Metody Programowania

Przetwarzanie tekstu w języku Java – wprowadzenie do ANTLR

23 wrzesnia 2017

Motywacja

$$p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$$

- Jak wczytać taki napis?
- Jak go reprezentować w pamięci?
- Jak go zinterpretować?

Język formalny

Definicja: język formalny

Język formalny jest zbiorem ciągów symboli (znaków) zwanych *słowami*. Przykładami języków formalnych są napisy i wyrażenia, które wczytujemy i przetwarzamy w programach. Do opisu języka używamy gramatyki.

Gramatyka formalna

Na potrzeby wykładu zdefiniujmy gramatykę następująco:

Definicja: gramatyka bezkontekstowa

Gramatykę bezkontekstową określamy za pomocą:

1. skończonego alfabetu symboli zawierającego *symbol pusty*,
2. skończonego zbioru *zmiennych* (rozłącznego z powyższym),
3. wyróżnionej *zmiennej początkowej*,
4. skończonego zbioru reguł o postaci
   $A \rightarrow B$
   gdzie A jest zmienną, a B jest napisem składającym się ze zmiennych i symboli z alfabetu.

Gramatyka rachunku zdań

Niech alfabet symboli to zbiór $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow, p, q\}$,
a zbiór zmiennych to: $\{S\}$

Zbiór reguł:
(1) $S ::= p$
(2) $S ::= q$
(3) $S ::= \neg S$
(4) $S ::= S \vee S$
(5) $S ::= S \wedge S$
(6) $S ::= S \rightarrow S$
(7) $S ::= S \leftrightarrow S$

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$S \rightarrow S$

(6)

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$p \rightarrow S$

$(6) \rightarrow (1)$

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$p \rightarrow S \leftrightarrow S$

$(6) \rightarrow (1) \rightarrow (7)$

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$p \rightarrow q \leftrightarrow S$

$(6) \rightarrow (1) \rightarrow (7) \rightarrow (2)$

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$p \rightarrow q \leftrightarrow S \rightarrow S$

$(6) \rightarrow (1) \rightarrow (7) \rightarrow (2) \rightarrow (6)$

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$p \rightarrow q \leftrightarrow \neg S \rightarrow S$

$(6) \rightarrow (1) \rightarrow (7) \rightarrow (2) \rightarrow (6) \rightarrow (3)$

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$p \rightarrow q \leftrightarrow \neg p \rightarrow S$

$(6) \rightarrow (1) \rightarrow (7) \rightarrow (2) \rightarrow (6) \rightarrow (3) \rightarrow (1)$

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$p \rightarrow q \leftrightarrow \neg p \rightarrow \neg S$

$(6) \rightarrow (1) \rightarrow (7) \rightarrow (2) \rightarrow (6) \rightarrow (3) \rightarrow (1) \rightarrow (3)$

Wyprowadzenie

Ciąg reguł prowadzący od zmiennej początkowej do pewnego słowa nazwywamy wyprowadzeniem.

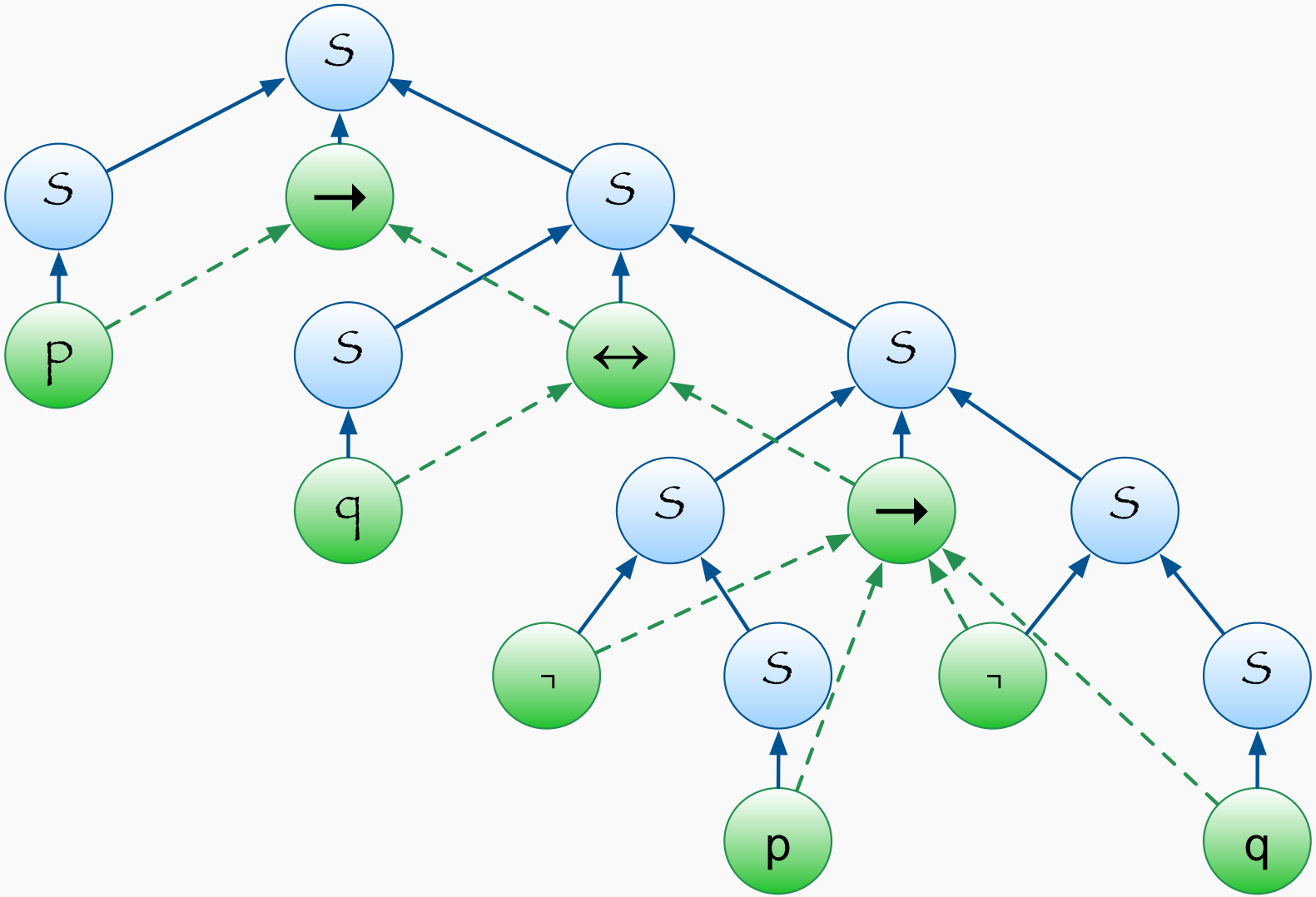
Jedno z możliwych wyprowadzeń dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$:

$p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$

$(6) \rightarrow (1) \rightarrow (7) \rightarrow (2) \rightarrow (6) \rightarrow (3) \rightarrow (1) \rightarrow (3) \rightarrow (2)$

Drzewo wyprowadzenia

Kolejne reguły w wyprowadzeniu tworzą drzewo wyprowadzenia.



Poprawiona gramatyka rachunku zdań

Poprzednia gramatyka nie jest jednoznaczna - istnieje więcej niż jedno wyprowadzenie dla danego słowa. Ponadto nie uwzględnia kolejności wyliczania formuł.

Przyjmujemy następujące priorytety:

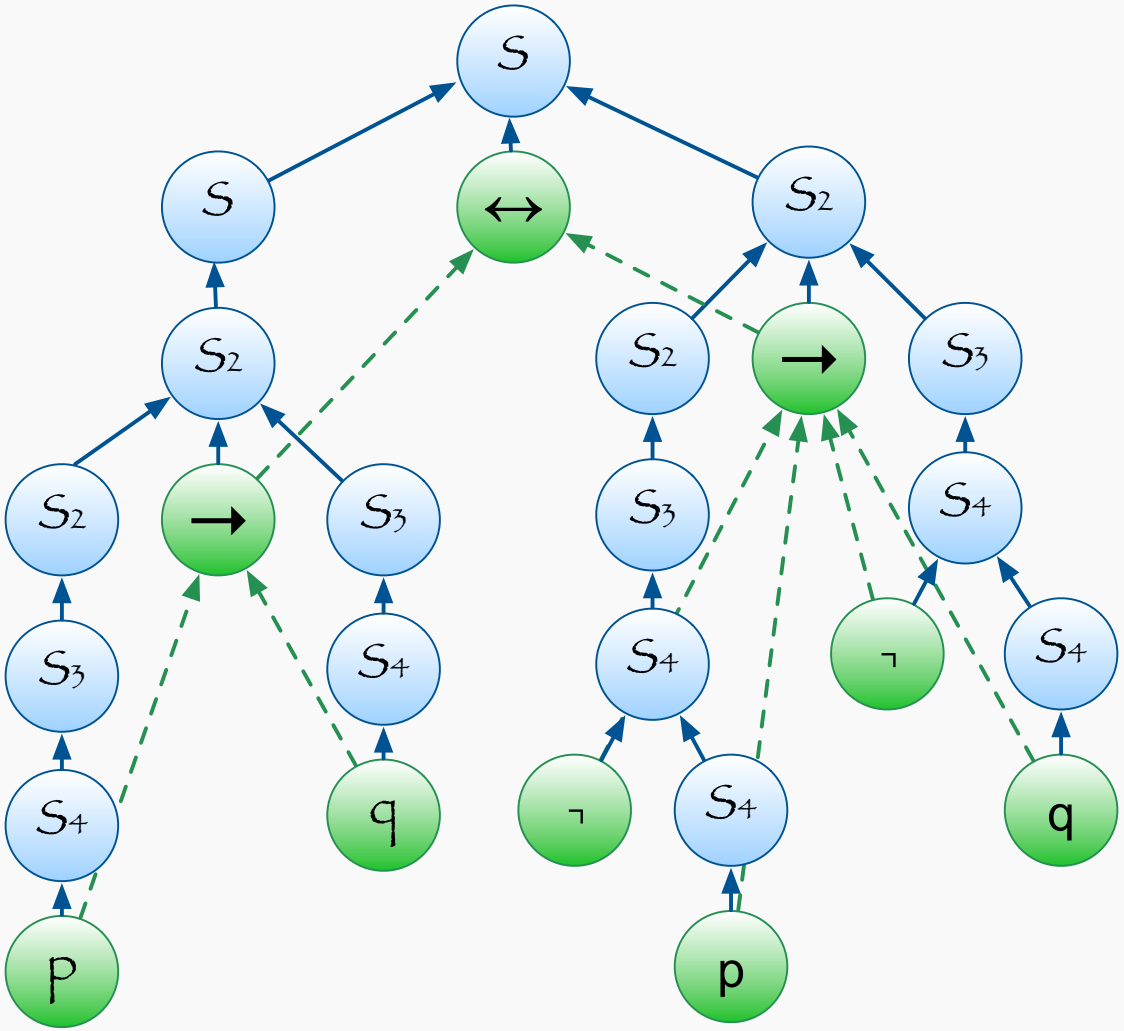
1. $\neg$
2. $\vee$ i $\wedge$
3. $\rightarrow$
4. $\leftrightarrow$

Poprawiona gramatyka rachunku zdań

Aby nasza gramatyka odzwierciedlała priorytety w rachunku zdań musimy wprowadzić dodatkowe zmienne: S_2, S_3 i S_4.
Albabet symboli to zbiór: $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow, p, q\}$
Zbiór zmiennych to: $\{S, S_2, S_3, S_4\}$
Zbiór reguł:
(1) $S_4 ::= p$
(2) $S_4 ::= q$
(3) $S_4 ::= \neg S_4$
(4) $S_4 ::= (S)$
(5) $S_3 ::= S_4$
(6) $S_3 ::= S_3 \vee S_4$
(7) $S_3 ::= S_3 \wedge S_4$
(8) $S_2 ::= S_3$
(9) $S_2 ::= S_2 \rightarrow S_3$
(10) $S ::= S_2$
(11) $S ::= S \leftrightarrow S_2$

Drzewo wyprowadzenia z poprawioną gramatyką

Teraz drzewo wyprowadzenia dla $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$ będzie wyglądało następująco:



Analizator leksykalny

Definicja: analizator leksykalny (ang. *lexer*)

Analizator leksykalny to moduł, ktory przetwarza symbole z wejścia i grupuje je w słowa, które są najmniejszymi elementami składni języka.

Analizator leksykalny podzieli $p \rightarrow q \leftrightarrow \neg p \rightarrow \neg q$ na następujące słowa:

$\boxed{p}$ $\boxed{\rightarrow}$ $\boxed{q}$ $\boxed{\leftrightarrow}$ $\boxed{\neg}$ $\boxed{p}$ $\boxed{\rightarrow}$ $\boxed{\neg}$ $\boxed{q}$

Parser

Definicja: parser

Parser (inaczej analizator syntaktyczny) to moduł, który generuje drzewo wyprowadzenia. Na wejściu otrzymuje słowa, które są wynikiem pracy analizatora leksykalnego. Parser jest budowany za pomocą generatora parserów, któremu dajemy na wejściu gramatykę języka, który ma być rozpoznawany przez wynikowy analizator syntaktyczny.

ANTLR

- ANother Tool for Language Recognition
- Generator analizatorow leksykalnych i parserów.
- Wejściem jest gramatyka.
- Wyjściem jest analizator leksykalny i parser w wybranym języku programowania (Java, C, C#, Python...)
- Używany przez: Twitter, Netbeans, IntelliJ Idea

Struktura gramatyki dla ANTLR

```
grammar GrammarName;

options {...}
import ... ;
tokens {...}
@<actionName >{...}

rule_1
...
rule_n
```

Struktura gramatyki dla ANTLR

```
options {language=Java; superClass=MySuperClass;
         tokenVocab=MyLexer.tokens;}
```

- *language* – język, w którym ANTLR ma wygenerować analizator leksykalny i parser.
- *superClass* – klasa, po której ma dziedziczyć główna klasa parsera.
- *tokenVocab* – słownik ze słowami z innego analizatora leksykalnego.

Struktura gramatyki dla ANTLR

```
import GrammarPart1;
import GrammarPart2;
```

- Służy do importowania innych gramatyk.
- Pozwala na dzielenie gramatyki na części, które można użyć w innych gramatykach.
- Importowane są jedynie te reguły, które nie są zdefiniowane w gramatyce importującej.

Struktura gramatyki dla ANTLR

```
tokens { IF, THEN, ELSE, FOR, WHILE }
```

- Służy do zdefiniowania nowych typów słów, dla których nie mamy reguł, np. słowa kluczowe.
- Pozwala na uniknięcie ostrzeżeń przy generowaniu (implicit definition of token X in parser').
- Najczęściej używamy tej instrukcji dla słów, które chcemy obsłużyc w specjalny sposób.

Struktura gramatyki dla ANTLR

Akcje:

- Dla parserów generowanych do Javy zaimplementowane są 2 akcje:
  - @*header*{...} - umożliwia dodanie kodu na początku pliku z klasą główną do analizatora leksykalnego i/lub parsera
  - @*member*{...} - umożliwia dodanie pól i metod do klasy głównej analizatora leksykalnego i/lub parsera
- Domyślnie akcje są wykonywane zarówno dla głównej klasy analizatora leksykalnego i parsera.
- Prefiks lexer:: ogranicza akcje do analizatora leksykalnego, np.: @*lexer* :: *header*{...}.
- Prefiks parser:: ogranicza akcje do parsera, np.: @*parser* :: *member*{...}.

Struktura gramatyki dla ANTLR

Reguły gramatyki:

```
formula_3 : fomula_4
          | formula 'or' formula
          | formula 'and' formula
          ;
```

Regułom możemy nadawać etykiety:

```
formula_3 : fomula_4                  # F4
          | formula 'or' formula      # Or
          | formula 'and' formula     # And
          ;
```

Jak używać ANTLR?

PropCalc.g4 – nasza etykietowana gramatyka ruchunku zdań.
Generujemy analizator leksykalny i parser używając polecenia:

```
> antlr4 -visitor PropCalc.g4
```

W naszym programie uzyskujemy drzewo wyprowadzenia przez następujące instrukcje:

```
PropCalcLexer lexer = new PropCalcLexer(input);
CommonTokenStream tokens = new CommonTokenStream(lexer);
PropCalcParser parser = new PropCalcParser(tokens);
ParseTree tree = parser.formula(); // parse
```

ANTLR - w praktyce

Kod dostępny pod adresem:

- `https://bitbucket.org/pawlokrz/pjwstk-metody-programowania/src`
  - Folder ANTLR – gramatyka
  - Folder PropositionalCalculus – kod programu