Portable Lexical Analysis for Parsing of Morphologically-Rich Languages

Marek Medveď

Natural Language Processing Centre Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic

6.12.2013

Lexical analysis

- before syntactic analysis we have to provide a lexical analysis of given language

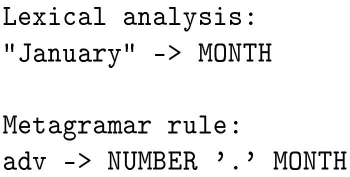
Lexical analysis

- before syntactic analysis we have to provide a lexical analysis of given language
- based on word, lemma, tag, word index the lexical analysis is able to assign a pre-terminal to each word

Lexical analysis

- before syntactic analysis we have to provide a lexical analysis of given language
- based on word, lemma, tag, word index the lexical analysis is able to assign a pre-terminal to each word
- the pre-terminal is then used in formal grammar of system `Synt`

Lexical analysis



```
Lexical analysis:
"January" -> MONTH

Metagramar rule:
adv -> NUMBER '.' MONTH
```

Figure: The basis of lexical analysis

Lexical analysis

- up to now the lexical analysis was hard-coded in C module and simulates decision tree operating on word, lemma and tag

Lexical analysis

- up to now the lexical analysis was hard-coded in C module and simulates decision tree operating on word, lemma and tag
- this approach has several disadvantages:

Lexical analysis

- up to now the lexical analysis was hard-coded in C module and simulates decision tree operating on word, lemma and tag
- this approach has several disadvantages:
  - needed programing skills

Lexical analysis

- up to now the lexical analysis was hard-coded in C module and simulates decision tree operating on word, lemma and tag
- this approach has several disadvantages:
    - needed programing skills
    - wide character strings

Lexical analysis

- up to now the lexical analysis was hard-coded in C module and simulates decision tree operating on word, lemma and tag
- this approach has several disadvantages:
  - needed programing skills
  - wide character strings
  - decision tree handling

Lexical analysis

- up to now the lexical analysis was hard-coded in C module and simulates decision tree operating on word, lemma and tag
- this approach has several disadvantages:
  - needed programing skills
  - wide character strings
  - decision tree handling
- we decide to replace current lexical analysis with more affable form

Lexical analysis

- up to now the lexical analysis was hard-coded in C module and simulates decision tree operating on word, lemma and tag
- this approach has several disadvantages:
  - needed programing skills
  - wide character strings
  - decision tree handling
- we decide to replace current lexical analysis with more affable form
- the new form consists of word, lemma, tag, word index and corresponding pre-terminal for word, lemma, tag and word index

Lexical analysis

- we tried to exploit tree tools for automatic generation lexical scanners: the lex, flex and re2c

Lexical analysis

- we tried to exploit tree tools for automatic generation lexical scanners: the lex, flex and re2c
- all thee of them crates a lexical analyzer from defined lexical rules

Lexical analysis

- we tried to exploit tree tools for automatic generation lexical scanners: the lex, flex and re2c
- all thee of them crates a lexical analyzer from defined lexical rules
- lexical rule: regular expression + action (in C language)

Lexical analysis

- usage of Unicode characters

Lexical analysis

- usage of Unicode characters
- better support of Unicode in re2c and flex, we ruled out the lex

Lexical analysis

- usage of Unicode characters
- better support of Unicode in re2c and flex, we ruled out the lex
- the input for re2c and flex is lexical rule and macro definition

Lexical analysis

```
def scan(string){
------ macro part ------
    IS_NUMBER = [0-9]+;
         ...
------ rule  part ------
    .*"\t".*"\t"{IS_NUMBER} {
------   action   ------
          preterm=NUM;
          }
         ...
 }

 def analyze(input_str){
     scan(input_str);
 }
```

Figure: Input for re2c and flex

Lexical analysis

- output of both tools is final automata in C code

Lexical analysis

- output of both tools is final automata in C code
- at the end we picked out much faster re2c tool

preprocessing script

- the Synt can be used for many languages

preprocessing script

- the `Synt` can be used for many languages
- the adaptation required programming skills, that lots of language specialists don't have

preprocessing script

- the `Synt` can be used for many languages
- the adaptation required programming skills, that lots of language specialists don't have
- we create a preprocessing script that converts an input mapping definition file into a re2c source file

preprocessing script

- the `Synt` can be used for many languages
- the adaptation required programming skills, that lots of language specialists don't have
- we create a preprocessing script that converts an input mapping definition file into a re2c source file
- definition file is plain text, that contains lexical rules: tag, lemma, word, word index and pre-terminal

Macro

- the macro format is following:
  `m=NameOfMacro RegularExpression`

Macro

- the macro format is following:
  `m=NameOfMacro RegularExpression`
- the NameOfMacro is then used in lexical rules and re2c substitute the NameOfMacro with RegularExpression in final scanner

Macro

- the macro format is following:
  `m=NameOfMacro RegularExpression`
- the NameOfMacro is then used in lexical rules and re2c substitute the NameOfMacro with RegularExpression in final scanner
- comments starts with #= character

Macro

- the macro format is following: `m=NameOfMacro RegularExpression`
- the NameOfMacro is then used in lexical rules and re2c substitute the NameOfMacro with RegularExpression in final scanner
- comments starts with #= character
- definition file supports top macro comments and line macro comments

Macro

```
#=top macro comment
m=IS_NUMBER     [0-9]+    #=line macro comment
```

Figure: Definition of macro with user comments

Lexical rule

- lexical rule contains of two parts: the first describe the pattern we want to match and the second part defines the action that provide after the first part is successfully matched

Lexical rule

- lexical rule contains of two parts: the first describe the pattern we want to match and the second part defines the action that provide after the first part is successfully matched
- the first part of lexical rule consists of tag, lemma, word, word index which are regular expressions with predefined macros

Lexical rule

- lexical rule contains of two parts: the first describe the pattern we want to match and the second part defines the action that provide after the first part is successfully matched
- the first part of lexical rule consists of tag, lemma, word, word index which are regular expressions with predefined macros
- the action contains pre-terminal that have to be assigned for this structure

Lexical rule

- lexical rule contains of two parts: the first describe the pattern we want to match and the second part defines the action that provide after the first part is successfully matched
- the first part of lexical rule consists of tag, lemma, word, word index which are regular expressions with predefined macros
- the action contains pre-terminal that have to be assigned for this structure
- the comments is same as in macro definition

Lexical rule

```
#=lexica rule top comment
.*   {IS_MONTH}   .*  WI   preterm=MONTH #=comment
```

Figure: Lexical rule with user comments

Lexical rule

- in second part (the action) the user can use predefined variables word, lemma, morf_info,word_index and lemma_index.

```
"k1".*  .*  {FU} WI  if(word_index>0){preterm=NPR;}
```

Figure: Lexical rule with predefined variable

Evaluation

Tool	Time of generating scanner
`flex`	22.9 min
`re2c`	1.4 min

Evaluation (6160 sentences in brief format)

Lexical analyzer	Analysis time
former `Synt` lexer	3.33 min
`re2c`	3.19 min

Thank you for attention.