

Kapitola V.

Lexikální analýza

Lexikální analyzátor (Scanner)

Zdrojový program

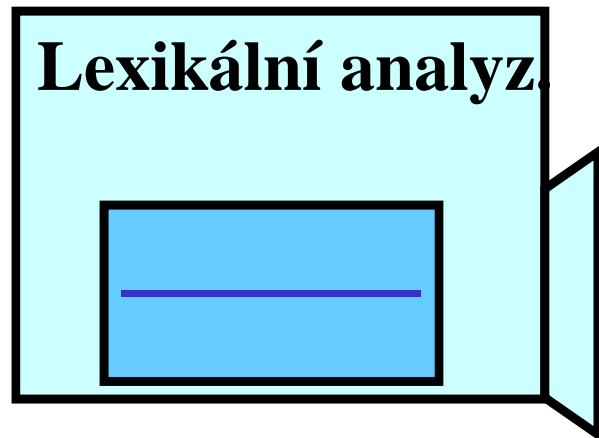
↓ Čti další znak



Příklad:

Zdrojový program:

`Pos := Rate*60`



Syntaktický analyz.

Lexikální analyzátor (Scanner)

Zdrojový program

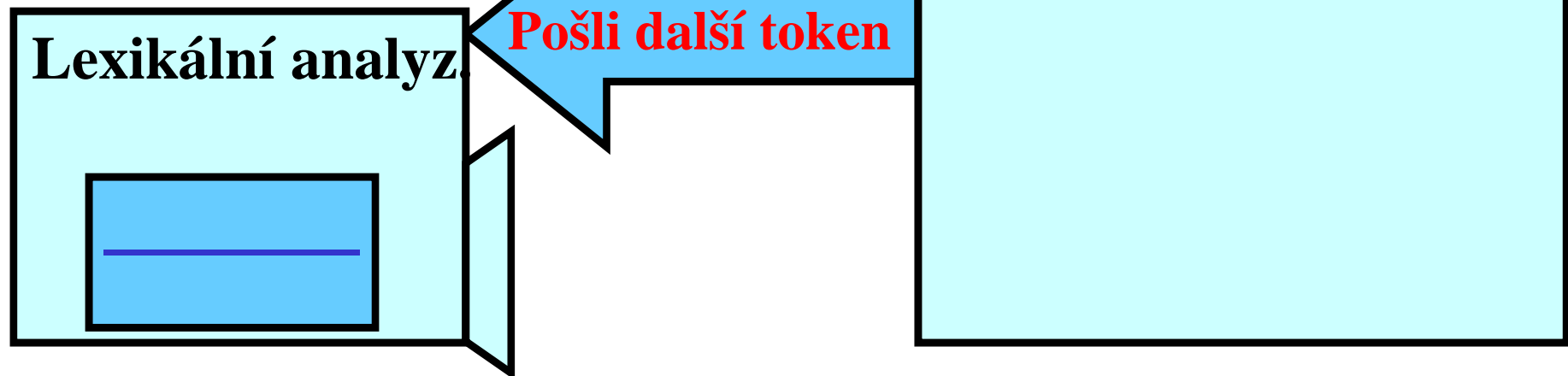
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

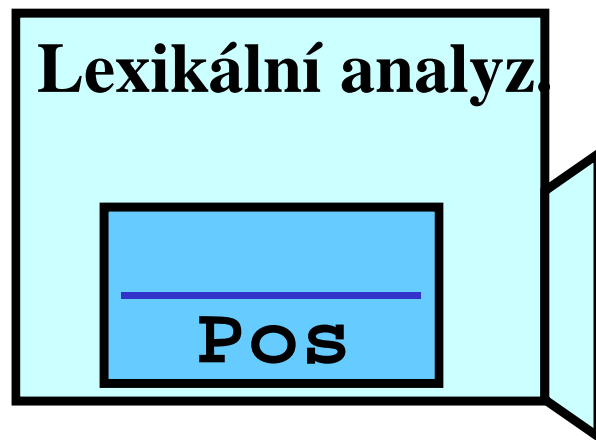
↓ Čti další znak



Příklad:

Zdrojový program:

`Pos := Rate * 60`



Syntaktický analyz.

Lexikální analyzátor (Scanner)

Zdrojový program

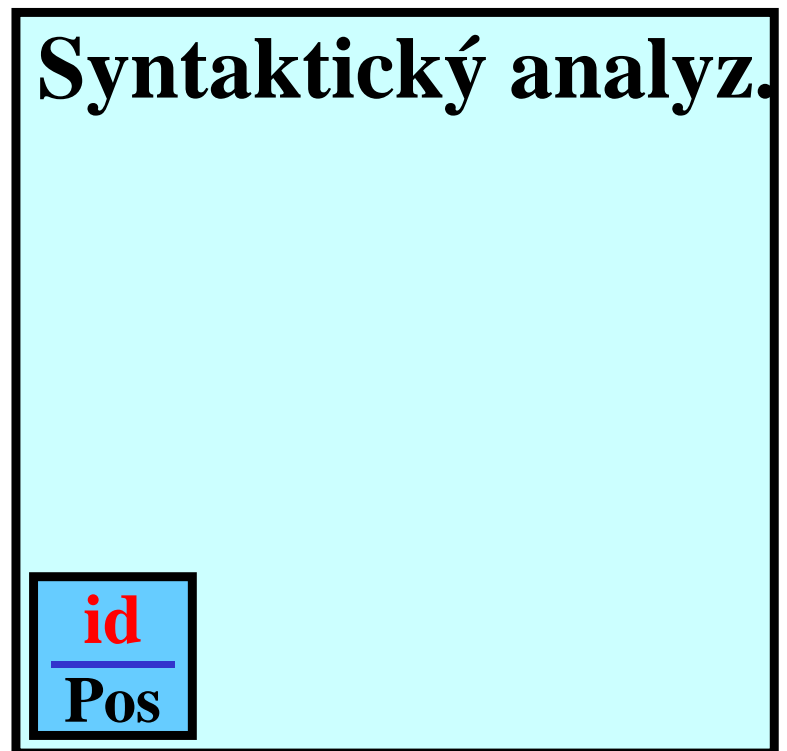
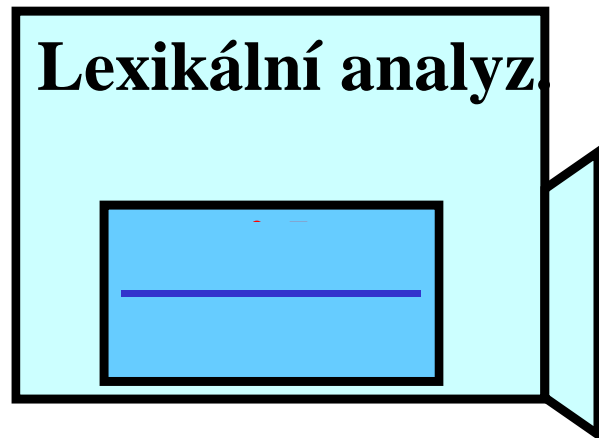
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate * 60



Lexikální analyzátor (Scanner)

Zdrojový program

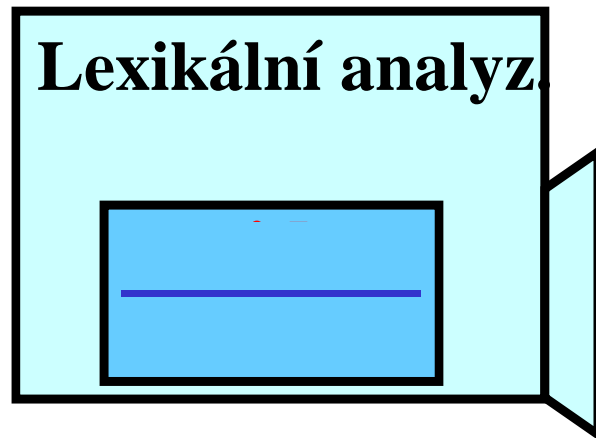
↓ Čti další znak



Příklad:

Zdrojový program:

`Pos := Rate*60`



Syntaktický analyz.

Assignment

Expr

id

:=

id

Pos



Lexikální analyzátor (Scanner)

Zdrojový program

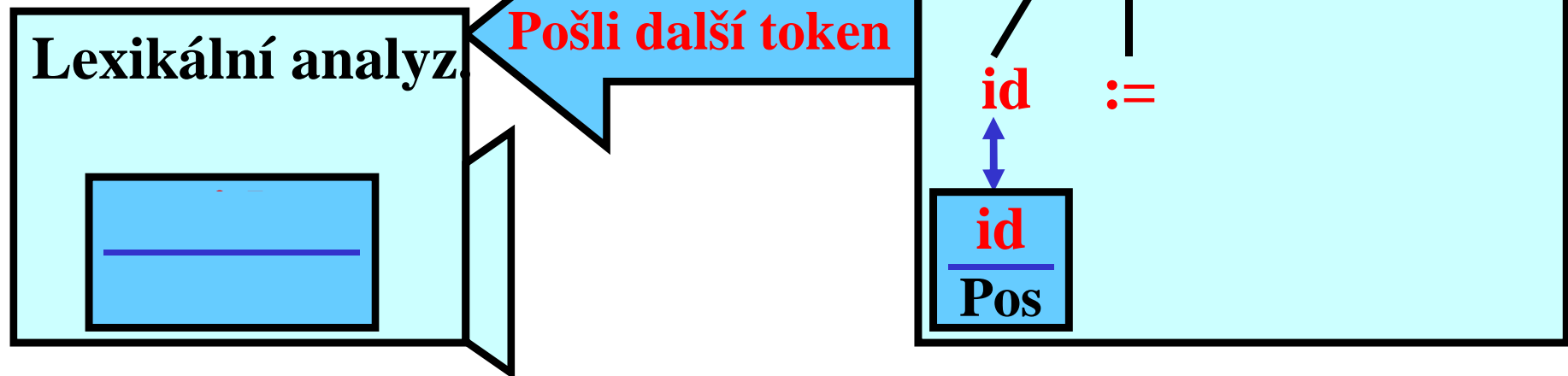
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

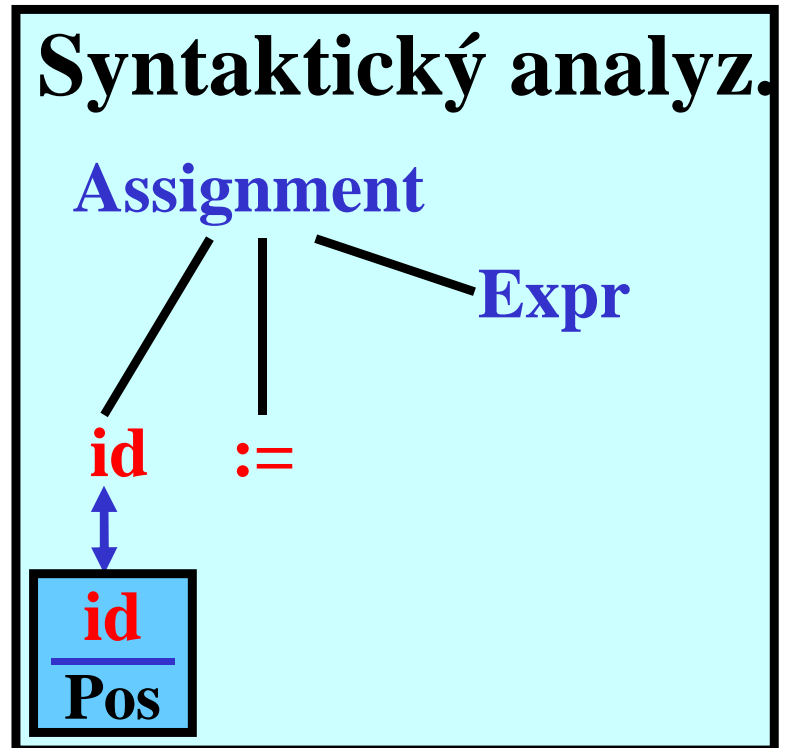
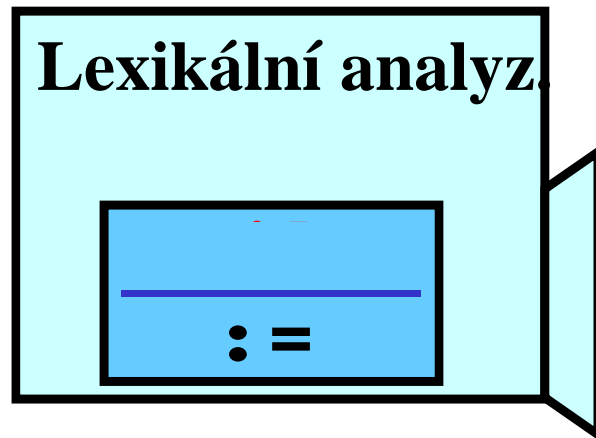
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate * 60



Lexikální analyzátor (Scanner)

Zdrojový program

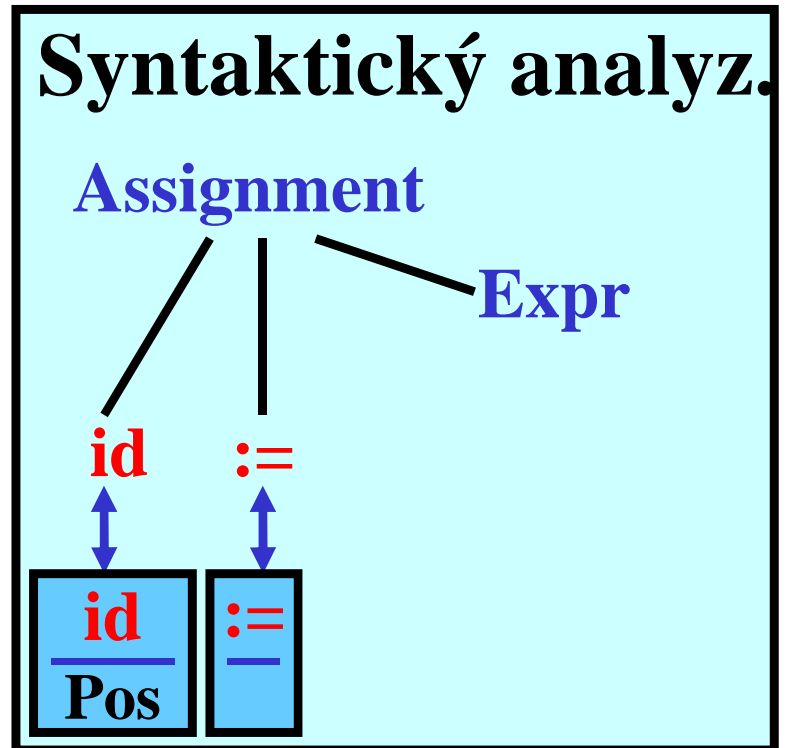
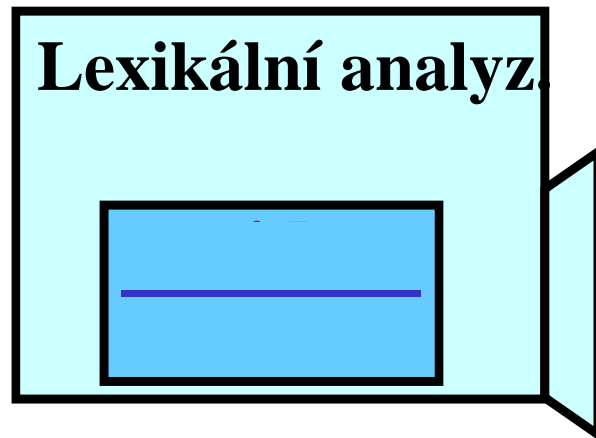
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate * 60



Lexikální analyzátor (Scanner)

Zdrojový program

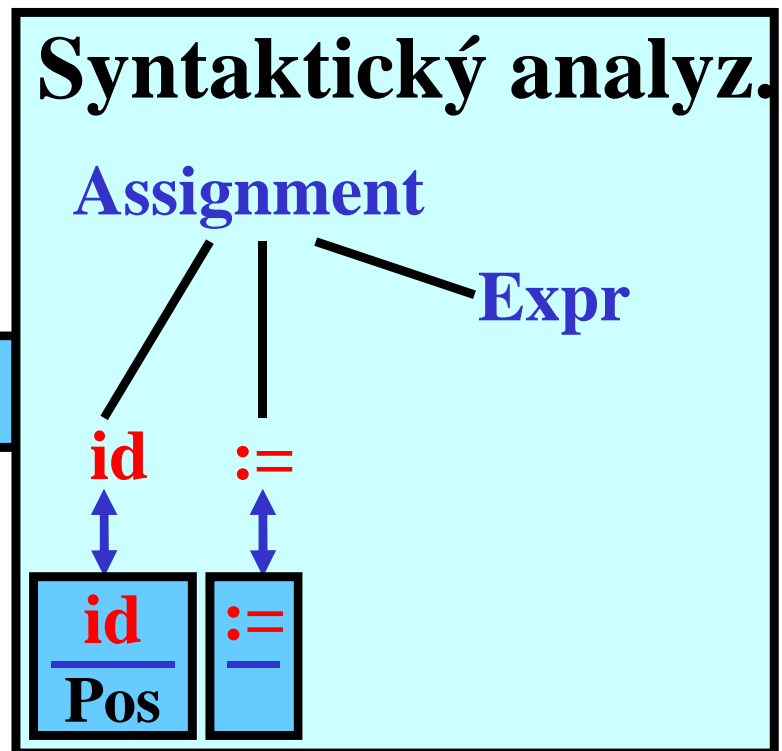
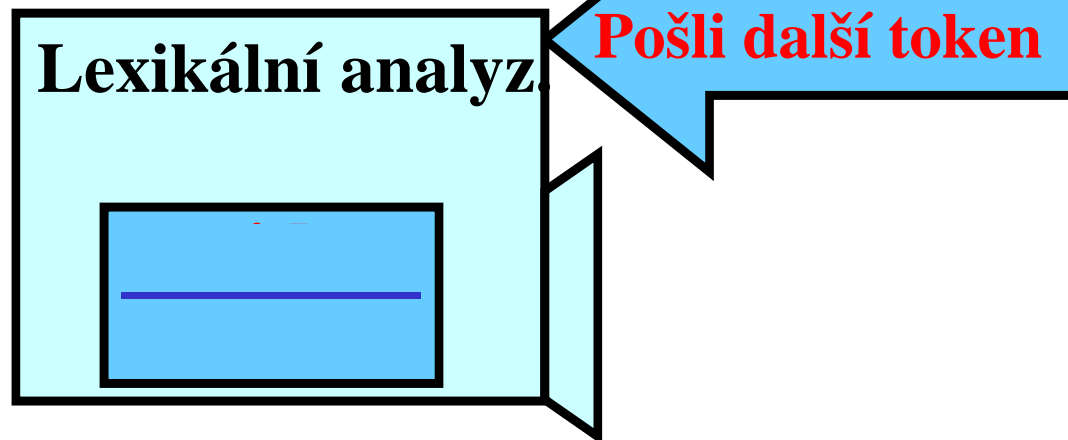
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate * 60



Lexikální analyzátor (Scanner)

Zdrojový program

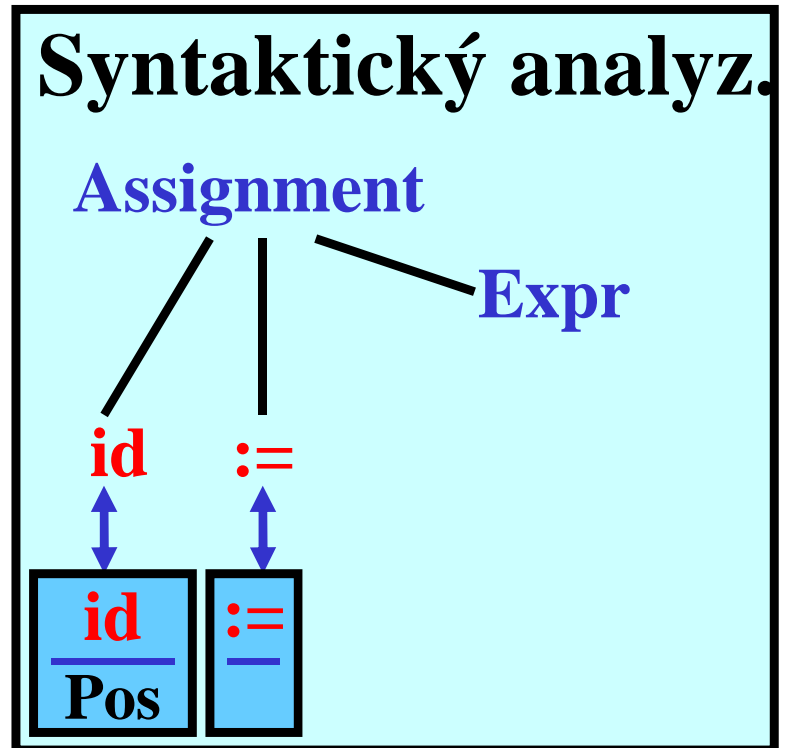
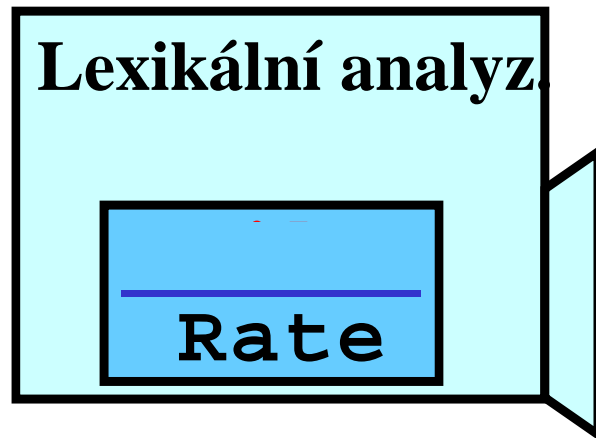
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

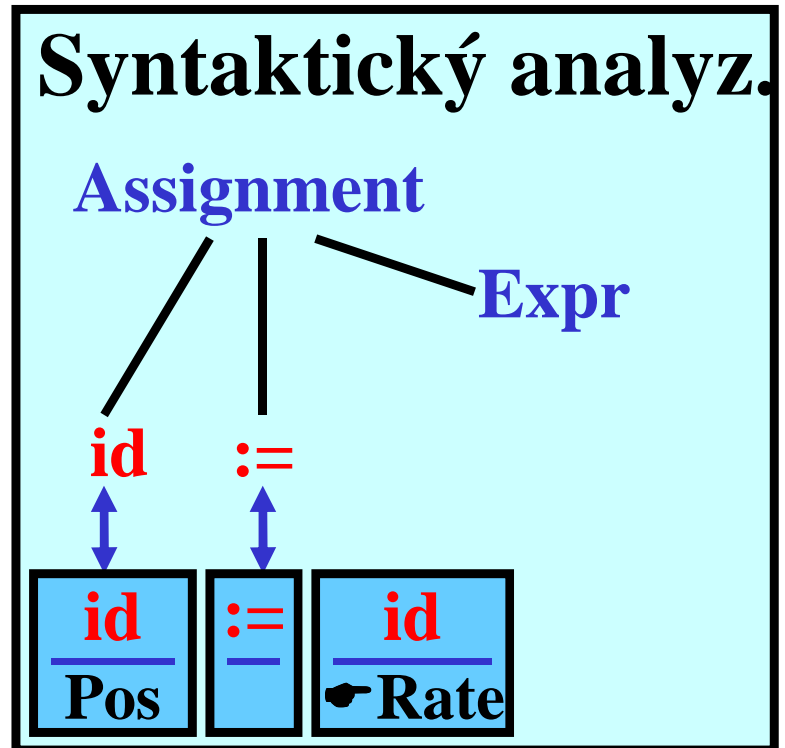
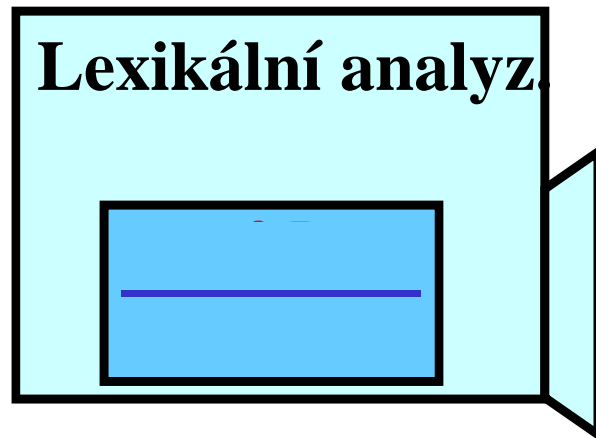
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*6.0



Lexikální analyzátor (Scanner)

Zdrojový program

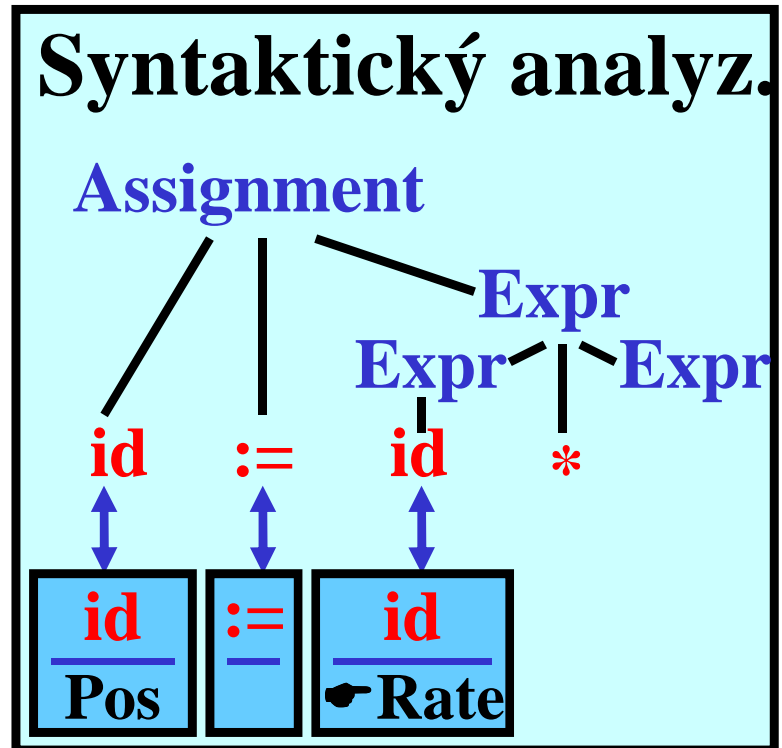
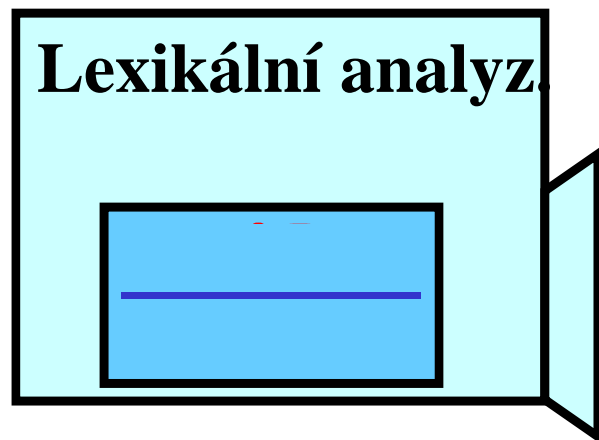
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

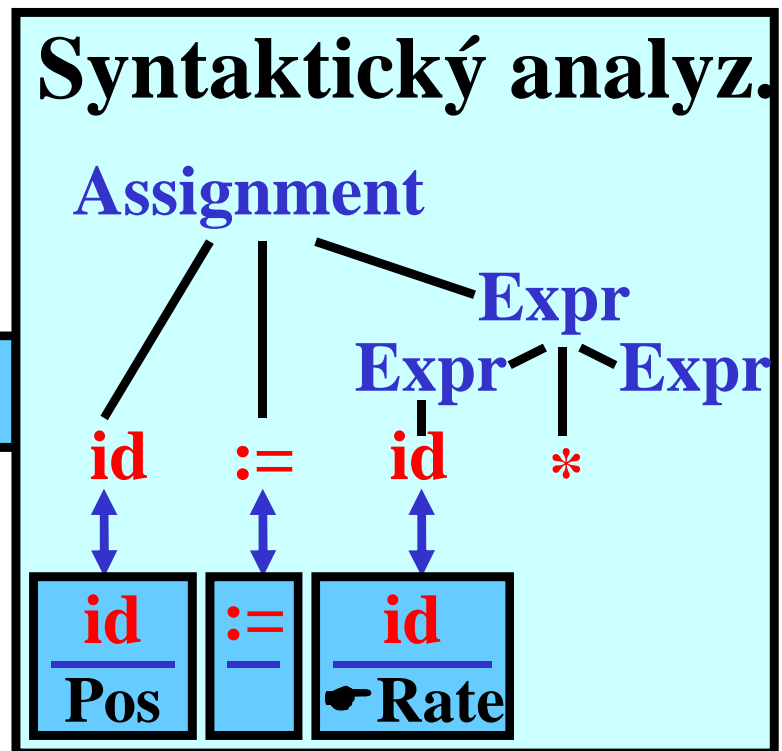
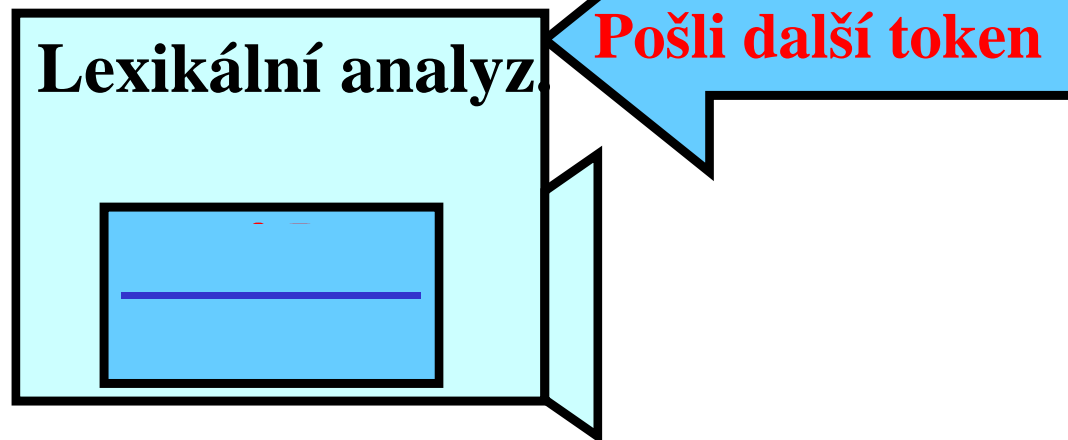
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

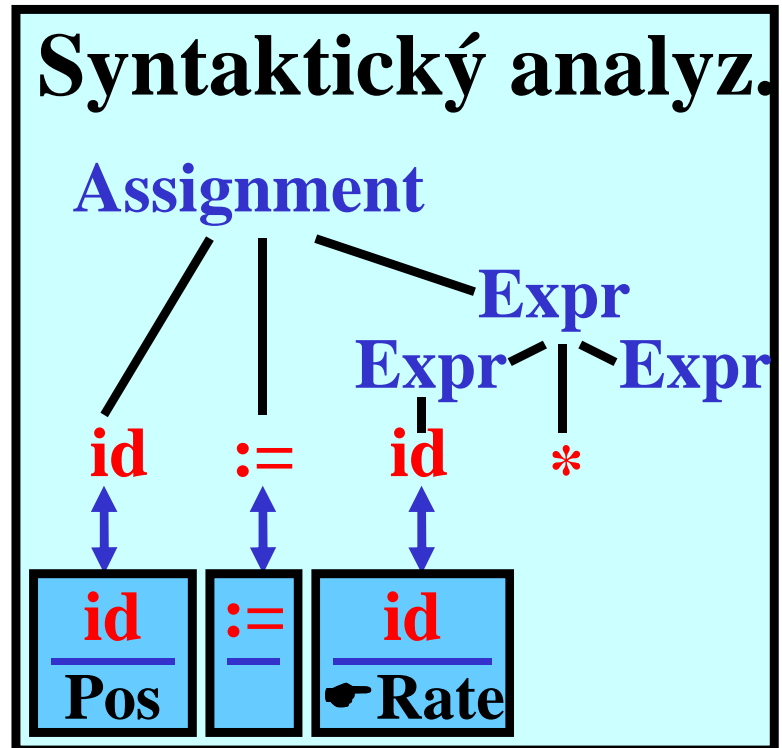
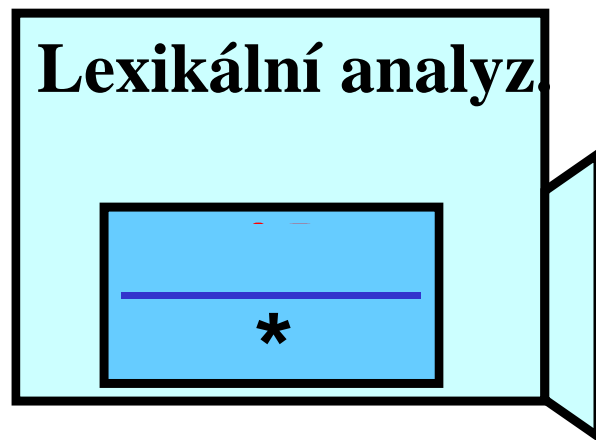
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

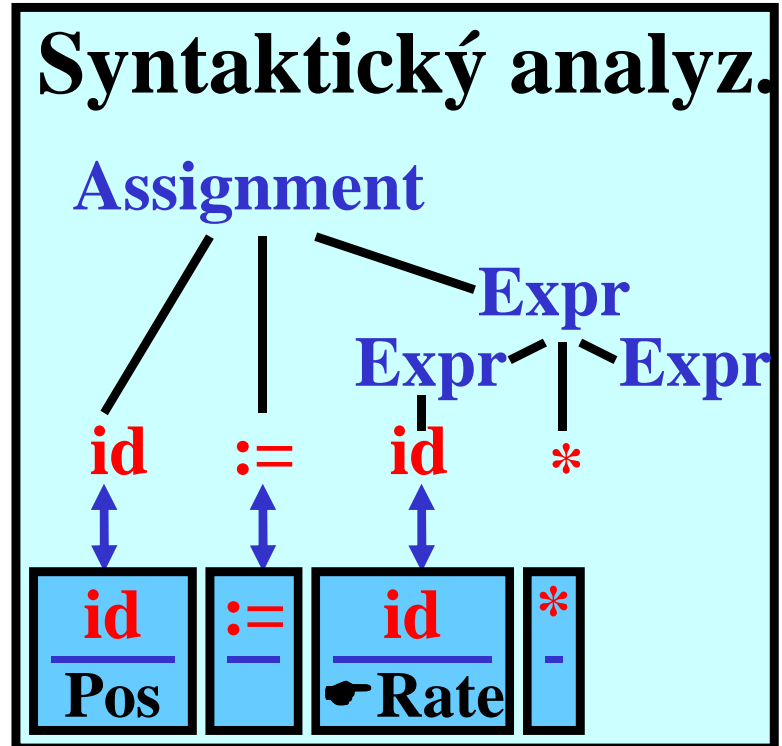
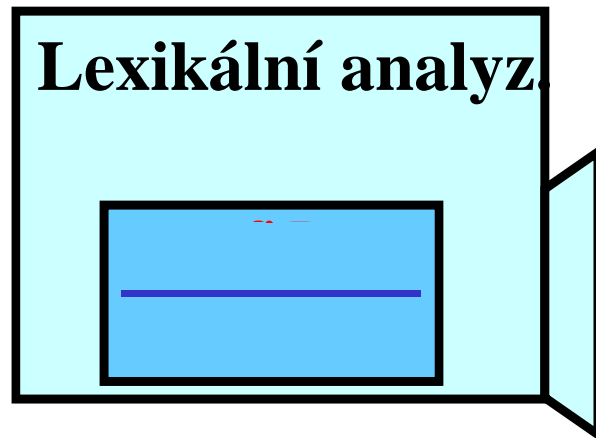
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

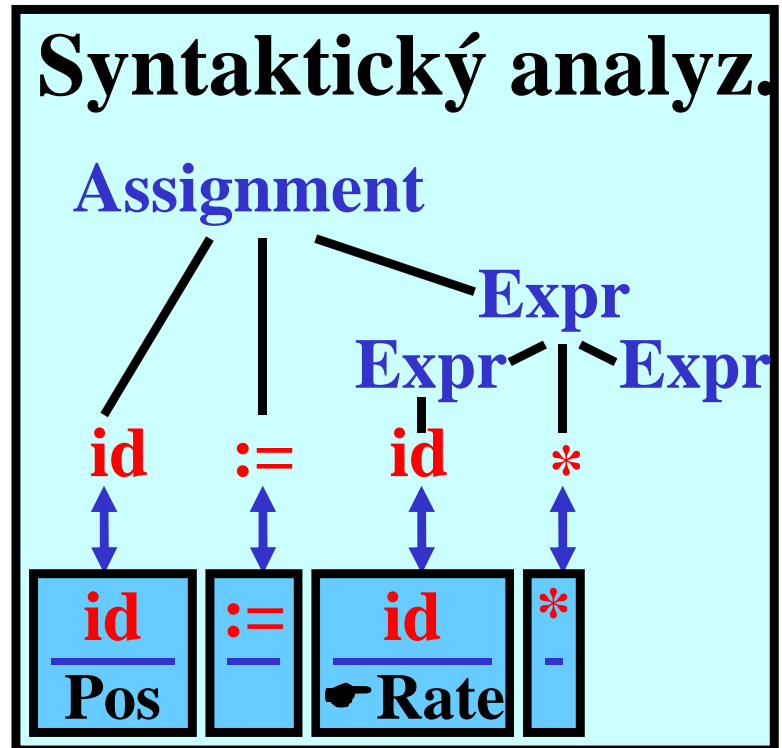
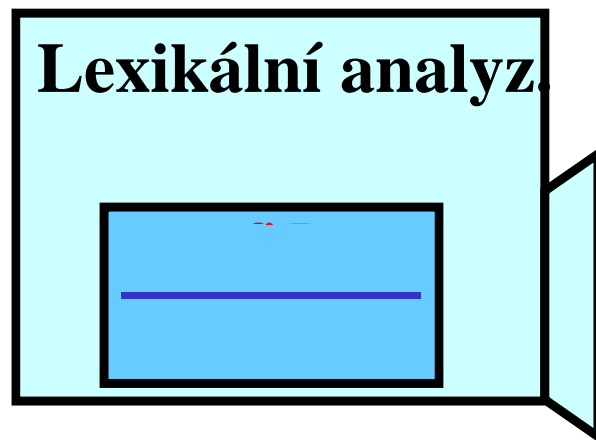
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

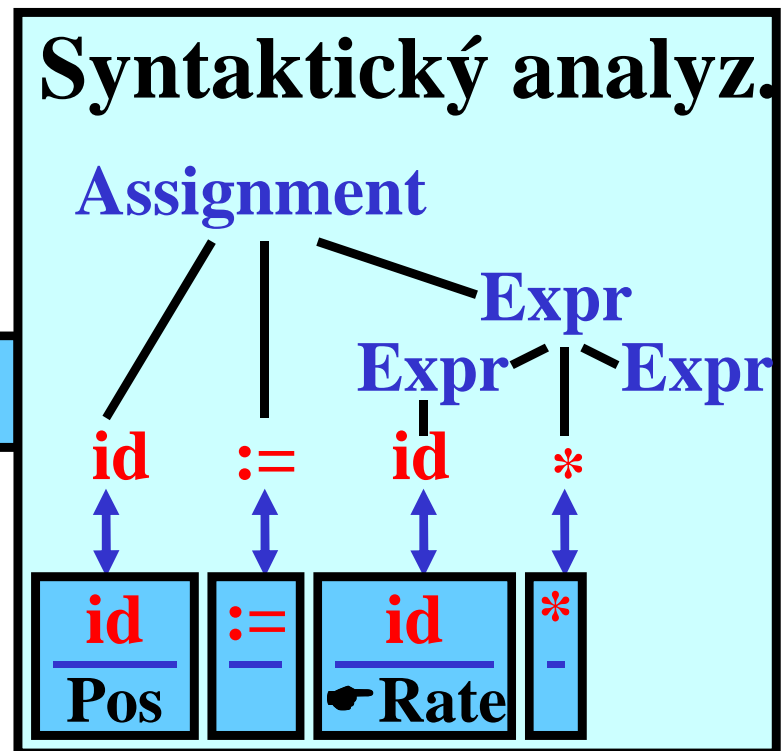
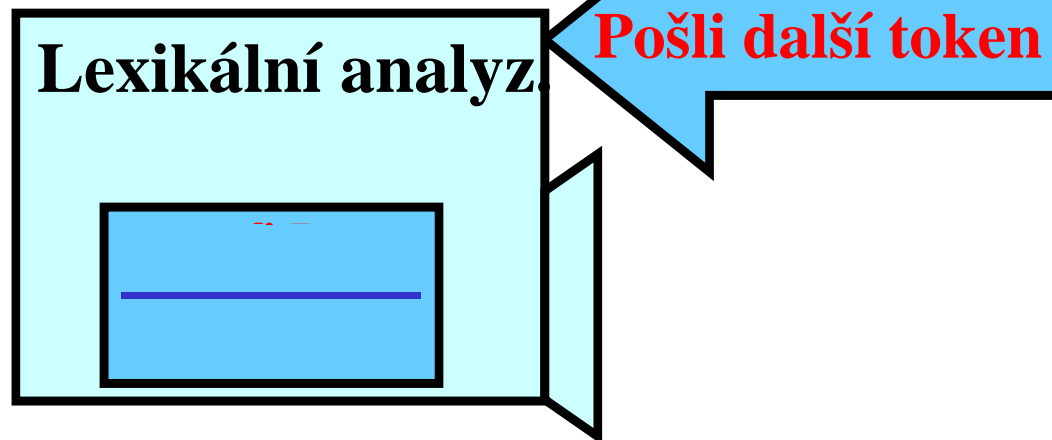
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

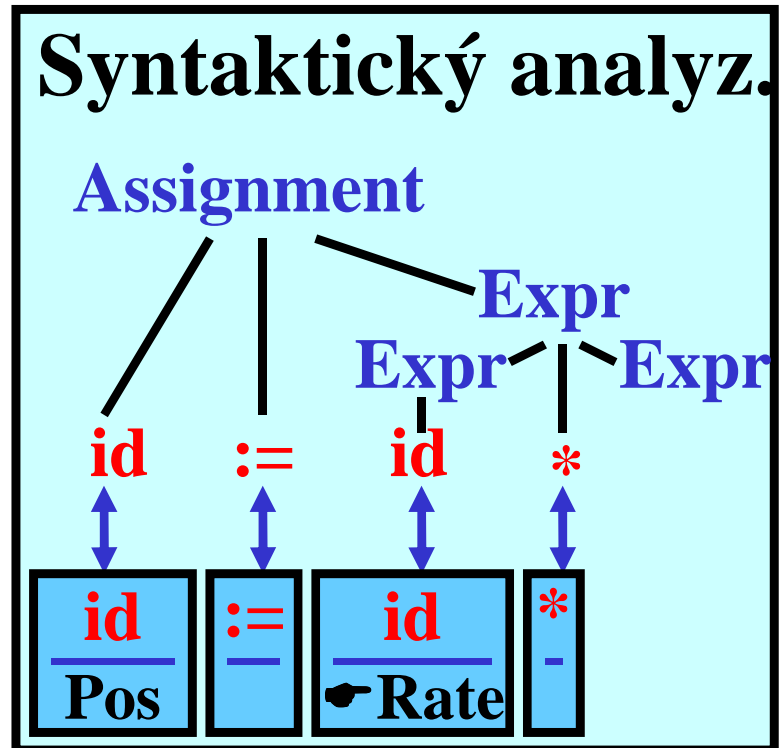
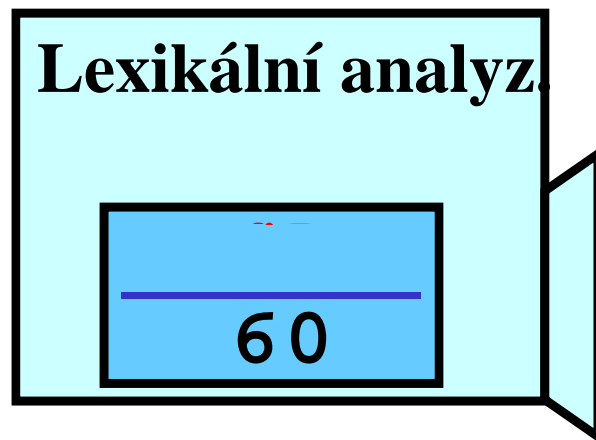
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

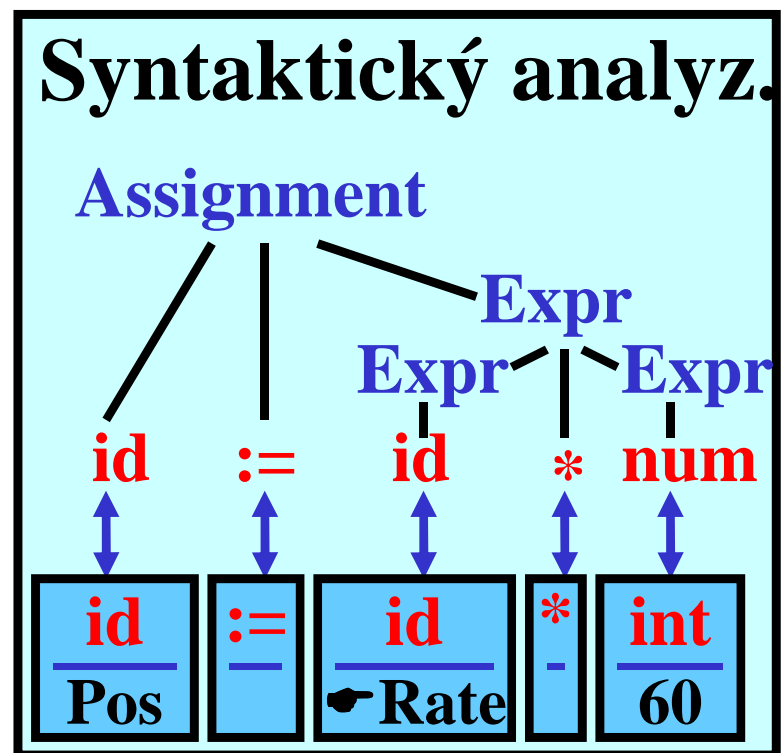
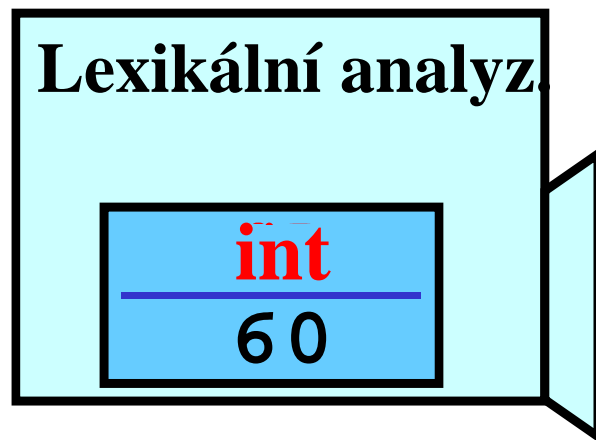
↓ Čti další znak



Příklad:

Zdrojový program:

Pos := Rate*60



Lexikální analyzátor (Scanner)

Zdrojový program

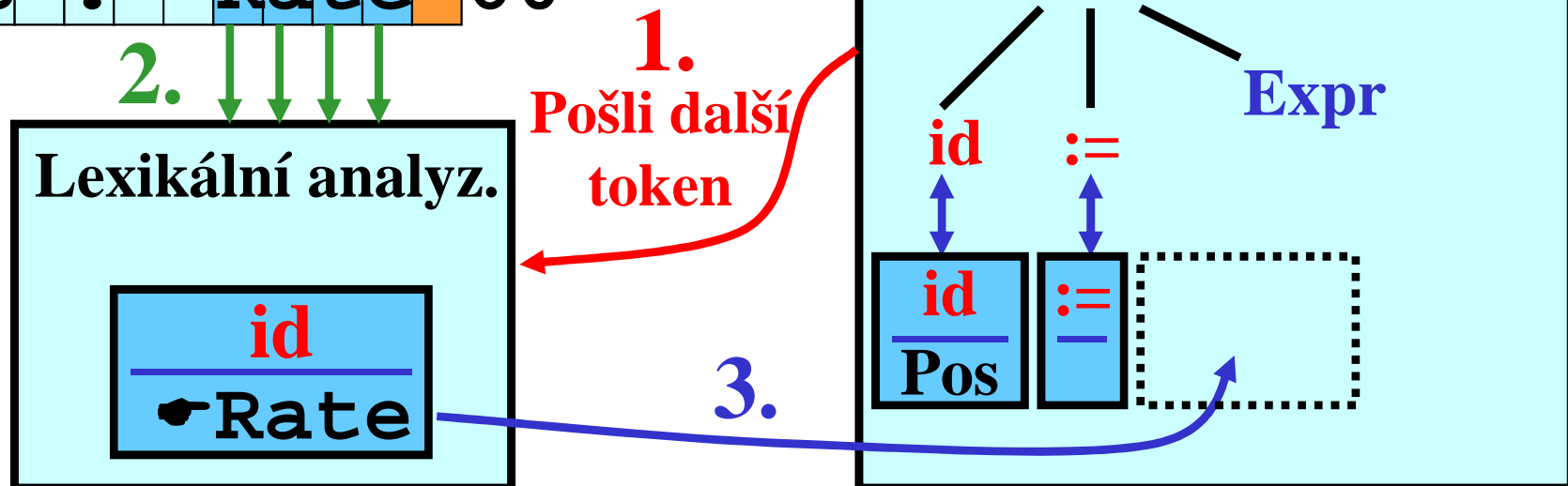
↓ Čti další znak



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:

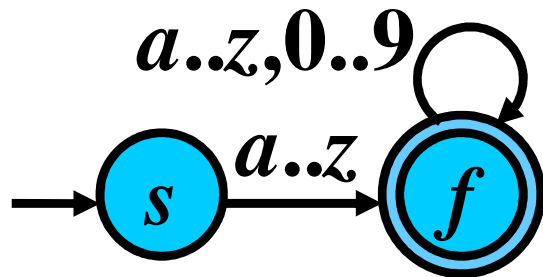
- **Regulární výrazy** specifikují lexémy
- Na **DKA** je založena implementace

Rozpoznávání lexémů pomocí DKA 1/2

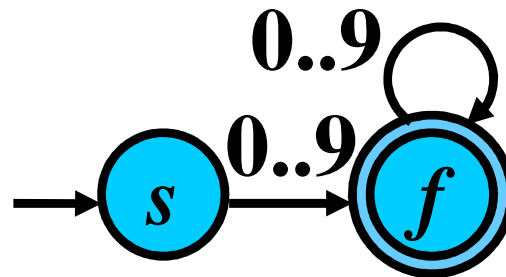
1) Rozpoznávání jednotlivých lexémů pomocí DKA:

Příklad:

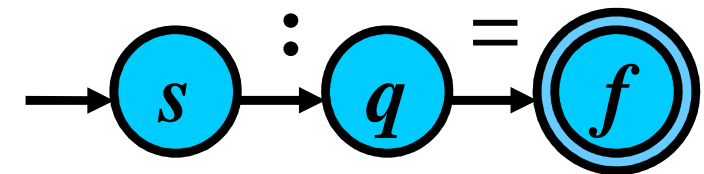
Identifikátor:



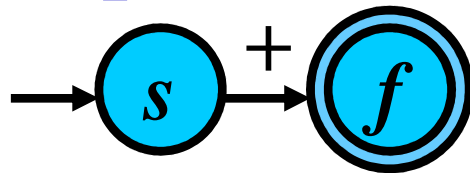
Celé číslo:



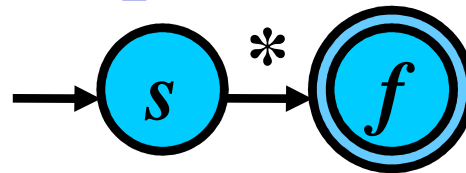
Přiřazení:



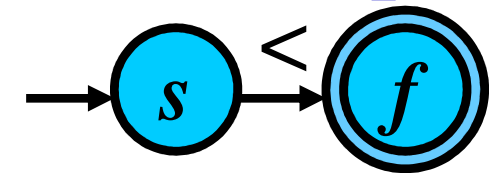
Operátor +:



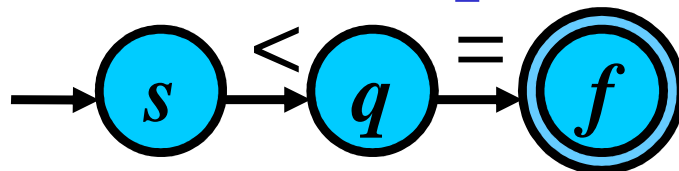
Operátor *:



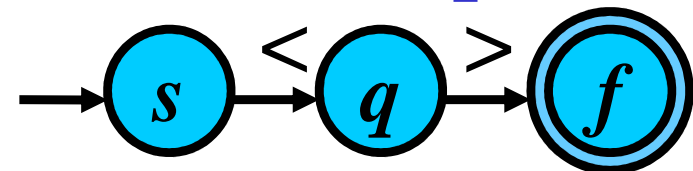
Relační op. <:



Relační op. <=:



Relační op. <>:



Rozpoznávání lexémů pomocí DKA 2/2

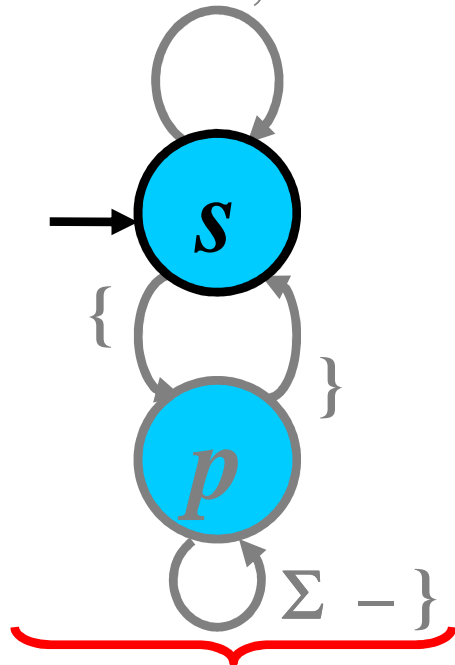
2) Konstrukce KA, který rozpoznává všechny lexémy:

Rozpoznávání lexémů pomocí DKA 2/2

2) Konstrukce KA, který rozpoznává všechny lexémy:

Space, Tab,

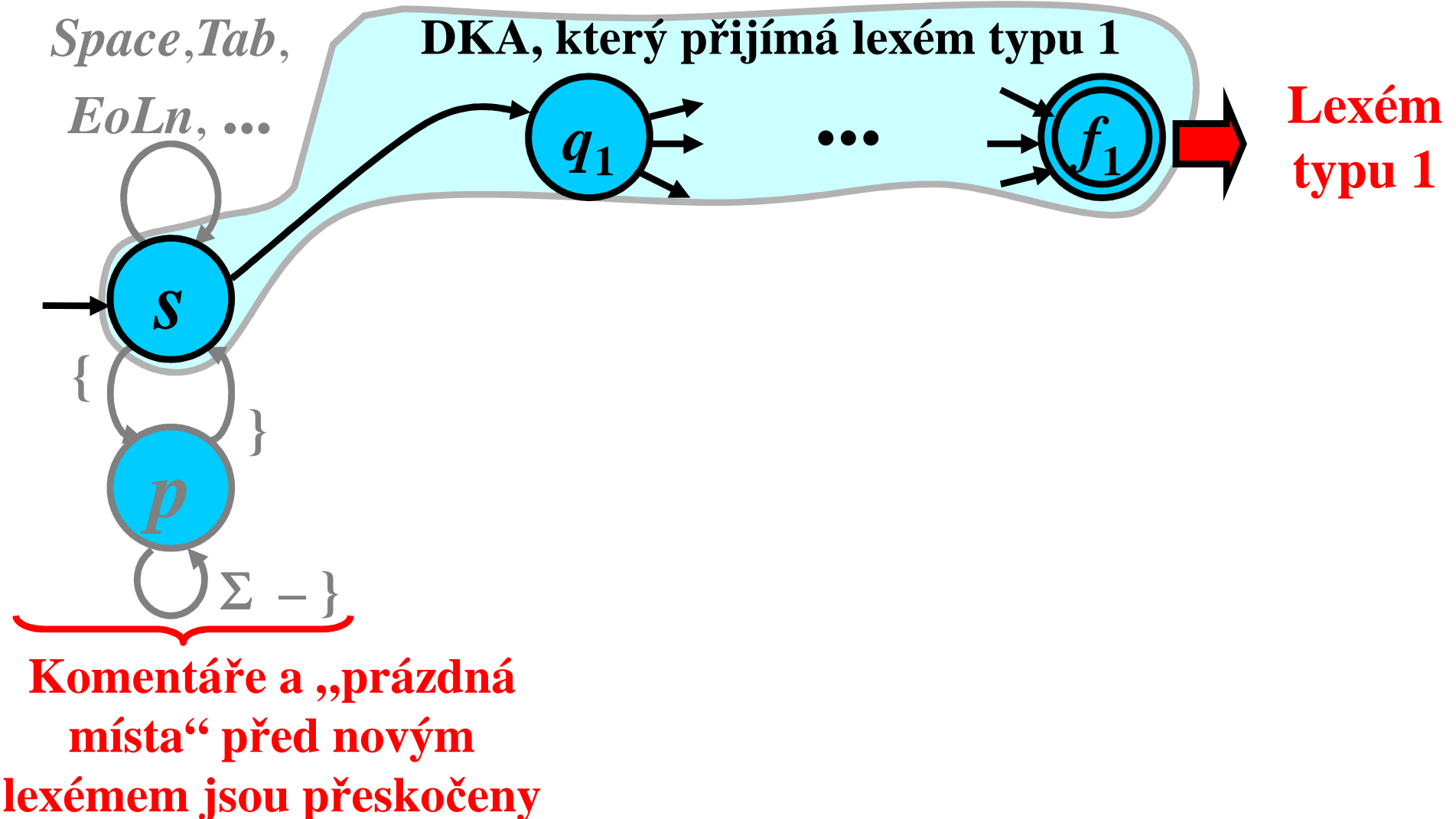
EoLn, ...



**Komentáře a „prázdná
místa“ před novým
lexémem jsou přeskočeny**

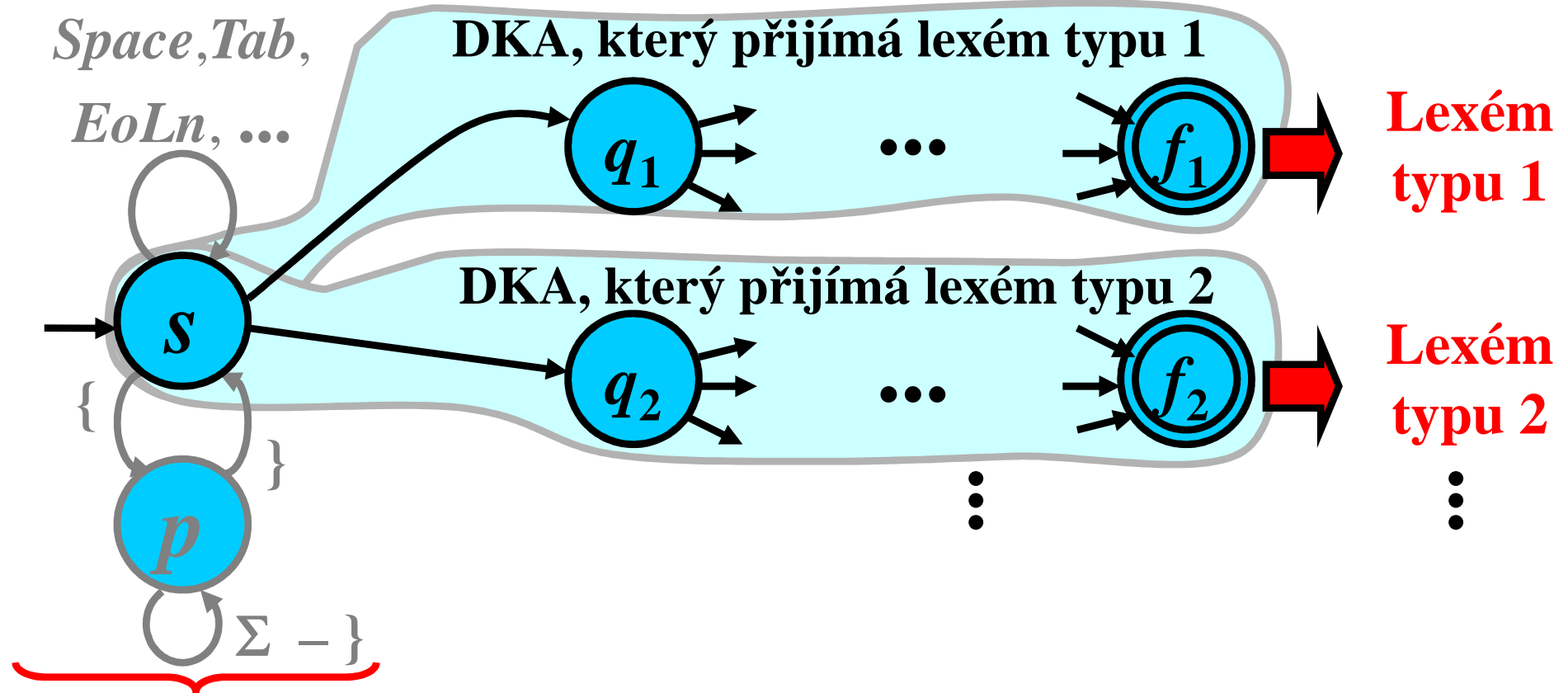
Rozpoznávání lexémů pomocí DKA 2/2

2) Konstrukce KA, který rozpoznává všechny lexémy:



Rozpoznávání lexémů pomocí DKA 2/2

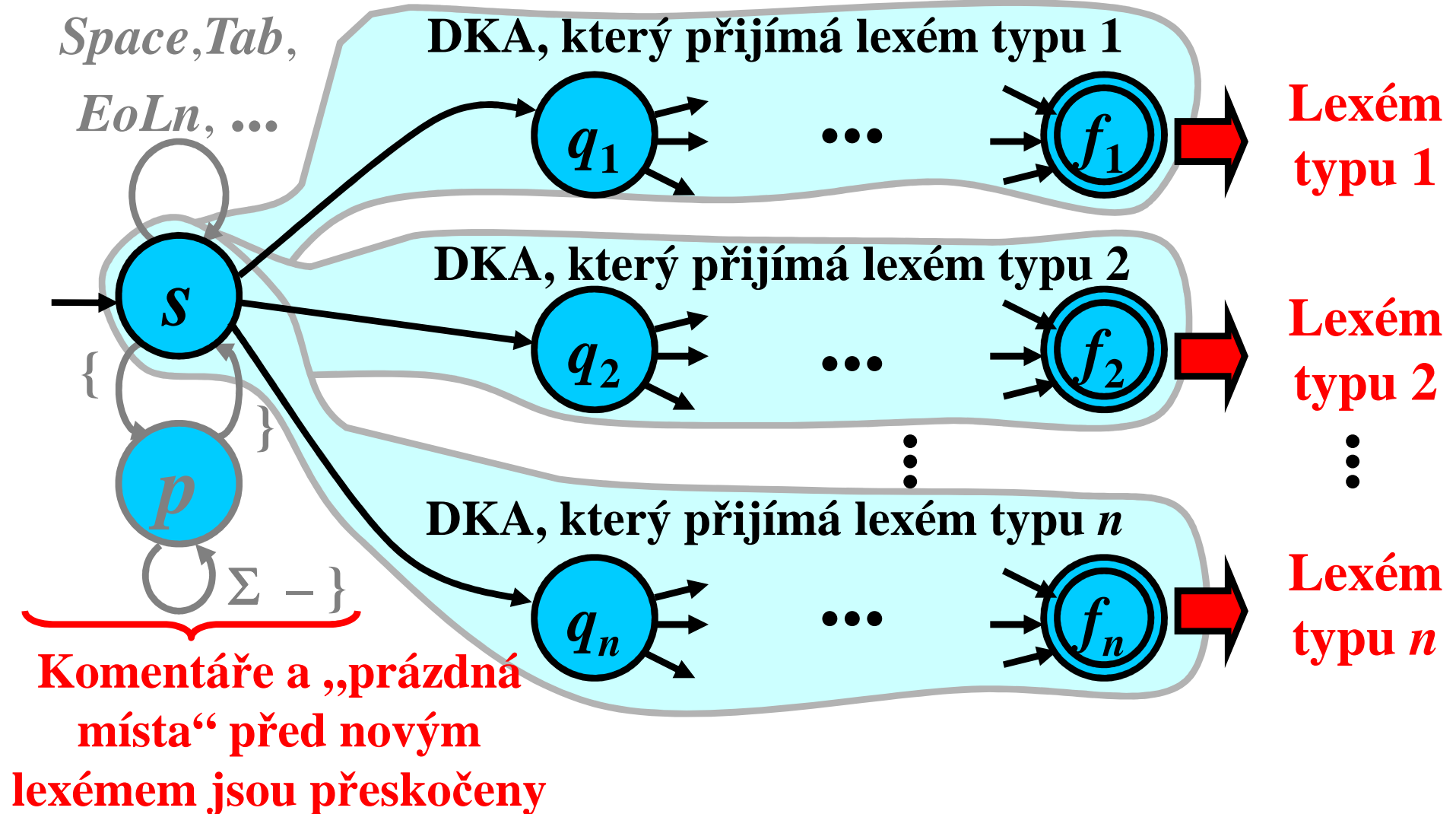
2) Konstrukce KA, který rozpoznává všechny lexémy:



Komentáře a „prázdná místa“ před novým lexémem jsou přeskočeny

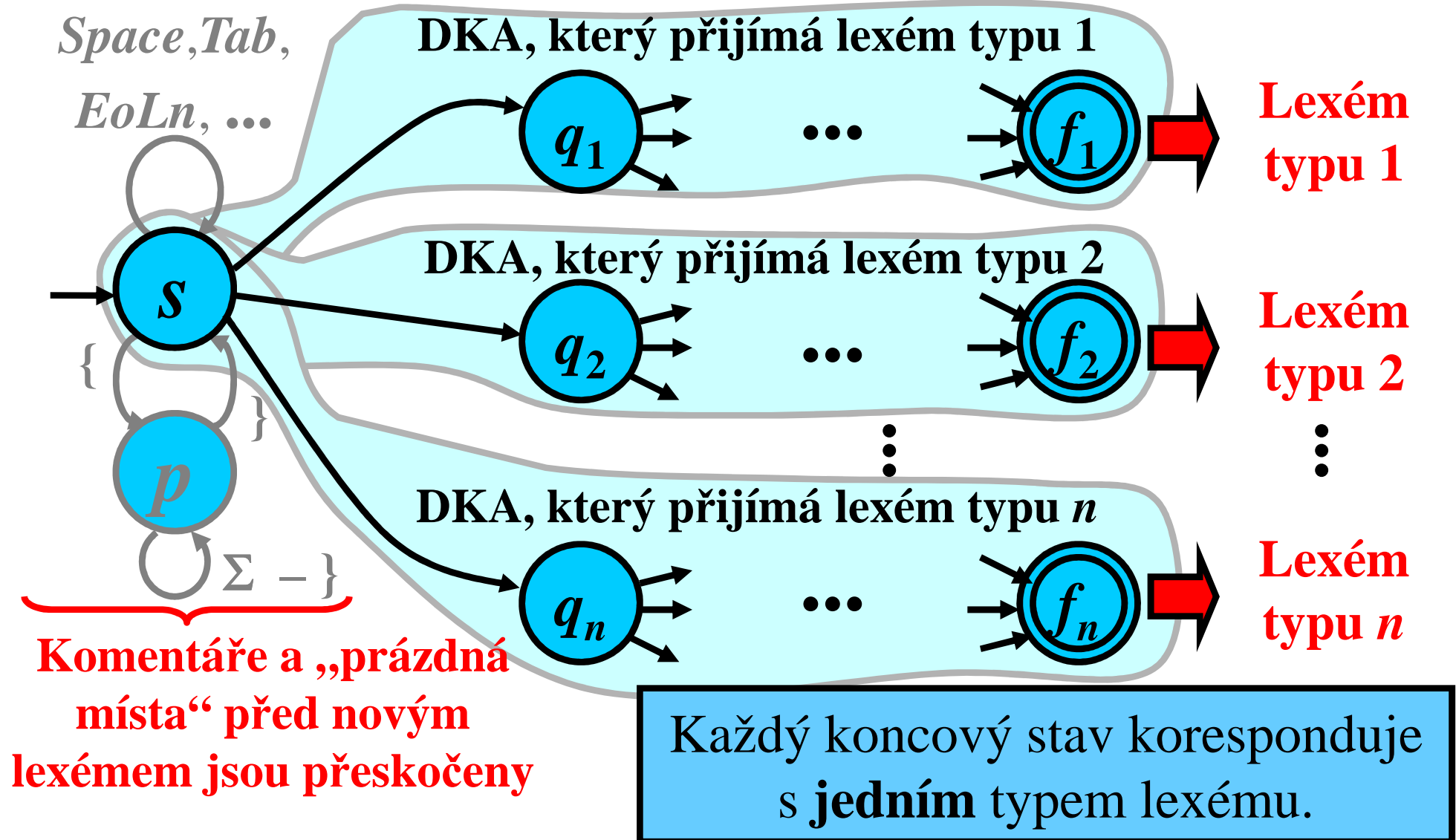
Rozpoznávání lexémů pomocí DKA 2/2

2) Konstrukce KA, který rozpoznává všechny lexémy:



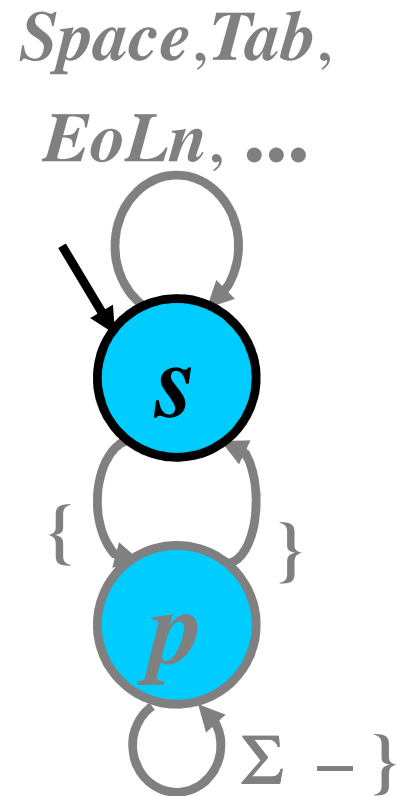
Rozpoznávání lexémů pomocí DKA 2/2

2) Konstrukce KA, který rozpoznává všechny lexémy:



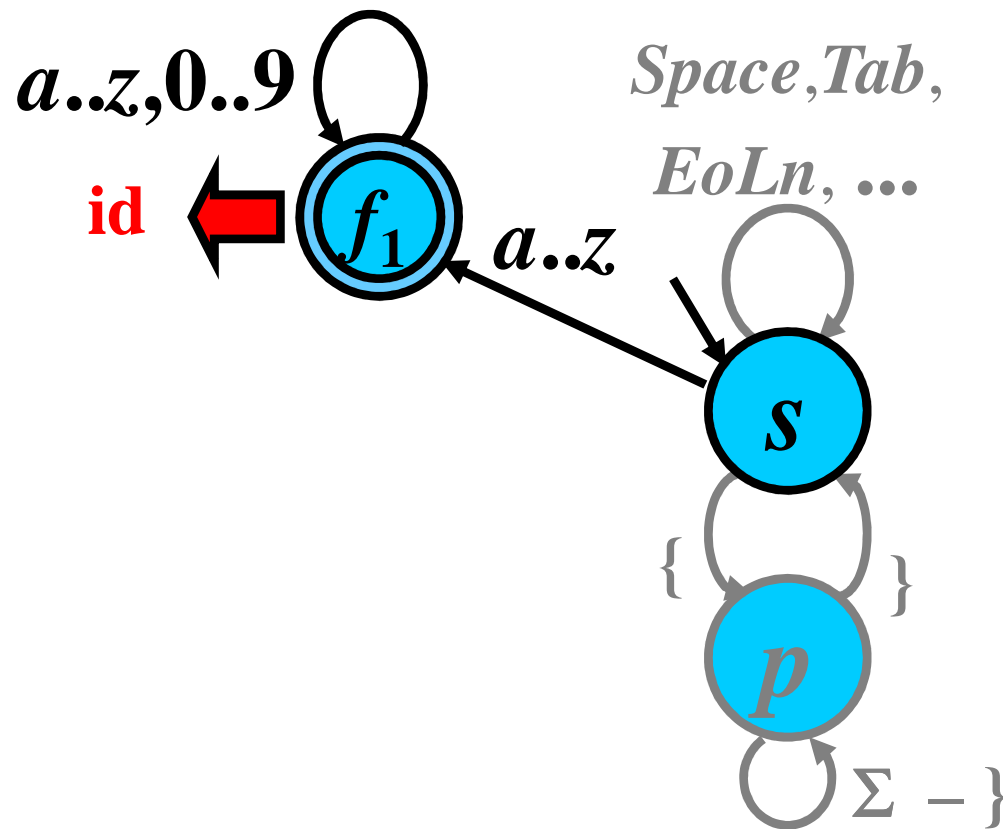
DKA přijímací lexémy: Příklad 1/2

- KA, který přijímá následující lexémy:



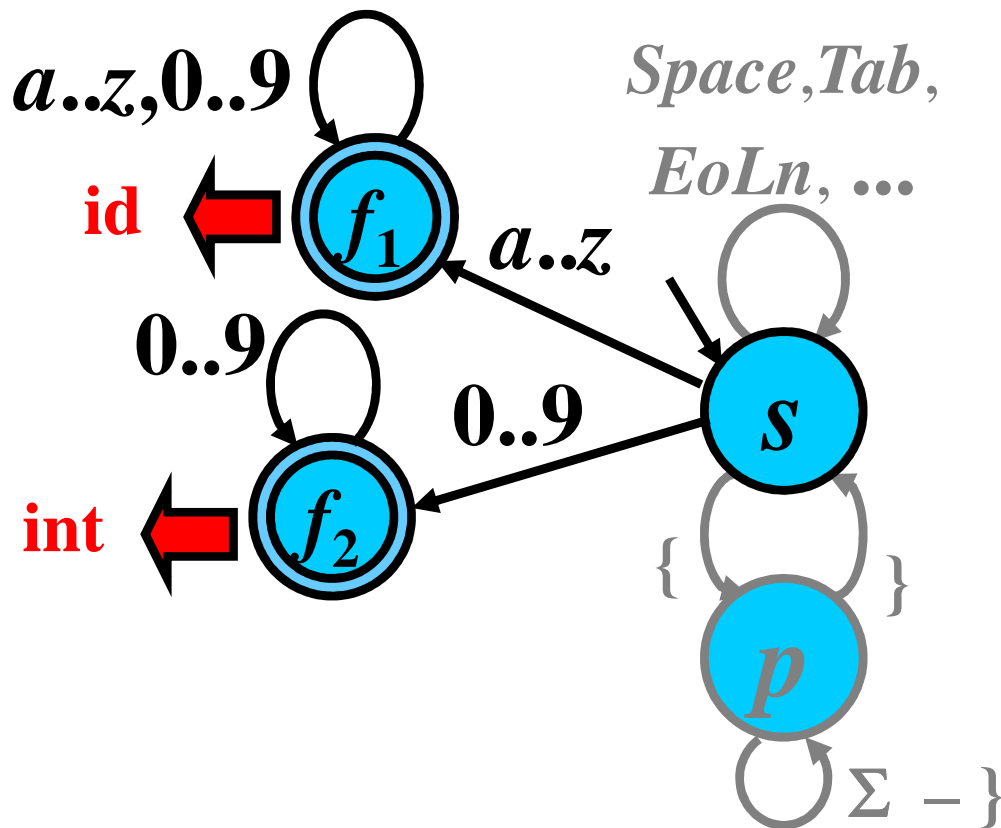
DKA přijímací lexémy: Příklad 1/2

- KA, který přijímá následující lexémy:
identifikátory,



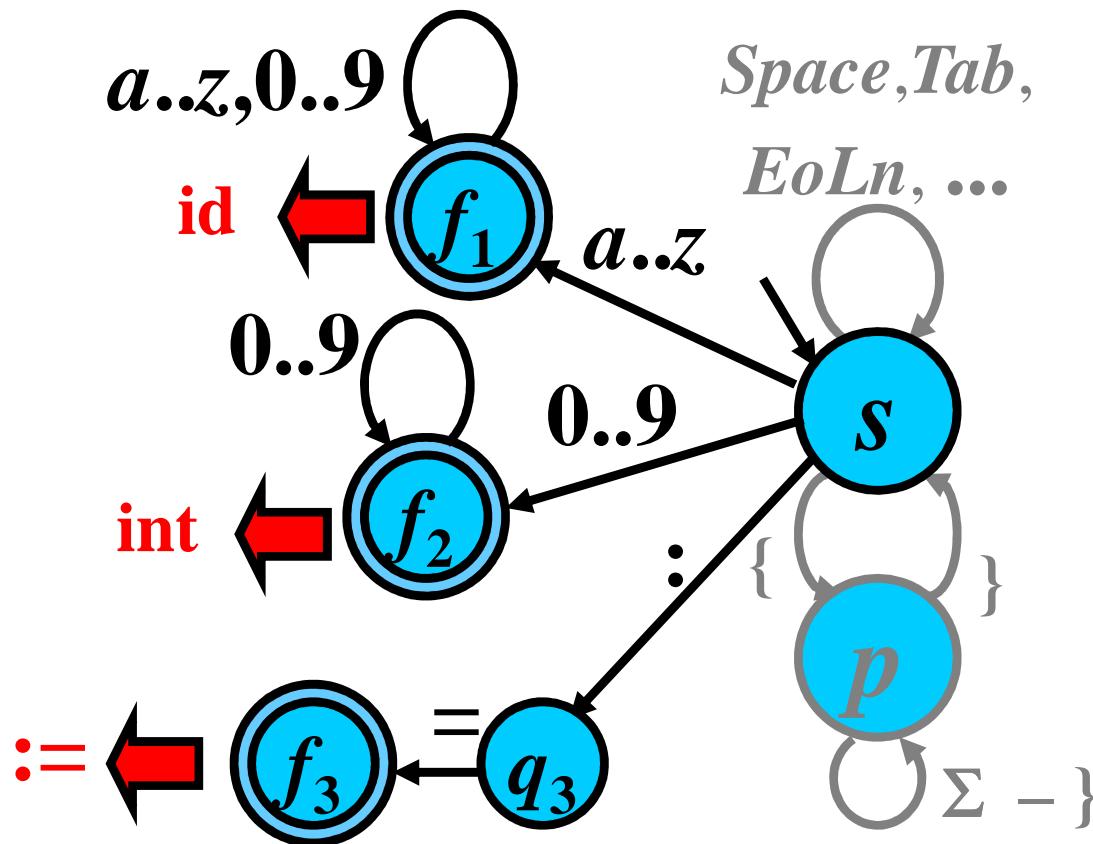
DKA přijímací lexémy: Příklad 1/2

- KA, který přijímá následující lexémy:
identifikátory, celá čísla,



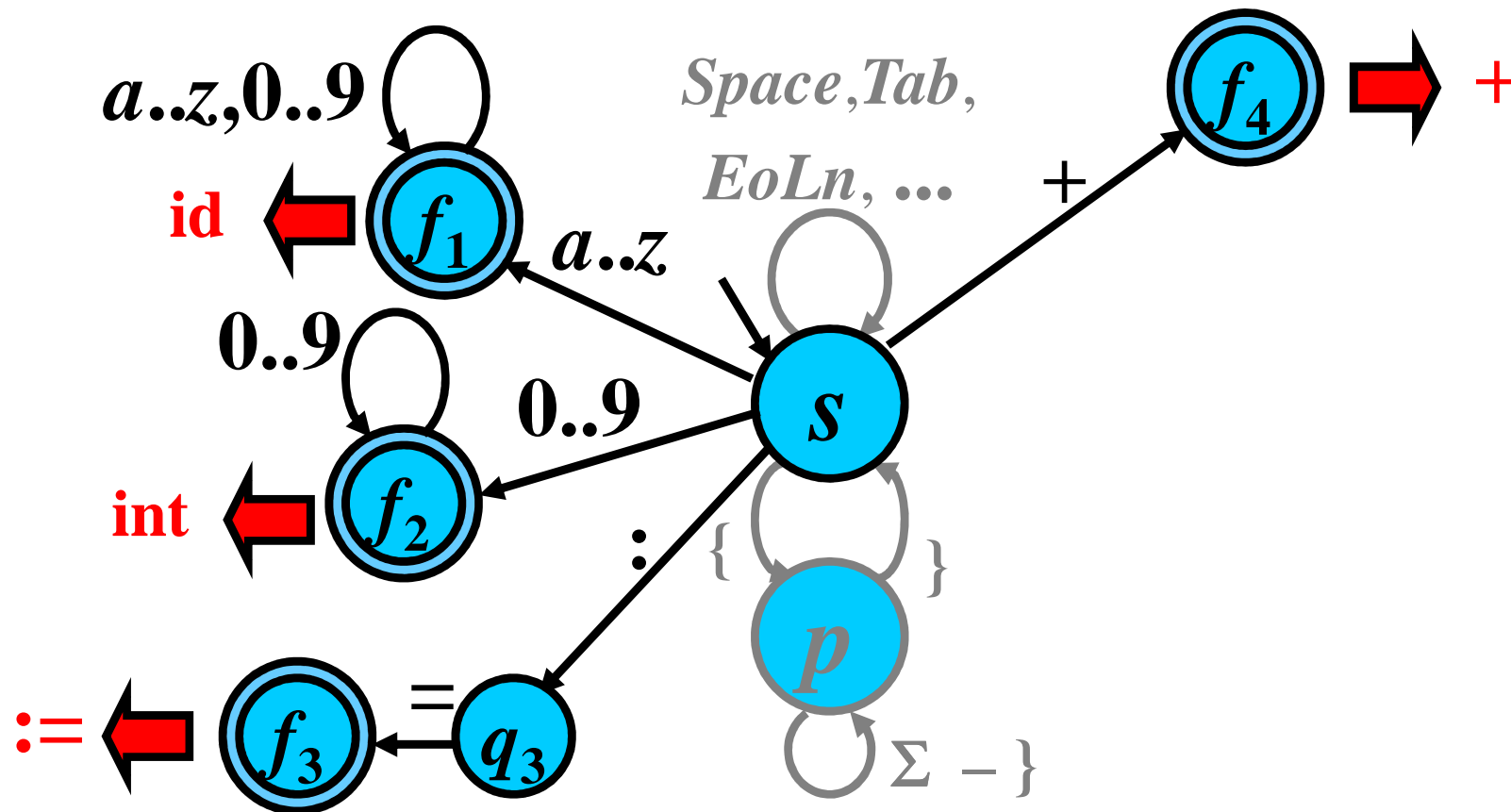
DKA přijímací lexémy: Příklad 1/2

- KA, který přijímá následující lexémy:
identifikátory, celá čísla, $:=$,



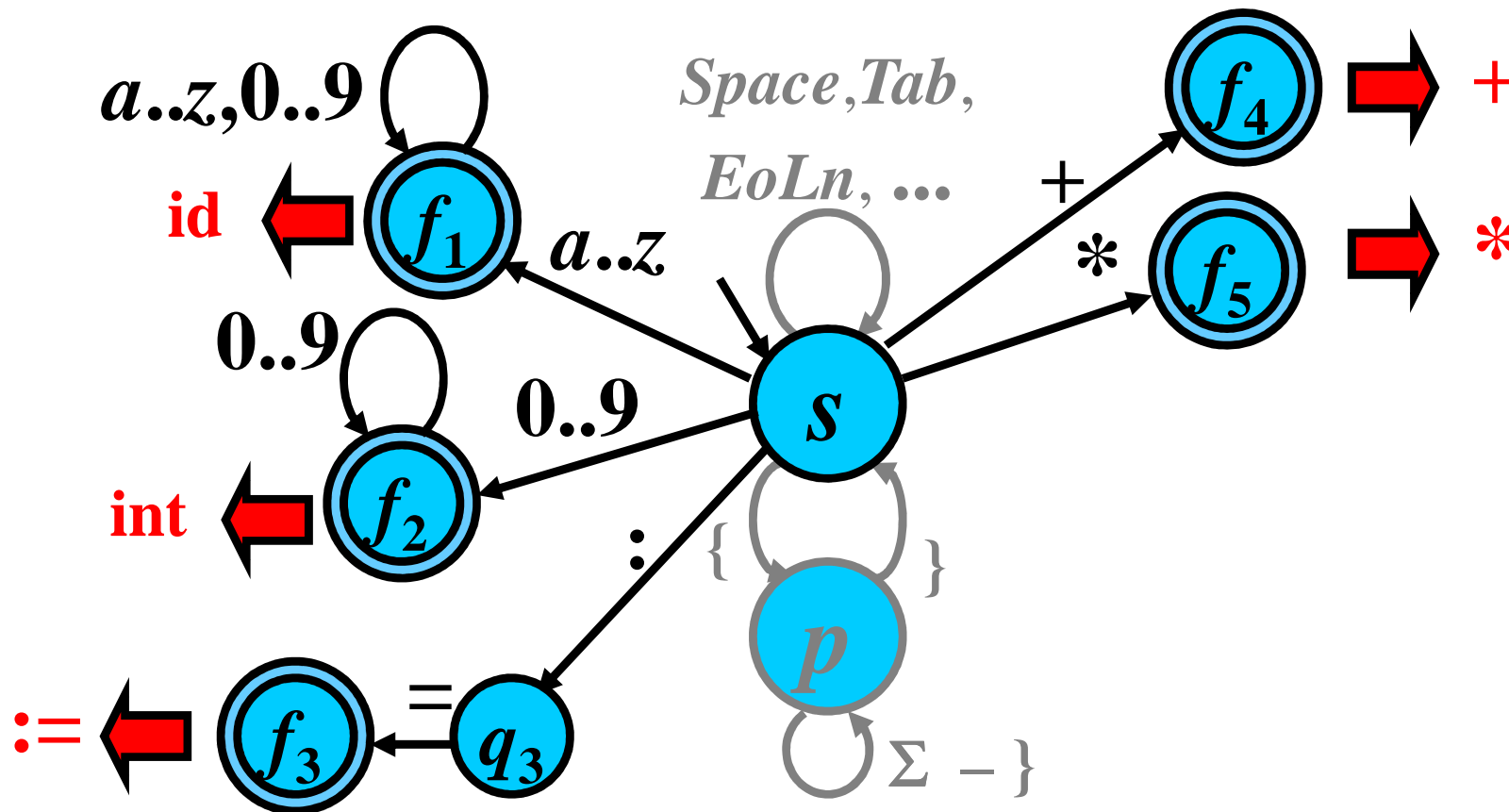
DKA přijímací lexémy: Příklad 1/2

- KA, který přijímá následující lexémy:
identifikátory, celá čísla, $:=$, $+$,



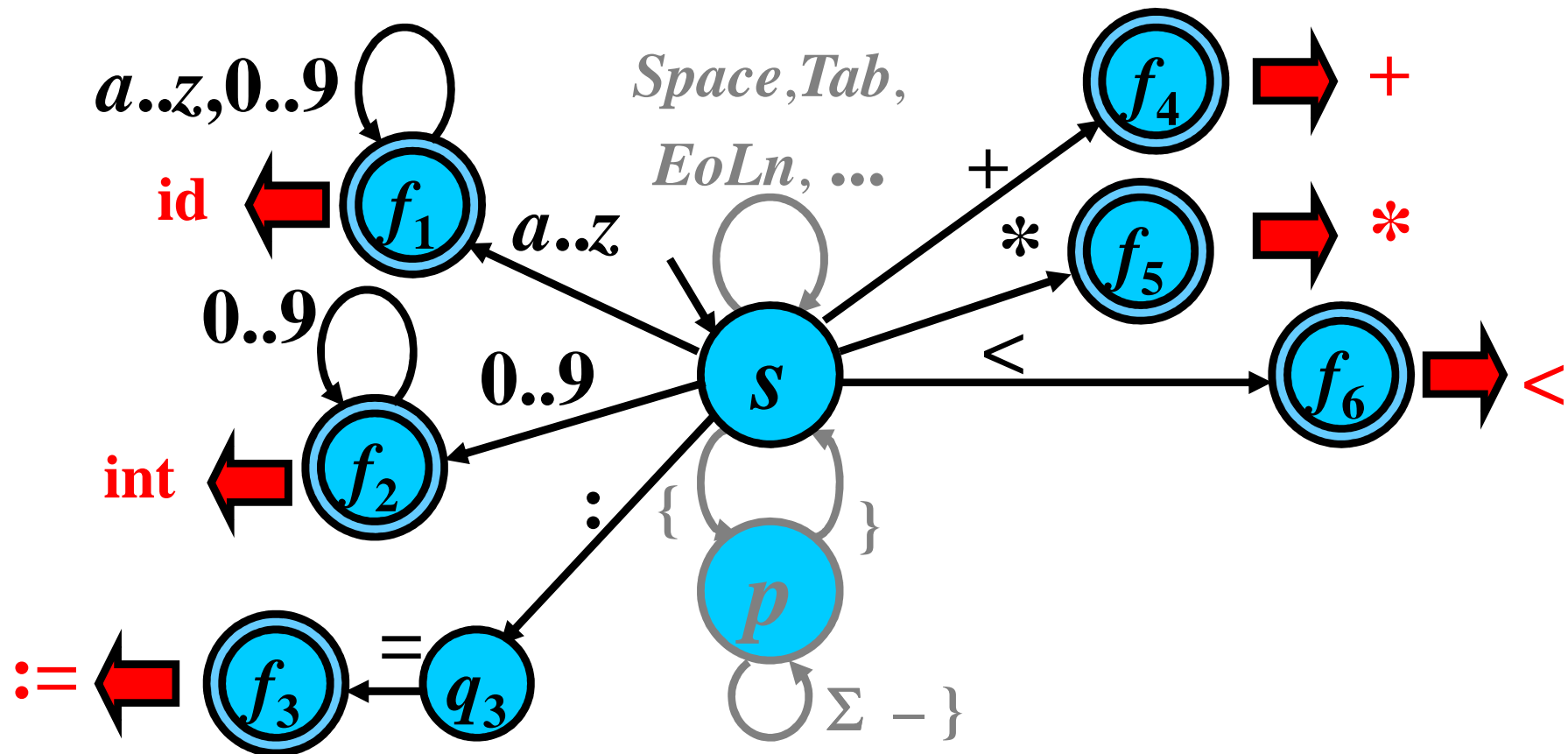
DKA přijímací lexémy: Příklad 1/2

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



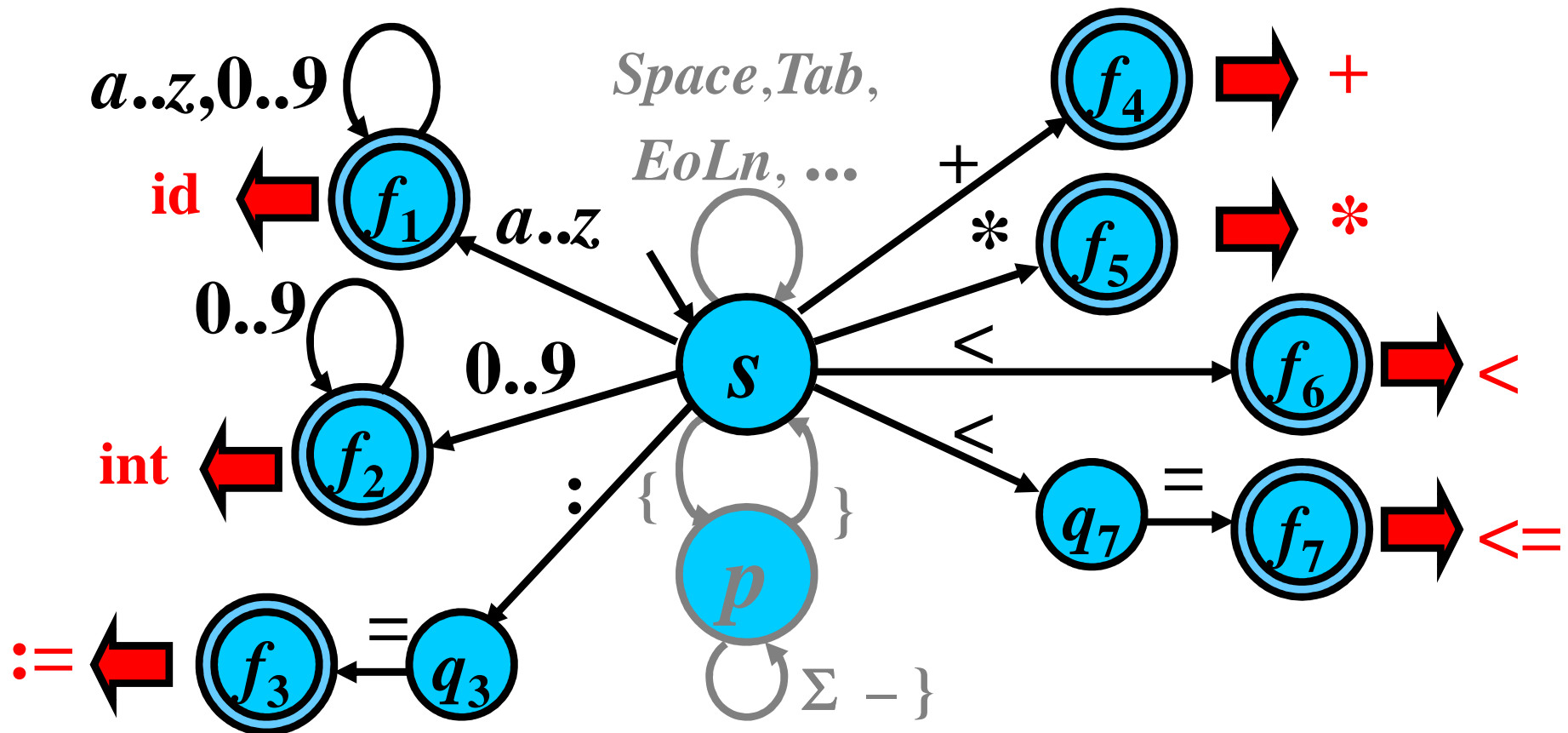
DKA přijímací lexémy: Příklad 1/2

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



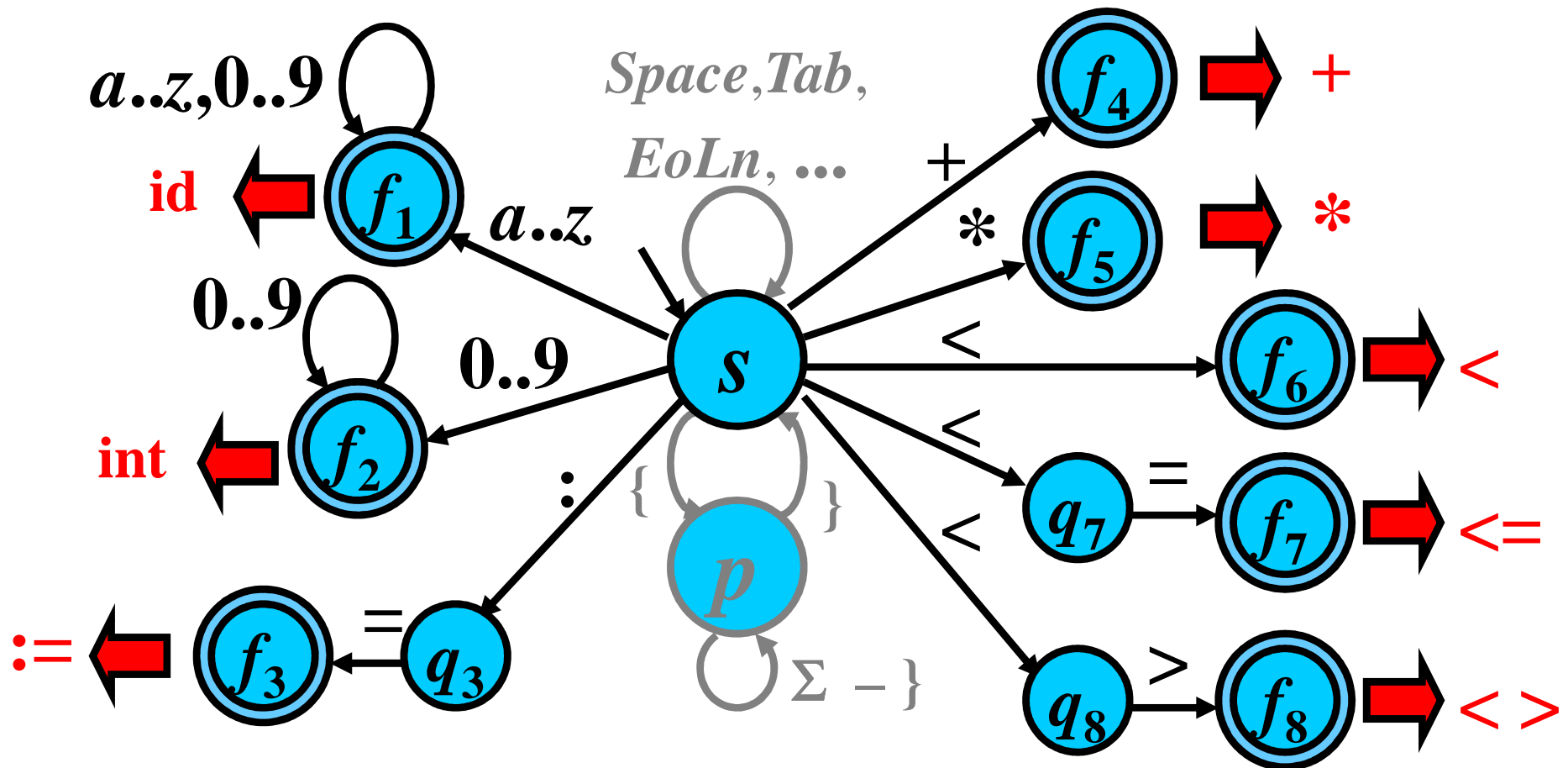
DKA přijímací lexémy: Příklad 1/2

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



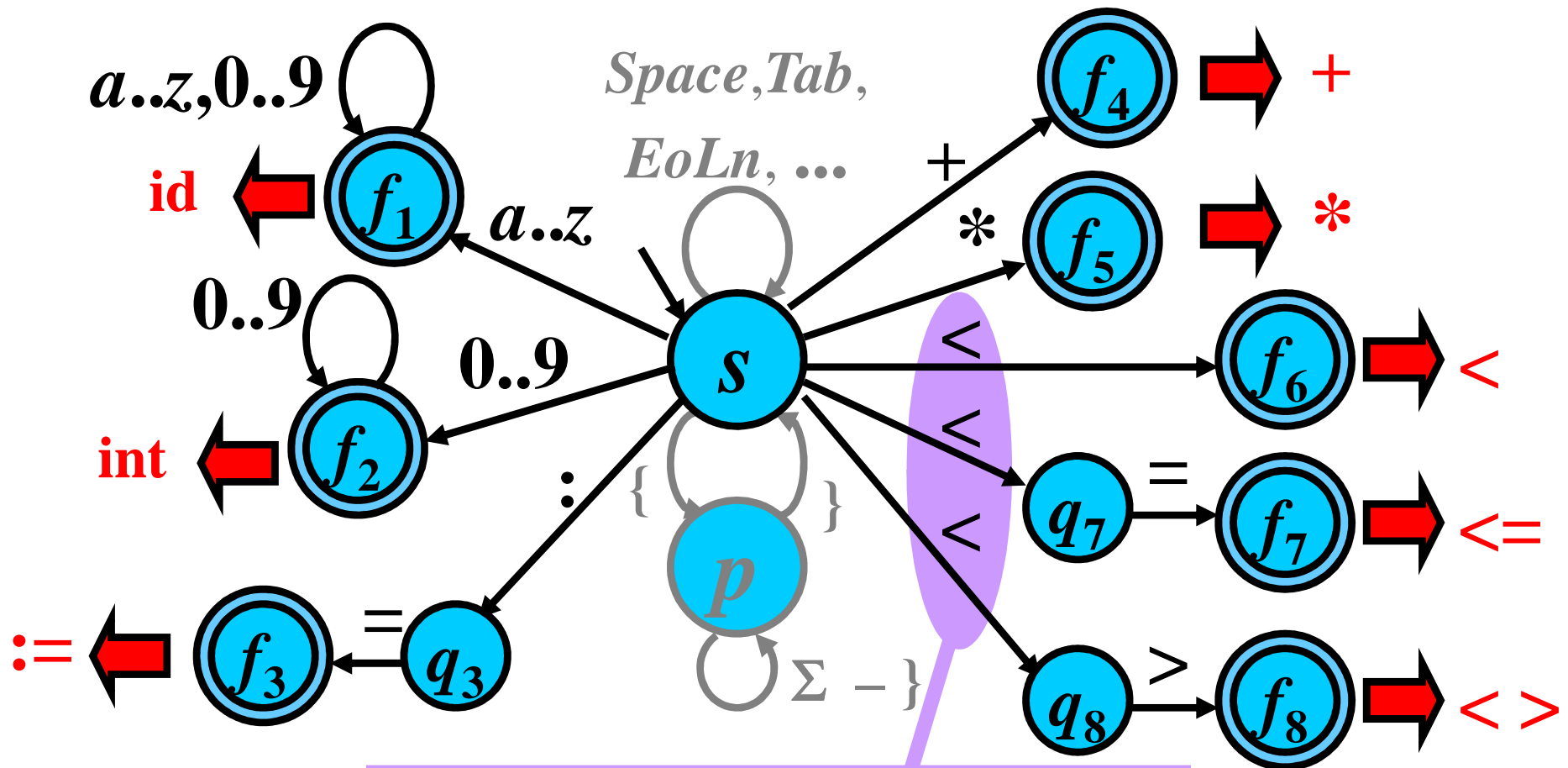
DKA přijímací lexémy: Příklad 1/2

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



DKA přijímací lexémy: Příklad 1/2

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

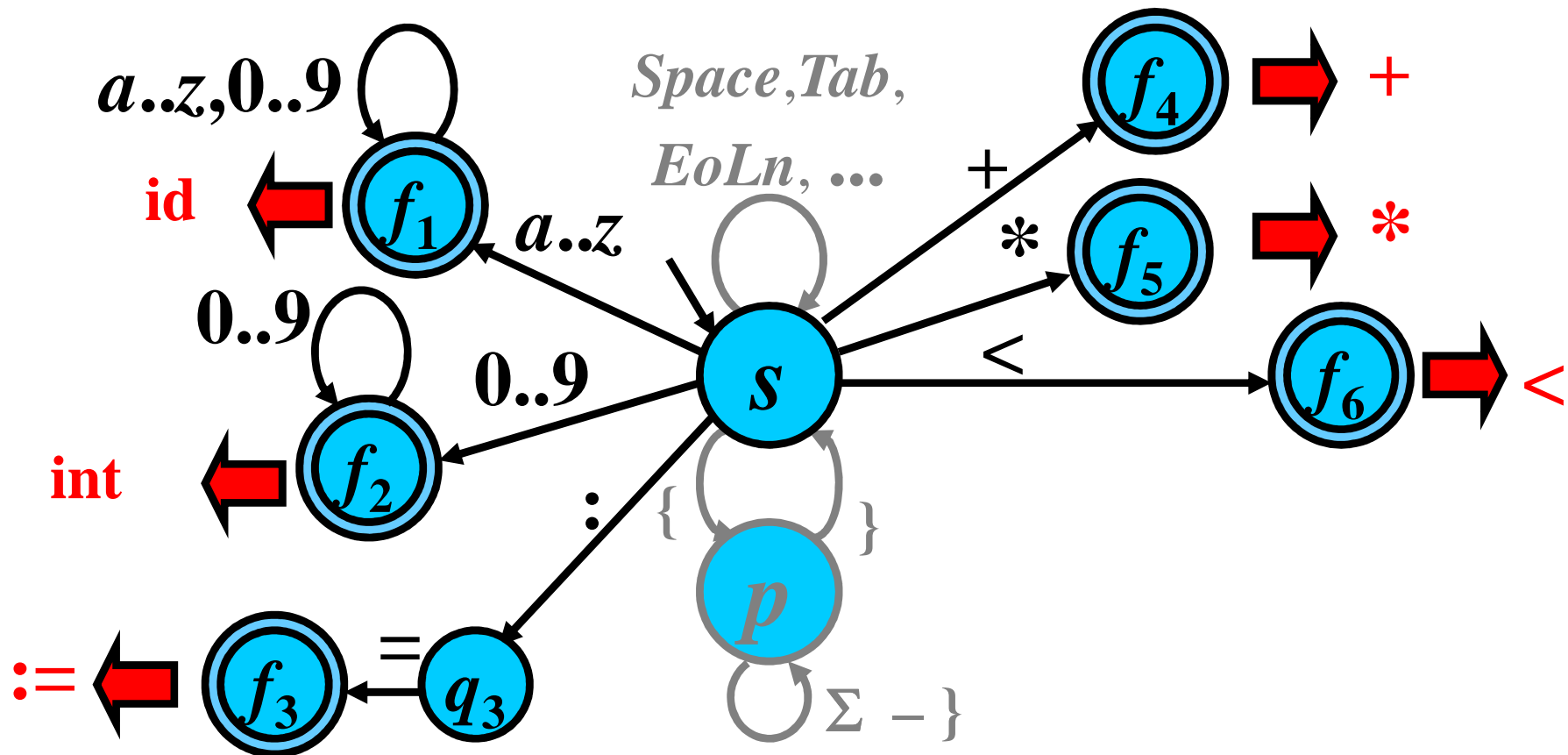


Převést tento KA na DKA.

DKA přijímací lexémy: Příklad 2/2

- Ekvivalentní DKA:

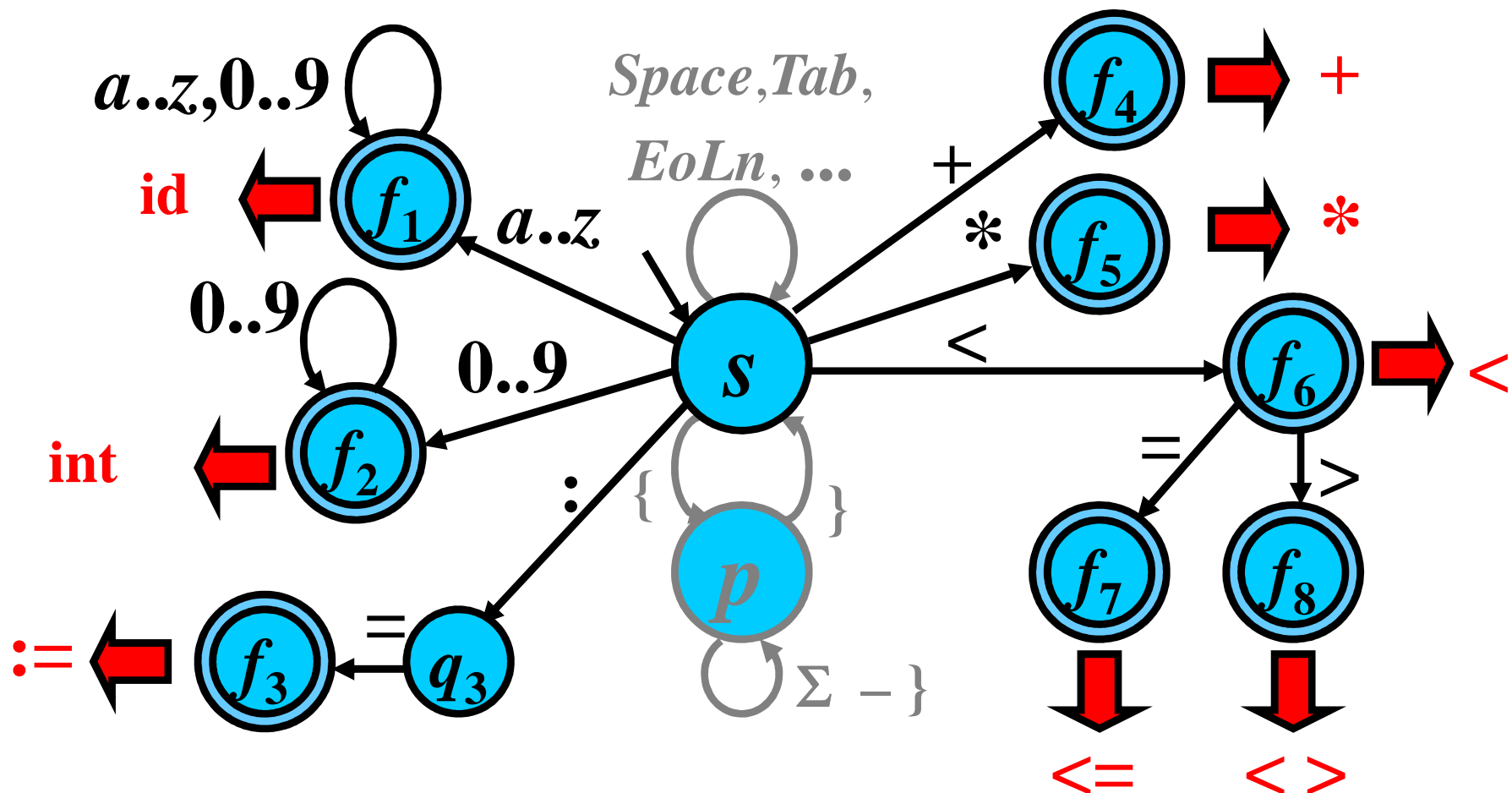
identifikátory, celá čísla, $:=$, $+$, $*$, $<$, $<=$, $< >$



DKA přijímací lexémy: Příklad 2/2

- Ekvivalentní DKA:

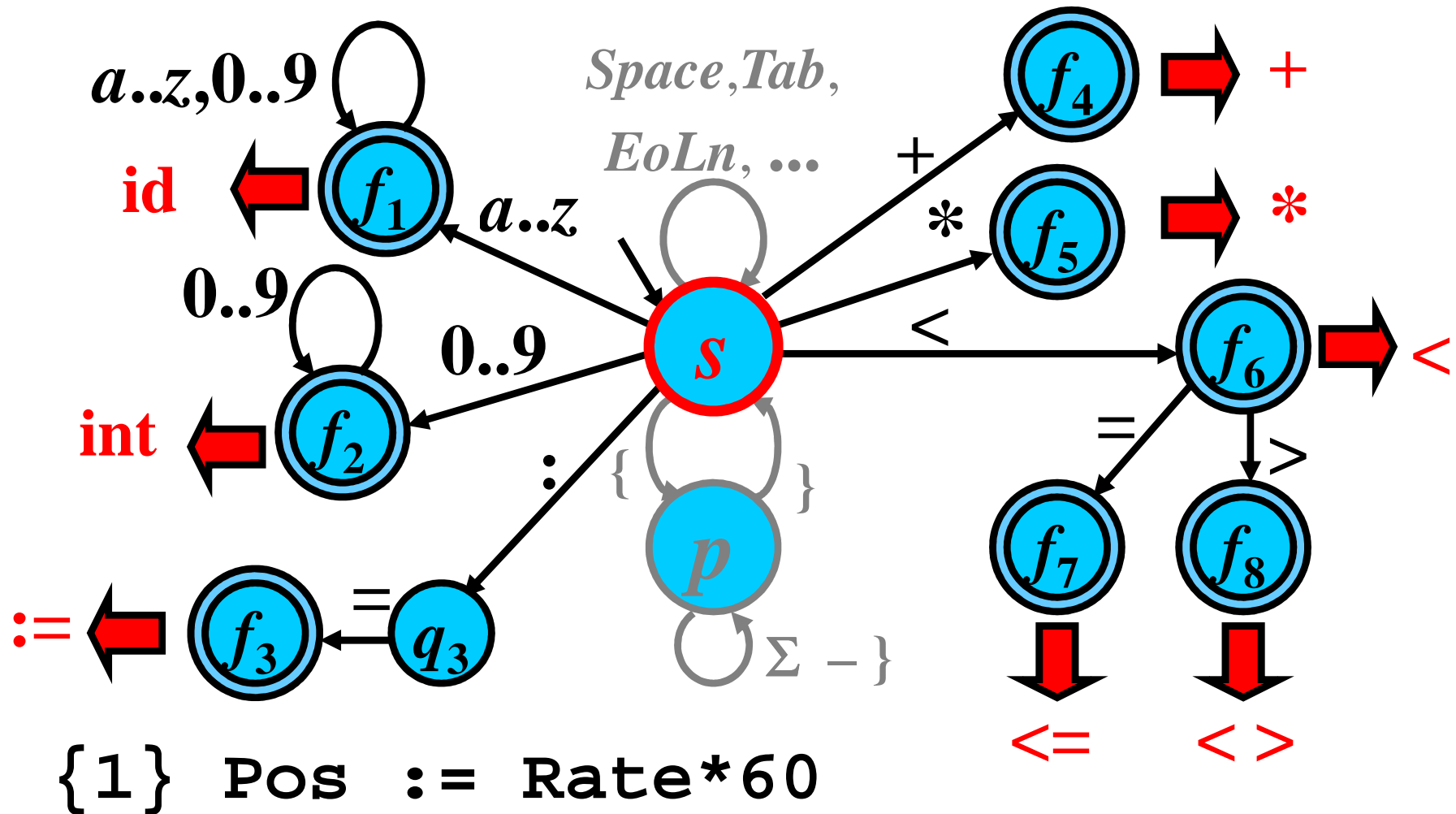
identifikátory, celá čísla, $:=$, $+$, $*$, $<$, $<=$, $<>$



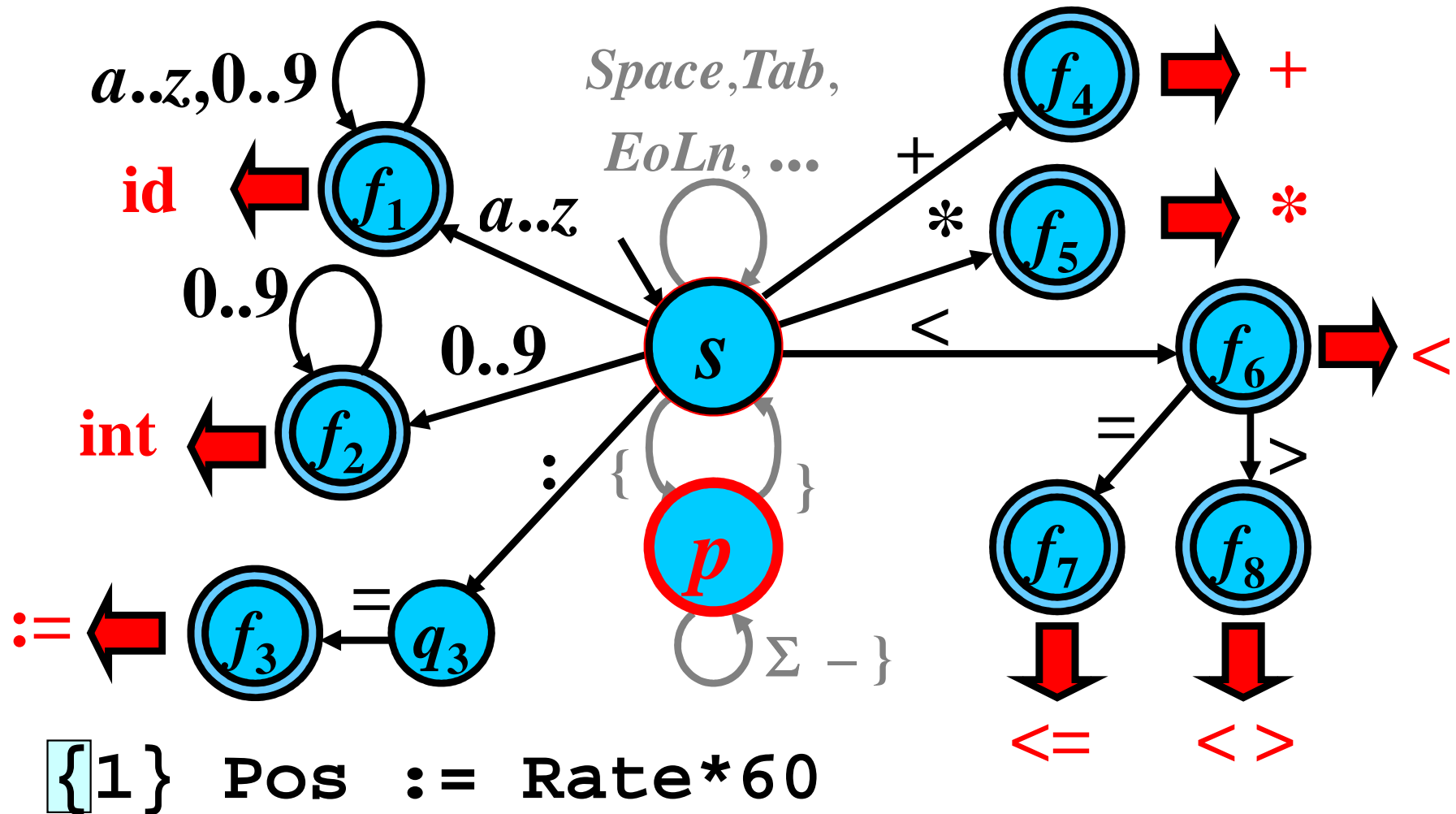
Algoritmus: Určení typu lexému

- **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 lexikální chybu (napiš zprávu atd.)

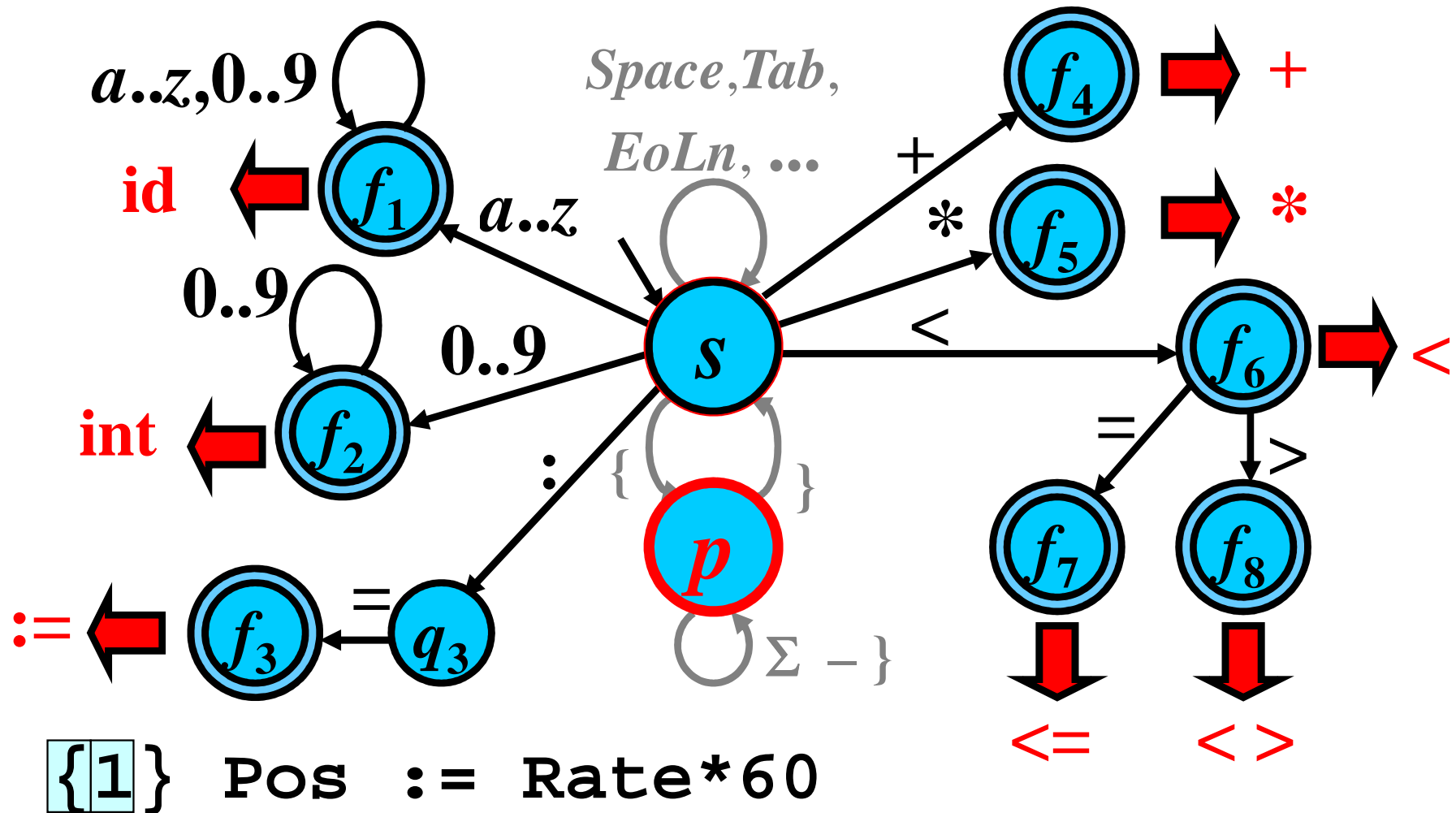
Určení typu lexému: Příklad



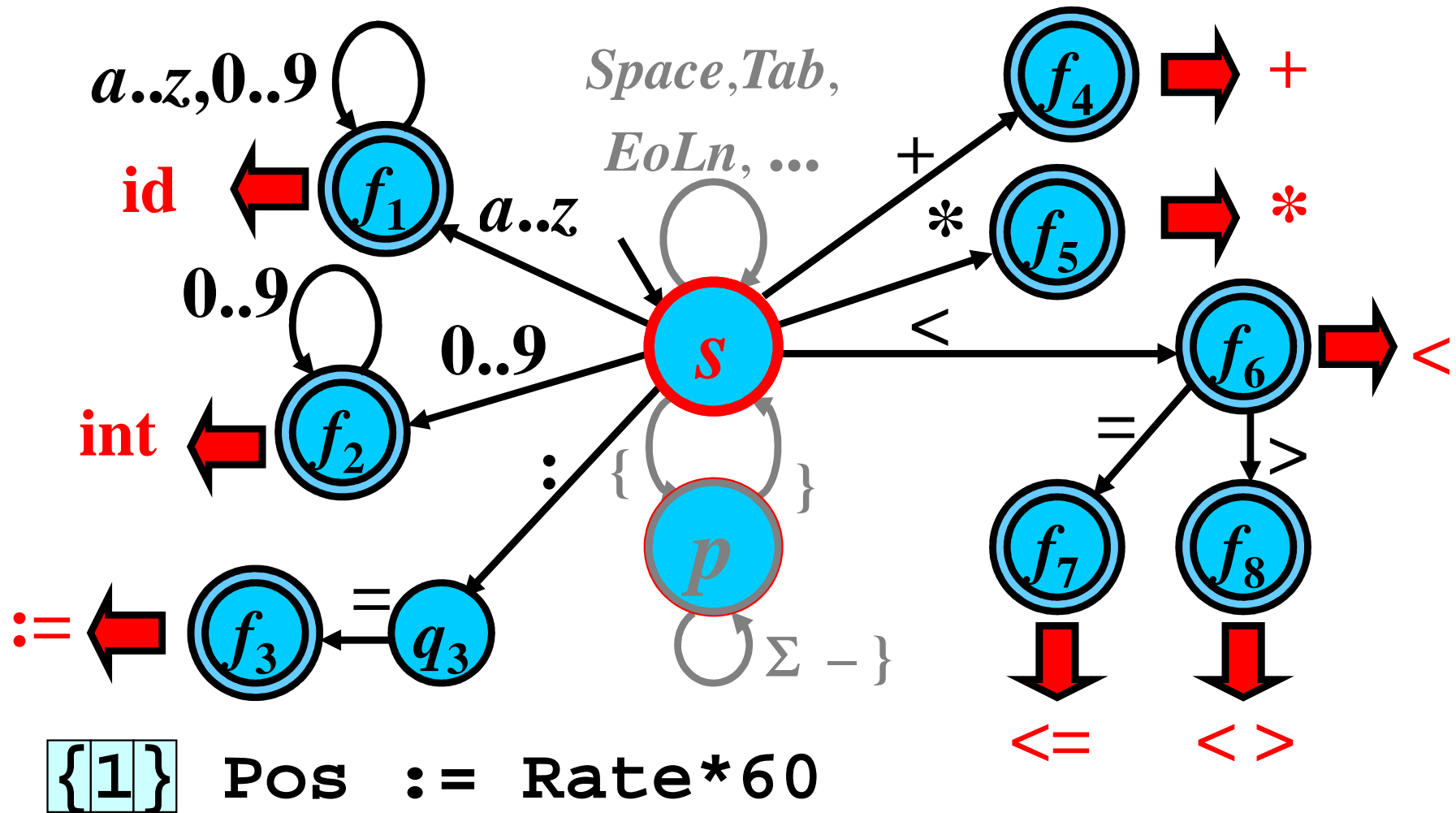
Určení typu lexému: Příklad



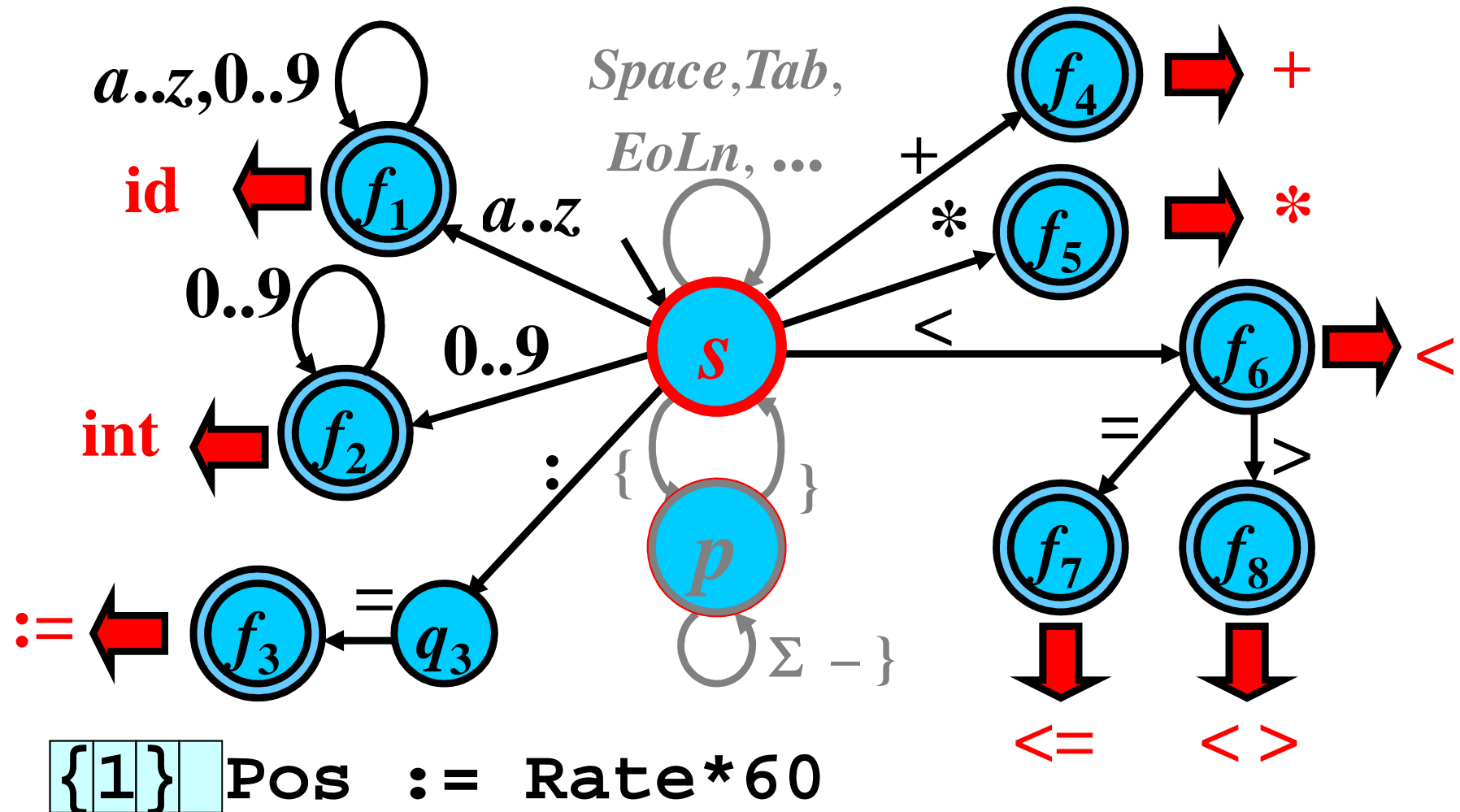
Určení typu lexému: Příklad



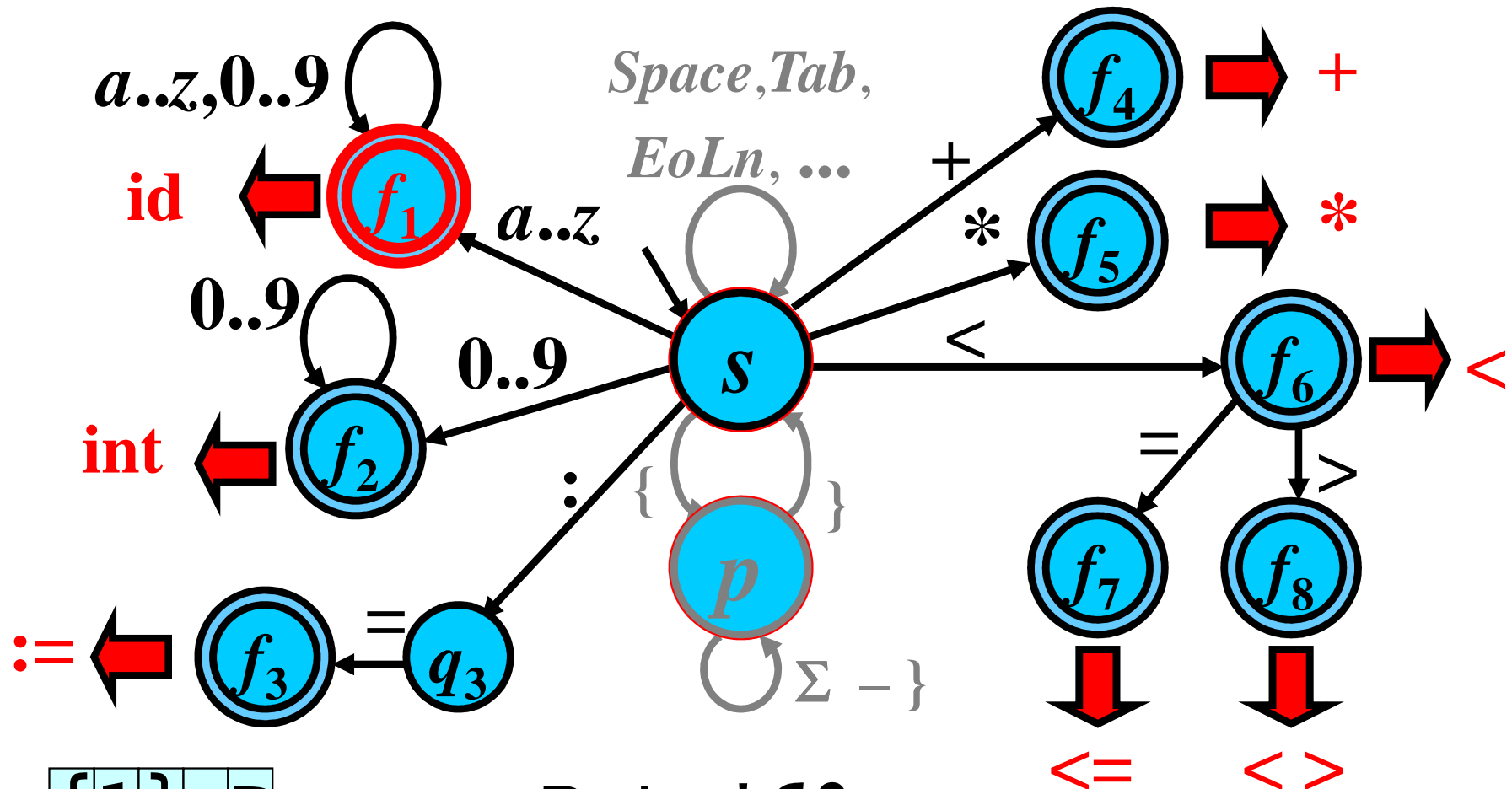
Určení typu lexému: Příklad



Určení typu lexému: Příklad

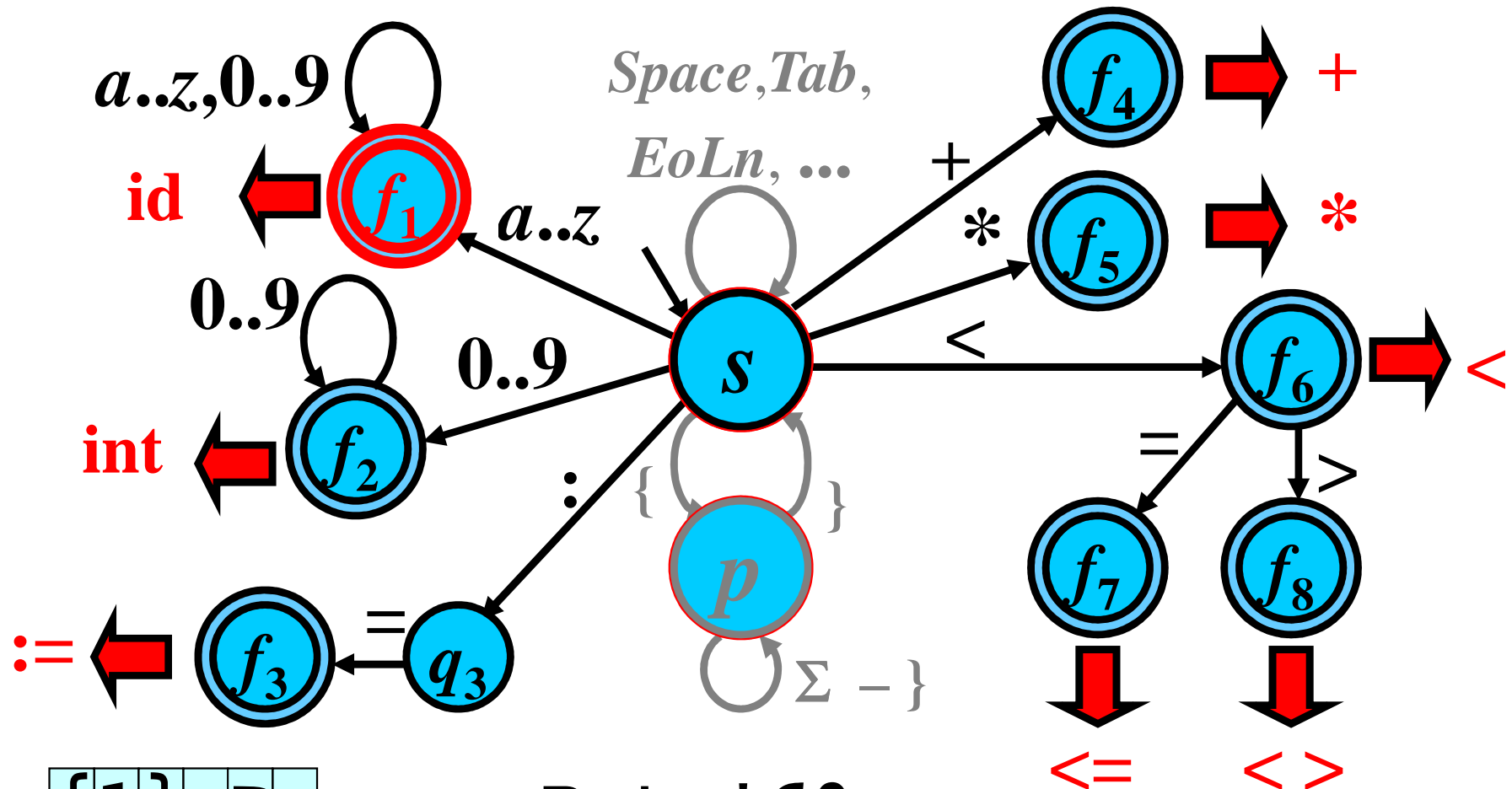


Určení typu lexému: Příklad



`{1} Pos := Rate*60`

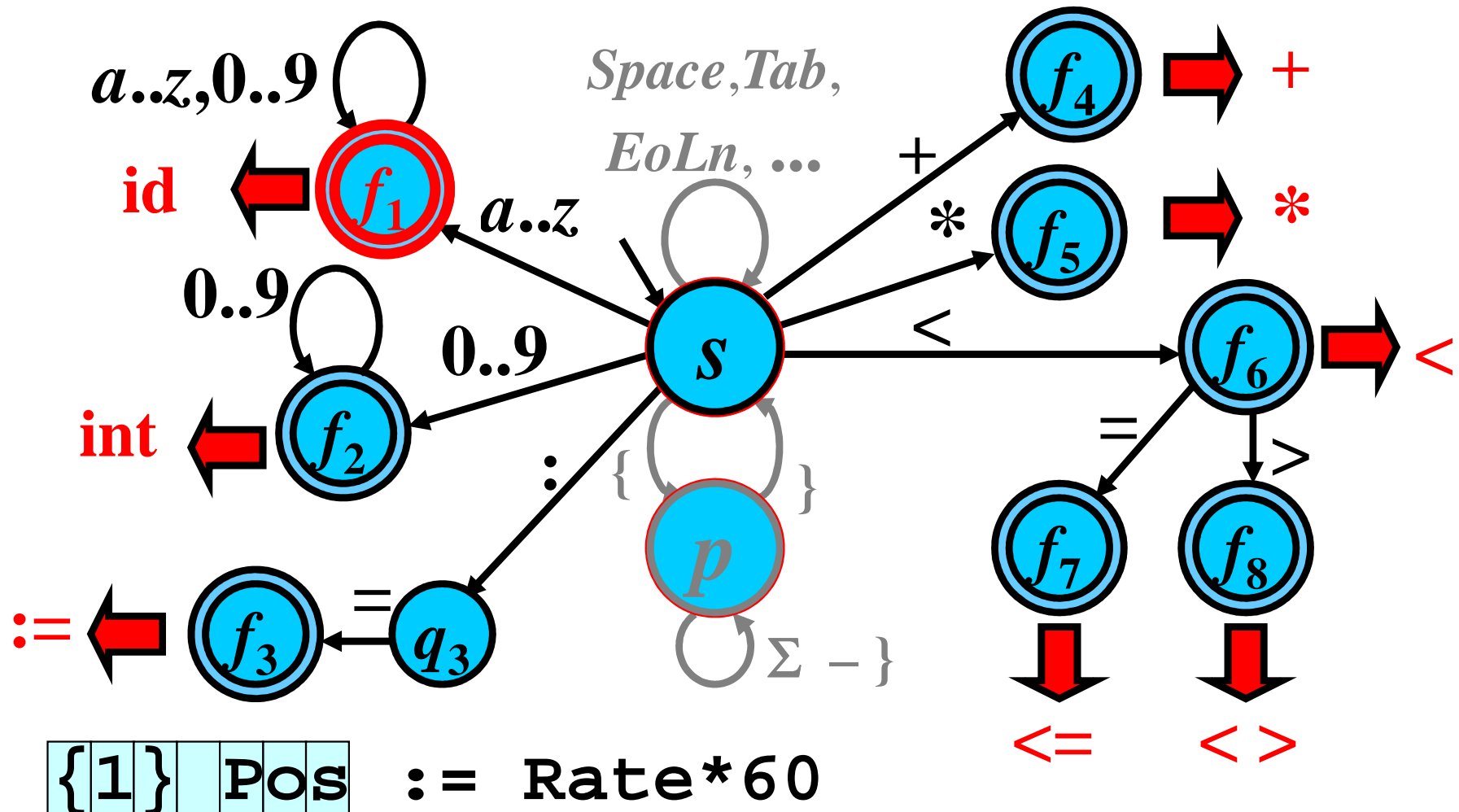
Určení typu lexému: Příklad



`{1} Pos := Rate*60`

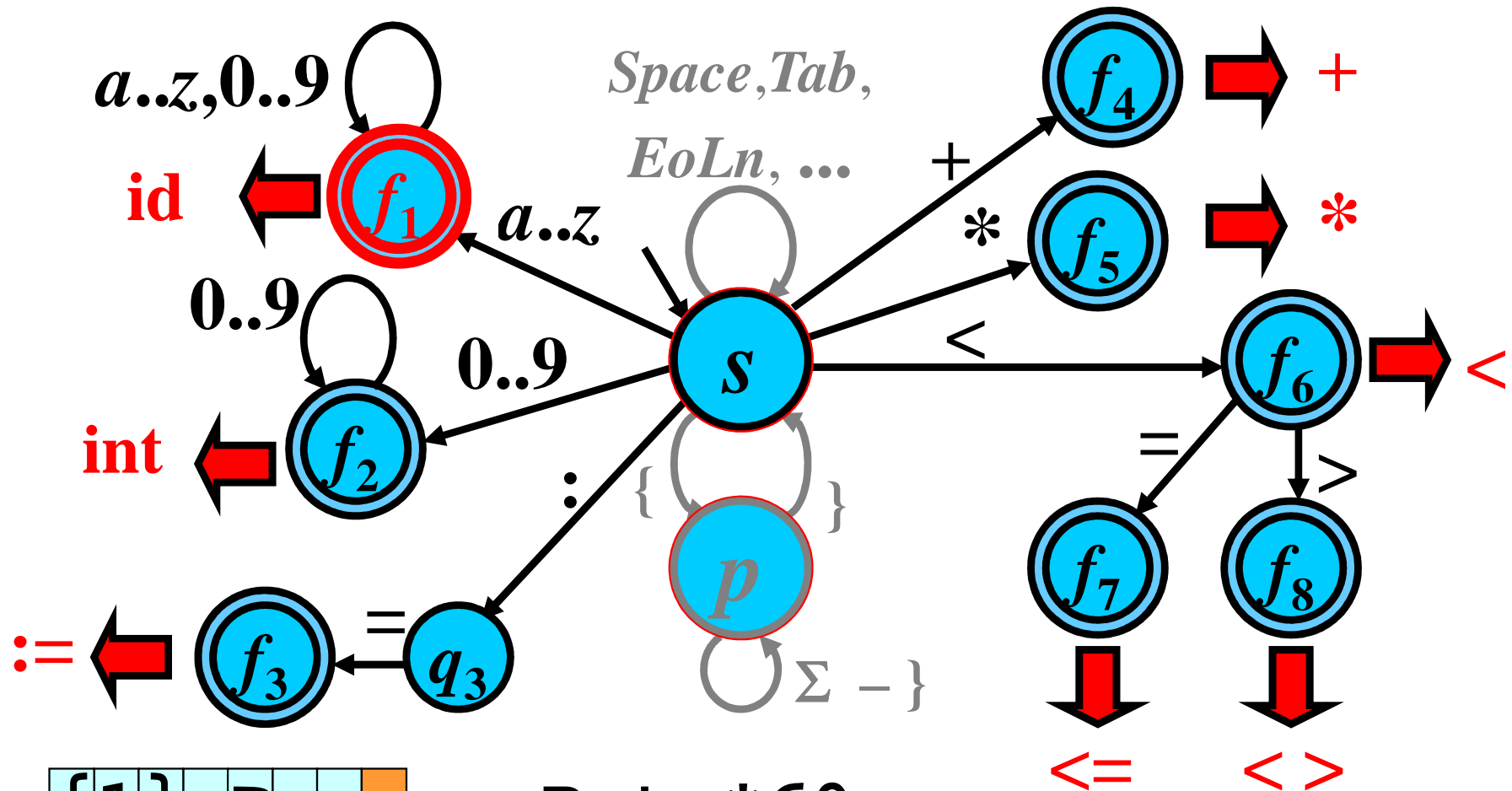
Pos

Určení typu lexému: Příklad



Pos

Určení typu lexému: Příklad

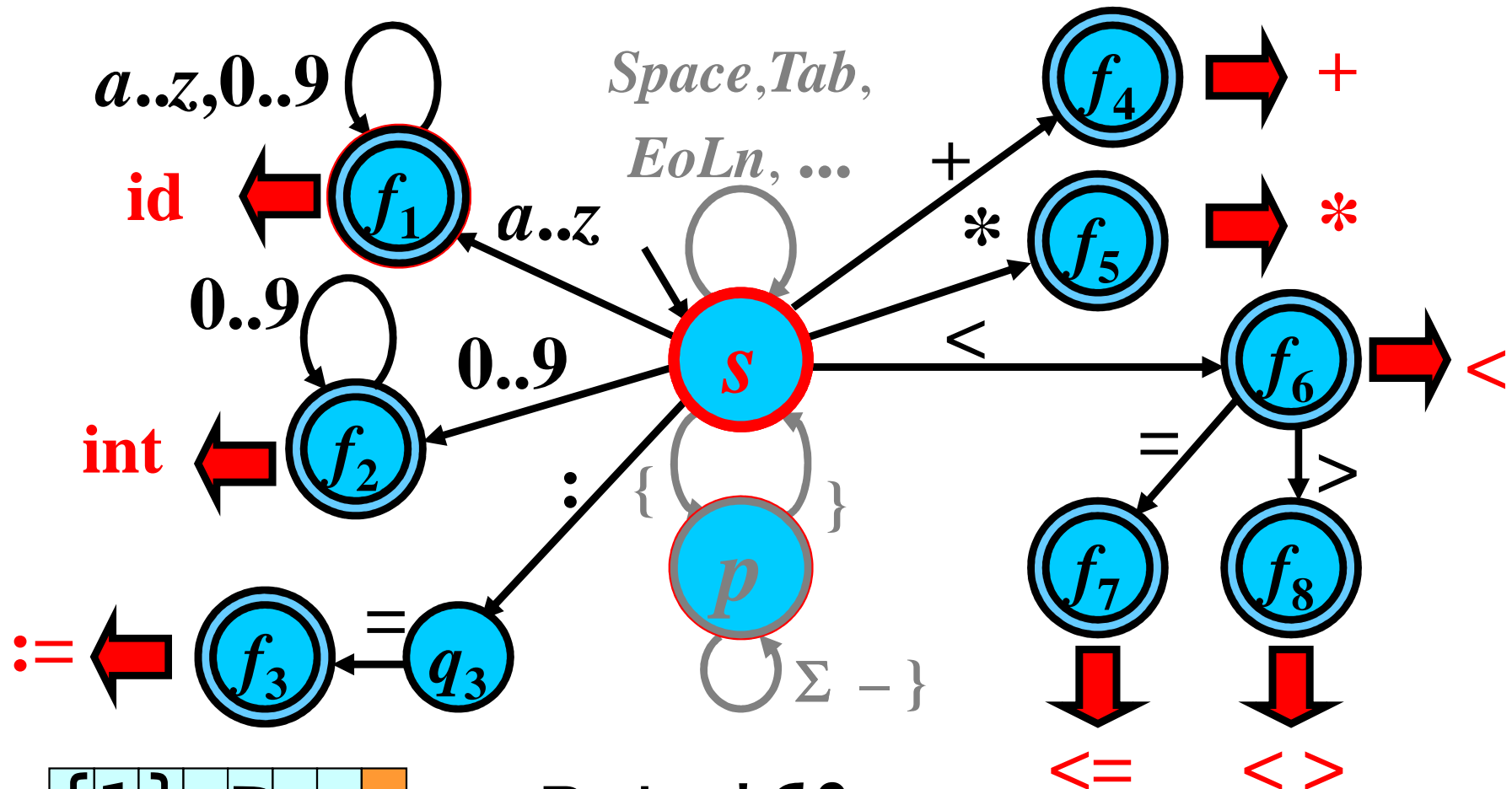


`{1} Pos := Rate*60`

id
Pos

Neexistuje další
konfig.!

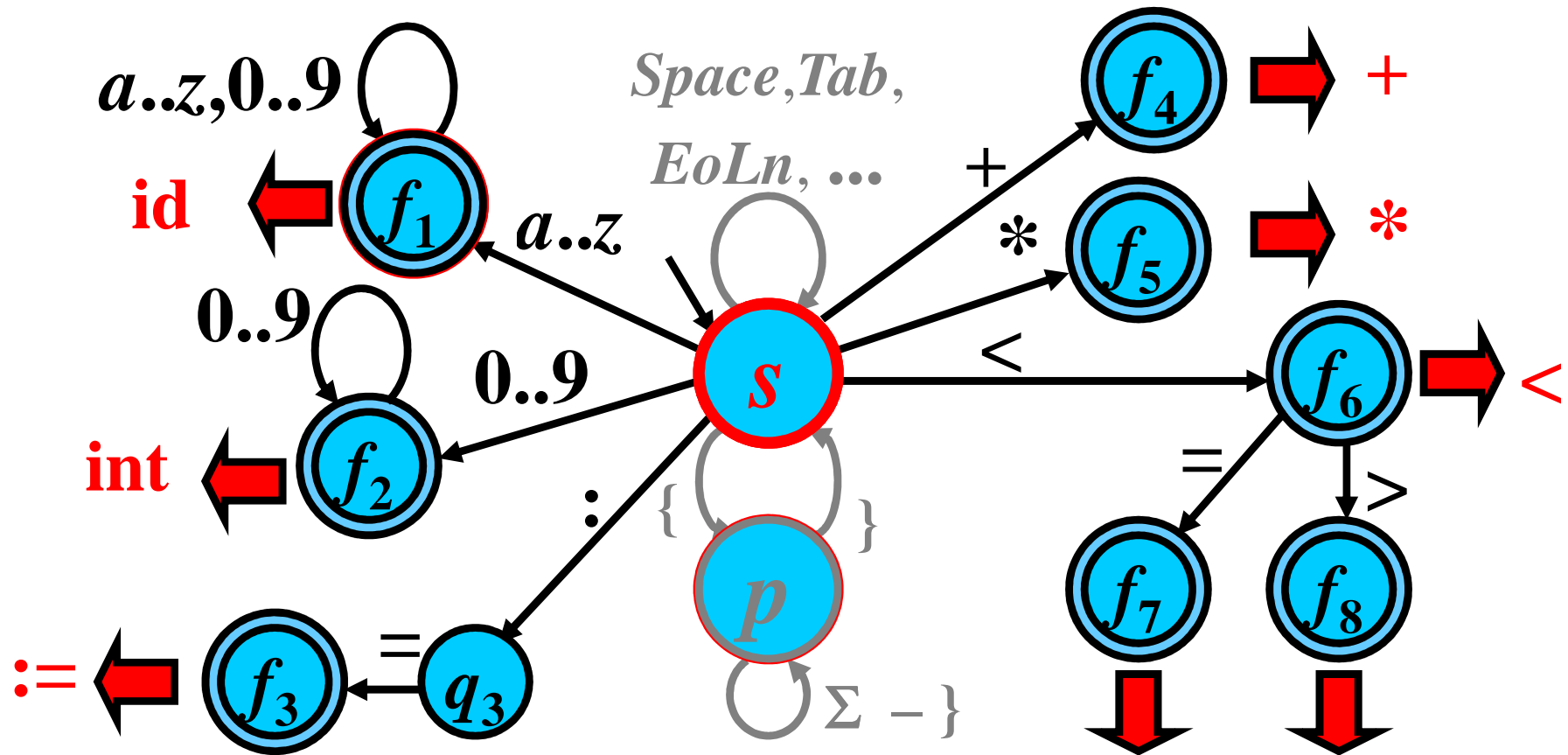
Určení typu lexému: Příklad



`{1} Pos := Rate*60`

`id`
`Pos`

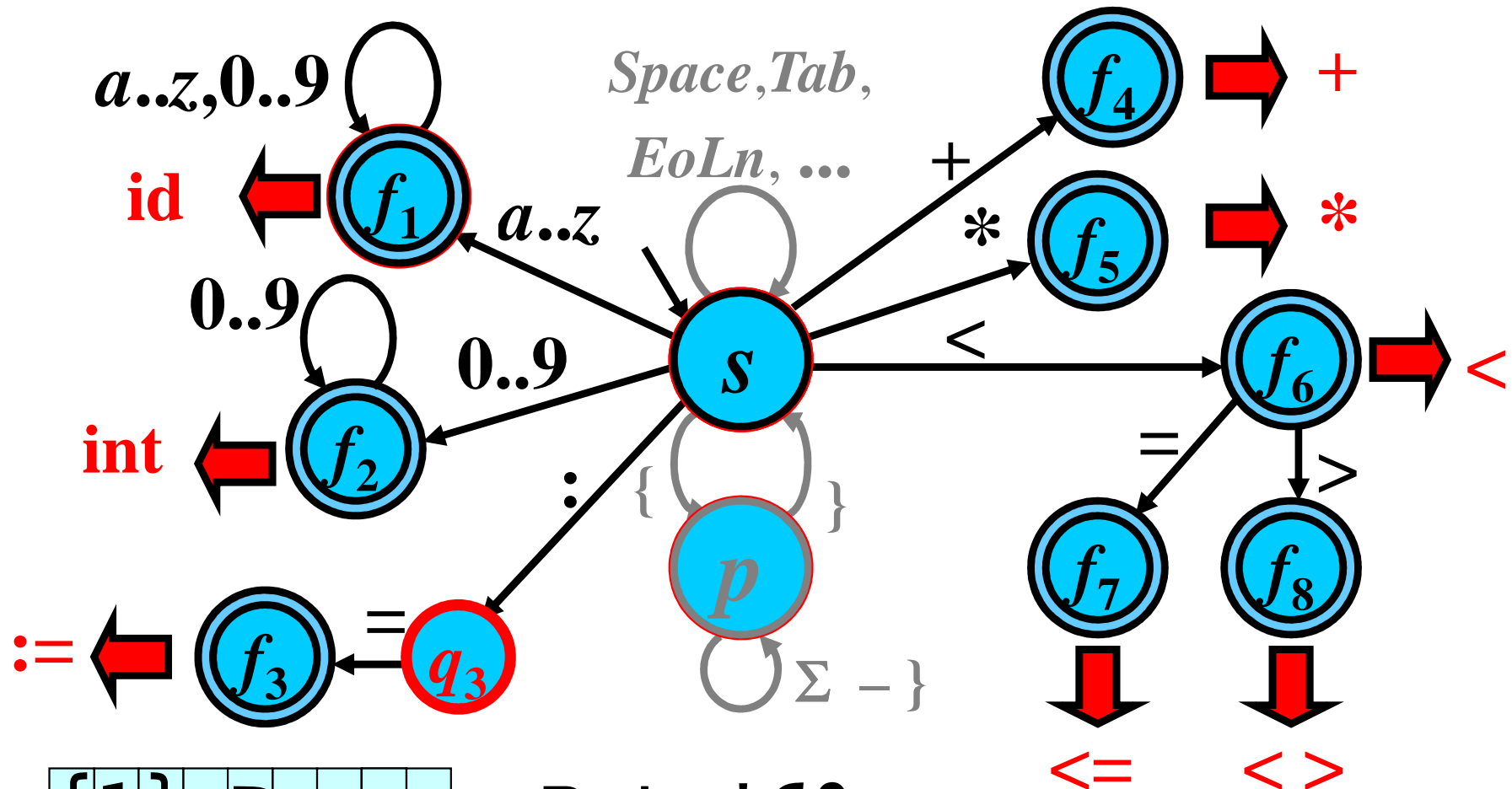
Určení typu lexému: Příklad



`{1} Pos := Rate*60`

id
Pos

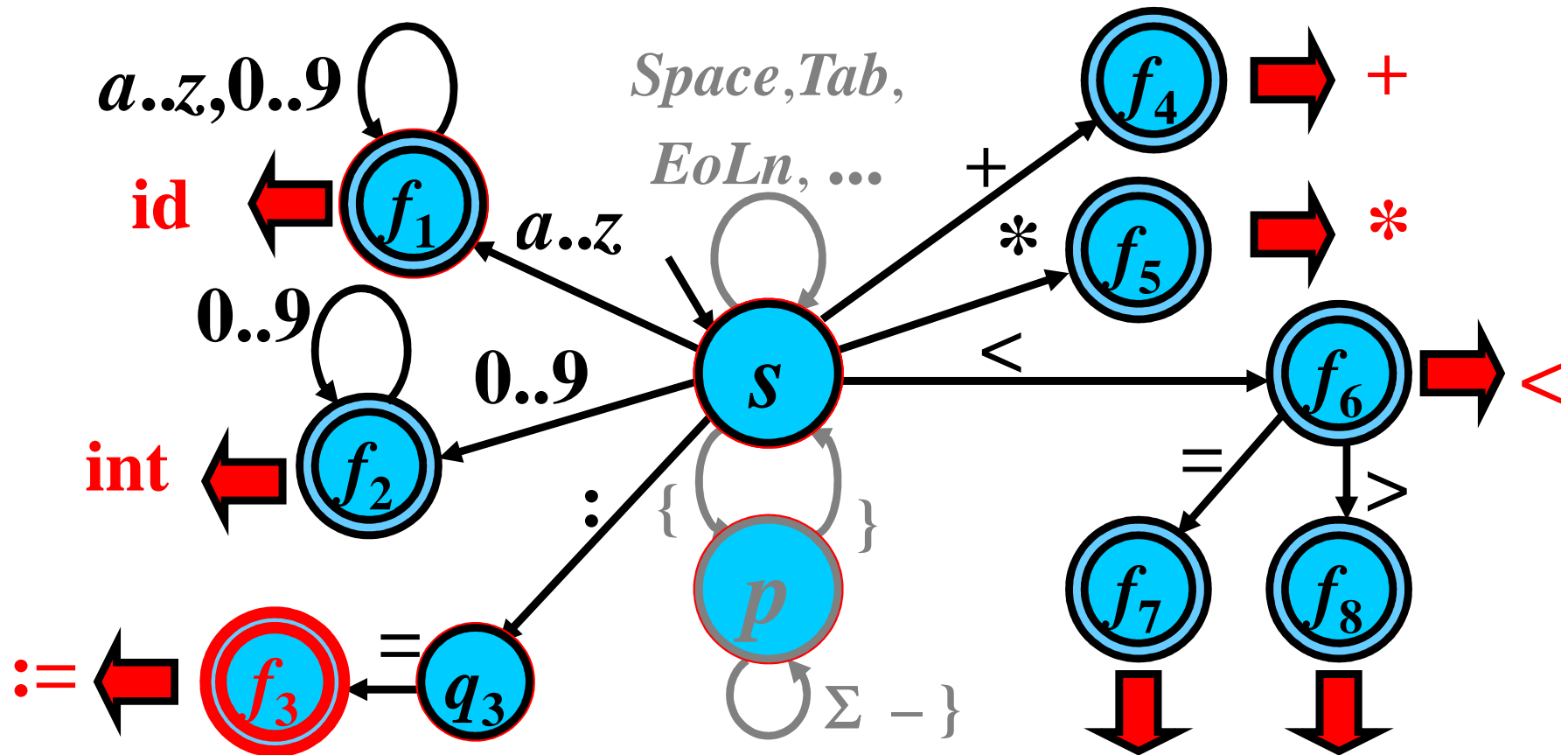
Určení typu lexému: Příklad



`{1} Pos := Rate*60`

`id`
Pos :

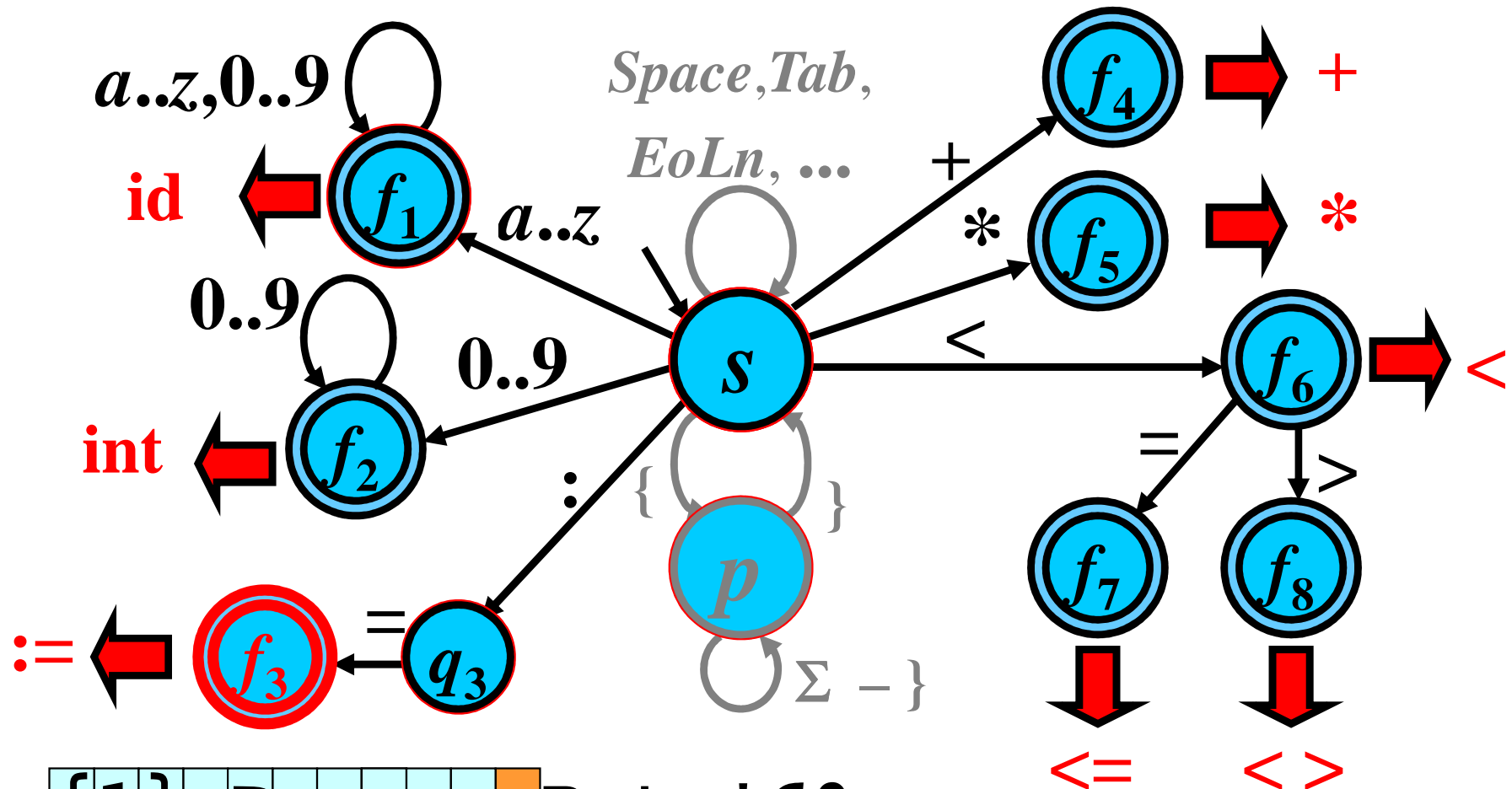
Určení typu lexému: Příklad



{1} Pos := Rate*60

id
Pos :=

Určení typu lexému: Příklad



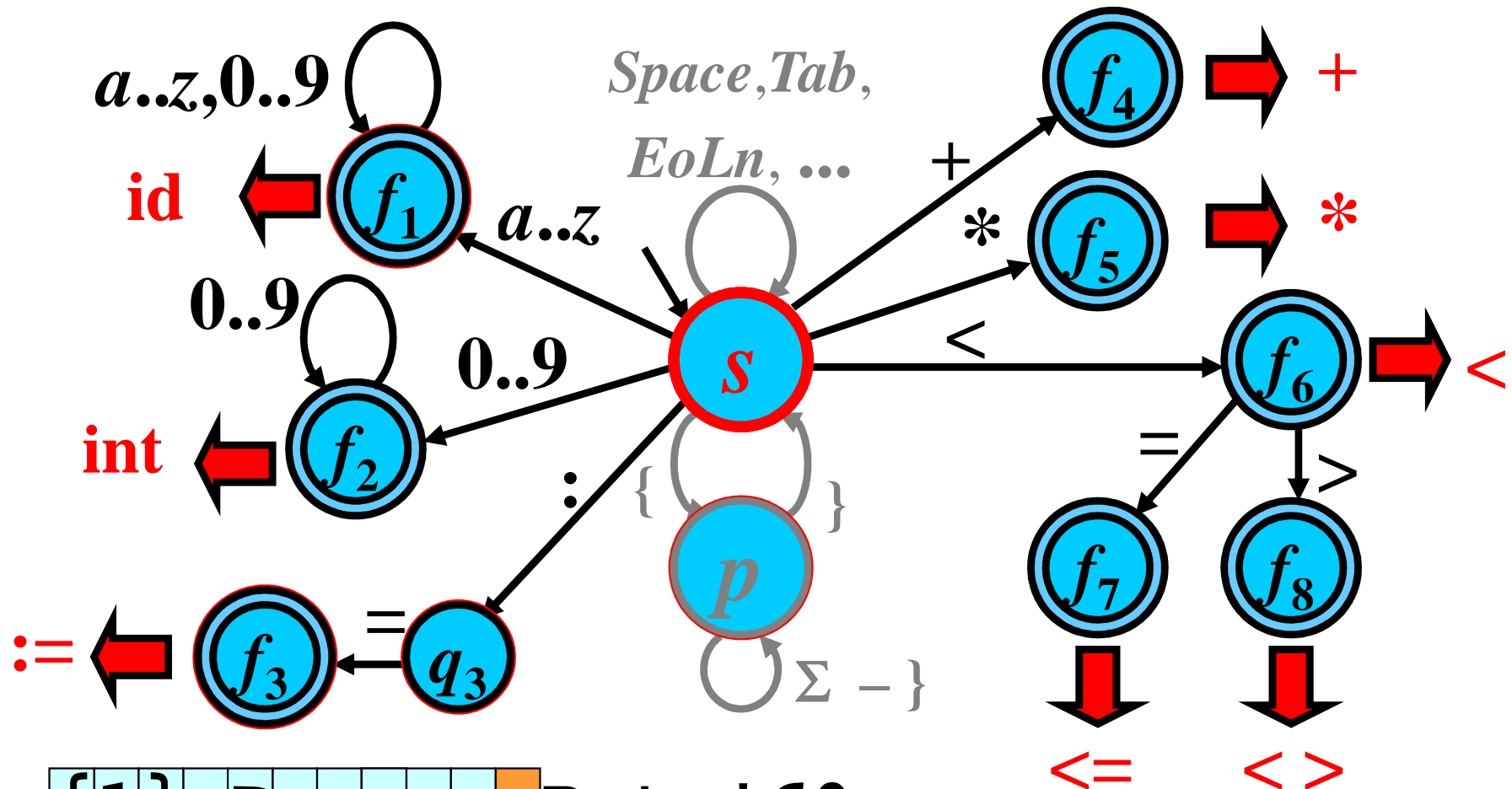
{1} Pos := Rate * 60

id
Pos

::=

Neexistuje další
konfig.!

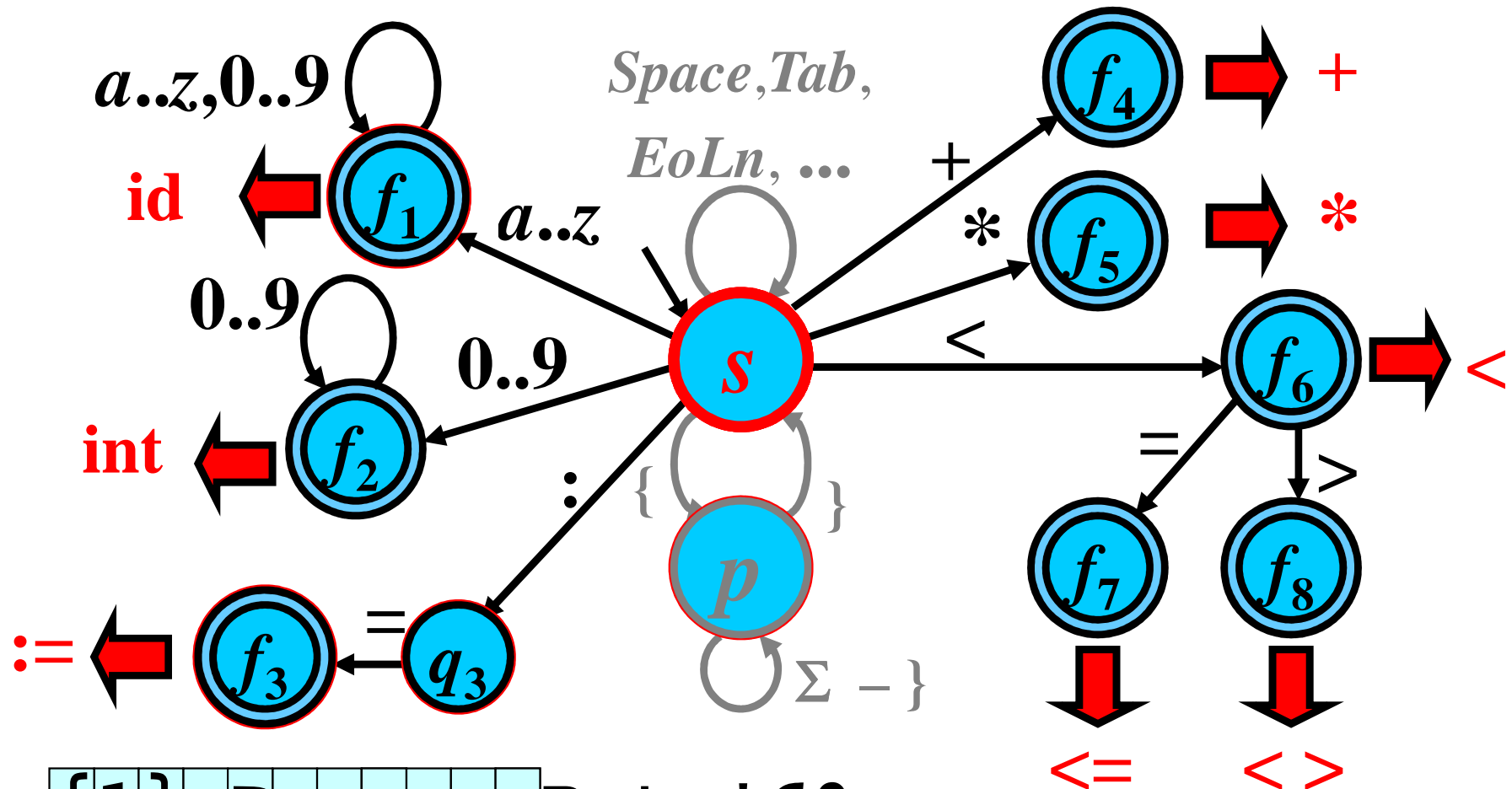
Určení typu lexému: Příklad



{1} Pos := Rate * 60

id	:=
Pos	

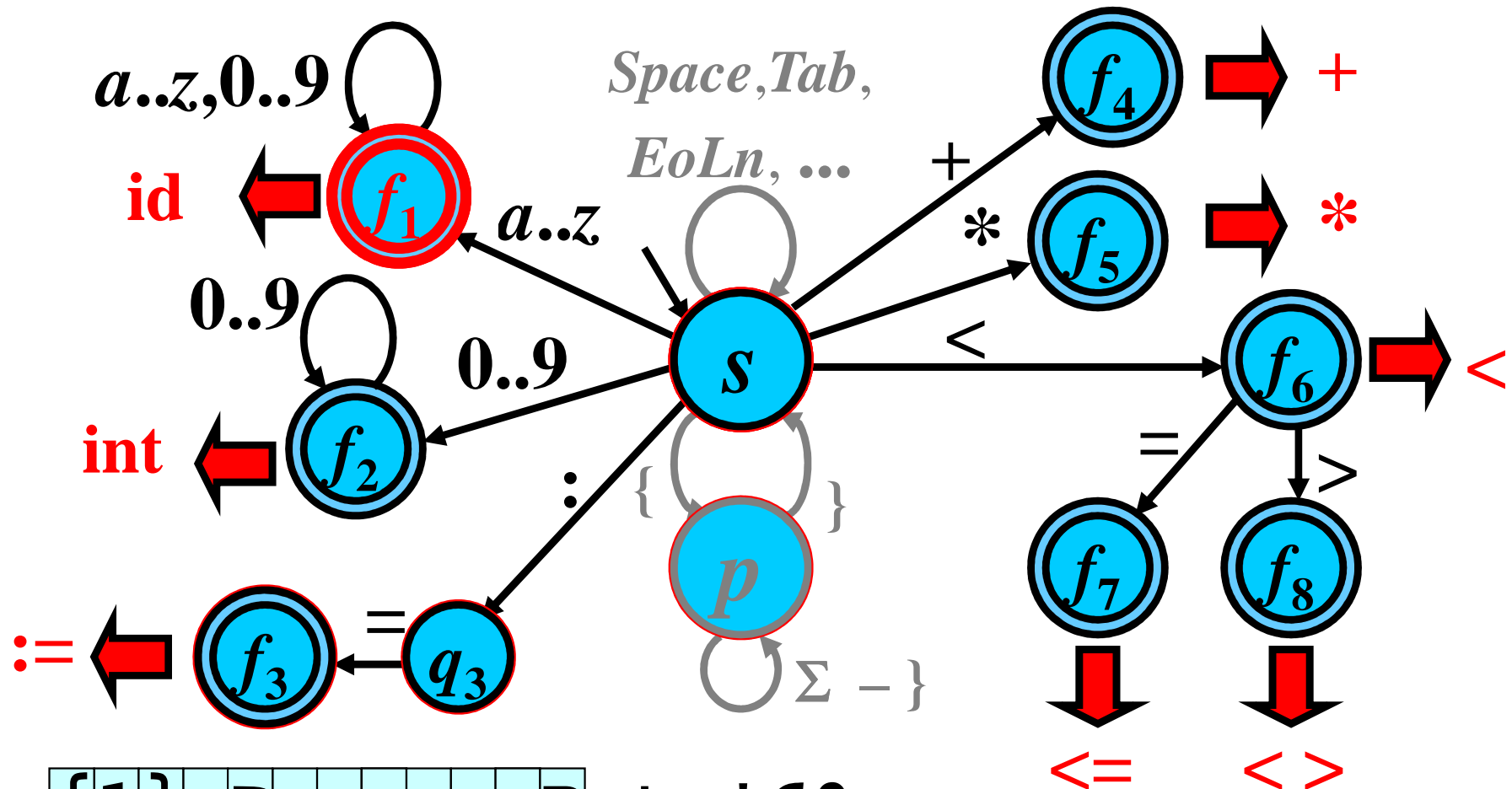
Určení typu lexému: Příklad



{1} Pos := Rate*60

id	:=
Pos	

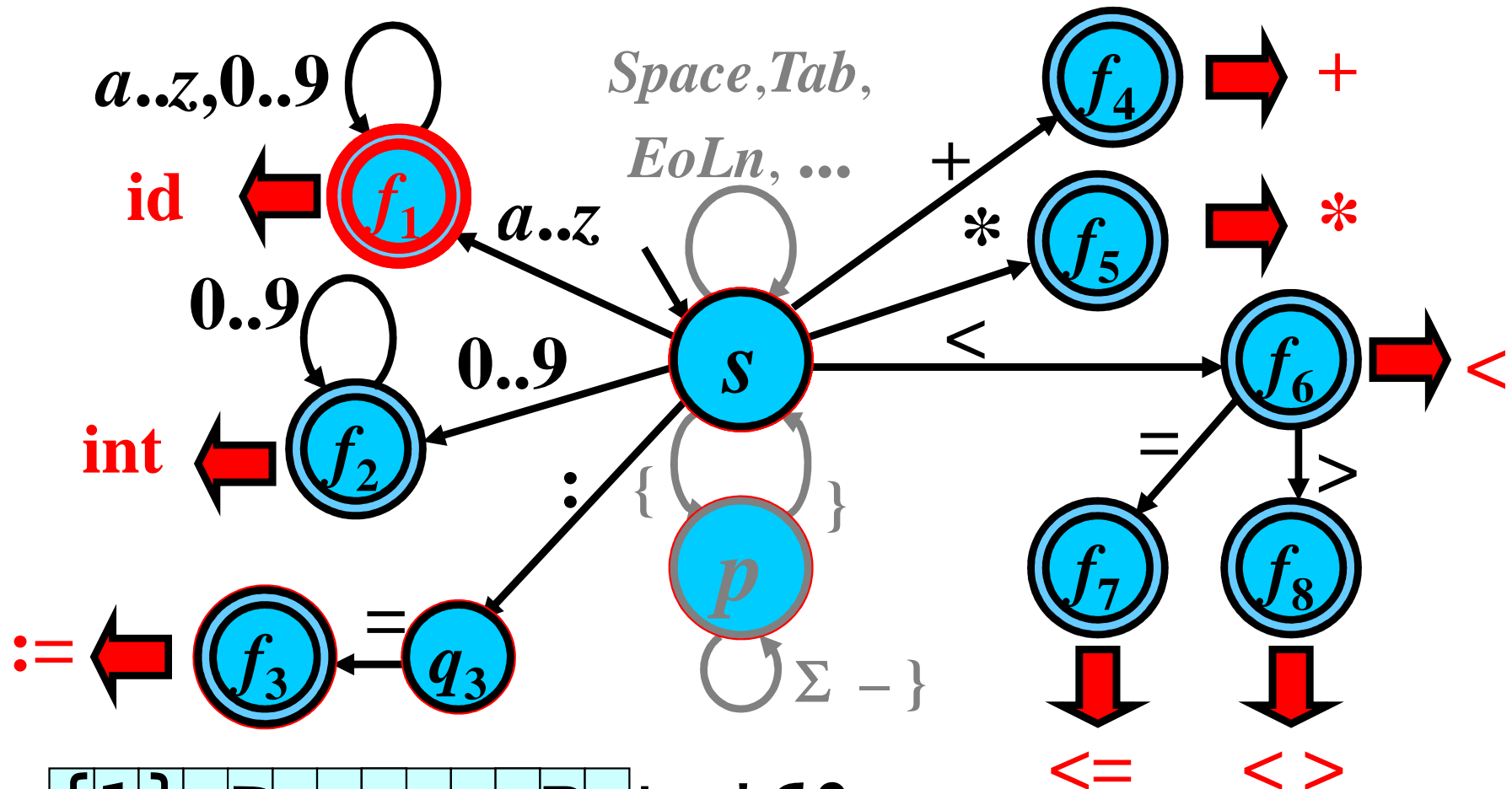
Určení typu lexému: Příklad



{1} Pos := Rate*60

id
Pos :=
R

Určení typu lexému: Příklad



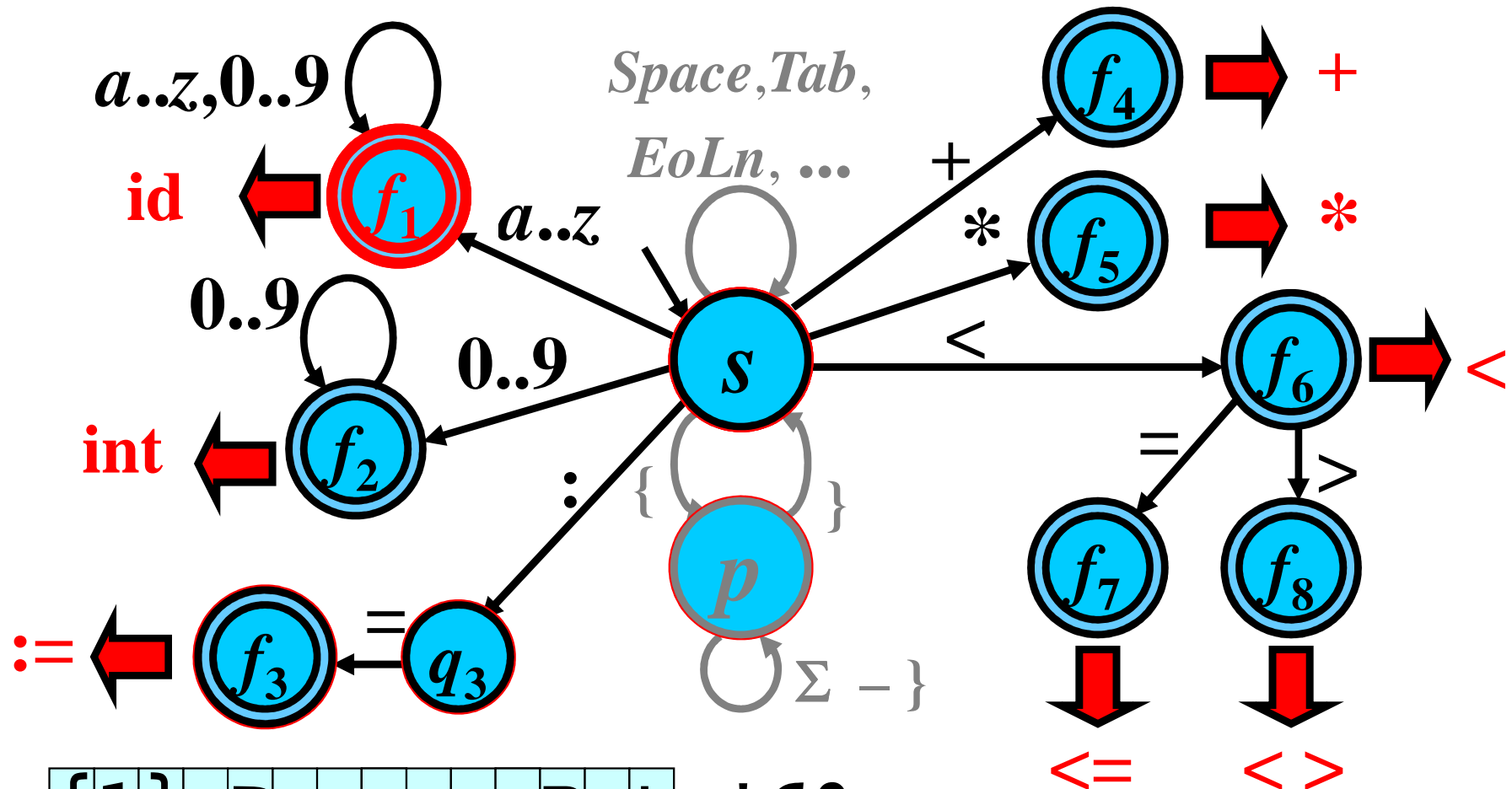
{1} Pos := Rate*60

id
Pos

:=

Ra

Určení typu lexému: Příklad



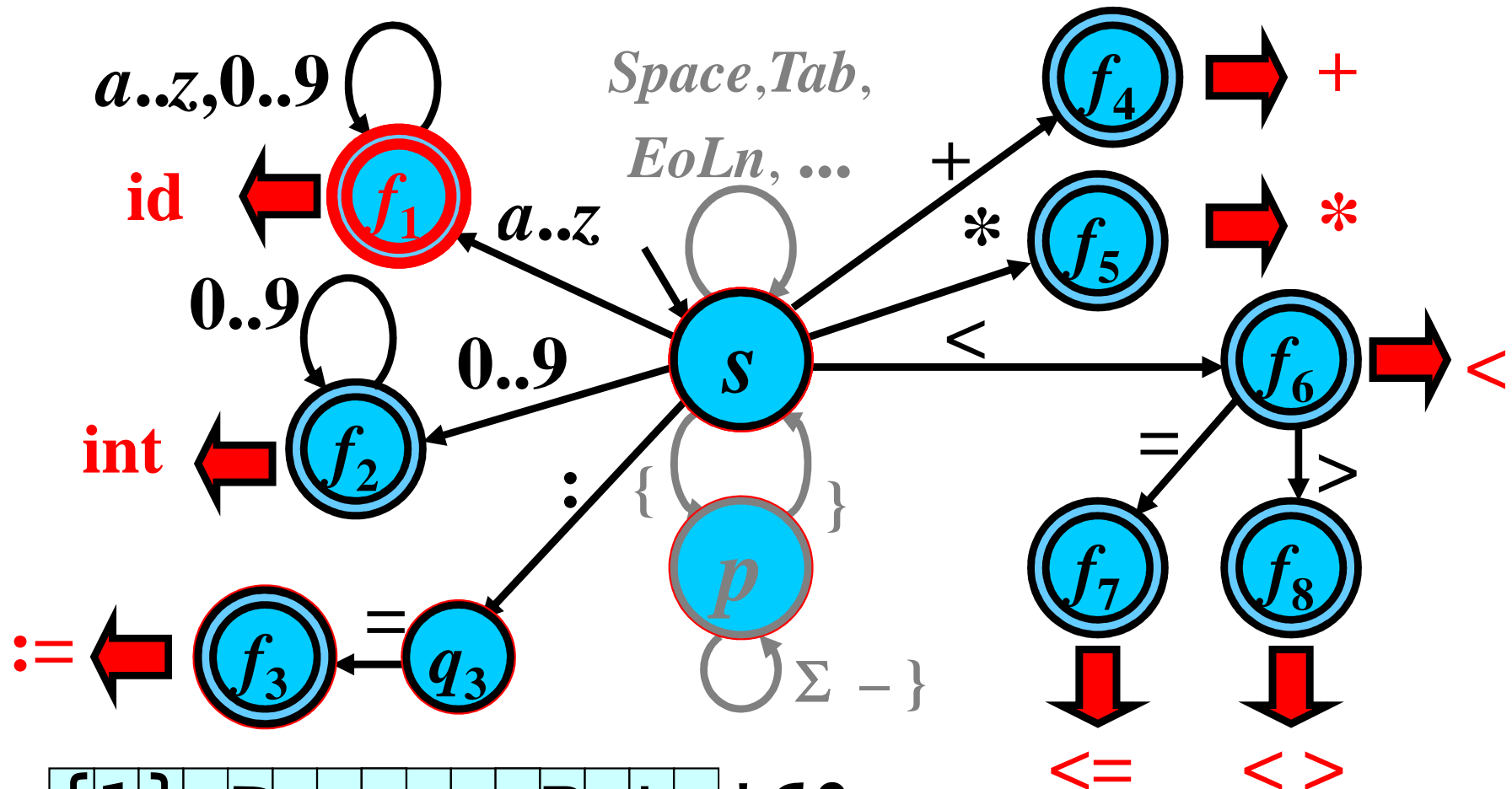
{1} Pos := Rate*60

id
Pos

:=

Rat

Určení typu lexému: Příklad



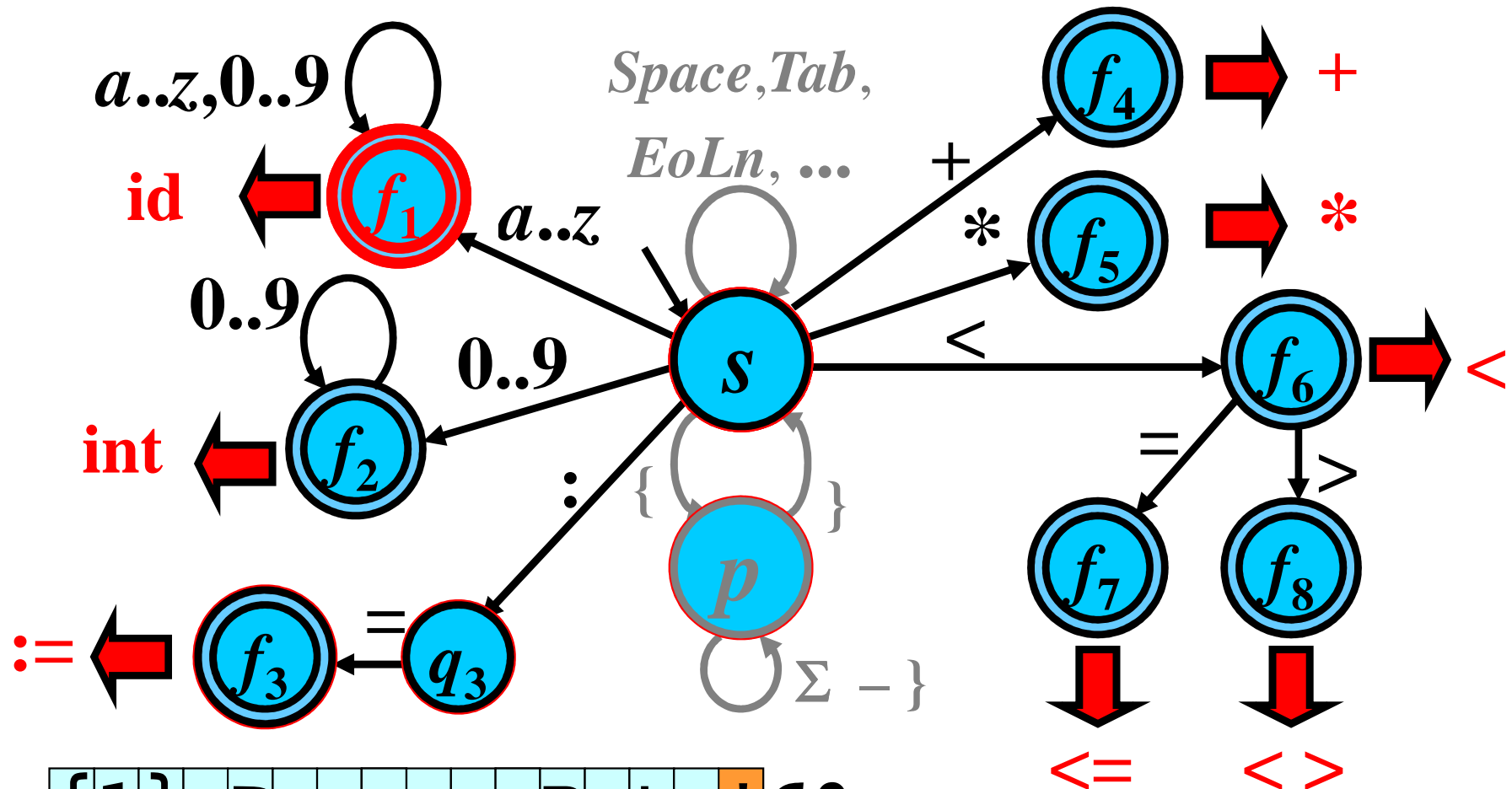
{1} Pos := Rate*60

id
Pos

::=

Rate

Určení typu lexému: Příklad



{1} Pos := Rate*60

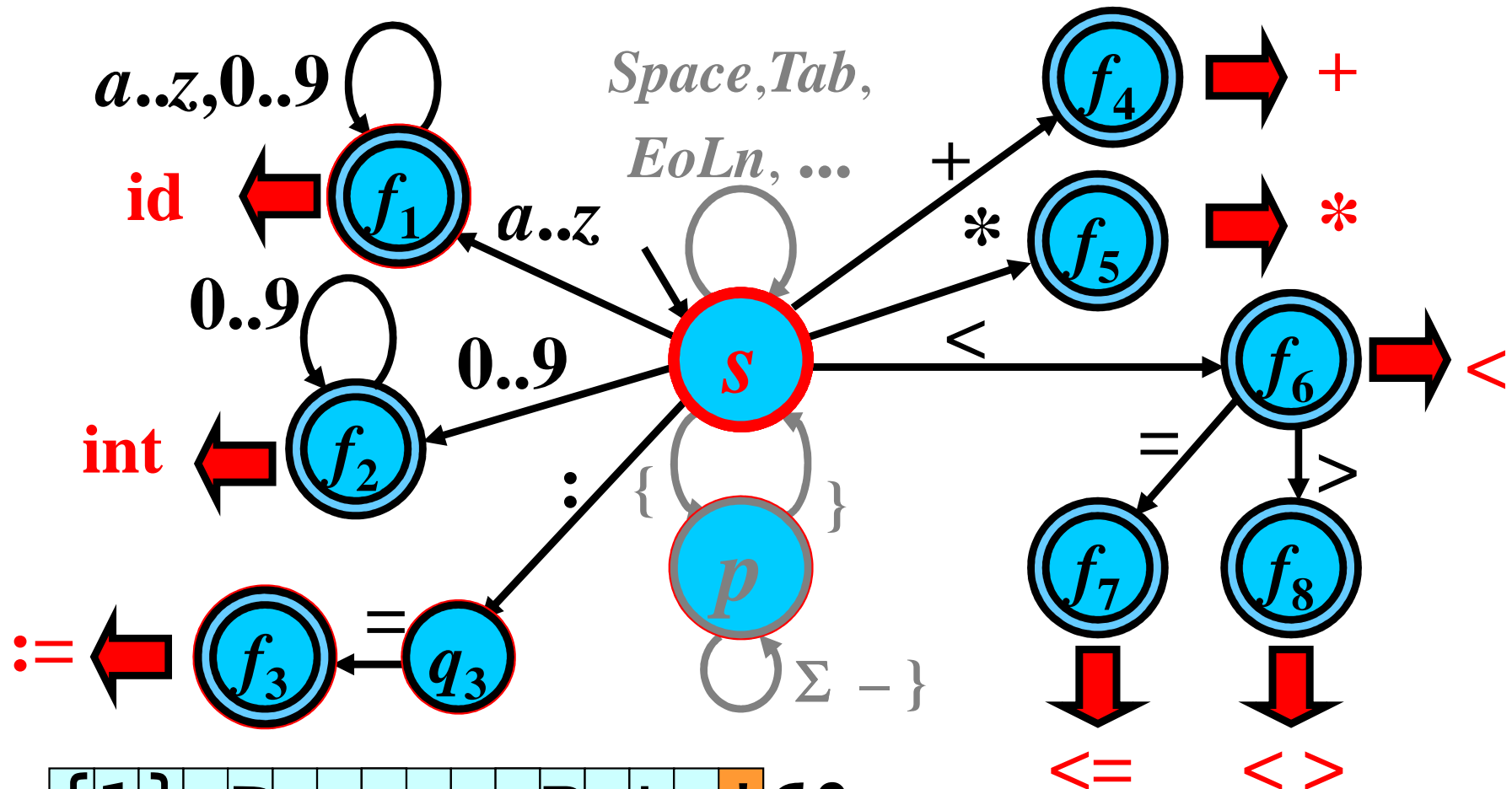
id
Pos

:=

id
Rate

Neexistuje další
konfig.!

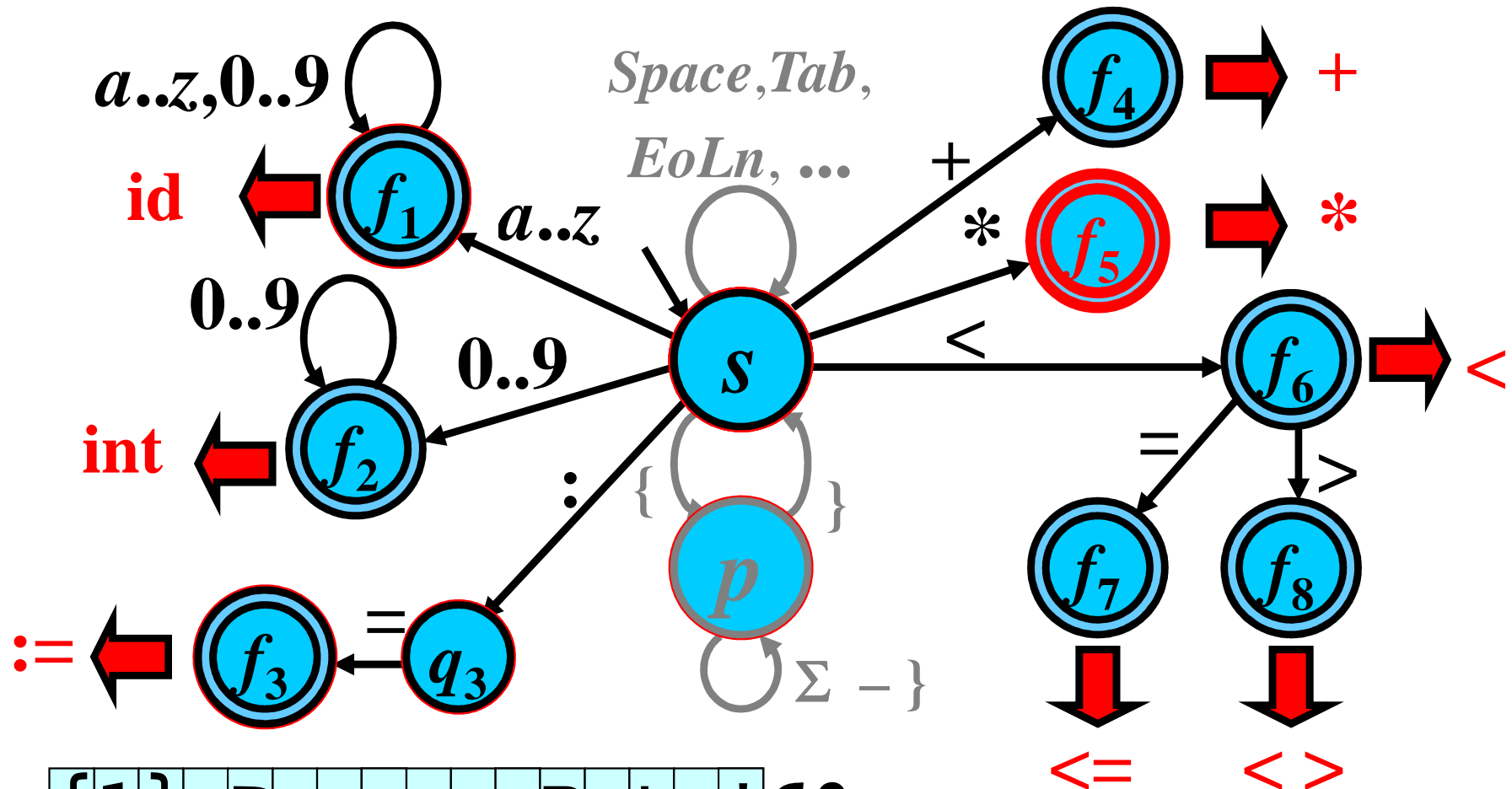
Určení typu lexému: Příklad



{1} Pos := Rate*60

id	:=	id
Pos		Rate

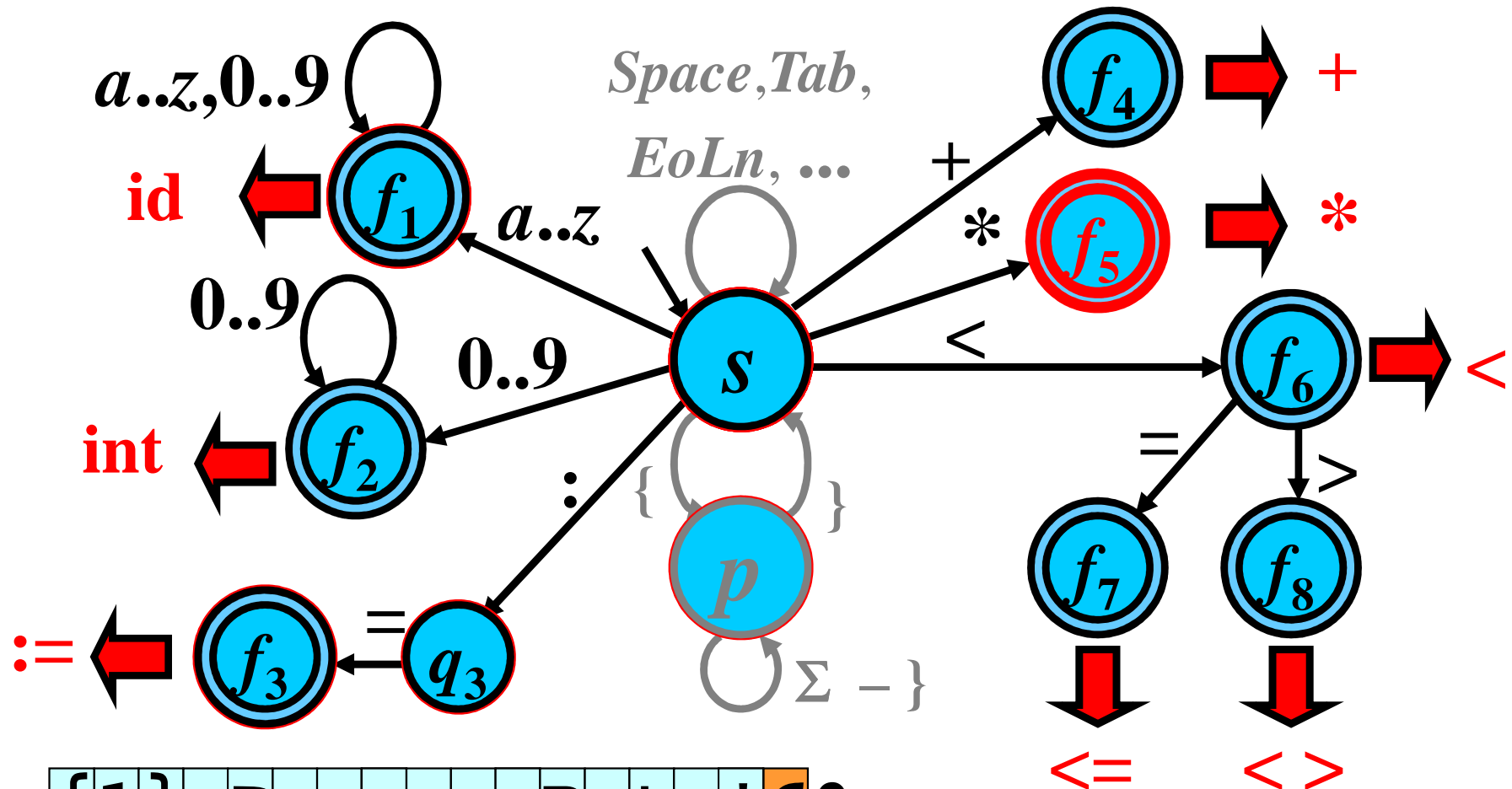
Určení typu lexému: Příklad



{1} Pos := Rate*60

id Pos := id Rate *

Určení typu lexému: Příklad



{1} Pos := Rate*60

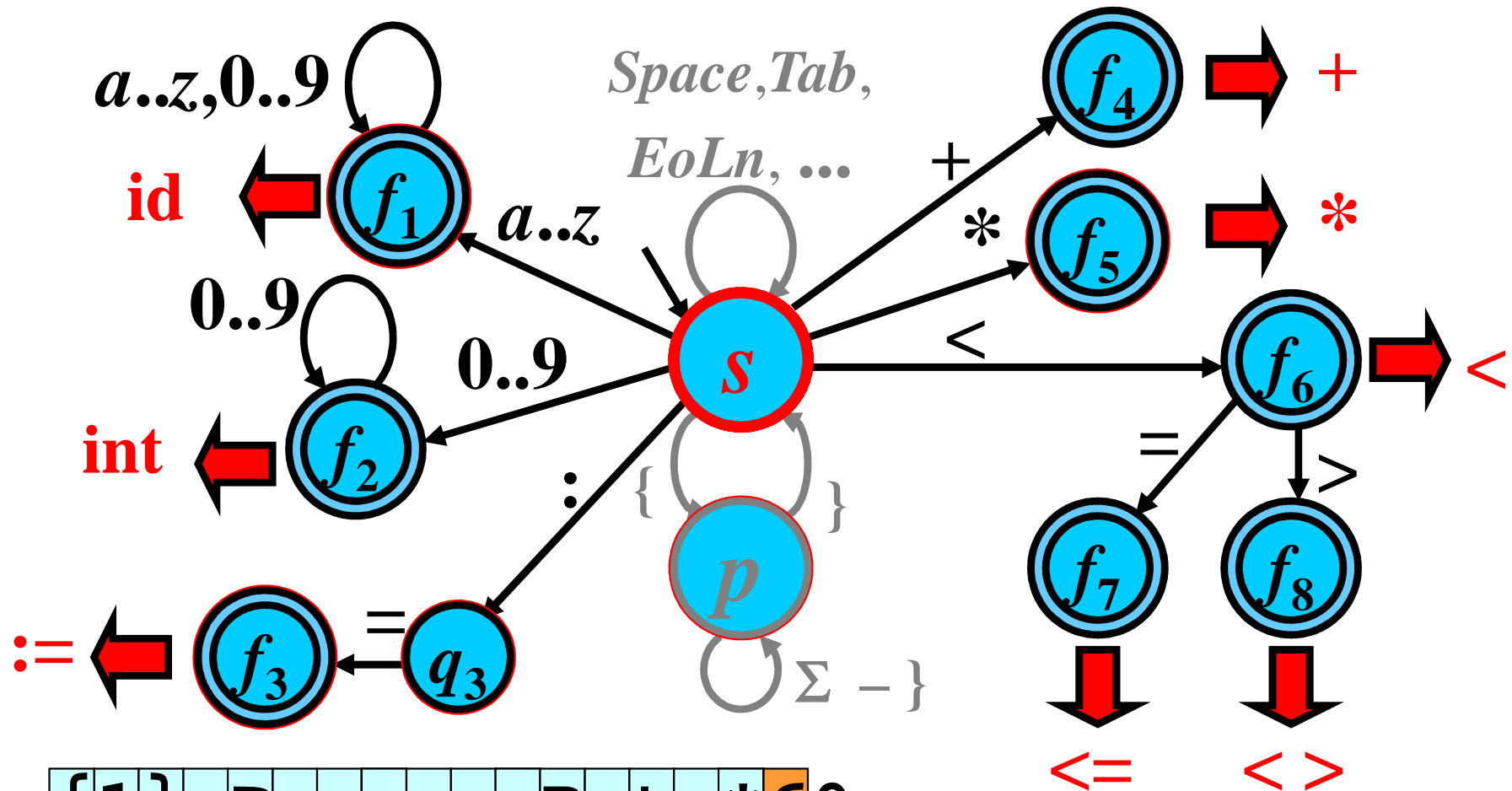
id
Pos

::=

id
Rate

Neexistuje další
konfig.!

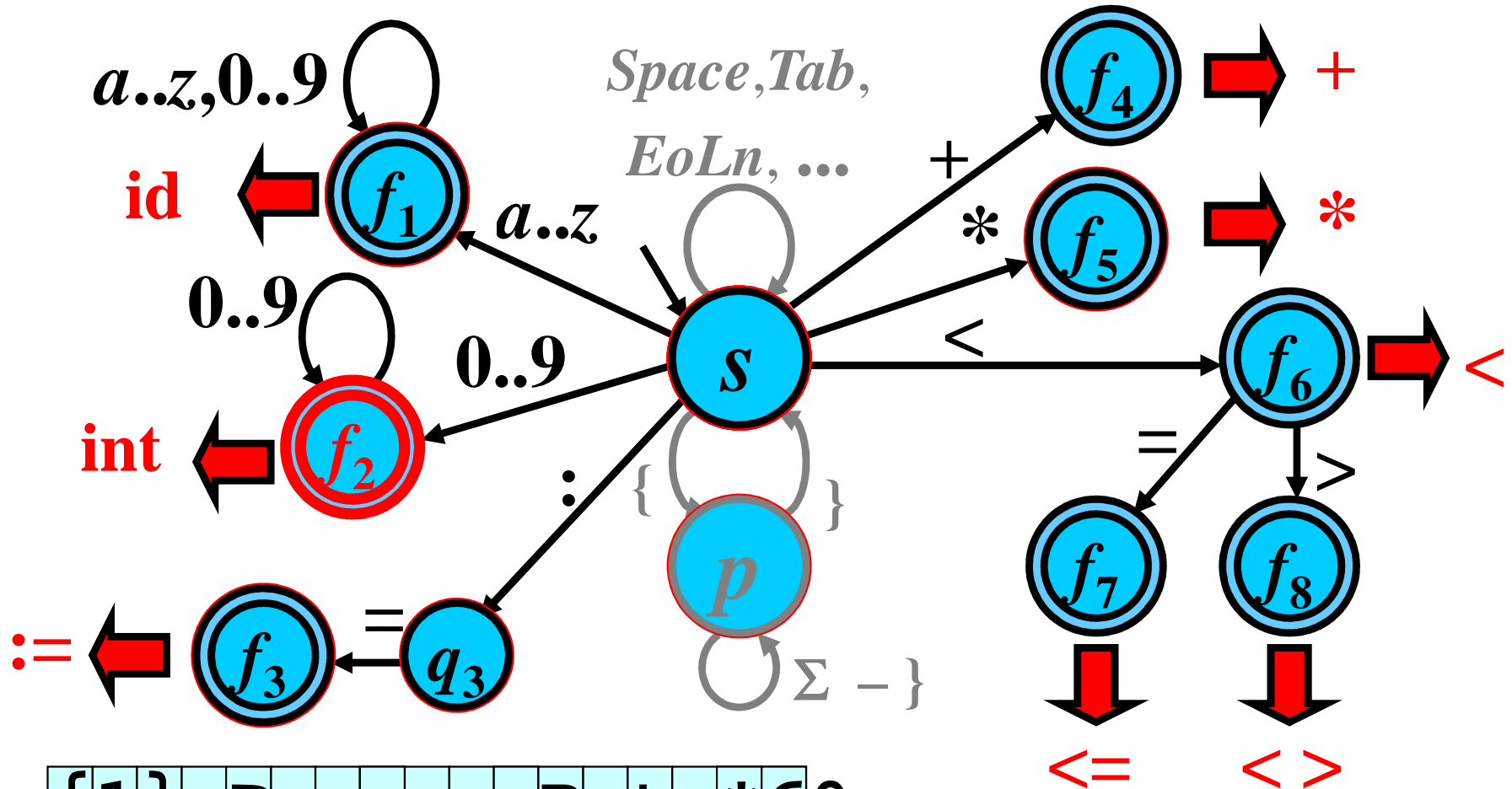
Určení typu lexému: Příklad



{1} Pos := Rate*60

id Pos := id Rate *

Určení typu lexému: Příklad



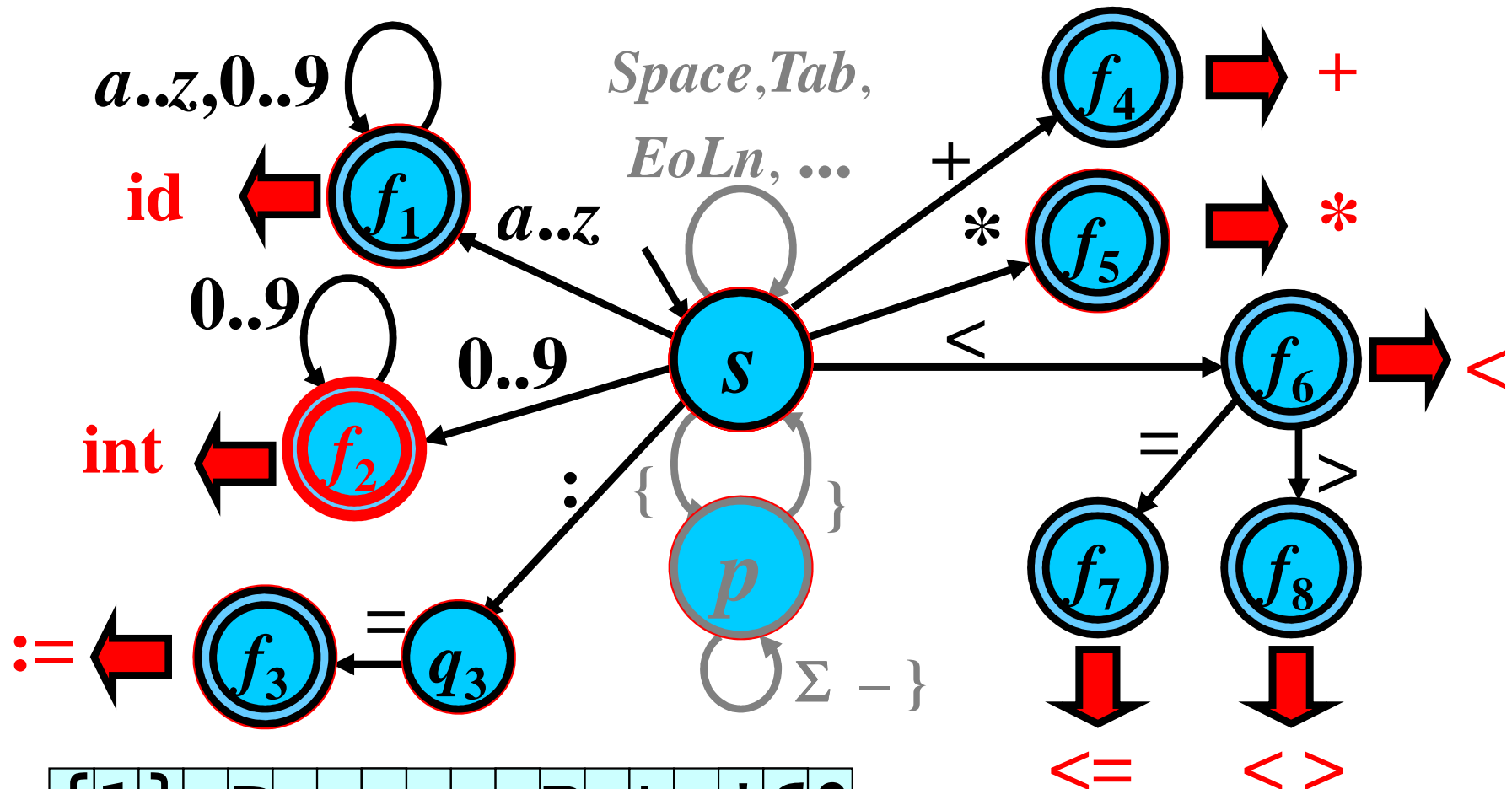
{1}	Pos	:=	Rate*60
-----	-----	----	---------

Diagram illustrating the evaluation of the expression `id := id * 6`. The expression is broken down into four components, each in a blue box:

- `id` (red text) over `Pos` (black text)
- `:=` (red text)
- `id` (red text) over `Rate` (black text)
- `*` (red text)

The value `6` (black text) is shown to the right of the multiplication box.

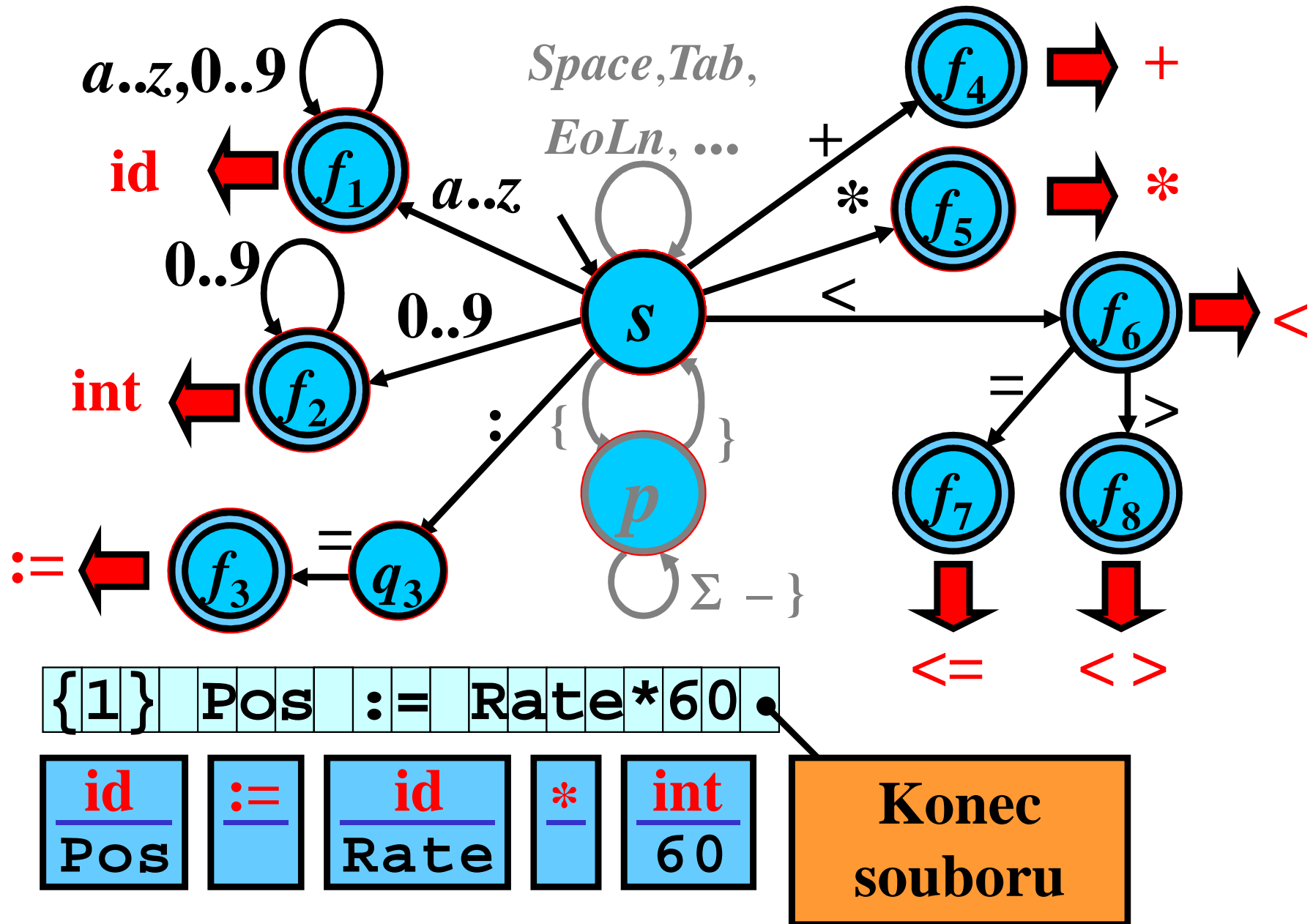
Určení typu lexému: Příklad



{1} Pos := Rate*60

id Pos := id Rate * 60

Určení typu lexému: Příklad



Implementace DKA 1/10

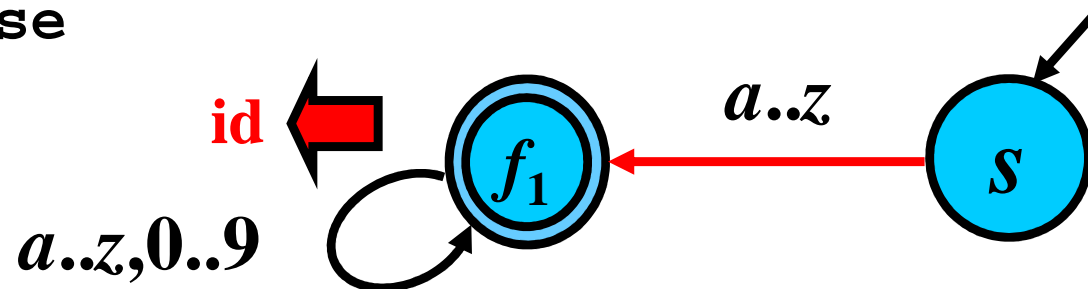
```

procedure get_Next-Token(var TOKEN: ....);

...                                     {deklarace, ...}

str      := '';                       {čtený řetězec}
state    := s;                         {aktuální stav}
repeat
  symbol = getchar();                 {čtení dalšího znaku}
  case state of
    s : begin                           {počáteční stav}
      if symbol in ['a'..'z'] then
        begin
          state := f1;                  {identifikátor}
          str    := symbol;
        end else

```



Implementace DKA 2/10

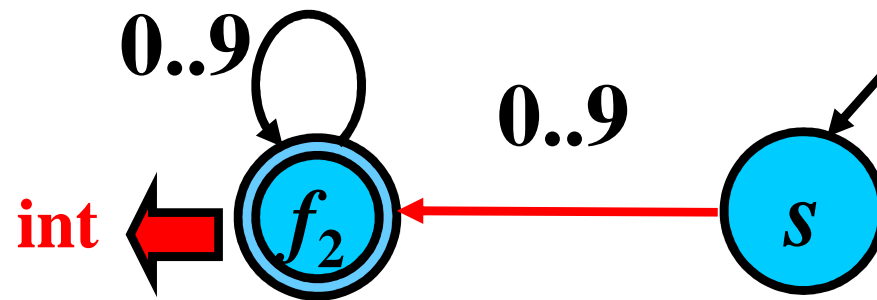
```

case state of
  s : begin                                {počáteční stav}

    ...

    if symbol in ['0'..'9'] then
    begin
      state := f2;                        {celé číslo}
      str   := symbol;
    end else

```

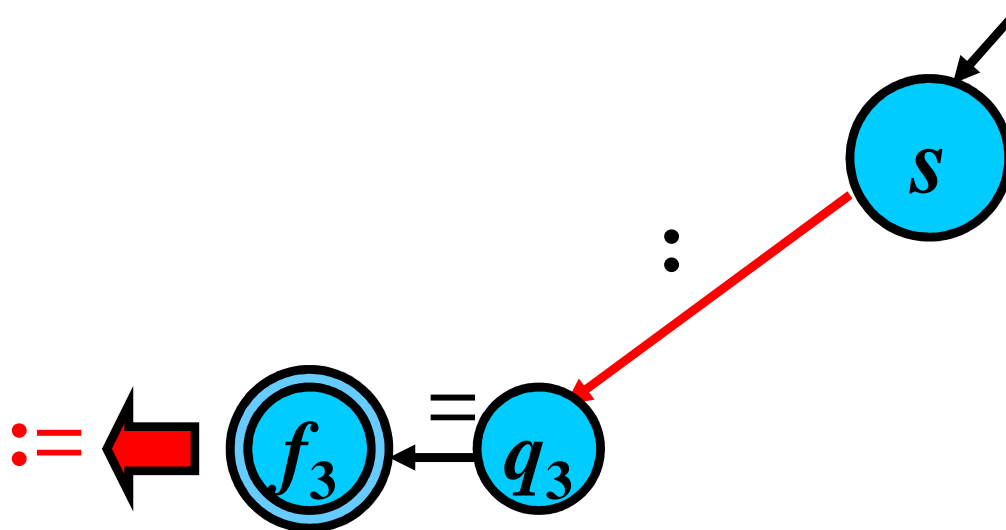


Implementace DKA 3/10

```
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
```

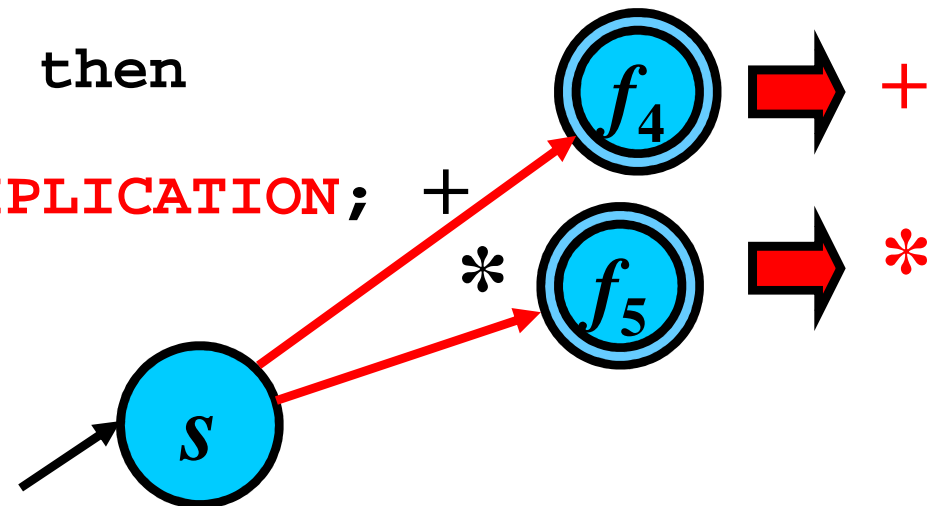
```
if symbol = '*' then
```

```
begin
```

```
  TOKEN := MULTIPLICATION;
```

```
  break;
```

```
end else
```



Implementace DKA 5/10

```
case state of
  s : begin
```

{**počáteční stav**}

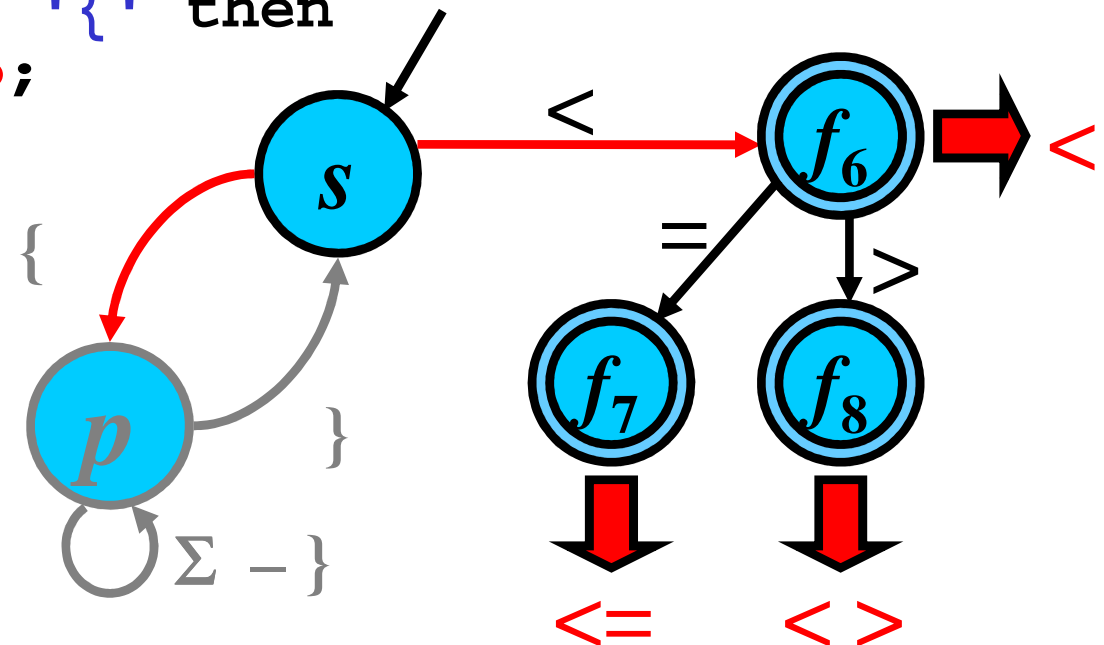
...

```
  if symbol = '<' then
    state := f6;
```

```
  else
```

```
    if symbol = '{' then
      state := p;
```

```
  end; {stav s}
```



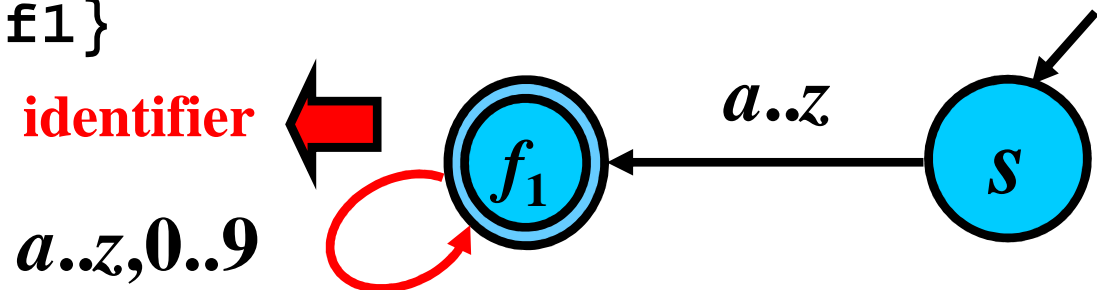
Implementace DKA 6/10

case **state** of

```

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

```



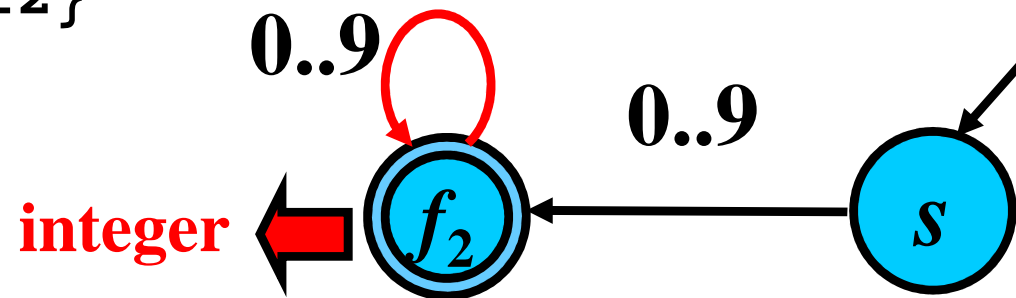
Implementace DKA 7/10

case **state** of

```

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

```



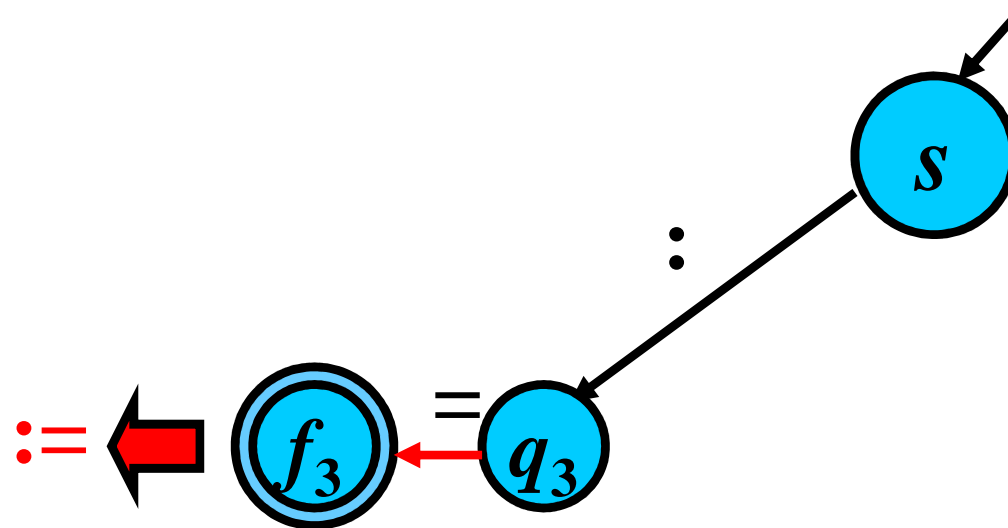
Implementace DKA 8/10

case **state** of

```

...
q3: begin                                {přiřazení}
    if symbol = '=' then
    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

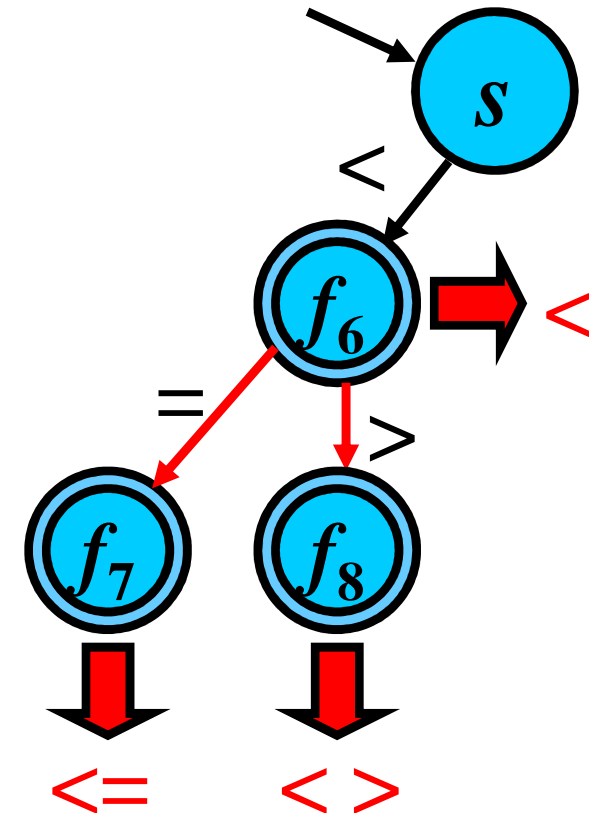
ungetchar(symbol); {návrat symbolu}

TOKEN := **LTN**; {<}

break;

end;

end; {stav f6}



Implementace DKA 10/10

case **state** of

```

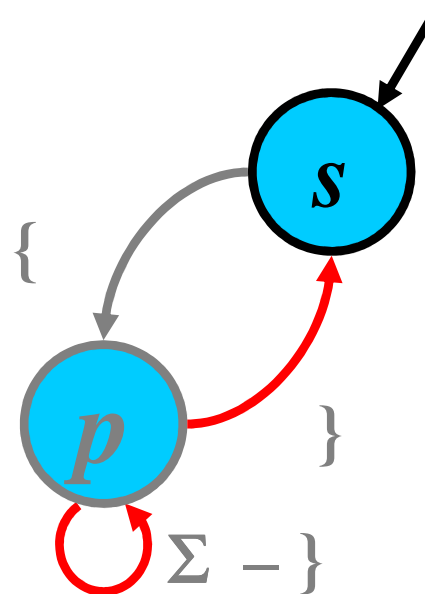
...
p : begin                                {komentář}
    if symbol = '}' then                {počáteční stav}
        state := s;
    end; {stav p}

```

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, <u>60</u>],	[*, <u> </u>]
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

[**id**,  **Pos**]


[,]

①	
Tabulka id :	
1:	Id1

[**int**, **60**]


[,]

②	
Tabulka int :	
1:	25
2:	10000

[*****,]


[,]

③	
Tabulka op :	
1:	*
2:	/
3:	+
4:	-

Řešení stejného typu atributů

[**id**,  **Pos**]

↓
[**1**,]

<div> <div>1</div> <div>Tabulka id:</div> </div>
1: Id1

[**int**, **60**]

↓
[,]

<div> <div>2</div> <div>Tabulka int:</div> </div>
1: 25
2: 10000

[*****,]

↓
[,]

<div> <div>3</div> <div>Tabulka op:</div> </div>
1: *
2: /
3: +
4: −

Řešení stejného typu atributů

[**id**,  **Pos**]

[**int**, **60**]

[*****,]

[, 

[,]

[,]

<div> <div>1</div> <div>Tabulka id:</div> </div>	
1:	Id1
2:	Pos

<div> <div>2</div> <div>Tabulka int:</div> </div>	
1:	25
2:	10000

<div> <div>3</div> <div>Tabulka op:</div> </div>	
1:	*
2:	/
3:	+
4:	-

Řešení stejného typu atributů

[**id**,  **Pos**]

[**1**, **2**]

<div> <div>1</div> <div>Tabulka id:</div> </div>	
1:	Id1
2:	Pos

[**int**, **60**]

[**2**,]

<div> <div>2</div> <div>Tabulka int:</div> </div>	
1:	25
2:	10000

[*****,]

[,]

<div> <div>3</div> <div>Tabulka op:</div> </div>	
1:	*
2:	/
3:	+
4:	-

Řešení stejného typu atributů

[**id**,  **Pos**]

[**1**, **2**]

<div> <div>1</div> <div>Tabulka id:</div> </div>	
1:	Id1
2:	Pos

[**int**, **60**]

[**2**, **3**]

<div> <div>2</div> <div>Tabulka int:</div> </div>	
1:	25
2:	10000
3:	60

[*****,]

[,]

<div> <div>3</div> <div>Tabulka op:</div> </div>	
1:	*
2:	/
3:	+
4:	-

Řešení stejného typu atributů

[**id**,  **Pos**]

[**1**, **2**]

Tabulka id :	
1 :	Id1
2 :	Pos

[**int**, **60**]

[**2**, **3**]

Tabulka int :	
1 :	25
2 :	10000
3 :	60

[*****,]

[**3**,]

Tabulka op :	
1 :	*
2 :	/
3 :	+
4 :	-

Řešení stejného typu atributů

[**id**,  **Pos**]

[**1**, **2**]

Tabulka id :	
1 :	Id1
2 :	Pos

[**int**, **60**]

[**2**, **3**]

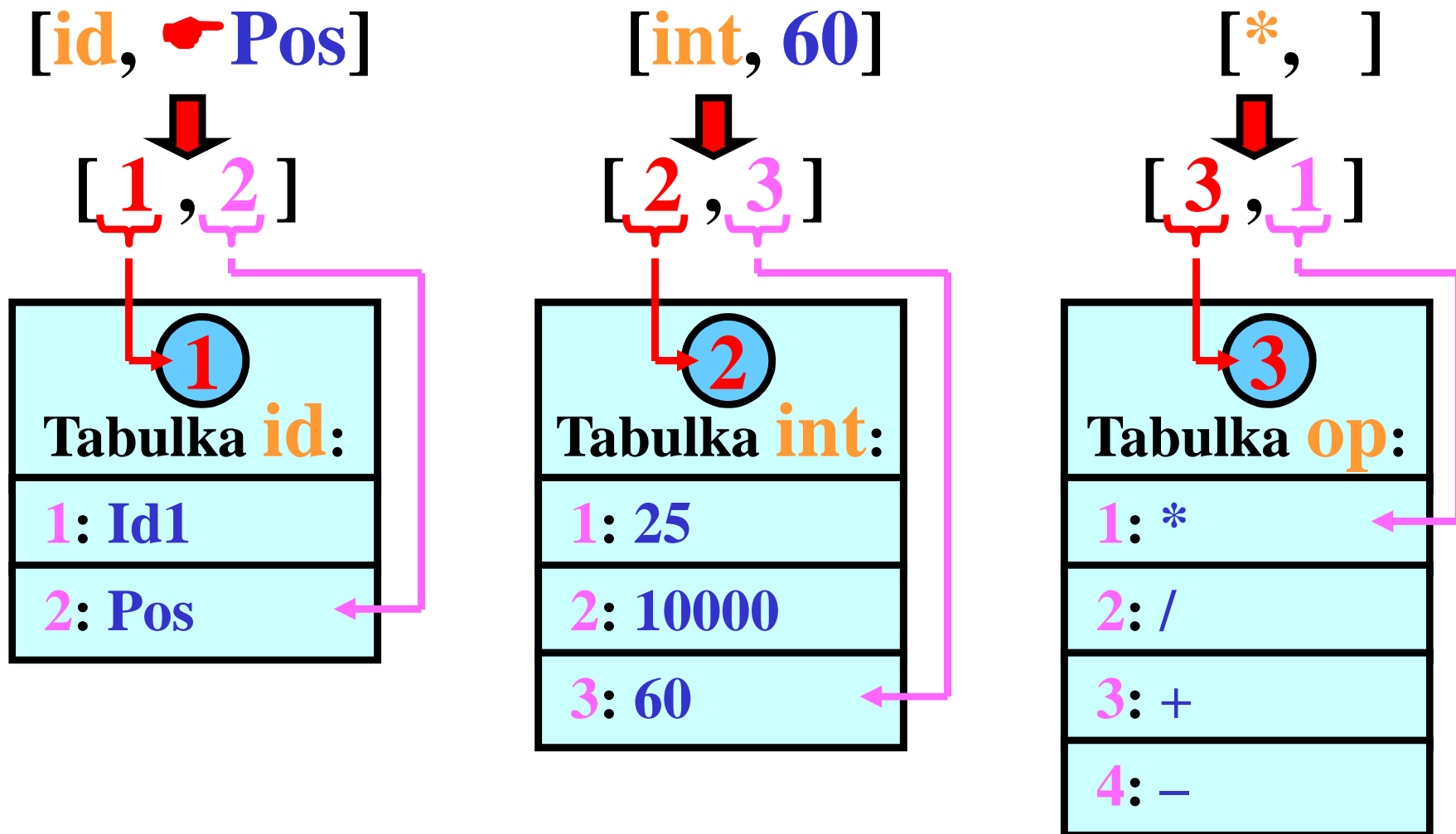
Tabulka int :	
1 :	25
2 :	10000
3 :	60

[*****,]

[**3**, **1**]

Tabulka op :	
1 :	*
2 :	/
3 :	+
4 :	-

Řešení stejného typu atributů



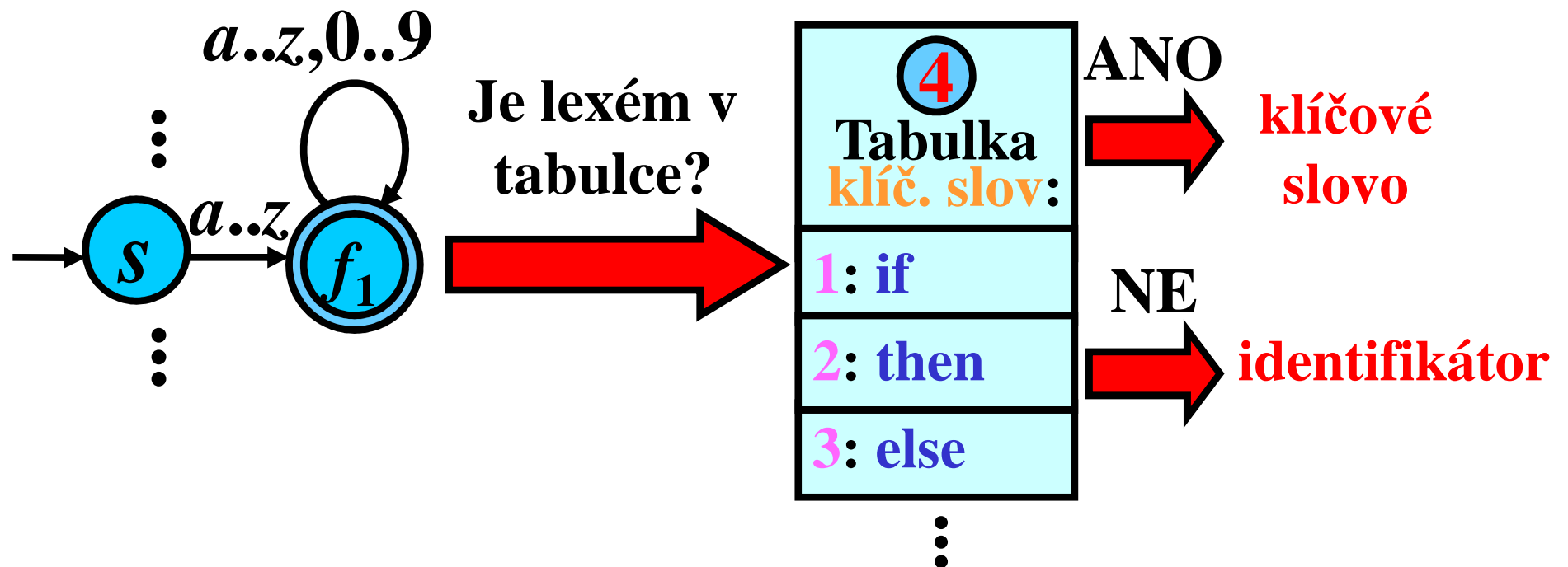
Uniformní forma tokenů: $[1, 2]; [2, 3]; [3, 1]$
 Homogenní struktura

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.	...	<i>n.</i>
1:	Id	1	—	—	—	...	—
2:	Pos	—	—	—	—	...	—
3:	X	—	—	—	—	...	—
	⋮						⋮

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

1:	•
2:	•
3:	•
	⋮

Id1#Pos#X#...

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
-

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:**

Program P1;

Tabulka symbolů

Zásobníková struktura tabulky symbolů

- Problém:**

```
Program P1;  
var x, y: integer;
```

Tabulka symbolů		
1:	x	...
2:	y	...

Zásobníková struktura tabulky symbolů

- Problém:**

```
Program P1;  
var x, y: integer;  
Procedure Proc1;
```

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

Zásobníková struktura tabulky symbolů

- Problém:**

```
Program P1;  
var x, y: integer;  
  
Procedure Proc1;  
var x, y: integer;
```

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

Zásobníková struktura tabulky symbolů

- Problém:**

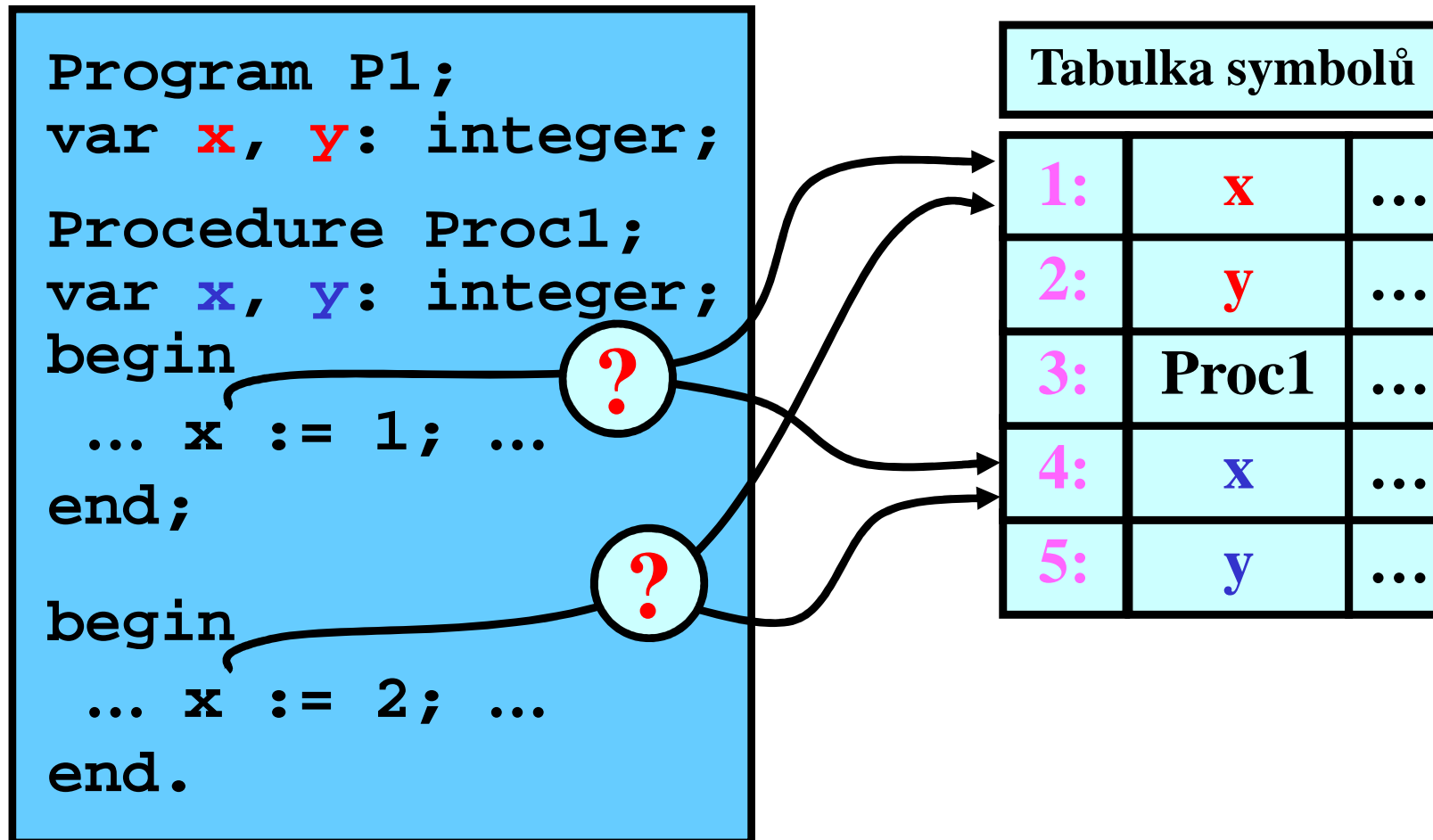
```
Program P1;  
var x, y: integer;  
Procedure Proc1;  
var x, y: integer;  
begin  
    ... x := 1; ...  
end;
```

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



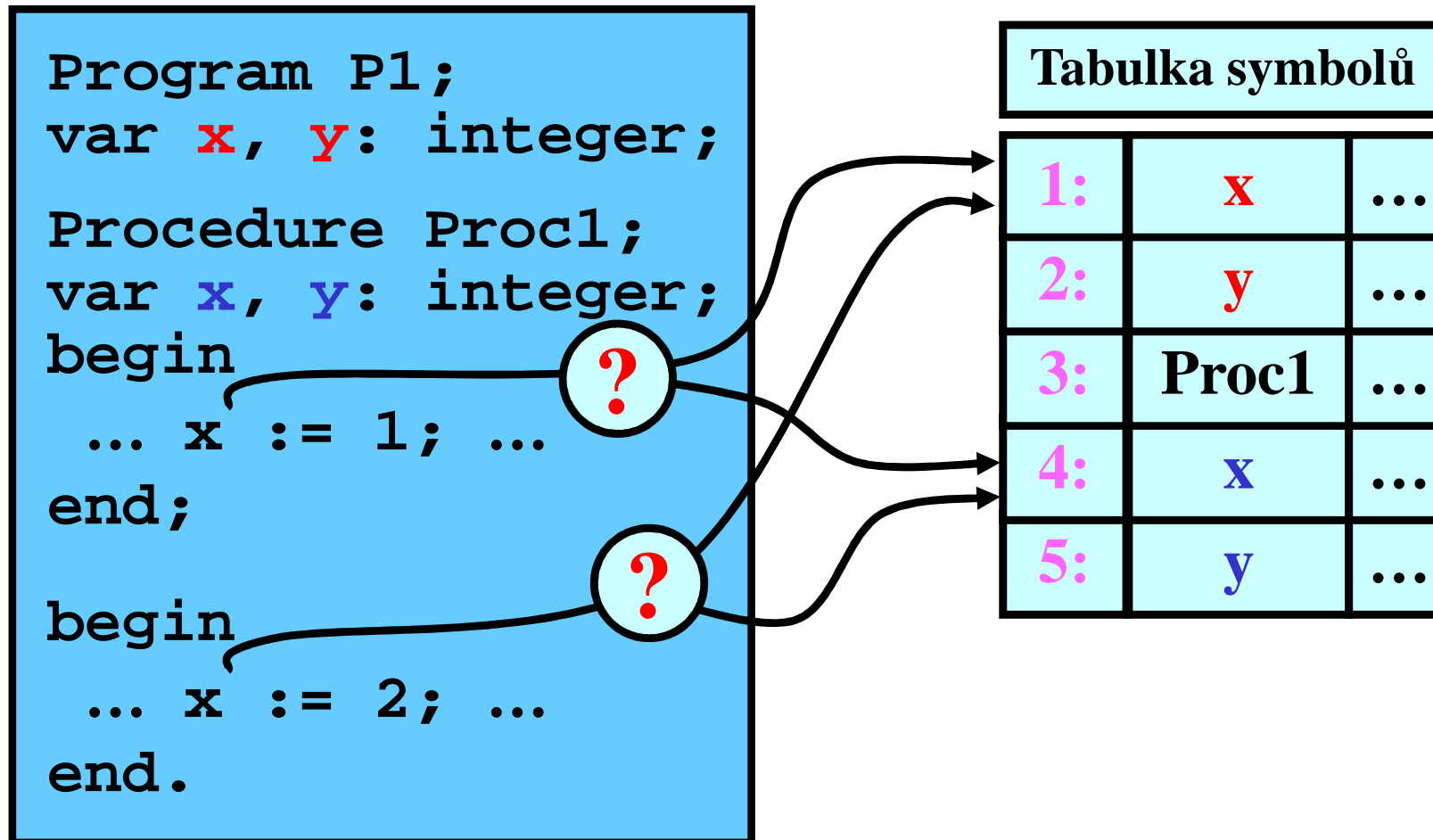
Zásobníková struktura tabulky symbolů

- Problém:**



Zásobníková struktura tabulky symbolů

- Problém:**



- Řešení:** Zásobníková struktura TS

Zásobníková struktura

**Tabulka symbolů =
TS-zásobník:**

**Pomocná
tabulka =
PT-zásobník:**



Zásobníková struktura

Hlavní blok (B0)

**Tabulka symbolů =
TS-zásobník:**

**Pomocná
tabulka =
PT-zásobník:**



Zásobníková struktura

Hlavní blok (B0)

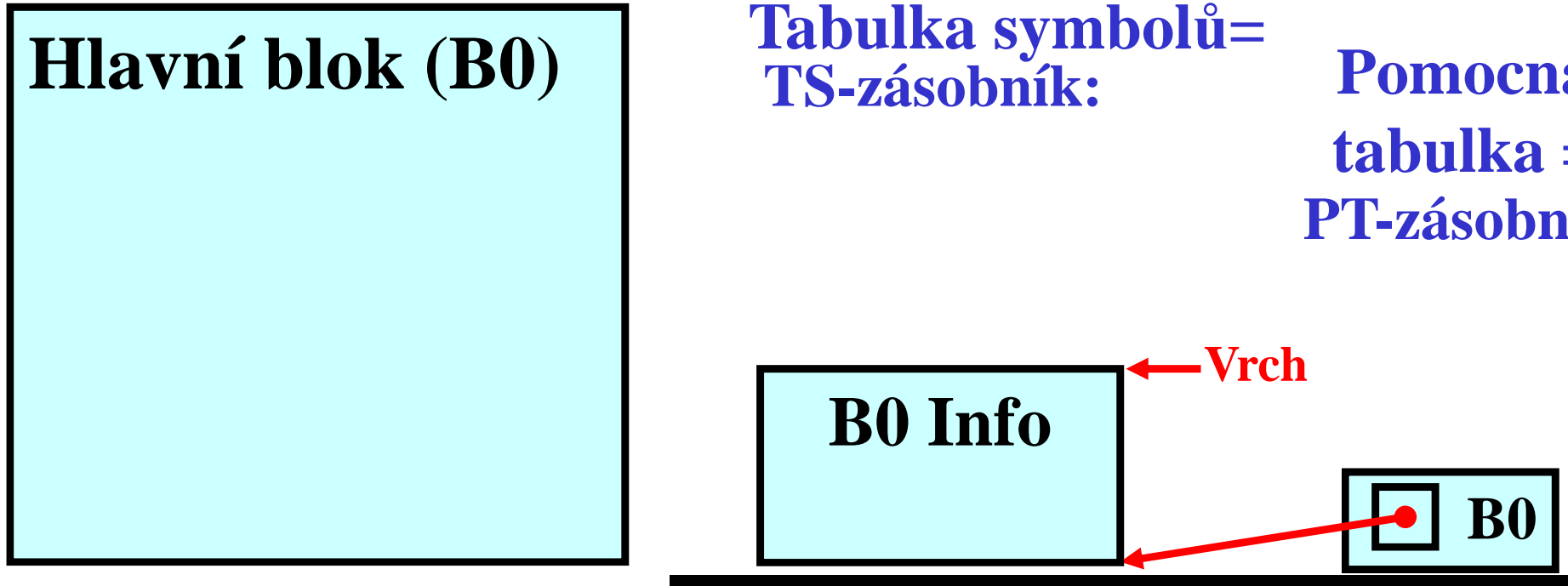
**Tabulka symbolů =
TS-zásobník:**

**Pomocná
tabulka =
PT-zásobník:**

B0 Info

← Vrch

B0



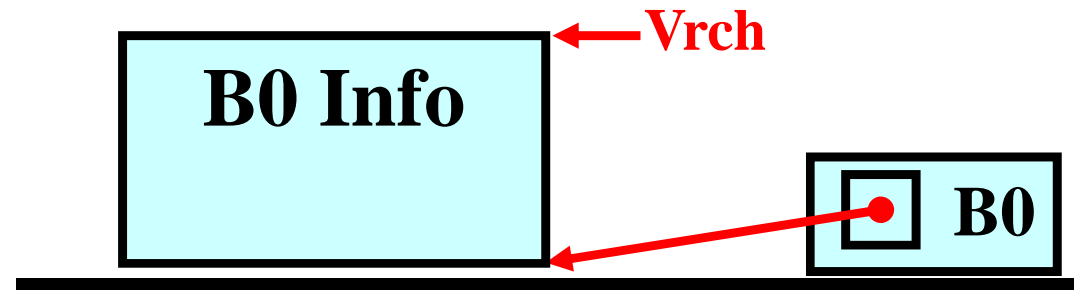
Zásobníková struktura

Hlavní blok (B0)

Blok 1 (B1)

**Tabulka symbolů =
TS-zásobník:**

**Pomocná
tabulka =
PT-zásobník:**



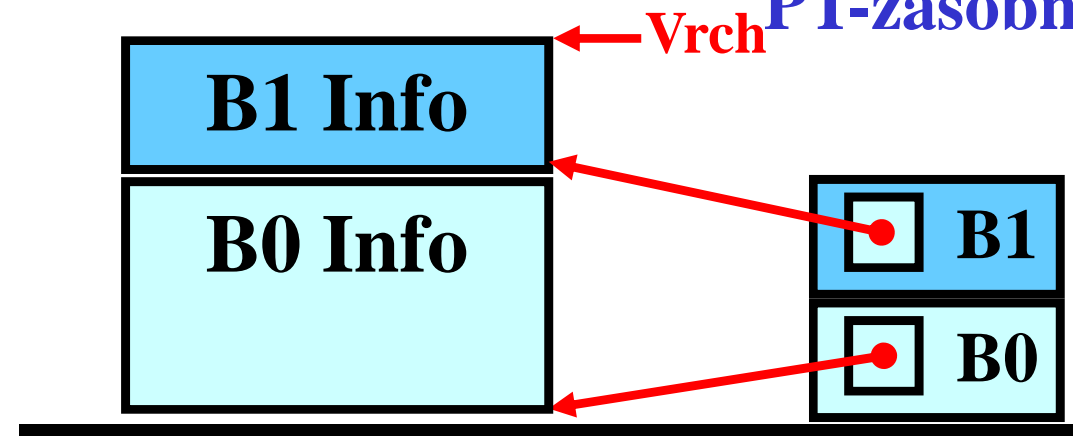
Zásobníková struktura

Hlavní blok (B0)

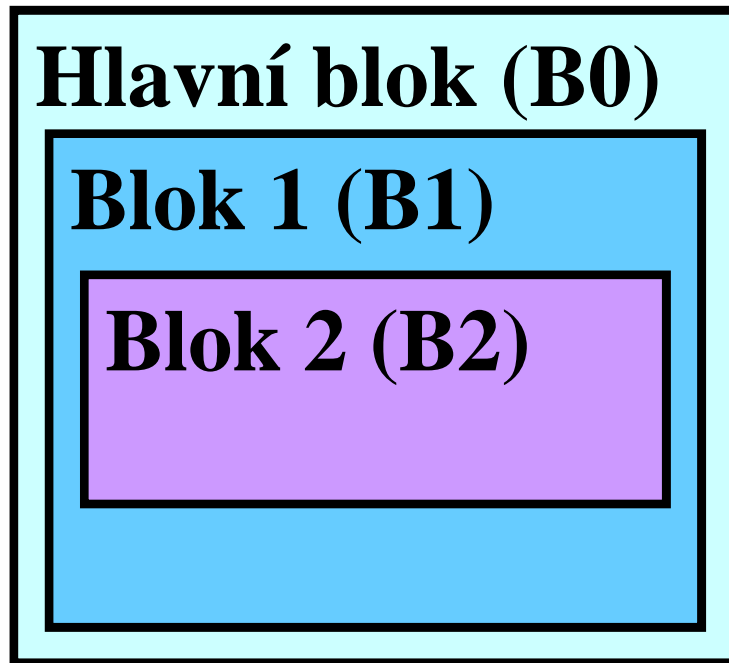
Blok 1 (B1)

**Tabulka symbolů =
TS-zásobník:**

**Pomocná
tabulka =
PT-zásobník:**

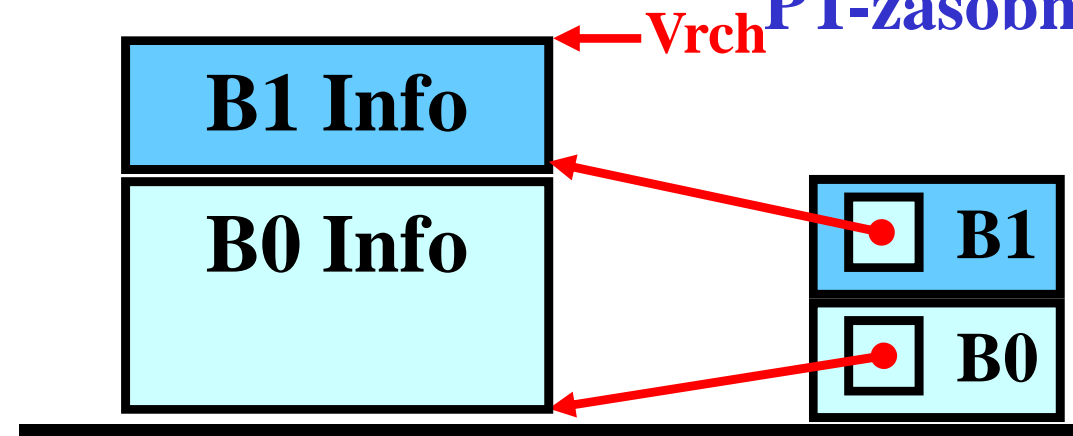


Zásobníková struktura

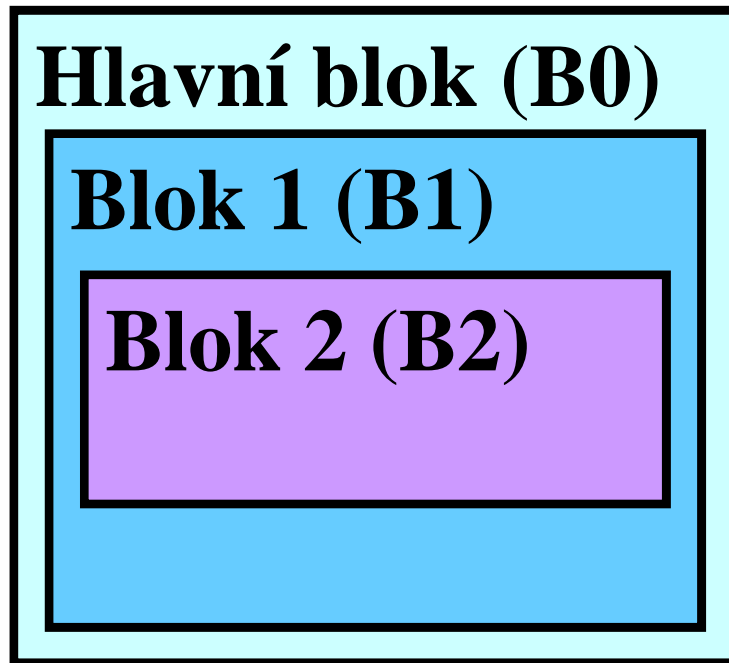


**Tabulka symbolů =
TS-zásobník:**

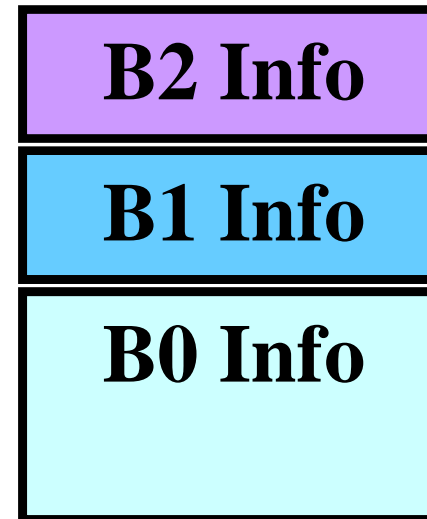
**Pomocná
tabulka =
PT-zásobník:**



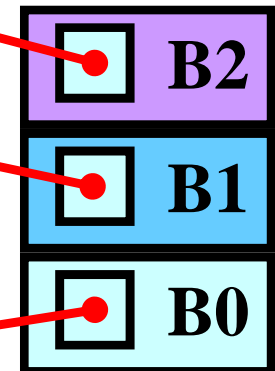
Zásobníková struktura



**Tabulka symbolů =
TS-zásobník:**



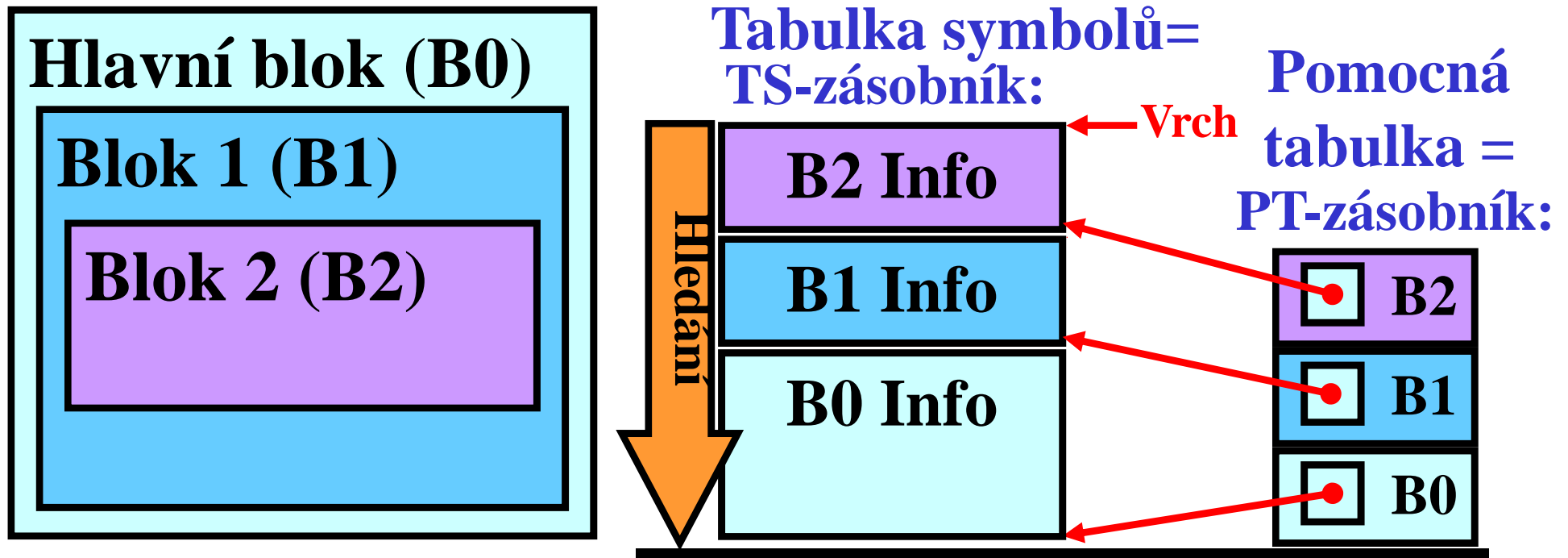
**Pomocná
tabulka =
PT-zásobník:**



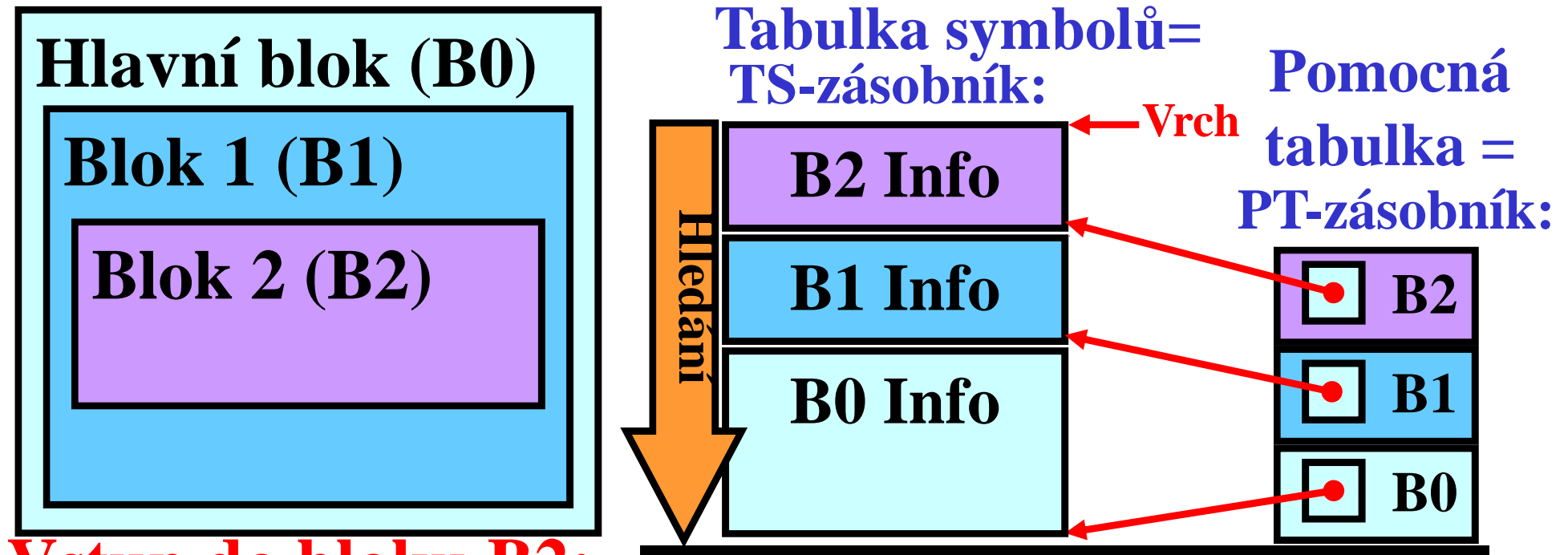
Vrch



Zásobníková struktura



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

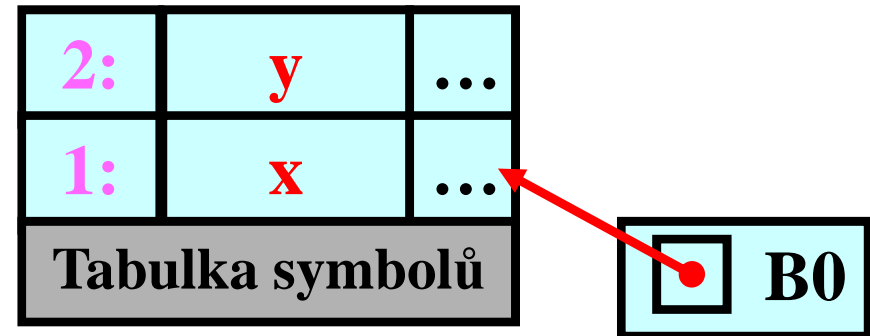
Zásobníková struktura: Příklad

Program P1;

Tabulka symbolů

Zásobníková struktura: Příklad

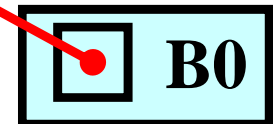
```
Program P1;  
var x, y: integer;
```



Zásobníková struktura: Příklad

```
Program P1;  
var x, y: integer;  
Procedure Proc1;
```

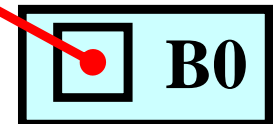
3:	Proc1	...
2:	y	...
1:	x	...
Tabulka symbolů		



Zásobníková struktura: Příklad

```
Program P1;  
var x, y: integer;  
  
Procedure Proc1;  
var x, y: integer;  
  
  ...  
  
end Proc1;  
  
begin  
  ...  
  
end P1;
```

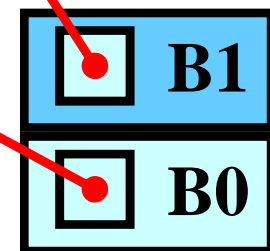
3:	Proc1	...
2:	y	...
1:	x	...
Tabulka symbolů		



Zásobníková struktura: Příklad

```
Program P1;  
var x, y: integer;  
  
Procedure Proc1;  
var x, y: integer;
```

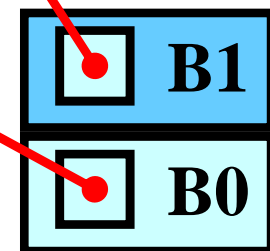
5:	y	...
4:	x	...
3:	Proc1	...
2:	y	...
1:	x	...
Tabulka symbolů		



Zásobníková struktura: Příklad

```
Program P1;  
var x, y: integer;  
  
Procedure Proc1;  
var x, y: integer;  
begin  
    ... x := 1; ...  
end;
```

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



Zásobníková struktura: Příklad

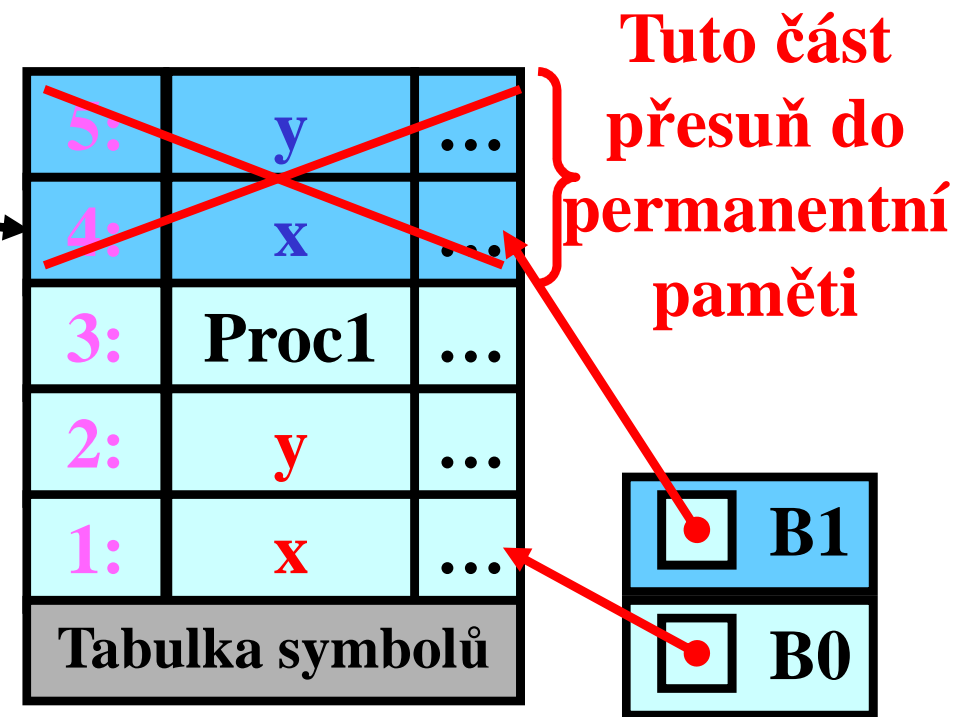
```

Program P1;
var x, y: integer;

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

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

```



Zásobníková struktura: Příklad

```

Program P1;
var x, y: integer;

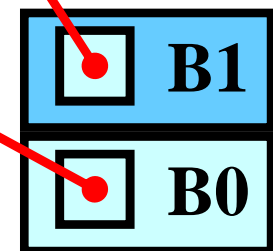
Procedure Proc1;
var x, y: integer;
begin
    ... x := 1; ...
end;

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

```

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

Tuto část
přesuň do
permanentní
paměti



Zásobníková struktura: Příklad

```

Program P1;
var x, y: integer;

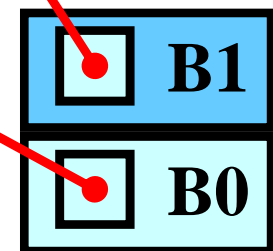
Procedure Proc1;
var x, y: integer;
begin
    ... x := 1; ...
end;

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

```

Tabulka symbolů:

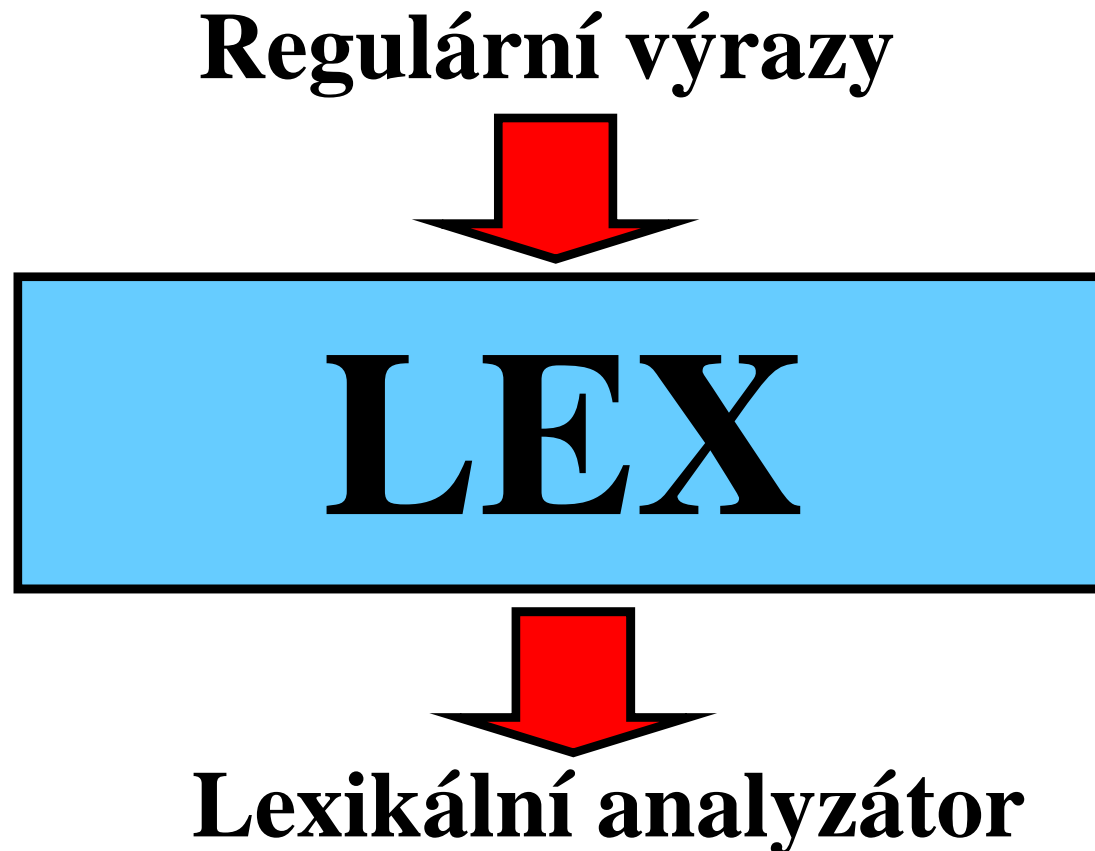
5:	y	...
4:	x	...
3:	Proc1	...
2:	y	...
1:	x	...
Tabulka symbolů		



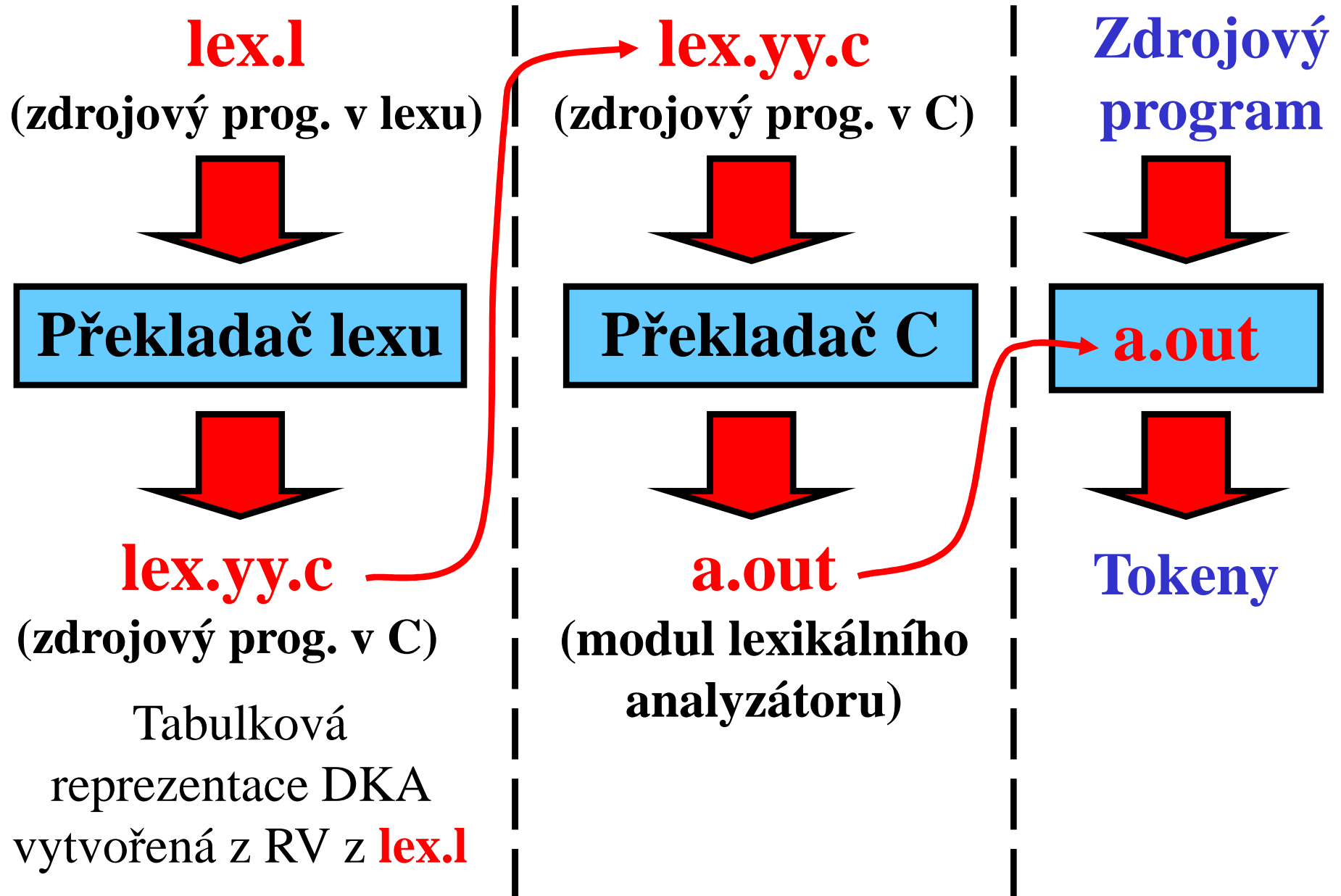
LEX: Hlavní myšlenka

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

Ilustrace:



LEX: Fáze kompilace



Struktura zdrojového programu v LEXu

/* Sekce I: Deklarace */

d_1, d_2, \dots, d_i

%% **/* Konec sekce I*/**

/* Sekce II: Překládová 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	<i>a</i>
rs	<i>r.s</i>
r s	<i>r + s</i>
r*	<i>r*</i>
r+	<i>r⁺</i>
r?	<i>r + ε</i>
[a-z]	<i>a + b + c + ... + z</i>
[0-9]	<i>0 + 1 + 2 + ... + 9</i>

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.

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
-

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

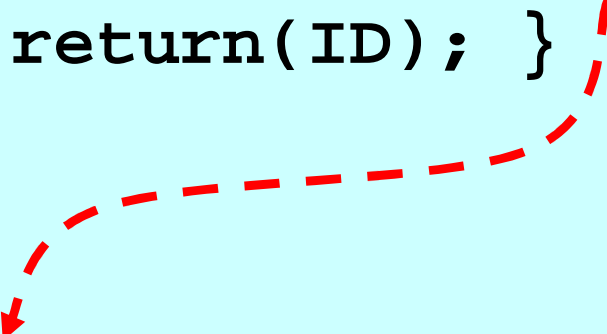
- Pomocné procedury jsou potřeba pro překladová pravidla
-

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 */  
    /* Proměnná yytext obsahuje načtený lexém */  
}  
...
```



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() { ... }

```