-	-		...	-
2:	Pos		-	-		...	-
3:	X	-	-	-	-	...	-
	⋮						⋮

Řešení:



Tabulka symbolů: Struktura

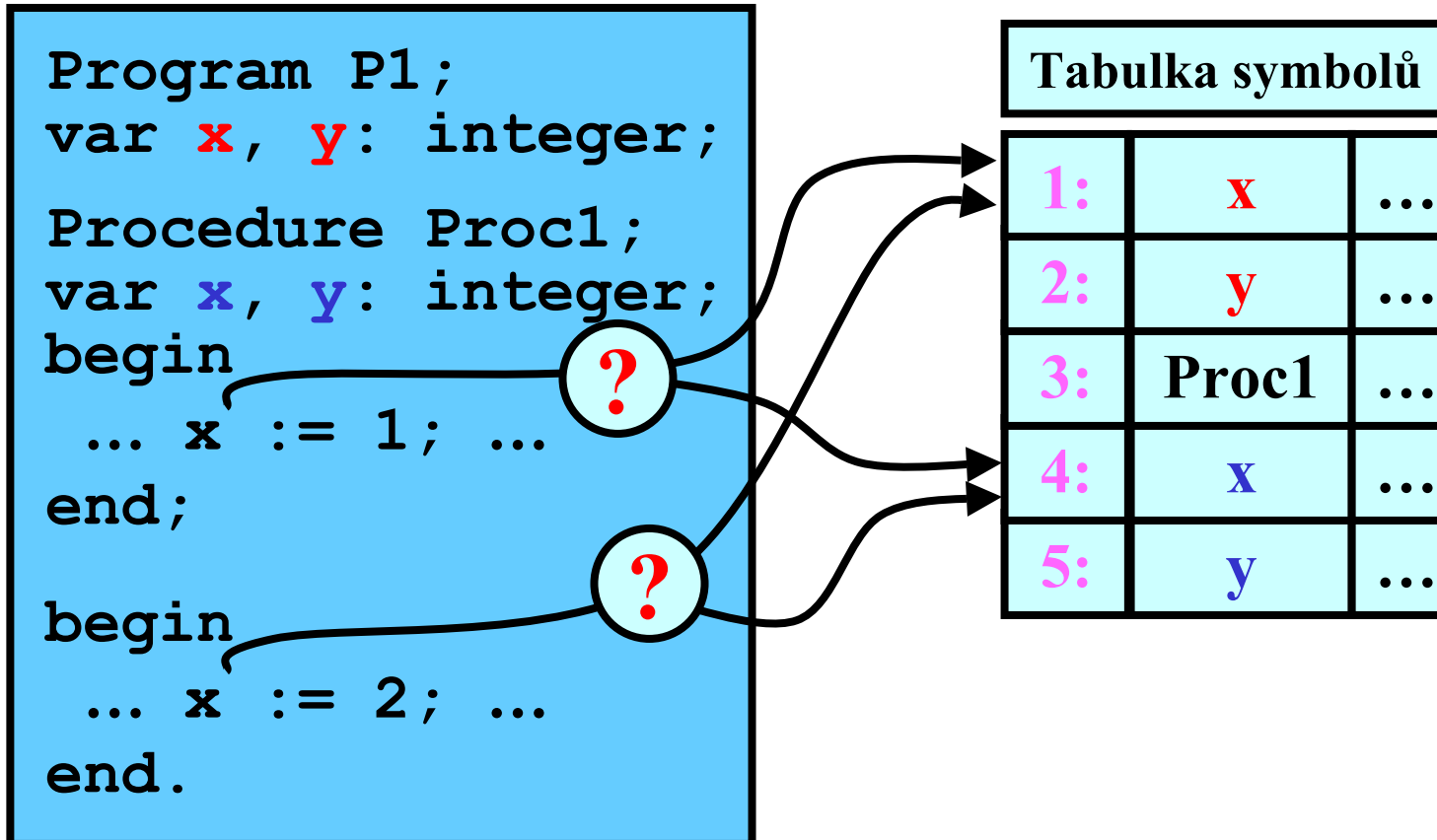
- Je potřeba uchovávat různé typy informací o identifikátorech v tabulce symbolů
 - **Proměnná**: jméno, typ, délka, ...
 - **Konstanta**: typ a hodnota konstanty
 - **Procedura**: počet a typ jednotlivých parametrů
 - \vdots

Struktura tabulky symbolů:

Tabulka symbolů		
	Název	Info
1:	Id1	Proměnná ; Typ: integer
2:	Pi	Konstanta ; Typ: real , Hodnota: 3.14159

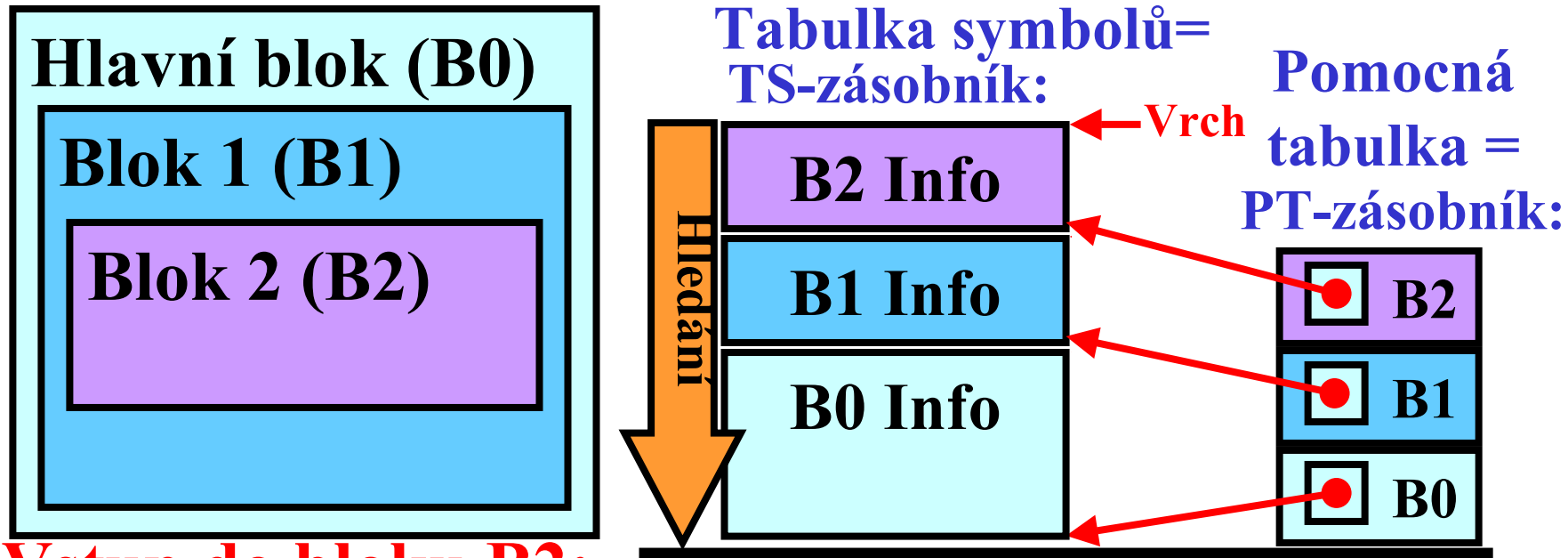
Zásobníková struktura tabulky symbolů

- Problém:**



- Řešení:** Zásobníková struktura TS

Zásobníková struktura



Vstup do bloku B2:

- Vlož ukazatel vrcholu TS-zásobníku na PT-zásobník

Výstup z bloku B2:

- Vrchol B1 Info se stane vrcholem TS-zásobníku
- Odtraň B2-ukazatel z vrcholu PT-zásobníku

Hledání v tabulce symbolů:

- Tabulku prohledávat z vrcholu směrem ke dnu zásobníku

Zásobníková struktura: Příklad

```

Program P1;
var x, y: integer;

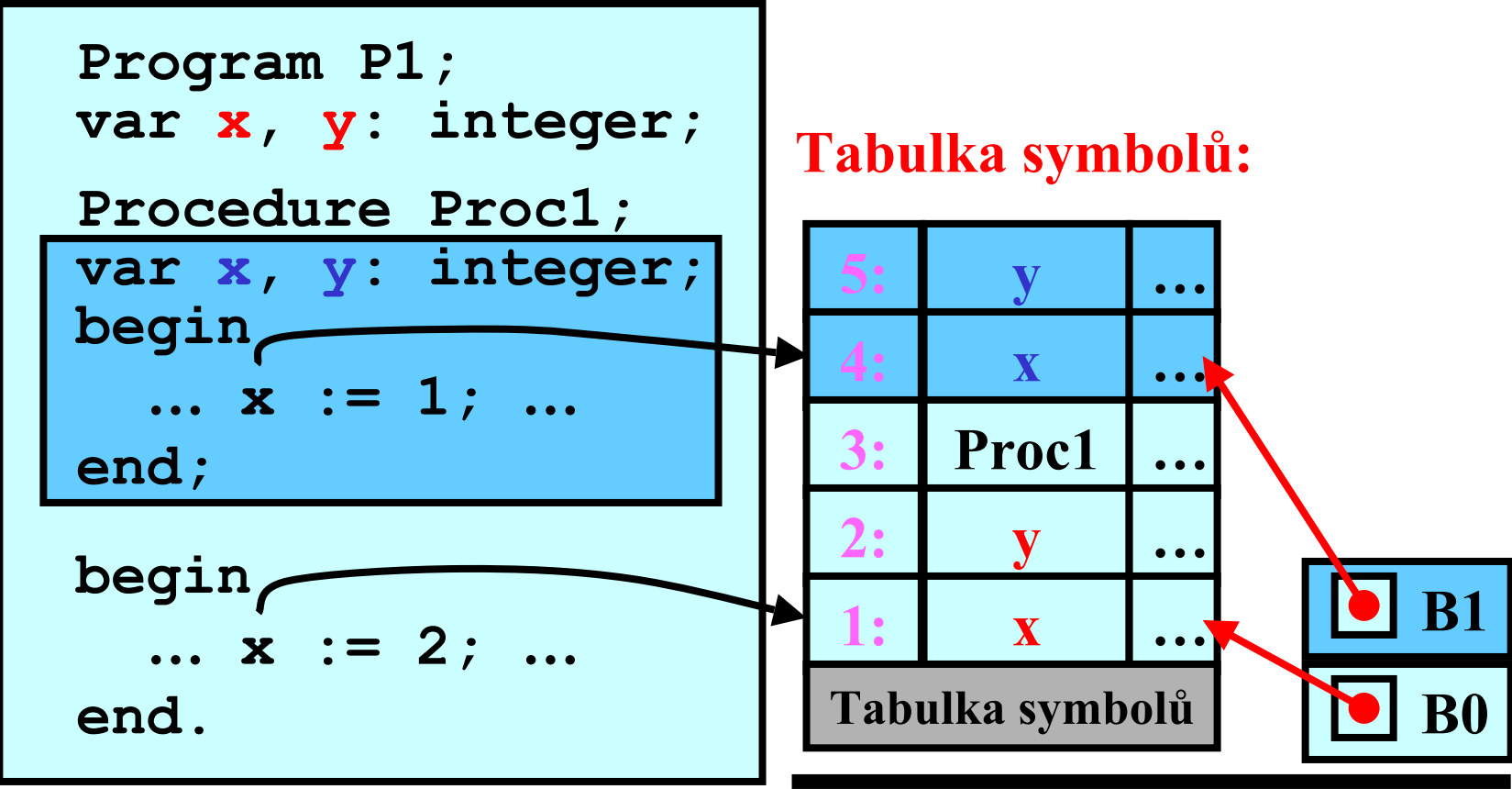
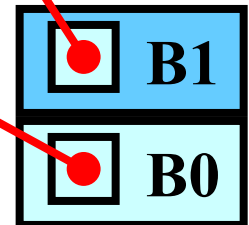
Procedure Proc1;
var x, y: integer;
begin
    ... x := 1; ...
end;

begin
    ... x := 2; ...
end.

```

Tabulka symbolů:

5:	y	...
4:	x	...
3:	Proc1	...
2:	y	...
1:	x	...
Tabulka symbolů		

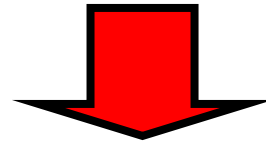


LEX: Hlavní myšlenka

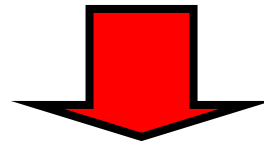
- Automatické vytvoření **lexikálního analyzátoru** z **RV**
 - Lex jako překladač × Lex jako jazyk
-

Ilustrace:

Regulární výrazy

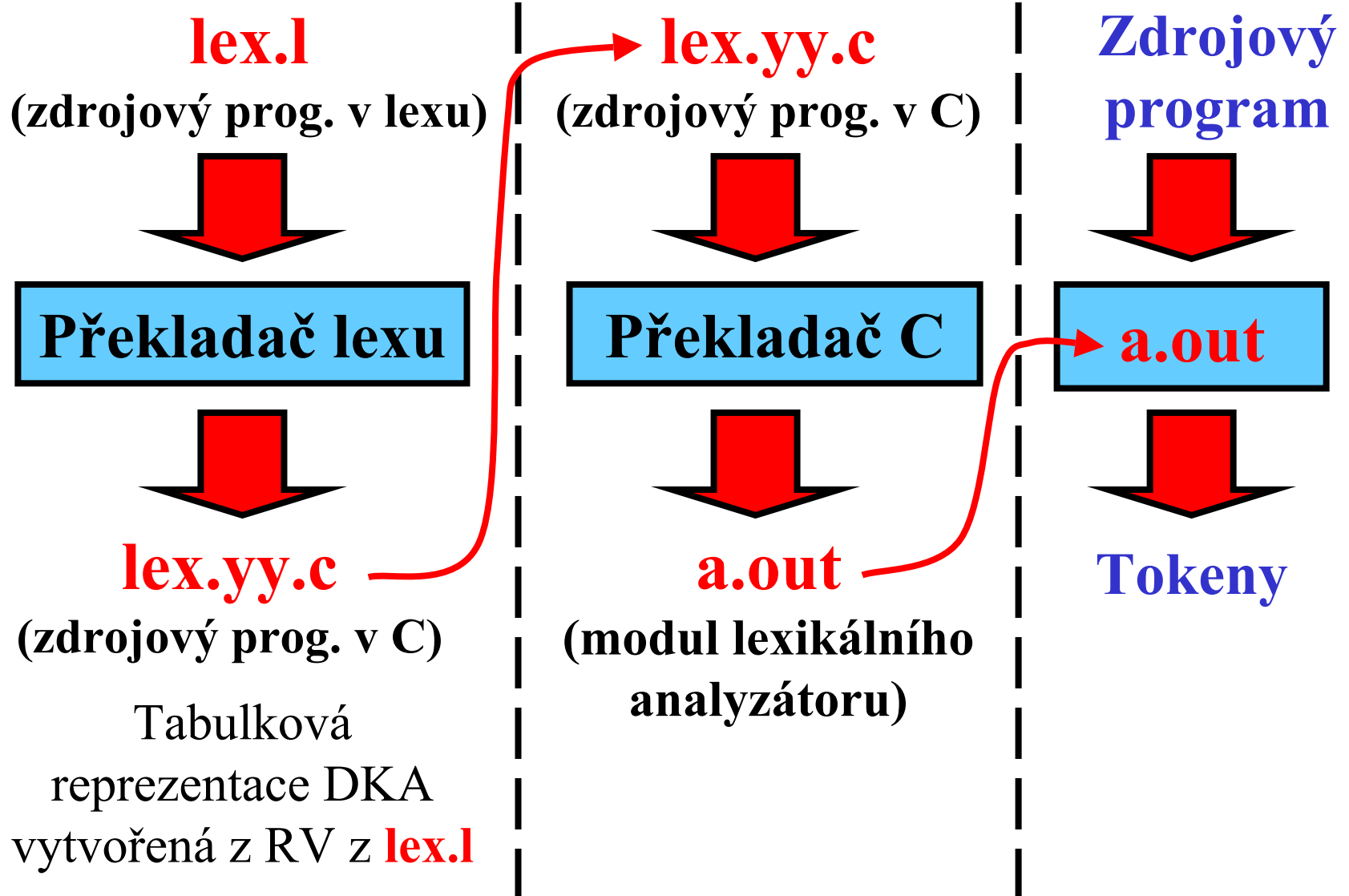


LEX



Lexikální analyzátor

LEX: Fáze kompilace



Struktura zdrojového programu v LEXu

/* Sekce I: Deklarace */

d_1, d_2, \dots, d_i

%% /* Konec sekce I*/

/* Sekce II: Překladová pravidla */

r_1, r_2, \dots, r_j

%% /* Konec sekce II*/

/* Sekce III: Pomocné procedury */

p_1, p_2, \dots, p_k

Základní regulární výrazy v LEXu

RV v LEXu	Ekvivalentní RV v teorii formálních jazyků
a	a
rs	$r \cdot s$
$r s$	$r + s$
r^*	r^*
r^+	r^+
$r?$	$r + \varepsilon$
$[a-z]$	$a + b + c + \dots + z$
$[0-9]$	$0 + 1 + 2 + \dots + 9$

Sekce I: Deklarace

- 1) Definice konstant = typů tokenů
- 2) Definice jsou založeny na RV a jsou ve tvaru:

Nazev_RV	RV
----------	----

- **Nazev_RV** reprezentuje **RV**
- **{Nazev_RV}** je odkaz na regulární výraz **Nazev_RV**, který může být použit v dalších RV.

Příklad:

```

#define      IF      256    /* konstanta pro IF */
#define      THEN    257    /* konstanta pro THEN */
#define      ID      258    /* konstanta pro ID */
#define      INT     259    /* konstanta pro NUM */
letter     [a-z]
digit     [0-9]
id        {letter} ({letter} | {digit}) *
integer   {digit}+
  
```


Sekce II: Překládová pravidla

- Překládová pravidla jsou tvaru:

RV **Akce**

- **Akce** je podprogram, který je zavolán, pokud **RV** popisuje aktuální lexém

Příklad:

```
if                    return (IF) ;
then                  return (THEN) ;
{id}                  { yylval = install_id() ;
                          return (ID) ; }
{integer}            { yylval = install_int() ;
                          return (INT) ; }
```

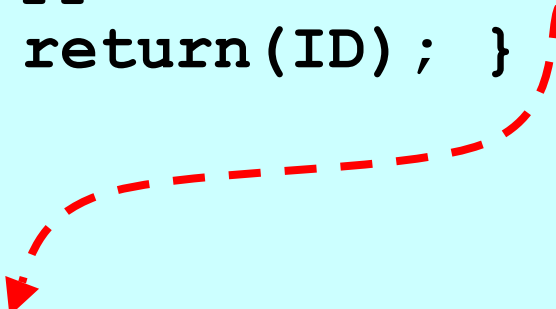
yylval: hodnota vrácená `install_id()` = atribut tokenu

Sekce III: Pomocné procedury

- Pomocné procedury jsou potřeba pro překladová pravidla

Příklad:

```
...
{id}      { yylval = install_id();
           return(ID); }
...
%%
...
int install_id() {
    /* Procedura, která vloží lexém do tabulky
       symbolů a vrátí na něj ukazatel */
}
...
```



Zdrojový program v LEXu

```

#define      IF      256  /* konstanta pro IF */
#define      THEN    257  /* konstanta pro THEN */
#define      ID      258  /* konstanta pro ID */
#define      INT     259  /* konstanta pro NUM */
int yylval;          /* yylval je viditelná pro parser */
letter      [a-z]
digit       [0-9]
id          {letter} ({letter} | {digit}) *
integer     {digit}+
%%
if          return (IF) ;
then       return (THEN) ;
{id}       {yylval = install_id() ;return (ID) ;}
{integer}  {yylval = install_int() ;return (INT) ;}
%%
int install_id()  { ... }
int install_int() { ... }

```