Kompilátory

Cvičenie 2: Lexikálna analýza všeobecne a v ANTLR4

Peter Kostolányi

10. októbra 2017

Oznamy

- V utorok 31. októbra je rektorské voľno

Oznamy

- V utorok 31. októbra je rektorské voľno
- Drobné zmeny v pláne cvičení

Oznamy

- V utorok 31. októbra je rektorské voľno
- Drobné zmeny v pláne cvičení

Týždeň	Dátum	Téma
1.	26.9.2017	Úvod
3.	10.10.2017	Lexikálna analýza, tvorba lexerov v ANTLR
5.	24.10.2017	$\mathrm{SLL}(k)$, $\mathrm{LL}(k)$, $\mathrm{LL}(\mathrm{Reg})$, $\mathrm{LL}(*)$ a $\mathrm{ALL}(*)$
7.	7.11.2017	ANTLR
8.	14.11.2017	ANTLR
9.	21.11.2017	LLVM
10.	28.11.2017	Generovanie kódu

Oznamy

- V utorok 31. októbra je rektorské voľno
- Drobné zmeny v pláne cvičení

Týždeň	Dátum	Téma
1.	26.9.2017	Úvod
3.	10.10.2017	Lexikálna analýza, tvorba lexerov v ANTLR
5.	24.10.2017	$\mathrm{SLL}(k)$, $\mathrm{LL}(k)$, $\mathrm{LL}(\mathrm{Reg})$, $\mathrm{LL}(*)$ a $\mathrm{ALL}(*)$
7.	7.11.2017	ANTLR
8.	14.11.2017	ANTLR
9.	21.11.2017	LLVM
10.	28.11.2017	Generovanie kódu

Aktuálny harmonogram cvičení je na stránke predmetu:
`https://micro.dcs.fmph.uniba.sk/dokuwiki/sk:dcs:co:start`

Projekt

- Detailné zadanie je na stránke predmetu

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

- Kompilátor vlastného vyššieho programovacieho jazyka

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

- Kompilátor vlastného vyššieho programovacieho jazyka
- Napísaný s využitím ANTLR, kompilácia do LLVM

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

- Kompilátor vlastného vyššieho programovacieho jazyka
- Napísaný s využitím ANTLR, kompilácia do LLVM
- Musí podporovať určité dátové typy a operácie na nich

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

- Kompilátor vlastného vyššieho programovacieho jazyka
- Napísaný s využitím ANTLR, kompilácia do LLVM
- Musí podporovať určité dátové typy a operácie na nich
- Musí podporovať určité konštrukcie (podmienky, aspoň 1 druh cyklu, ...)

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

- Kompilátor vlastného vyššieho programovacieho jazyka
- Napísaný s využitím ANTLR, kompilácia do LLVM
- Musí podporovať určité dátové typy a operácie na nich
- Musí podporovať určité konštrukcie (podmienky, aspoň 1 druh cyklu, ...)
- ...(viac na stránke predmetu)

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

- Kompilátor vlastného vyššieho programovacieho jazyka
- Napísaný s využitím ANTLR, kompilácia do LLVM
- Musí podporovať určité dátové typy a operácie na nich
- Musí podporovať určité konštrukcie (podmienky, aspoň 1 druh cyklu, ...)
- ... (viac na stránke predmetu)

Špecifikácia by mala obsahovať:

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

- Kompilátor vlastného vyššieho programovacieho jazyka
- Napísaný s využitím ANTLR, kompilácia do LLVM
- Musí podporovať určité dátové typy a operácie na nich
- Musí podporovať určité konštrukcie (podmienky, aspoň 1 druh cyklu, ...)
- ... (viac na stránke predmetu)

Špecifikácia by mala obsahovať:

- Neformálny popis všetkých podporovaných syntaktických konštrukcií (ideálne aj s príkladmi použitia)

Projekt

- Detailné zadanie je na stránke predmetu
- Termín odovzdania neformálnej špecifikácie sa (kvôli rektorskému voľnu) posúva na 6. novembra 2017

Približné zadanie projektu:

- Kompilátor vlastného vyššieho programovacieho jazyka
- Napísaný s využitím ANTLR, kompilácia do LLVM
- Musí podporovať určité dátové typy a operácie na nich
- Musí podporovať určité konštrukcie (podmienky, aspoň 1 druh cyklu, ...)
- ... (viac na stránke predmetu)

Špecifikácia by mala obsahovať:

- Neformálny popis všetkých podporovaných syntaktických konštrukcií (ideálne aj s príkladmi použitia)
- Zdrojové kódy programov riešiacich tri predpísané úlohy (viac na stránke)

Časť 1:

Lexikálna analýza všeobecne

Úloha lexikálnej analýzy pri tvorbe kompilátorov

- Postupnosť symbolov na vstupe je rozdelená na (z pohľadu syntaxe) minimálne zmysluplné podpostupnosti – lexémy

Úloha lexikálnej analýzy pri tvorbe kompilátorov

- Postupnosť symbolov na vstupe je rozdelená na (z pohľadu syntaxe) minimálne zmysluplné podpostupnosti – lexémy
- Na lexémy sa možno pozerať podobne ako na slová v prirodzenom jazyku

Úloha lexikálnej analýzy pri tvorbe kompilátorov

- Postupnosť symbolov na vstupe je rozdelená na (z pohľadu syntaxe) minimálne zmysluplné podpostupnosti – lexémy
- Na lexémy sa možno pozerať podobne ako na slová v prirodzenom jazyku
- Ku každej lexéme je vygenerovaný tzv. token (žiaden slovenský ekvivalent sa nezaužíval)

Úloha lexikálnej analýzy pri tvorbe kompilátorov

- Postupnosť symbolov na vstupe je rozdelená na (z pohľadu syntaxe) minimálne zmysluplné podpostupnosti – lexémy
- Na lexémy sa možno pozerať podobne ako na slová v prirodzenom jazyku
- Ku každej lexéme je vygenerovaný tzv. token (žiaden slovenský ekvivalent sa nezaužíval)
- Každý token obsahuje:

Úloha lexikálnej analýzy pri tvorbe kompilátorov

- Postupnosť symbolov na vstupe je rozdelená na (z pohľadu syntaxe) minimálne zmysluplné podpostupnosti – lexémy
- Na lexémy sa možno pozerať podobne ako na slová v prirodzenom jazyku
- Ku každej lexéme je vygenerovaný tzv. token (žiaden slovenský ekvivalent sa nezaužíval)
- Každý token obsahuje:
  - Typ tokenu – napríklad všetky reťazce znakov slúžiace ako mená premenných alebo funkcií môžu mať typ tokenu „`Id`“

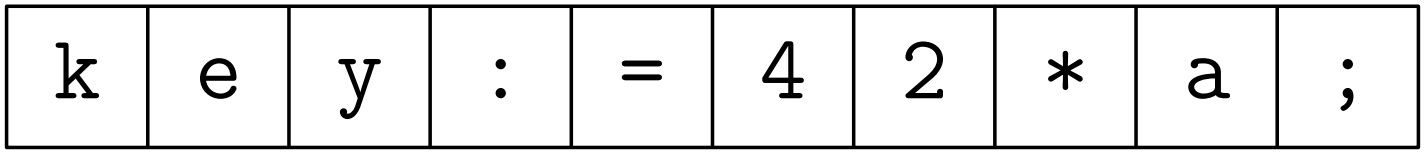
Úloha lexikálnej analýzy pri tvorbe kompilátorov

- Postupnosť symbolov na vstupe je rozdelená na (z pohľadu syntaxe) minimálne zmysluplné podpostupnosti – lexémy
- Na lexémy sa možno pozerať podobne ako na slová v prirodzenom jazyku
- Ku každej lexéme je vygenerovaný tzv. token (žiaden slovenský ekvivalent sa nezaužíval)
- Každý token obsahuje:
  - Typ tokenu – napríklad všetky reťazce znakov slúžiace ako mená premenných alebo funkcií môžu mať typ tokenu „`Id`“
  - Atribúty – napríklad zodpovedajúci text (lexéma), počiatočný a koncový index v postupnosti vstupných symbolov, ...

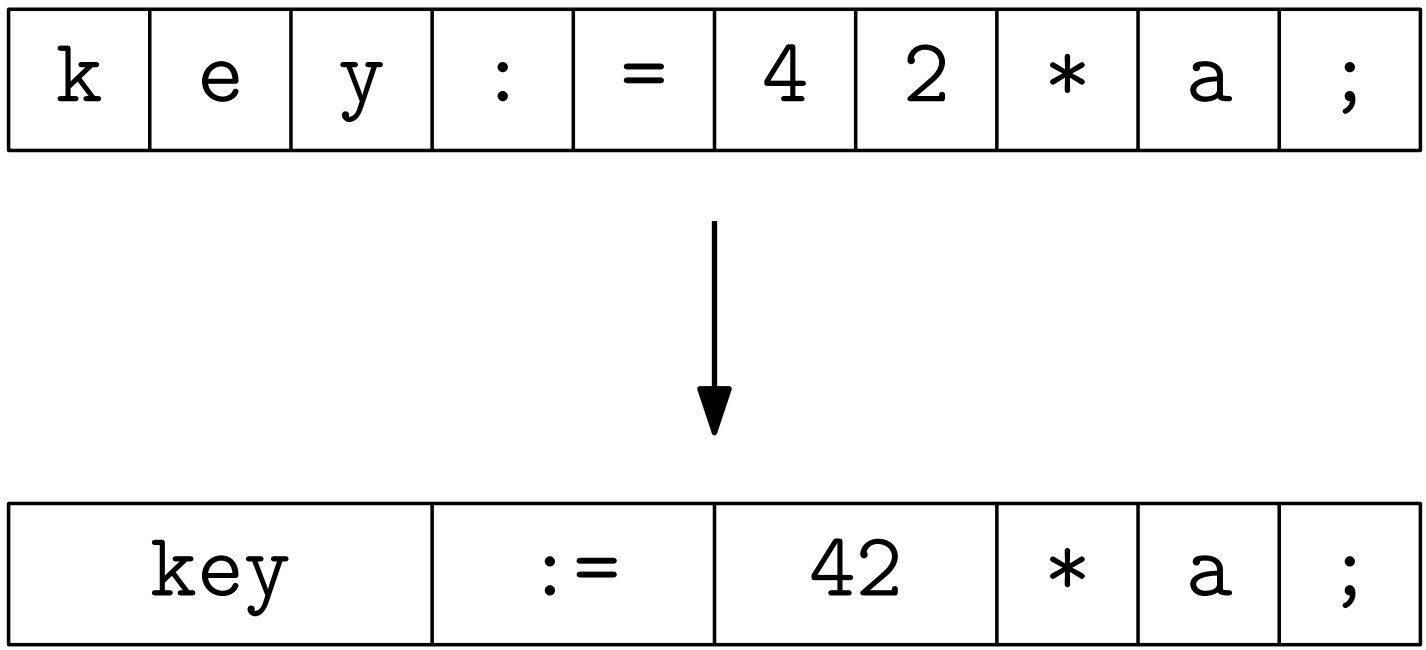
Úloha lexikálnej analýzy pri tvorbe kompilátorov

- Postupnosť symbolov na vstupe je rozdelená na (z pohľadu syntaxe) minimálne zmysluplné podpostupnosti – lexémy
- Na lexémy sa možno pozerať podobne ako na slová v prirodzenom jazyku
- Ku každej lexéme je vygenerovaný tzv. token (žiaden slovenský ekvivalent sa nezaužíval)
- Každý token obsahuje:
  - Typ tokenu – napríklad všetky reťazce znakov slúžiace ako mená premenných alebo funkcií môžu mať typ tokenu „`Id`“
  - Atribúty – napríklad zodpovedajúci text (lexéma), počiatočný a koncový index v postupnosti vstupných symbolov, . . .
  - V ANTLR4 je každý token objekt; klasickejším prístupom je ale uloženie atribútov v podobe smerníka do tzv. tabuľky symbolov

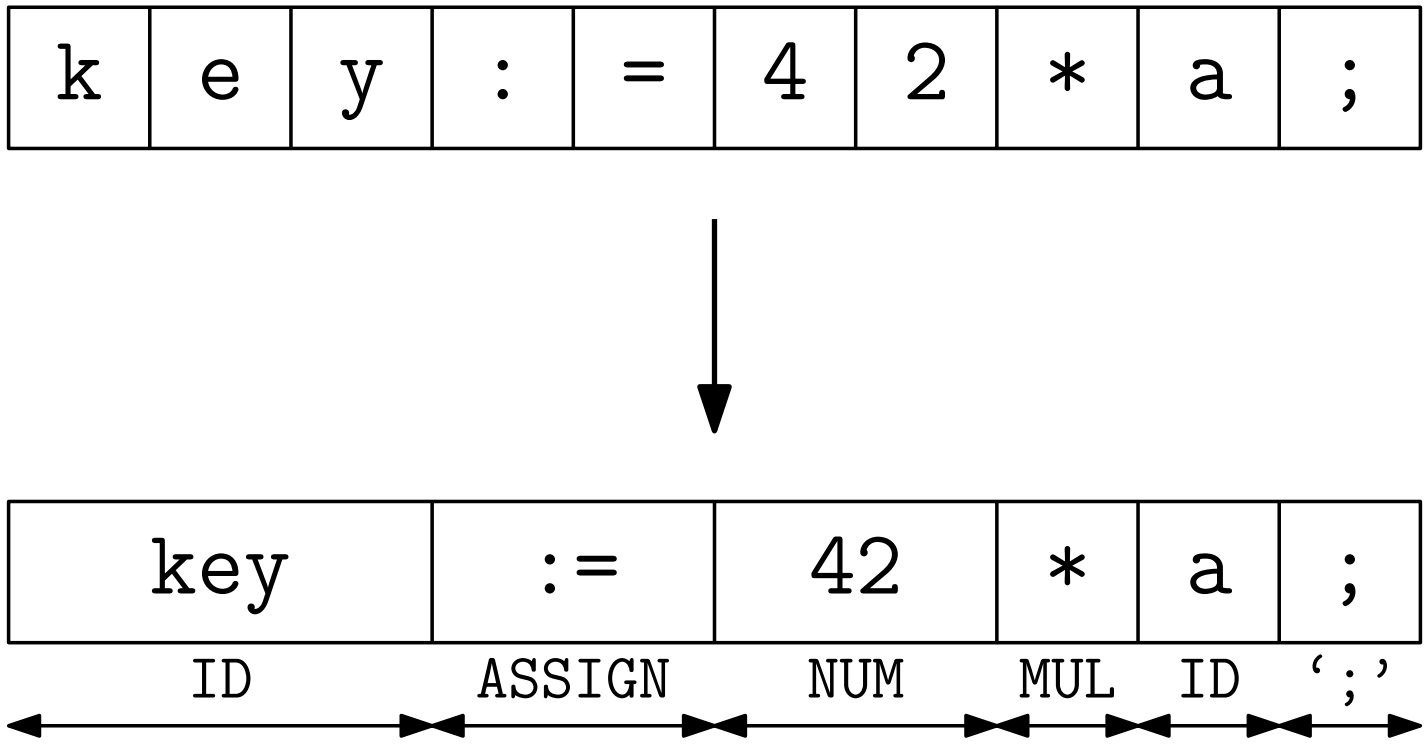
Úloha lexikálnej analýzy pri tvorbe kompilátorov



Úloha lexikálnej analýzy pri tvorbe kompilátorov



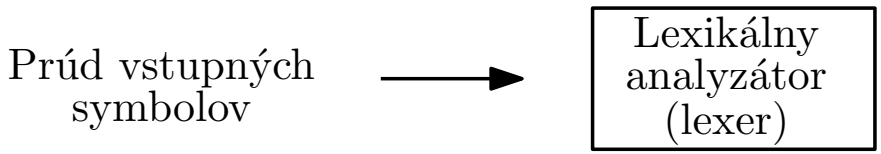
Úloha lexikálnej analýzy pri tvorbe kompilátorov



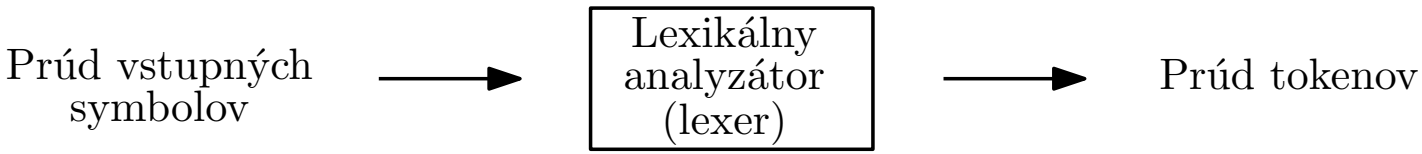
Úloha lexikálnej analýzy pri tvorbe kompilátorov

Prúd vstupných
symbolov

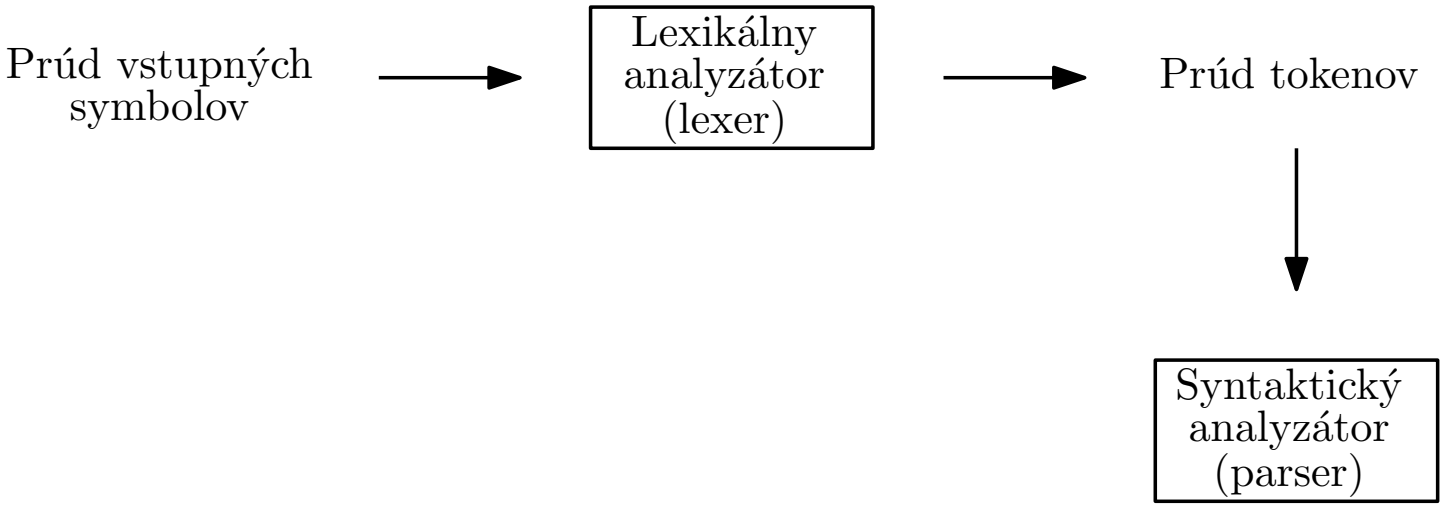
Úloha lexikálnej analýzy pri tvorbe kompilátorov



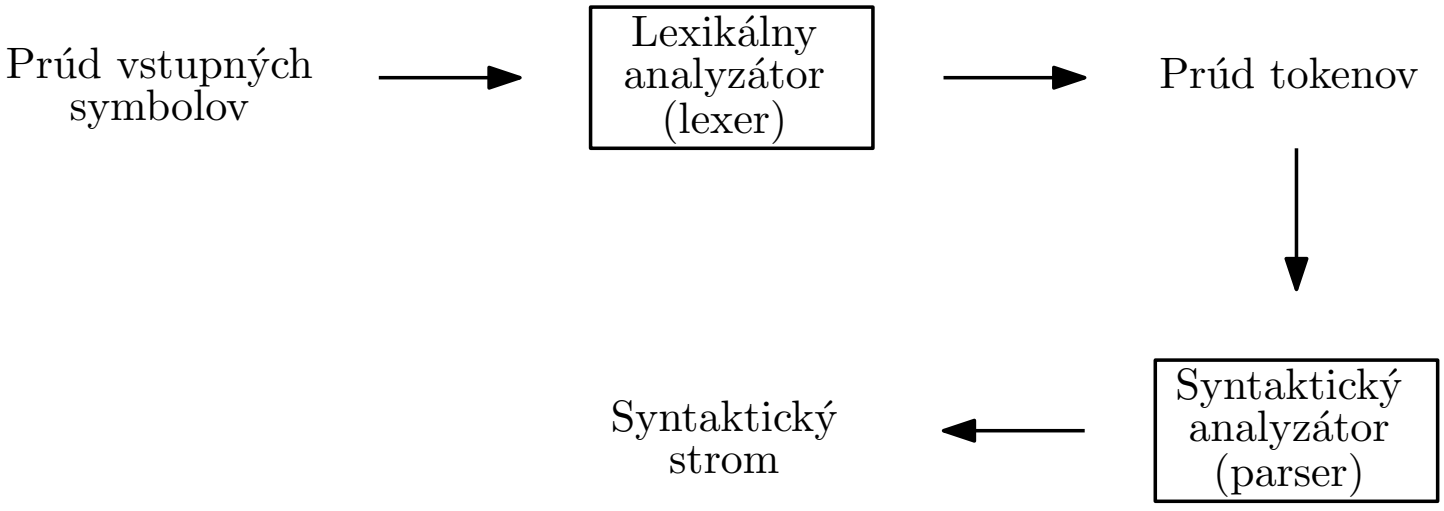
Úloha lexikálnej analýzy pri tvorbe kompilátorov



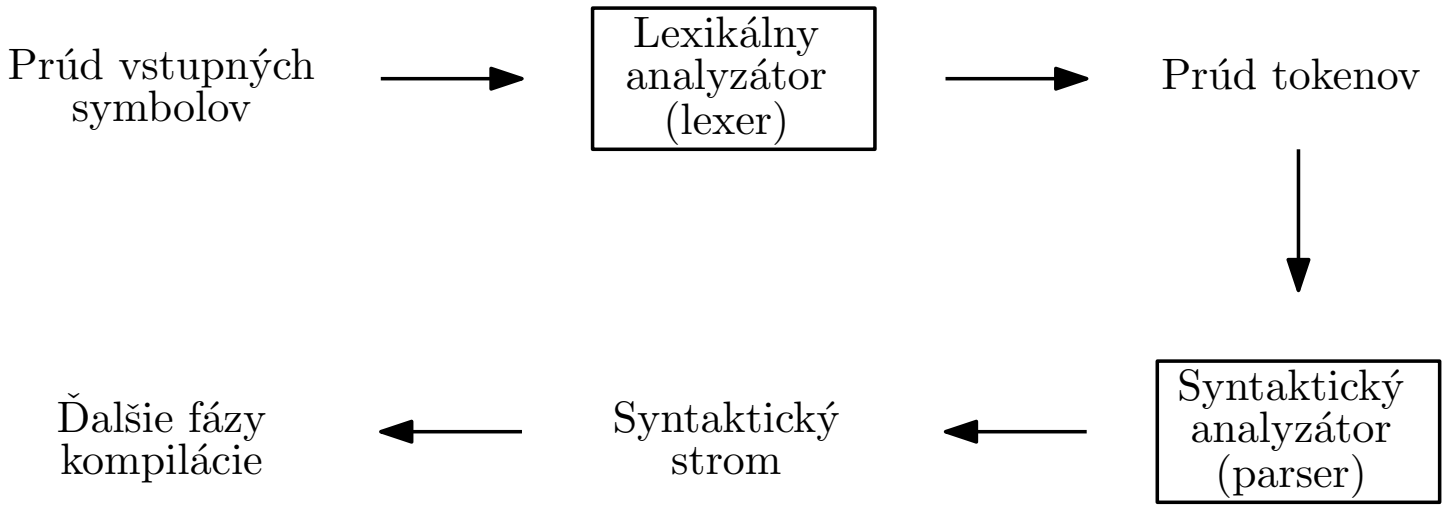
Úloha lexikálnej analýzy pri tvorbe kompilátorov



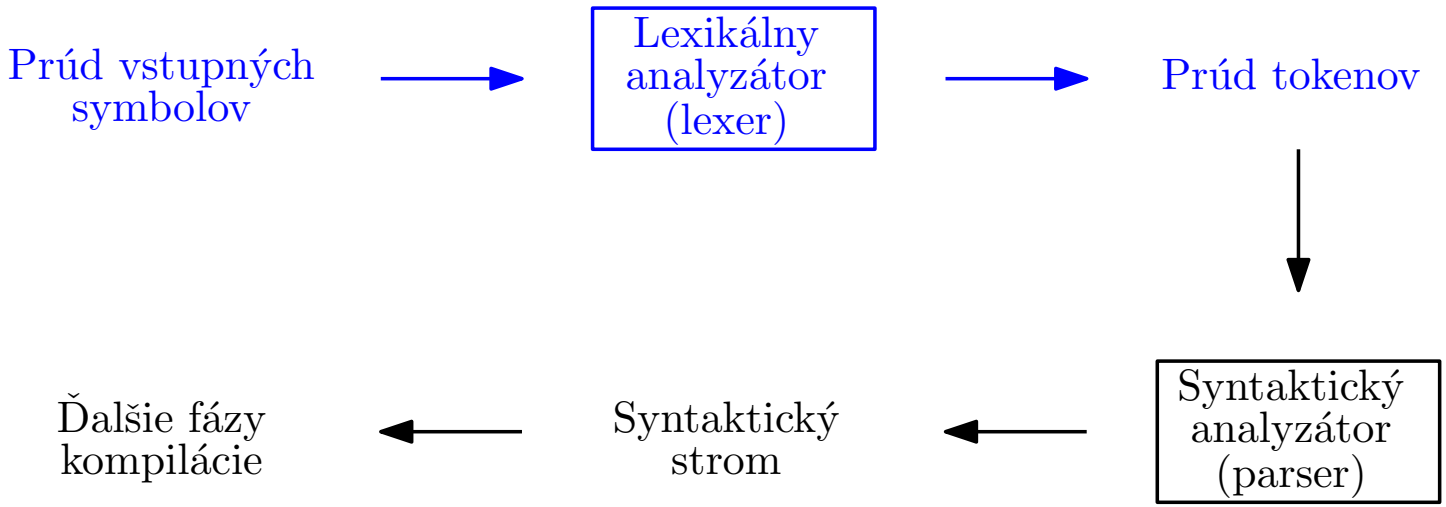
Úloha lexikálnej analýzy pri tvorbe kompilátorov



Úloha lexikálnej analýzy pri tvorbe kompilátorov



Úloha lexikálnej analýzy pri tvorbe kompilátorov



Lexikálna analýza pomocou konečných automatov

- Základná funkcia lexera: procedúra „vráť ďalší token“

Lexikálna analýza pomocou konečných automatov

- Základná funkcia lexera: procedúra „vráť ďalší token“
- Dá sa implementovať pomocou konečných automatov

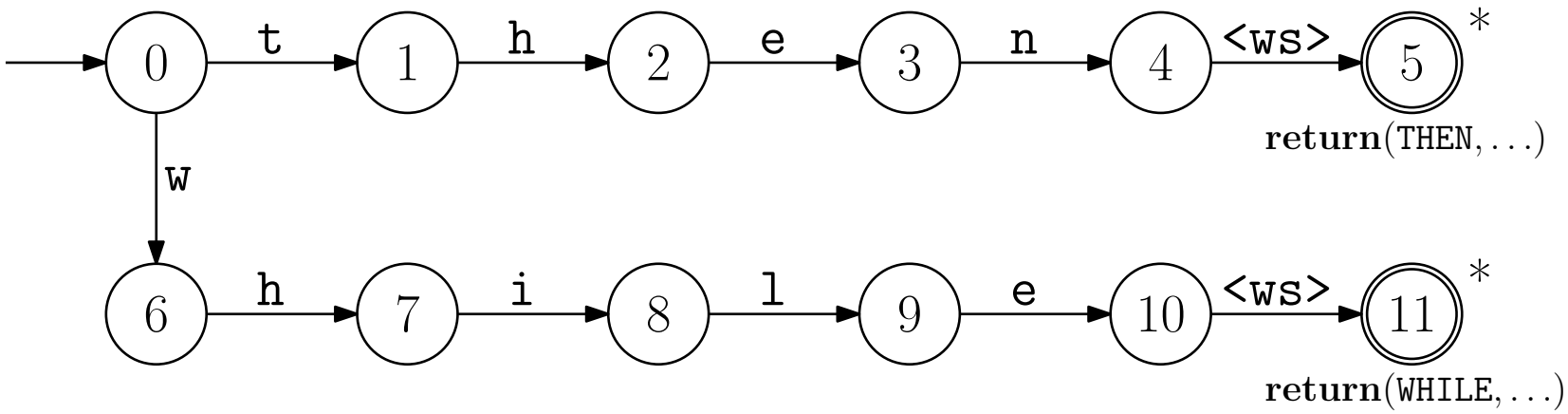
Lexikálna analýza pomocou konečných automatov

- Základná funkcia lexera: procedúra „vráť ďalší token“
- Dá sa implementovať pomocou konečných automatov
- Deterministické konečné automaty: simulácia

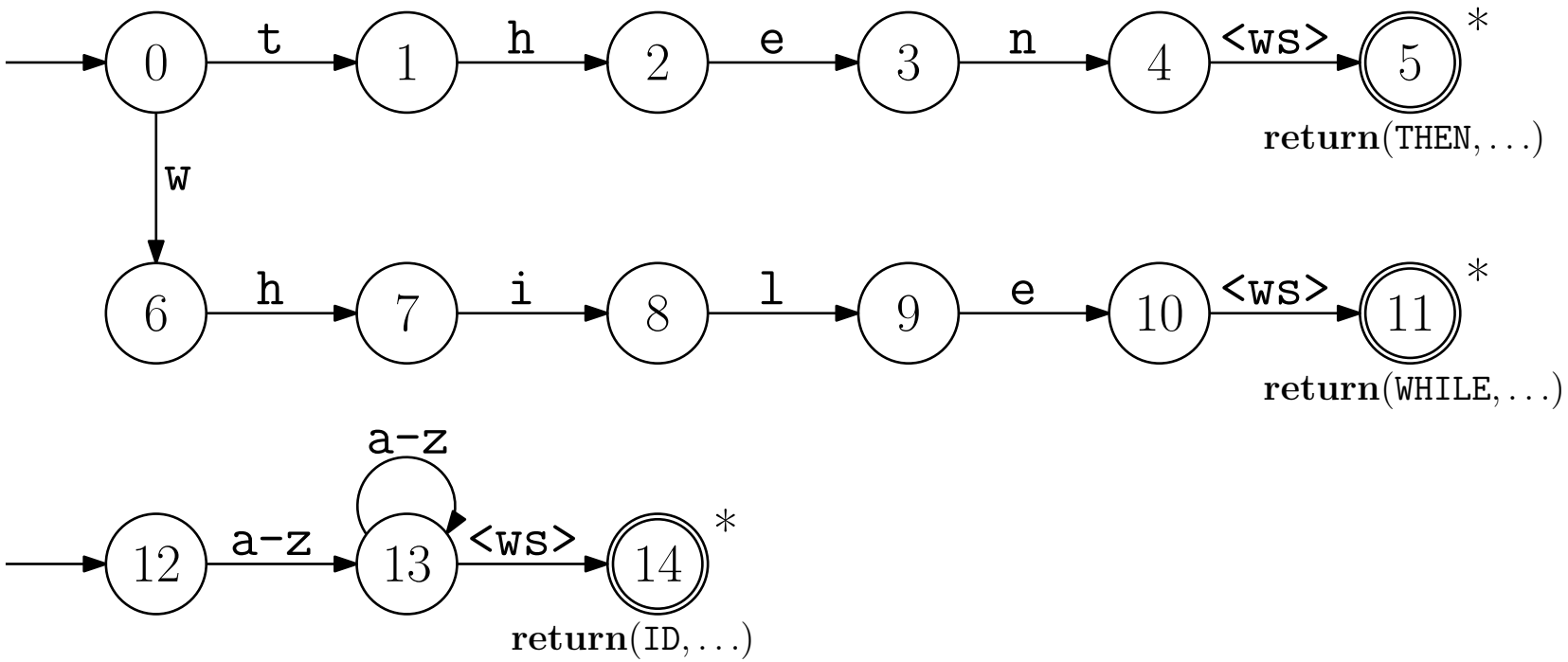
Lexikálna analýza pomocou konečných automatov

- Základná funkcia lexera: procedúra „vráť ďalší token“
- Dá sa implementovať pomocou konečných automatov
- Deterministické konečné automaty: simulácia
- Nedeterministické konečné automaty: prehľadávanie + priority

Lexikálna analýza pomocou konečných automatov



Lexikálna analýza pomocou konečných automatov



Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

- Procedúra „vráť ďalší token“ sa implementuje ručne v nejakom programovacom jazyku

Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

- Procedúra „vráť ďalší token“ sa implementuje ručne v nejakom programovacom jazyku
- Typicky môže ísť o nejakú implementáciu metódy využívajúcej konečné automaty

Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

- Procedúra „vráť ďalší token“ sa implementuje ručne v nejakom programovacom jazyku
- Typicky môže ísť o nejakú implementáciu metódy využívajúcej konečné automaty
- Výhody: pri dobrej implementácii efektívnosť výsledného kódu

Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

- Procedúra „vráť ďalší token“ sa implementuje ručne v nejakom programovacom jazyku
- Typicky môže ísť o nejakú implementáciu metódy využívajúcej konečné automaty
- Výhody: pri dobrej implementácii efektívnosť výsledného kódu
- Nevýhody: prácnosť, slabá prenositeľnosť, ...

Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

- Procedúra „vráť ďalší token“ sa implementuje ručne v nejakom programovacom jazyku
- Typicky môže ísť o nejakú implementáciu metódy využívajúcej konečné automaty
- Výhody: pri dobrej implementácii efektívnosť výsledného kódu
- Nevýhody: prácnosť, slabá prenositeľnosť, ...

Automatické generátory lexerov:

Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

- Procedúra „vráť ďalší token“ sa implementuje ručne v nejakom programovacom jazyku
- Typicky môže ísť o nejakú implementáciu metódy využívajúcej konečné automaty
- Výhody: pri dobrej implementácii efektívnosť výsledného kódu
- Nevýhody: prácnosť, slabá prenositeľnosť, ...

Automatické generátory lexerov:

- Nástroje, ktoré na základe nejakej špecifikácie lexikálnych pravidiel vygenerujú zdrojový kód lexera

Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

- Procedúra „vráť ďalší token“ sa implementuje ručne v nejakom programovacom jazyku
- Typicky môže ísť o nejakú implementáciu metódy využívajúcej konečné automaty
- Výhody: pri dobrej implementácii efektívnosť výsledného kódu
- Nevýhody: prácnosť, slabá prenositeľnosť, ...

Automatické generátory lexerov:

- Nástroje, ktoré na základe nejakej špecifikácie lexikálnych pravidiel vygenerujú zdrojový kód lexera
- Napríklad `Lex` a `ANTLR4`

Lexer: ručná implementácia vs. automatické generátory

Ručná implementácia lexera:

- Procedúra „vráť ďalší token“ sa implementuje ručne v nejakom programovacom jazyku
- Typicky môže ísť o nejakú implementáciu metódy využívajúcej konečné automaty
- Výhody: pri dobrej implementácii efektívnosť výsledného kódu
- Nevýhody: prácnosť, slabá prenositeľnosť, ...

Automatické generátory lexerov:

- Nástroje, ktoré na základe nejakej špecifikácie lexikálnych pravidiel vygenerujú zdrojový kód lexera
- Napríklad `Lex` a `ANTLR4`
- V súčasnosti preferovaná metóda

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Regulárne definície:

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Regulárne definície:

- Sady pravidiel $d_1 \rightarrow r_1, \ldots, d_n \rightarrow r_n$

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Regulárne definície:

- Sady pravidiel $d_1 \rightarrow r_1, \ldots, d_n \rightarrow r_n$
- r_1 je regulárny výraz nad nejakou abecedou Σ

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Regulárne definície:

- Sady pravidiel $d_1 \rightarrow r_1, \ldots, d_n \rightarrow r_n$
- r_1 je regulárny výraz nad nejakou abecedou Σ
- Pre $i = 1, \ldots, n$ je r_i regulárny výraz nad $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$ a d_i nepatrí do $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Regulárne definície:

- Sady pravidiel $d_1 \to r_1, \ldots, d_n \to r_n$
- r_1 je regulárny výraz nad nejakou abecedou Σ
- Pre $i = 1, \ldots, n$ je r_i regulárny výraz nad $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$ a d_i nepatrí do $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$

Lexikálne gramatiky na spôsob ANTLR:

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Regulárne definície:

- Sady pravidiel $d_1 \to r_1, \ldots, d_n \to r_n$
- r_1 je regulárny výraz nad nejakou abecedou Σ
- Pre $i = 1, \ldots, n$ je r_i regulárny výraz nad $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$ a d_i nepatrí do $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$

Lexikálne gramatiky na spôsob ANTLR:

- Technicky podobne silné ako bezkontextové gramatiky

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Regulárne definície:

- Sady pravidiel $d_1 \to r_1, \ldots, d_n \to r_n$
- r_1 je regulárny výraz nad nejakou abecedou Σ
- Pre $i = 1, \ldots, n$ je r_i regulárny výraz nad $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$ a d_i nepatrí do $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$

Lexikálne gramatiky na spôsob ANTLR:

- Technicky podobne silné ako bezkontextové gramatiky
- Zvyčajne sa ale používajú podobne ako regulárne definície

Špecifikácia lexikálnych pravidiel

Regulárne výrazy:

- Môžu byť transformované na NKA napríklad pomocou Thompsonovej konštrukcie
- Výsledný NKA môže byť transformovaný na DKA alebo priamo prehľadávaný
- Často sa povoľujú rozširujúce operácie ako napr. +, ?, atď.

Regulárne definície:

- Sady pravidiel $d_1 \to r_1, \ldots, d_n \to r_n$
- r_1 je regulárny výraz nad nejakou abecedou Σ
- Pre $i = 1, \ldots, n$ je r_i regulárny výraz nad $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$ a d_i nepatrí do $\Sigma \cup \{d_1, \ldots, d_{i-1}\}$

Lexikálne gramatiky na spôsob ANTLR:

- Technicky podobne silné ako bezkontextové gramatiky
- Zvyčajne sa ale používajú podobne ako regulárne definície
- Viac detailov o chvíľu

Časť 2:

Lexikálna analýza v ANTLR4

ANTLR: základné princípy

ANTLR: základné princípy

- Vstup: „gramatika“ v metajazyku ANTLR4

ANTLR: základné princípy

- Vstup: „gramatika“ v metajazyku ANTLR4
- Typicky ide o súbor s príponou `.g4` alebo `.g`

ANTLR: základné princípy

- Vstup: „gramatika“ v metajazyku ANTLR4
- Typicky ide o súbor s príponou `.g4` alebo `.g`

```
grammar Vyrazy;

expr
    :   expr op=(MUL|DIV) expr
    |   expr op=(ADD|SUB) expr
    |   '(' expr ')'
    |   INT
    ;

MUL
    :   '*'
    ;

DIV
    :   '/'
    ;

ADD
    :   '+'
    ;

SUB
    :   '-'
    ;

INT
    :   DIGIT+
    ;

WS
    :   [ \t\n\r]+   -> skip
    ;

fragment DIGIT
    :   [0-9]
    ;
```

ANTLR: základné princípy

- Vstup: „gramatika“ v metajazyku ANTLR4
- Typicky ide o súbor s príponou `.g4` alebo `.g`

```
1  lexer grammar Vyrazy;

3  LEFT
       :     '('
5      ;

7  RIGHT
       :     ')'
9      ;

11 MUL
       :     '*'
13     ;

15 DIV
       :     '/'
17     ;

   ADD
20     :     '+'
       ;
22
   SUB
24     :     '-'
       ;
26
   INT
28     :     DIGIT+
       ;
30
   WS
32     :     [ \t\n\r]+   -> skip
       ;
34
   fragment DIGIT
36     :     [0-9]
       ;
```

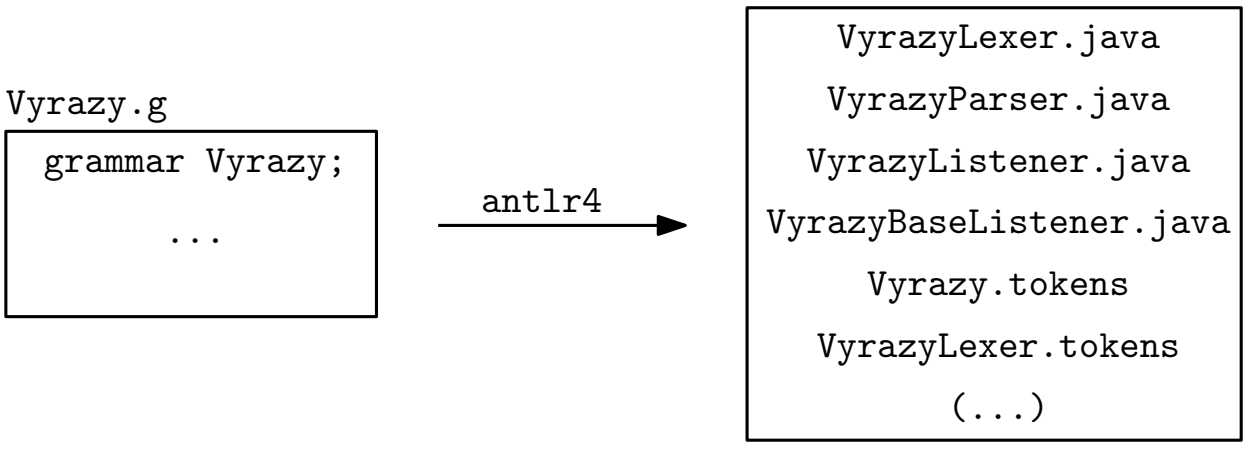
ANTLR: základné princípy

Vyrazy.g

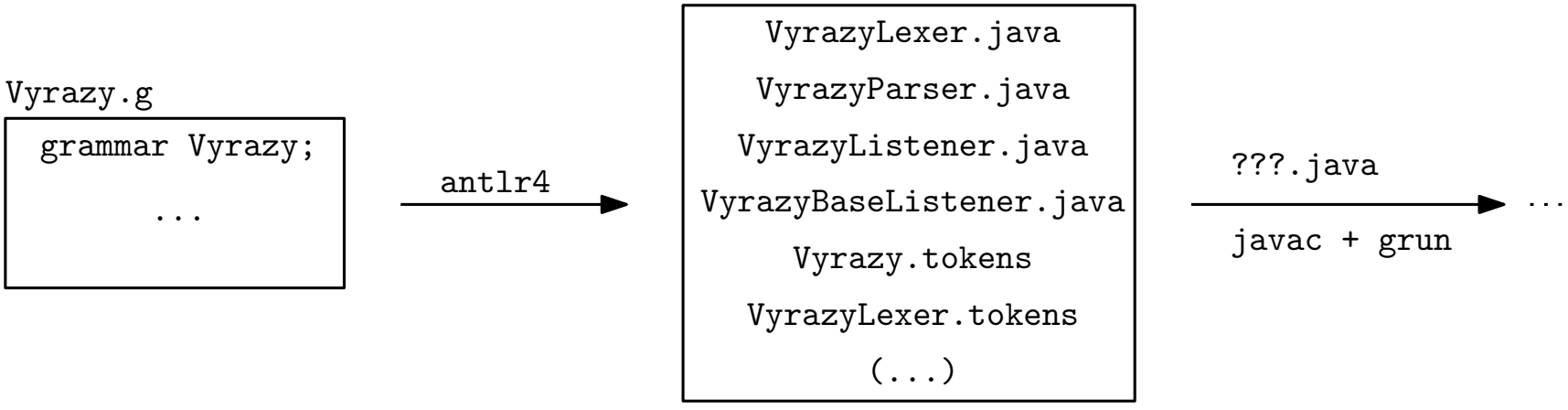
```
grammar Vyrazy;

    ...
```

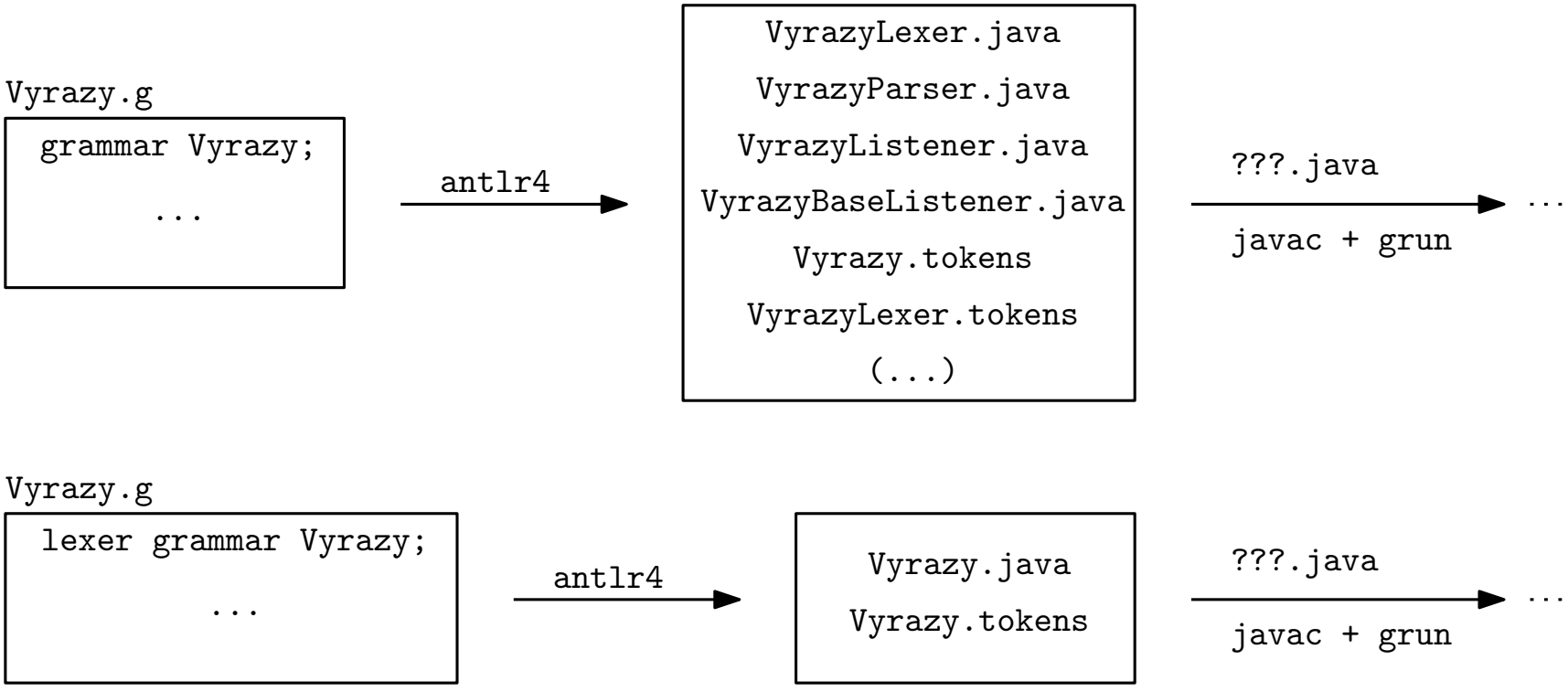
ANTLR: základné princípy



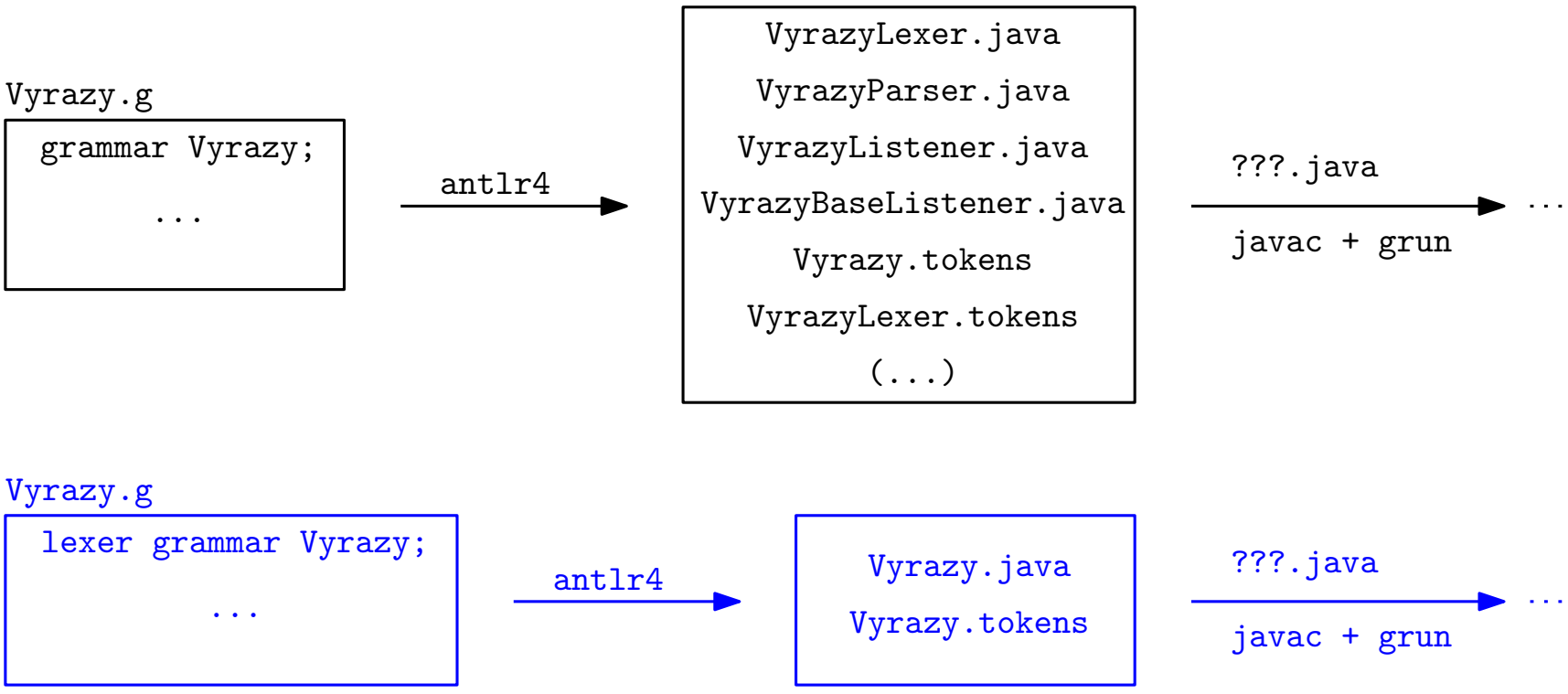
ANTLR: základné princípy



ANTLR: základné princípy



ANTLR: základné princípy



ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

- ANTLR4 Tool

ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

- ANTLR4 Tool
- ANTLR4 Runtime API

ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

- ANTLR4 Tool
- ANTLR4 Runtime API

Treba pritom rozlišovať medzi dvoma jazykmi:

ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

- ANTLR4 Tool
- ANTLR4 Runtime API

Treba pritom rozlišovať medzi dvoma jazykmi:

- Metajazyk ANTLR4 (špecifikácia gramatík)

ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

- ANTLR4 Tool
- ANTLR4 Runtime API

Treba pritom rozlišovať medzi dvoma jazykmi:

- Metajazyk ANTLR4 (špecifikácia gramatík)
- Cieľový jazyk (napr. Java) – nie úplne štandardná terminológia

ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

- ANTLR4 Tool
- ANTLR4 Runtime API

Treba pritom rozlišovať medzi dvoma jazykmi:

- Metajazyk ANTLR4 (špecifikácia gramatík)
- Cieľový jazyk (napr. Java) – nie úplne štandardná terminológia

Nová generácia generátorov parserov a lexerov:

ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

- ANTLR4 Tool
- ANTLR4 Runtime API

Treba pritom rozlišovať medzi dvoma jazykmi:

- Metajazyk ANTLR4 (špecifikácia gramatík)
- Cieľový jazyk (napr. Java) – nie úplne štandardná terminológia

Nová generácia generátorov parserov a lexerov:

- Návrhové kritériá sú často uprednostňované pred výkonnostnými

ANTLR: základné princípy

Dve najhlavnejšie súčasti ANTLR4:

- ANTLR4 Tool
- ANTLR4 Runtime API

Treba pritom rozlišovať medzi dvoma jazykmi:

- Metajazyk ANTLR4 (špecifikácia gramatík)
- Cieľový jazyk (napr. Java) – nie úplne štandardná terminológia

Nová generácia generátorov parserov a lexerov:

- Návrhové kritériá sú často uprednostňované pred výkonnostnými
- Generovanie parserov v ANTLR4: neskôr v priebehu semestra

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:
`https://github.com/antlr/antlr4/blob/master/doc/index.md`

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:
`https://github.com/antlr/antlr4/blob/master/doc/index.md`

Relevantné oddiely k dnešnému cvičeniu:

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:
`https://github.com/antlr/antlr4/blob/master/doc/index.md`

Relevantné oddiely k dnešnému cvičeniu:

- Getting Started with ANTLR v4

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:
`https://github.com/antlr/antlr4/blob/master/doc/index.md`

Relevantné oddiely k dnešnému cvičeniu:

- Getting Started with ANTLR v4
- Grammar Lexicon

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:
`https://github.com/antlr/antlr4/blob/master/doc/index.md`

Relevantné oddiely k dnešnému cvičeniu:

- Getting Started with ANTLR v4
- Grammar Lexicon
- Lexer Rules

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:
`https://github.com/antlr/antlr4/blob/master/doc/index.md`

Relevantné oddiely k dnešnému cvičeniu:

- Getting Started with ANTLR v4
- Grammar Lexicon
- Lexer Rules
- Wildcard Operator and Nongreedy Subrules

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:
`https://github.com/antlr/antlr4/blob/master/doc/index.md`

Relevantné oddiely k dnešnému cvičeniu:

- Getting Started with ANTLR v4
- Grammar Lexicon
- Lexer Rules
- Wildcard Operator and Nongreedy Subrules

Dokumentácia k Runtime API:

Dokumentácia k ANTLR

Všeobecná kvázi-dokumentácia k ANTLR:
`https://github.com/antlr/antlr4/blob/master/doc/index.md`

Relevantné oddiely k dnešnému cvičeniu:

- Getting Started with ANTLR v4
- Grammar Lexicon
- Lexer Rules
- Wildcard Operator and Nongreedy Subrules

Dokumentácia k Runtime API:
`http://www.antlr.org/api/Java/`

„Lexikálne gramatiky“ v ANTLR

Súbor `NazovGramatiky.g4`:

„Lexikálne gramatiky“ v ANTLR

Súbor `NazovGramatiky.g4`:

```
1  lexer grammar NazovGramatiky;

3  SpecialnySymbol
       :    '-'                    // Komentar
5      |    '_'
       ;
7
   Specialita
9      :    ('(' | '[') SpecialnySymbol (')' | ']')
       ;
11
   Slovo
13     :    [a-zA-Z]+
       ;
15
   Cislo
17     :    ('0'..'9')+
       ;
19
   WS
21     :    [ \t\n\r]+
       ;
```

„Lexikálne gramatiky“ v ANTLR

Súbor `NazovGramatiky.g4`:

```
1  lexer grammar NazovGramatiky;

3  SpecialnySymbol
       :    '-'                    // Komentar
5      |    '_'
       ;
7
   Specialita
9      :    ('(' | '[') SpecialnySymbol (')' | ']')
       ;
11
   Slovo
13     :    [a-zA-Z]+
       ;
15
   Cislo
17     :    ('0'..'9')+
       ;
19
   WS
21     :    [ \t\n\r]+
       ;
```

- Technicky sú podobne silné ako bezkontextové gramatiky, odporúča sa ale pracovať s nimi ako s definíciami regulárnych jazykov

Použitie ANTLR4 a nástroja TestRig

```
1 antlr4 NazovGramatiky.g4
2 javac *.java
3 grun NazovGramatiky tokens -tokens
```

Použitie ANTLR4 a nástroja TestRig

```
1 antlr4 NazovGramatiky.g4
2 javac *.java
3 grun NazovGramatiky tokens -tokens
```

- Nástroj TextRig po tomto volaní vypíše postupnosť tokenov zadanú na štandardný vstup

Použitie ANTLR4 a nástroja TestRig

```
1 antlr4 NazovGramatiky.g4
2 javac *.java
3 grun NazovGramatiky tokens -tokens
```

- Nástroj TextRig po tomto volaní vypíše postupnosť tokenov zadanú na štandardný vstup
- Prvý parameter pre grun je vždy názov gramatiky

Použitie ANTLR4 a nástroja TestRig

```
1 antlr4 NazovGramatiky.g4
2 javac *.java
3 grun NazovGramatiky tokens -tokens
```

- Nástroj `TextRig` po tomto volaní vypíše postupnosť tokenov zadanú na štandardný vstup
- Prvý parameter pre `grun` je vždy názov gramatiky
- Druhý parameter je pri lexikálnych gramatikách vždy „`tokens`“; pri kombinovaných lexikálno-syntaktických gramatikách je to počiatočný neterminál

Použitie ANTLR4 a nástroja TestRig

```
1 antlr4 NazovGramatiky.g4
2 javac *.java
3 grun NazovGramatiky tokens -tokens
```

- Nástroj `TextRig` po tomto volaní vypíše postupnosť tokenov zadanú na štandardný vstup
- Prvý parameter pre `grun` je vždy názov gramatiky
- Druhý parameter je pri lexikálnych gramatikách vždy „`tokens`“; pri kombinovaných lexikálno-syntaktických gramatikách je to počiatočný neterminál
- Je možné čítať aj zo súboru – jeho názov je potom ďalší parameter

Použitie ANTLR4 a nástroja TestRig

```
1 antlr4 NazovGramatiky.g4
2 javac *.java
3 grun NazovGramatiky tokens -tokens
```

- Nástroj `TextRig` po tomto volaní vypíše postupnosť tokenov zadanú na štandardný vstup
- Prvý parameter pre `grun` je vždy názov gramatiky
- Druhý parameter je pri lexikálnych gramatikách vždy „`tokens`“; pri kombinovaných lexikálno-syntaktických gramatikách je to počiatočný neterminál
- Je možné čítať aj zo súboru – jeho názov je potom ďalší parameter
- EOF je špeciálny token (na štandardnom vstupe ho treba zadať ako `Ctrl+Z` resp. `Ctrl+D` v závislosti od systému)

Príkazy pre lexer

V našom príklade lexikálnej gramatiky sme mali aj nasledujúce pravidlo:

Príkazy pre lexer

V našom príklade lexikálnej gramatiky sme mali aj nasledujúce pravidlo:

```
20 WS
        :       [ \t\n\r]+
22      ;
```

Príkazy pre lexer

V našom príklade lexikálnej gramatiky sme mali aj nasledujúce pravidlo:

```
20 WS
        :       [ \t\n\r]+
22      ;
```

- Je vysoká pravdepodobnosť, že tokeny `WS` budeme ignorovať

Príkazy pre lexer

V našom príklade lexikálnej gramatiky sme mali aj nasledujúce pravidlo:

```
20 WS
          :      [ \t\n\r]+
22        ;
```

- Je vysoká pravdepodobnosť, že tokeny `WS` budeme ignorovať

ANTLR4 umožňuje zadávať do gramatiky príkazy pre lexer

Príkazy pre lexer

V našom príklade lexikálnej gramatiky sme mali aj nasledujúce pravidlo:

```
20 WS
       :      [ \t\n\r]+
22     ;
```

- Je vysoká pravdepodobnosť, že tokeny `WS` budeme ignorovať

ANTLR4 umožňuje zadávať do gramatiky príkazy pre lexer

- Jedným z nich je príkaz `skip`

Príkazy pre lexer

V našom príklade lexikálnej gramatiky sme mali aj nasledujúce pravidlo:

```
20 WS
       :     [ \t\n\r]+
22     ;
```

- Je vysoká pravdepodobnosť, že tokeny `WS` budeme ignorovať

ANTLR4 umožňuje zadávať do gramatiky príkazy pre lexer

- Jedným z nich je príkaz `skip`

```
20 WS
       :     [ \t\n\r]+   -> skip
22     ;
```

Priorita lexikálnych pravidiel

Nasledujúci token na vstupe (vrátený vygenerovaným lexerom) možno popísať ako token popisovaný pravidlom získaným nasledovne:

Priorita lexikálnych pravidiel

Nasledujúci token na vstupe (vrátený vygenerovaným lexerom) možno popísať ako token popisovaný pravidlom získaným nasledovne:

1. Nájdi všetky pravidlá, ktoré popisujú nejaký prefix zostávajúceho vstupu

Priorita lexikálnych pravidiel

Nasledujúci token na vstupe (vrátený vygenerovaným lexerom) možno popísať ako token popisovaný pravidlom získaným nasledovne:

1. Nájdi všetky pravidlá, ktoré popisujú nejaký prefix zostávajúceho vstupu
2. Z nich vyber tie pravidlá, ktoré popisujú najdlhší prefix spomedzi všetkých pravidiel

Priorita lexikálnych pravidiel

Nasledujúci token na vstupe (vrátený vygenerovaným lexerom) možno popísať ako token popisovaný pravidlom získaným nasledovne:

1. Nájdi všetky pravidlá, ktoré popisujú nejaký prefix zostávajúceho vstupu
2. Z nich vyber tie pravidlá, ktoré popisujú najdlhší prefix spomedzi všetkých pravidiel
3. Z nich vyber pravidlo, ktoré je v súbore s gramatikou definované ako prvé

„Pažravé a striedme podpravidlá“

Nájdenie prefixu vstupu popisovaného daným pravidlom:

„Pažravé a striedme podpravidlá“

Nájdenie prefixu vstupu popisovaného daným pravidlom:

- Zvyčajne sa aplikuje „pažravá“ (greedy) stratégia

„Pažravé a striedme podpravidlá“

Nájdenie prefixu vstupu popisovaného daným pravidlom:

- Zvyčajne sa aplikuje „pažravá“ (greedy) stratégia
- Napríklad `'(' .* ')'` (kde „`.`“ je takzvaná „divoká karta“, čiže ľubovoľný symbol) vždy dočíta zvyšok vstupu začínajúci s „`(`“ až po jeho koniec

„Pažravé a striedme podpravidlá“

Nájdenie prefixu vstupu popisovaného daným pravidlom:

- Zvyčajne sa aplikuje „pažravá“ (greedy) stratégia
- Napríklad `'(' .* ')'` (kde „`.`“ je takzvaná „divoká karta“, čiže ľubovoľný symbol) vždy dočíta zvyšok vstupu začínajúci s „`(`“ až po jeho koniec
- Najväčší problém toto správanie predstavuje pri pravidlách typu `(...)+`, `(...)*` a `(...)?`

„Pažravé a striedme podpravidlá“

Nájdenie prefixu vstupu popisovaného daným pravidlom:

- Zvyčajne sa aplikuje „pažravá“ (greedy) stratégia
- Napríklad `'(' .* ')'` (kde „`.`“ je takzvaná „divoká karta“, čiže ľubovoľný symbol) vždy dočíta zvyšok vstupu začínajúci s „`(`“ až po jeho koniec
- Najväčší problém toto správanie predstavuje pri pravidlách typu `(...)+`, `(...)*` a `(...)?`
- Pri týchto pravidlách sa možno prepnúť do „striedmeho“ (nongreedy) módu použitím ďalšieho operátora „`?`“

„Pažravé a striedme podpravidlá“

Nájdenie prefixu vstupu popisovaného daným pravidlom:

- Zvyčajne sa aplikuje „pažravá“ (greedy) stratégia
- Napríklad `'(' .* ')'` (kde „`.`“ je takzvaná „divoká karta“, čiže ľubovoľný symbol) vždy dočíta zvyšok vstupu začínajúci s „`(`“ až po jeho koniec
- Najväčší problém toto správanie predstavuje pri pravidlách typu `(...)+`, `(...)*` a `(...)?`
- Pri týchto pravidlách sa možno prepnúť do „striedmeho“ (nongreedy) módu použitím ďalšieho operátora „`?`“
- Napríklad `'(' .*? ')'` bude čítať vstup len po prvý výskyt „`)`“

„Pažravé a striedme podpravidlá“

Nájdenie prefixu vstupu popisovaného daným pravidlom:

- Zvyčajne sa aplikuje „pažravá“ (greedy) stratégia
- Napríklad `'(' .* ')'` (kde „`.`“ je takzvaná „divoká karta“, čiže ľubovoľný symbol) vždy dočíta zvyšok vstupu začínajúci s „`(`“ až po jeho koniec
- Najväčší problém toto správanie predstavuje pri pravidlách typu `(...)+`, `(...)*` a `(...)?`
- Pri týchto pravidlách sa možno prepnúť do „striedmeho“ (nongreedy) módu použitím ďalšieho operátora „`?`“
- Napríklad `'(' .*? ')'` bude čítať vstup len po prvý výskyt „`)`“
- „Striedme“ podpravidlá ako napr. `.*?` sa „snažia popísať“ minimálny počet znakov tak, aby okolité pravidlo popisovalo nejaký prefix vstupu

„Pažravé a striedme podpravidlá“

Nájdenie prefixu vstupu popisovaného daným pravidlom:

- Zvyčajne sa aplikuje „pažravá“ (greedy) stratégia
- Napríklad `'(' .* ')'` (kde „`.`“ je takzvaná „divoká karta“, čiže ľubovoľný symbol) vždy dočíta zvyšok vstupu začínajúci s „`(`“ až po jeho koniec
- Najväčší problém toto správanie predstavuje pri pravidlách typu `(...)+`, `(...)*` a `(...)?`
- Pri týchto pravidlách sa možno prepnúť do „striedmeho“ (nongreedy) módu použitím ďalšieho operátora „`?`“
- Napríklad `'(' .*? ')'` bude čítať vstup len po prvý výskyt „`)`“
- „Striedme“ podpravidlá ako napr. `.*?` sa „snažia popísať“ minimálny počet znakov tak, aby okolité pravidlo popisovalo nejaký prefix vstupu
- Zvyšok pravidla nasledujúci za „striedmym“ podpravidlom sa vyhodnocuje inak, než zvyčajne (viď dokumentáciu)

Použitie vygenerovaného lexera

Po volaní `antlr4 Gramatika.g4`:

Použitie vygenerovaného lexera

Po volaní `antlr4 Gramatika.g4`:

- ANTLR4 vygeneruje lexer ako triedu `Gramatika.java`

Použitie vygenerovaného lexera

Po volaní `antlr4 Gramatika.g4`:

- ANTLR4 vygeneruje lexer ako triedu `Gramatika.java`
- Pri kombinovaných lexikálno-syntaktických gramatikách má názov `GramatikaLexer.java`

Použitie vygenerovaného lexera

Po volaní `antlr4 Gramatika.g4`:

- ANTLR4 vygeneruje lexer ako triedu `Gramatika.java`
- Pri kombinovaných lexikálno-syntaktických gramatikách má názov `GramatikaLexer.java`

```
1  import org.antlr.v4.runtime.*;
   import org.antlr.v4.runtime.tree.*;
3  import java.io.*;
   import java.util.*;
5
   public class HlavnaTrieda {
7
       public static void main(String[] args) throws Exception {
9          InputStream is = System.in;
           CharStream input = CharStreams.fromStream(is);
11         Gramatika lexer = new Gramatika(input);
           CommonTokenStream tokens = new CommonTokenStream(lexer);
13
           tokens.fill();
15     }
   }
17
```

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

- `public List<Token> getTokens()`

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

- `public List<Token> getTokens()`

Trieda `Token` obsahuje napríklad metódy:

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

- `public List<Token> getTokens()`

Trieda `Token` obsahuje napríklad metódy:

- `String getText()`

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

- `public List<Token> getTokens()`

Trieda `Token` obsahuje napríklad metódy:

- `String getText()`
- `int getType()`

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

- `public List<Token> getTokens()`

Trieda `Token` obsahuje napríklad metódy:

- `String getText()`
- `int getType()`

Trieda `Gramatika` obsahuje okrem iného (zdedenú) metódu:

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

- `public List<Token> getTokens()`

Trieda `Token` obsahuje napríklad metódy:

- `String getText()`
- `int getType()`

Trieda `Gramatika` obsahuje okrem iného (zdedenú) metódu:

- `public Vocabulary getVocabulary()`

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

- `public List<Token> getTokens()`

Trieda `Token` obsahuje napríklad metódy:

- `String getText()`
- `int getType()`

Trieda `Gramatika` obsahuje okrem iného (zdedenú) metódu:

- `public Vocabulary getVocabulary()`

Rozhranie `Vocabulary` definuje napríklad metódu:

Použitie vygenerovaného lexera

Trieda `CommonTokenStream` obsahuje napríklad metódu:

- `public List<Token> getTokens()`

Trieda `Token` obsahuje napríklad metódy:

- `String getText()`
- `int getType()`

Trieda `Gramatika` obsahuje okrem iného (zdedenú) metódu:

- `public Vocabulary getVocabulary()`

Rozhranie `Vocabulary` definuje napríklad metódu:

- `String getSymbolicName(int tokenType)`