Portable Lexical Analysis for Parsing of Morphologically-Rich Languages

Marek Medveď and Miloš Jakubíček

Natural Language Processing Centre
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
{xmedved1,jak}@fi.muni.cz

Abstract. In this paper we present new approach to lexical analysis in the `Synt` parser. We describe three fast lexical analyzers we have exploited for lexical analysis and advantages of the `re2c` fast lexical analyzer in comparison to others. This paper shows a new lexical analysis workflow which is both easy to maintain and portable to new languages. Finally we provide an evaluation of the new lexical analysis against the original lexical analysis.

Key words: lexical analysis, synt parser

1 Introduction

Any parser that is not fully lexicalized (i.e. operates on some – fine or coarse grained – categories of words) must deal with the issue of performing the lexical analysis – mapping of words into the respective categories – before it can actually perform any parsing. For analytical languages the mapping procedure may be very simple and may boil down to the mapping to part-of-speech categories, however for wide-coverage parsers dealing with morphologically-rich languages, this is not the case. For those it holds that precise and complex (fine-grained) lexical analysis is the starting point for a successful syntactic analysis.

As with any hand-written rule system, one has to decide what formalism is optimal for designing and developing the lexical analysis. We hereby list the requirements for both of these based on our past experience – the formalism for development of the lexical analysis should be:

- **simple** to edit and maintain for language specialists
- easily **portable** to new languages
- **expressive** enough so as to be appropriate for the task

As for the actual execution of the lexical analysis (i.e. evaluation of the mappings on particular input), there is a single straightforward condition in place: it must be fast enough so as not to harm the speed of the parser.

Aleš Horák, Pavel Rychlý (Eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2013, pp. 21–26, 2013.

In this paper we present a new approach to lexical analysis in the Czech parser Synt[1,2] that satisfies the conditions listed above. At first we describe the former lexical analysis in the parser and its deficiencies, then we discuss possible alternatives that we exploited and finally we describe in detail the new approach based on a re2c-generated lexical scanner.

2 The Synt parser

Syntactic analyzer Synt has been developed over the past years in the Natural Language Processing Centre at Faculty of Informatics, Masaryk University. The Synt parser is based on a context-free backbone enhanced with contextual actions and performs a stochastic agenda-based head-driven chart analysis.

The input of Synt is a sentence morphologically annotated by the morphological analyzer Ajka[3] which uses an attributive tagset described in [4][1].

The lexical analysis follows immediately after loading an input sentence and its morphological annotation. In the context of a phrase-structure grammar, the lexical analysis provides mapping from words to the so called *pre-terminals* – non-terminal leaves which are directly rewritten to the surface word in the resulting syntactic tree. Therefore correct assignment of a word's pre-terminal is crucial for the analysis to succeed.

After the lexical analysis, the Synt parser proceeds with two-step parsing: first with a basic context-free analysis of the sentence and then with evaluation of complex contextual actions finally producing one of the possible outputs, which might be:

- **a phrase-structure tree**
  This is the main output of Synt and consists of a set of phrase-structure trees ordered according to the tree ranking. The system makes it possible to retrieve n-best trees effectively.
- **a dependency graph**
  A dependency graph represents a packed structure which can be utilized to extract all possible dependency trees. It is created by using the head and dependency markers that might be tied with each rule in the grammar.
- **set of syntactic structures**
  The input sentence is decomposed into a set of unambiguous syntactic structures chosen by the user.

3 Lexical analysis

In the current Czech grammar used in Synt the lexical analysis exploited the following token properties to be able to assign the pre-terminal correctly: the **word** itself, its **lemma** and its **morphological tag**, mainly the part-of-speech including its subclassification as provided by the Majka morphological analyser.

[1] Current version of the tagset is available online at http://nlp.fi.muni.cz/ma/.

3.1 Tools

The former implementation was based on a hard-coded C module that simulated a decision tree operating on the three features (word, lemma, tag) used for the task. This approach, while being very efficient during the execution of the lexical analysis, suffered from several obvious drawbacks, the biggest one being that good knowledge of the system and programming skills were required in order to be able to develop the lexical analysis module, which made it practically impossible for a linguist specialist to do the task. Another, rather more engineering complications, included the necessity for special handling of wide-character strings and the fact that manual maintenance of a hand-written decision tree is quite error-prone.

For all these reasons we decided to replace the current lexical analysis module with a better formalism without the deficiencies listed above. We concluded that from the user perspective the mapping should be maintained "as is" in the most natural form consisting of lines mapping the word, lemma, tag triple to the pre-terminal. For this task, we tried to exploit three available tools for automatic generation of lexical scanners/analyzers – the lex[2], flex[3] and re2c[4].

A lexical scanner is a program which recognizes lexical patterns in text and after it matches a lexical pattern it executes an associated action. Input for all three tools is a description of the scanner in the form of pairs of regular expressions and C code actions, called rules. The output is then the generated scanner in the form of a C source file.

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

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

Fig. 1: Principle of lexical analysis

From these three tools we first ruled out the `lex` because of its limited support for Unicode character set (wide/multibyte character strings). From the remaining two, `flex` and `re2c`, the latter one turned out to be much faster and hence our implementation is based on that one.

[2] Current version of lex is available online at `http://dinosaur.compilertools.net/lex/`.

[3] Current version of flex is available online at `http://flex.sourceforge.net/`.

[4] Current version of re2c is available online at `http://re2c.org/`.

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

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

Fig. 2: Input for `re2c` and flex

3.2 New formalism for lexical analysis

The `Synt` parser can be used for many languages. The problem is that lot of language specialists (linguists) have basic or no skills in programing. Because of that we decided to create a preprocessing script that converts an input mapping definition file into a `re2c` source file. The definition file is a simple plain text file containing lines with lexical rules mapping a tag-, lemma- and word-triple to the respective pre-terminal.

Apart of the mapping lines, the definition file may contain comments and macros that are expanded in the mapping lines. Comment lines start with `#=`, macro lines take the format `m=Macro RegularExpresion` (replace any `Macro` instances with `RegularExpression` in the mapping lines (see example in Figure 3).

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

Fig. 3: Macro example with user comments

The lexical rule (exemplified in Figure 4) contains two parts: the first part describes the pattern to be matched and the second part is action to be executed upon the first part is successfully matched. The first part of the lexical rule contains tag, lemma and word in this order. The tag, lemma and word are regular expressions where predefined macros can be used, finally the 4^{th} field (second part of the lexical rule) is the action prescribing the pre-terminal to

be assigned (though it might be a short C snippet too). All the fields are tab-separated.

```
#=top comment
.*   {IS_MONTH}   .*   preterm=MONTH #=line comment
```

Fig. 4: Lexical rule example with user comments

In the lexical rules and macros predefined variables for word, lemma, tag, word index (denoted as `wi`) and lemma index (denoted as `li`) might be used – word, lemma and tag variables contain the respective input fields, the word (lemma) index is an integer representing word (lemma) position in the input sentence. For example if one wants to write a lexical rule for words that do not occur at the first position of the sentence, the following lexical rule might be appropriate:

```
"k1".*  .*  {ONLY_FIRST_UPPER} if(wi>0){preterm=NPR;}
```

Fig. 5: Lexical rule with predefined variables

4 Evaluation

Hereby we provide both a comparison of `flex` and `re2c` scanner generation and the actual speed of running the lexical analysis on a sample set of 6,160 sentences (see Table 1). From this evaluation it follows that the new lexical analysis system is not only easier to maintain and more portable, but also slightly faster than its hard-coded predecessor.

Table 1: Time of generating scanner from description of scanner

Lexical analyzer	Time of scanner generation
`flex`	22.9 min
`re2c`	2.5 min

Time of syntactic analysis with old and new lexical analysis

Lexical analyzer	Lexical analysis time
original `Synt` lexer	3.33 min
`re2c`	3.19 min

5 Conclusions

In this paper we have presented new lexical analysis used in the Czech parser `Synt`. We have discussed in detail the motivation behind this work: to have a portable, easy to maintain and fast to evaluate formalism for mapping input tokens to grammatical pre-terminals. We are convinced that the new approach based on the `re2c` lexical scanner will help us to port the parser to new languages faster and will be also less error-prone.

Acknowledgements This work has been partly supported by the Ministry of Education of CR within the LINDAT-Clarin project LM2010013.

References

1. Kadlec, V.: Syntactic analysis of natural languages based on context-free grammar backbone. PhD thesis, Fakulta informatiky, Masarykova univerzita, Brno (2007)
2. Jakubíček, M., Horák, A., Kovář, V.: Mining phrases from syntactic analysis. In: Text, Speech and Dialogue. (2009) 124–130
3. Šmerk, P.: Fast Morphological Analysis of Czech. In: Proceedings of the Raslan Workshop 2009, Brno (2009)
4. Jakubíček, M., Kovář, V., Šmerk, P.: Czech Morphological Tagset Revisited. Proceedings of Recent Advances in Slavonic Natural Language Processing 2011 (2011) 29–42