Using tgrep2 on XML

Hanne Moa

February 2, 2005

1 Introduction

As part of the LOGON project (Lønning et al., 2004), I have access to a digitized version of the *Engelsk stor ordbok* (Eek et al., 2001), a bilingual dictionary of English and Norwegian, encoded in XML. It will be neccessary to extract much from this dictionary, but unfortunately the nature of XML and the size[1] of the dictionary would make looking through the dictionary by hand a truly herculean task. Therefore, efficient and easy to use tools to search in and extract information from XML are needed.

In this paper I demonstrate how XML can be searched by *TGrep2* (Rohde, 2004) by conversion to trees encoded as s-expressions (McCarthy, 1960).

2 XML and how to search it

It is possible to use existing text-search tools like *grep* on XML, but these are generally unsuitable as they often are meant to search unstructrured text, while with XML the structure itself is content information that needs to be searchable.

One way of extracting bits and pieces of XML directly is to use XSLT (Clark et al., 1999). However, XSLT is complex and verbose, as it uses mostly functions and not syntax to do its searching, making it less useful for quick exploration from the command line[2].

2.1 XML as a tree

XML is, in essence, just a way of encoding a tree[3], as figure 1 illustrates.

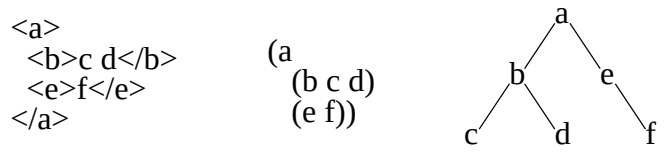

Figure 1: Equivalent trees: from left to right the same tree is encoded in XML, as s-expressions and visually.

[1] A single volume with over 200 000 words and 500 000 translations.
[2] Its predecessor DSSSL has much the same drawbacks, and has largely been phased out.
[3] With cross-references one can also encode more complex graphs.

One tool that makes the search in and extraction of trees easily possible is tgrep2, a follow-up to tgrep (tree-grep), that has for years been used to search in the Penn treebank. Tgrep2 works on trees encoded as s-expressions, see e.g. the middle figure of figure 1, and not directly on trees encoded in XML. S-expressions differ from XML in that subtrees are enclosed by parentheses instead of tags and the first word/token after the opening parenthesis is the root of the subtree, furthermore, s-expressions only has subtrees and hence no attributes.

2.2 XML as s-expressions

SXML (Kiselyov, 2002) at `http://ssax.sourceforge.net` is a full reimplementation of XML as s-expressions, however, it seemed to be overkill in this instance. Therefore it was decided to make a simpler alternative to just convert to and from generic XML, as described in this paper.

3 Converting between XML and s-expressions

3.1 From XML to s-expressions

This was quickily done with XSLT. Listing 1 shows the resulting generic stylesheet, which is small enough that adjustments "in the field" are possible.

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="text"/>
4  <xsl:strip-space elements="*"/>

6  <xsl:template match="text()">
   (LEAF <xsl:value-of select="normalize-space(.)"/>)</xsl:template>
8
   <xsl:template match="*">(<xsl:value-of select="local-name()"/>
10 <xsl:if test="@*">(@ <xsl:for-each select="@*">
   (<xsl:value-of select="local-name(.)"/><xsl:text> </xsl:text><xsl:value-of select="."/>)
12 </xsl:for-each>)</xsl:if>
   <xsl:apply-templates/>)
14 </xsl:template>

16 </xsl:stylesheet>
```

Listing 1: XSLT-stylesheet to convert from XML to s-expressions. Line numbers are included as a convenience to the reader.

Running the stylesheet in listing 1 on the example XML in listing 2 (page 7) produces the s-expressions in listing 3 (page 7), sans line numbers and indentation.

The stylesheet in listing 4 on page 8 has been tailored for use on the XML dictionary-example of listing 2, and produces the s-expressions of listing 5, sans line numbers and indentation.

3.2 From s-expressions to XML

For completeness' sake, a tool to go from s-expressions to XML was needed.

I made a standalone python program to serve this function, implemented by a simple finite state transducer with the addition of a stack for the xml-tags. The source for this is included in appendix B on page 9.

4 Usage and tips

To help illustrate usage and tips, the made-up example in listing 2 on page 7 will be used, an abstraction of the format used in *Engelsk stor ordbok*.

Conversion-results are in listings 3 on page 7 and 5 on page 9. The first is the result of using the XSLT stylesheet unchanged while the second is the result after applying most of the tips that follows.

4.1 Preparing the XML for tgrep2

Tgrep2 *will* consider all opening parentheses to branch off a new subtree. If preventing this is relevant, it becomes necessary to replace all parentheses in the XML by e.g. square brackets, for instance by using the standard search&replace-command of a text-editor.

Line 19 in the example XML in listing 2 is a case in point, compare lines 45 and 46 in listing 3 with line 31 in listing 5.

4.2 Selecting the trees we want to convert

An XML-file generally includes only a single tree and not several (aka. a forest), and therefore just a single root-node. In the example, that node is <dictionary>.

Now, we won't be needing this <dictionary>-node in any of our tgrep2-trees as we will be concentrating on the <entry>-nodes, so by adjusting line 10 in the XSLT from

```
<xsl:if test="@*">(@ <xsl:for-each select="@*">
```

to

```
<xsl:template match="dictionary//*">(<xsl:value-of select="local-name()" />
```

only the descendants of <dictionary> but not <dictionary> itself is included in our tgrep2-able trees. Compare lines 1 to 3 in listing 3 with line 1 in listing 5.

4.3 Ignoring attributes

None of the attributes of the example are of use so they will be stripped from the trees.

This can be accomplished by deleting[4] the lines 10 to 12 inclusive from listing 1 and no attributes will be used in the trees. Otherwise, the attributes of a node will be the first child subtree of the mother node, see figure 2. To see the effect of doing this, compare lines 3 to 5 in listing 3 with line 1 in listing 5.

```
<node attr1="1" attr2="2">  ⟺  (node (@ (attr1 1) (attr2 2)))
```

Figure 2: Conversion of attributes

[4] Alternatively: comment out by prepending <! − − and postpending − − >

4.4 What to do with free text

Free text, or #PCDATA in XML-jargon, is content without explicit XML structure; text-strings that are neither a tag nor an attribute. XML-relatives like HTML or MathML varies in how much and where they allow #PCDATA, if any.

#PCDATA cannot have daughters so I have decided to make them stand out by turning them into a subtree of sisters with the tag `LEAF` as mother, thus making them look different from other, non-#PCDATA nodes that happen to not branch.

By changing line 7 of listing 1 from

```
(LEAF <xsl:value-of select="normalize-space(.)"/>)</xsl:template>
```

to

```
<xsl:value-of select="normalize-space(.)"/></xsl:template>
```

the #PCDATA become daughters of their logical mother node instead. Compare line 4 in listing 2 and lines 6 and 7 in listng 3 with figure 1, which does not use `LEAF`.

4.5 Converting from XML

To use an XSLT-stylesheet to convert XML to something else, an XSLT-processor is necessary, and there are several available. Well-known opensource alternatives include *Saxon* and *Xalan*, but I use *xsltproc* from the GNOME project.

Simply running `xsltproc stylesheet xmlfile` will write the transformed xml to standard out, where it can be redirected to a file:

```
xsltproc stylesheet xmlfile > tgrep2ablefile
```

Then, it will be necessary to make the tgrep2-database from the s-expressions, by running

```
tgrep2 -p tgrep2ablefile tgrep2database.t2c
```

Consult the documentation for tgrep2 for more options.

4.6 Examples of use of tgrep2

Space-considerations limits the number of useful examples that can be shown. The last example is of particular interest in my work.

Showing all entries

```
tgrep2 -c tgrep2database.t2c 'entry'
```

The results are identical to listing 5 apart from the line-numbers and indentation.

Listing all possible part-of-speech tags used

```
tgrep2 -c tgrep2database.t2c 'pos' | sort -u
```

```
(pos (LEAF n))
(pos (LEAF vt))
```

Listing only trees that describe transitive verbs

```
tgrep2 -c tgrep2database.t2c 'entry << (pos << vt)'
```

```
(entry (word (LEAF abash))
  (pos (LEAF vt)) (definition (LEAF to make so. ashamed or embarrased))
  (expression (words (LEAF to abash so. by sneering))))
```

Listing only trees that contain "in expressions only"

```
tgrep2 -c tgrep2database.t2c 'entry << (in $ expressions $ only)'
```

```
(entry (word (LEAF abeyance))
  (pos (LEAF n))
  (definition (LEAF in expressions only))
  (expression
    (words (LEAF property in abeyance))
    (meaning (LEAF property without an owner)))
  (expression
    (words (LEAF hold smth. in abeyance))
    (meaning (LEAF temporarily suspend smth.))))
```

4.7 Converting back to XML

Use SXML or the program in appendix B. The latter is made by the author, and is a generic s-expressions-to-XML-converter. It does not treat subtrees whose roots are `@` or `LEAF` in any special way.

4.8 Bonus: Translation by tgrep2

By using the right pattern in tgrep2 on the real *Engelsk stor ordbok* database, one can get translation-suggestions:

Words: what is an "abbed"?

```
tgrep2 'ekv >> (artikkel << abbed)'
```

```
(ekv (LEAF abbot))
```

Expressions: what does "hulter til bulter" mean?

```
tgrep2 -a 'ekv >> (artikkel << (uttrykk << (hulter $ til $ bulter))'
```

```
(ekv (LEAF pell-mell))
(ekv (LEAF helter-skelter))
(ekv (LEAF at sixes and sevens))
(ekv (LEAF in a mess))
```

5 Conclusion

Using tgrep2 to extract information from XML-encoded databases by way of converting these to s-expressions is a perfectly viable solution to also find the information that is stored only as structure in the XML, and as experiments show it is also a solution that is relatively simple for this researcher.

References

James Clark et al. XSL Transformations (XSLT) Version 1.0. Technical report, W3C, 16 November 1999. URL `http://www.w3.org/TR/1999/REC-xslt-19991116`.

Øystein Eek et al., editors. *Engelsk stor ordbok: engelsk-norsk/norsk-engelsk*. Kunnskapsforlaget, 2001. ISBN 82-573-1288-6.

Oleg Kiselyov. A Better XML Parser through Functional Programming. In S. Krishnamurthi and C. R. Ramakrishnan, editors, *Practical Aspects of Declarative Languages: 4th International Symposium*, Lecture Notes in Computer Science, Portland, OR. USA, January 2002. Springer-Verlag Heidelberg. URL `http://okmij.org/ftp/papers/XML-parsing.ps.gz`.

Jan Tore Lønning, Stephan Oepen, Dorothee Beermann, Lars Hellan, John Carroll, Helge Dyvik, Dan Flickinger, Janne Bondi Johannsen, Paul Meurer, Torbjørn Nordgård, Victoria Rosén, and Erik Velldal. LOGON. A Norwegian MT effort. In *Proceedings of the Workshop in Recent Advances in Scandinavian Machine Translation*, page 6, Uppsala, Sweden, 2004. URL `http://stp.ling.uu.se/RASMAT/extended_abstracts/LOGON.pdf`.

John L. McCarthy. Recursive functions of symbolic expressions and their computation by machine, Part I. *Communications of the ACM*, 3(4):184–195, April 1960. URL `http://www-formal.stanford.edu/jmc/recursive/recursive.html`.

Douglas L.T. Rohde. *TGrep2 User Manual version 1.12*, 4 November 2004. URL `http://tedlab.mit.edu/~dr/Tgrep2/tgrep2.pdf`.

Appendices

A Examples used

All the listings in this appendix have been indented for readability and provided with line numbers for the convenience of the reader.

A.1 The beginnings of a monolingual dictionary, encoded in XML

```
<?xml version="1.0"?>
<dictionary>
    <entry number="1">
        <word>abacus</word><pos grammarpage="nouns">n</pos>
        <definition>ancient manual calculator</definition>
        <seealso>slide rule</seealso>
    </entry>
    <entry number="2">
        <word>abash</word><pos grammarpage="verbs">vt</pos>
        <definition>to make so. ashamed or embarrased</definition>
        <expression>
            <words>to abash so. by sneering</words>
        </expression>
    </entry>
    <entry number="3">
        <word>abeyance</word><pos grammarpage="expressions">n</pos>
        <definition>in expressions only</definition>
        <expression>
            <words>(property) in abeyance</words>
            <meaning>property without an owner</meaning>
        </expression>
        <expression>
            <words>hold smth. in abeyance</words>
            <meaning>temporarily suspend smth.</meaning>
        </expression>
    </entry>
</dictionary>
```

Listing 2: Example of a hypothetical dictionary of English encoded as XML.

A.2 The dictionary after direct conversion to s-expressions

```
(dictionary

  (entry
    (@
      (number 1))
    (word
      (LEAF abacus))
    (pos
      (@
        (grammarpage nouns))
      (LEAF n))
    (definition
      (LEAF ancient manual calculator))
    (seealso
      (LEAF slide rule)))

  (entry
```

```
18      (@
          (number 2))
20      (word
          (LEAF abash))
22      (pos
          (@
24          (grammarpage verbs))
          (LEAF vt))
26      (definition
          (LEAF to make so. ashamed or embarrased))
28      (expression
          (words
30          (LEAF to abash so. by sneering))))

32    (entry
        (@
34        (number 3))
        (word
36        (LEAF abeyance))
        (pos
38        (@
            (grammarpage expressions))
40        (LEAF n))
        (definition
42        (LEAF in expressions only))
        (expression
44        (words
            (LEAF
46            (property) in abeyance))
          (meaning
48          (LEAF property without an owner)))
        (expression
50        (words
            (LEAF hold smth. in abeyance))
52        (meaning
            (LEAF temporarily suspend smth.)))))
```

Listing 3: The results of converting the example XML without adapting the stylesheet.

A.3 A stylesheet tailored for the dictionary

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="text"/>
4  <xsl:strip-space elements="*"/>

6  <xsl:template match="text()">
   (LEAF <xsl:value-of select="normalize-space(.)"/>)</xsl:template>
8
   <xsl:template match="dictionary//*">(<xsl:value-of select="local-name()"/>
10
   <xsl:apply-templates/>)
12 </xsl:template>

14 </xsl:stylesheet>
```

Listing 4: XSLT stylesheet adjusted to the mock-up dictionary

A.4 The dictionary as s-expressions after conversion by the adjusted stylesheet

```
(entry
  (word
    (LEAF abacus))
  (pos
    (LEAF n))
  (definition
    (LEAF ancient manual calculator))
  (seealso
    (LEAF slide rule)))

(entry
  (word
    (LEAF abash))
  (pos
    (LEAF vt))
  (definition
    (LEAF to make so. ashamed or embarrased))
  (expression
    (words
      (LEAF to abash so. by sneering))))

(entry
  (word
    (LEAF abeyance))
  (pos
    (LEAF n))
  (definition
    (LEAF in expressions only))
  (expression
    (words
      (LEAF [property] in abeyance))
    (meaning
      (LEAF property without an owner))
    )
  (expression
    (words
      (LEAF hold smth. in abeyance))
    (meaning
      (LEAF temporarily suspend smth.))))
```

Listing 5: The final results of converting the example XML, after adapting the stylesheet.

B A program for converting s-expressions to XML

This short library-with-builtin-example-program needs Python 2.3 to run, get it from `http://www.python.org/`.

For usage, invoke it like so: `python lisp2xml.py -h`.

```
#!/usr/bin/env python
# Author : Hanne Moa
# Date   : 2004-12-10
# LICENSE: public domain

import string, sys
from optparse import OptionParser

class Lisp2XML_FST(object):

    def __init__(self, root='ROOT', empty='EMPTY', readfrom=''):
        self.emptytag = empty
```

```
        self.roottag = root
        if readfrom:
            self.readfrom = file(readfrom)
        else:
            self.readfrom = sys.stdin
        self._tags = []
        self._next = None
        self._prev = None

    def is_startparens(self):
        self.is_tag()
        self._tags.append([])

    def is_tag(self):
        prevtag = ''.join(self._tags[-1]) or self.emptytag
        self._tags[-1] = prevtag
        self.write("<%s>" % prevtag)

    def is_endparens(self):
        tag = ''.join(self._tags.pop())
        self.write("</%s>" % tag)

    def write(self, something):
        sys.stdout.write(something)

    def comment(self, char):
        if char == '\n':
            return self._prev
        else:
            return 'comment'

    def starttag(self, char):
        self._prev = 'starttag'
        if char in string.whitespace:
            return 'starttag'
        elif char == '(':
            self._tags.append([])
            return 'intag'
        elif char == '%':
            return 'comment'
        else:
            return 'error'

    def intag(self, char):
        self._prev = 'intag'
        if char == '(':
            self.is_startparens()
            return 'intag'
        elif char in string.whitespace:
            if not self._tags[-1]:
                return 'intag'
            else:
                self.is_tag()
                return 'indata'
        elif char == '%':
            return 'comment'
        elif char == ')':
            self.is_endparens()
            return 'indata'
        else:
            self._tags[-1].append(char)
            return 'intag'
```

```
    def indata(self, char):
        self._prev = 'indata'
        default = 'indata'
        if char == '(':
            self._tags.append([])
            return 'intag'
        elif char == ')':
            self.is_endparens()
            return default
        elif char in string.whitespace:
            self.write(char)
            return default
        elif char == '%':
            return 'comment'
        else:
            self.write(char)
            return default

    def error(self, char):
        return ''

    engine = {
        'comment': comment,
        'starttag': starttag,
        'intag': intag,
        'indata': indata,
        'error': error,
        }

def convert(root='ROOT', empty='EMPTY', readfrom=''):
    fst = Lisp2XML_FST(root, empty, readfrom)
    engine = fst.engine
    fst.write("<%s>" % fst.roottag)
    for line in fst.readfrom:
        for char in line:
            if not fst._tags:
                next = 'starttag'
                fst.write('\n')
            prev = next
            next = engine[next](fst, char)
            if not next:
                if fst._tags:
                    fst.is_endparens()
                break
    if fst._tags:
        fst.is_endparens()
    fst.write("</%s>\n" % fst.roottag)
    fst.readfrom.close()

if __name__ == '__main__':
    usage = """Usage: %prog [options] < FILE
Translate s-epressions into XML."""
    parser = OptionParser(usage=usage)
    parser.add_option("-f", "--file", metavar='FILE',
        help="convert s-expressions in FILE")
    parser.add_option("-e", "--empty", default='EMPTY',
        help="fallback tag in case of '(' in source")
    parser.add_option("-r", "--root", default='ROOT',
        help="tag to use as root, if none")

    (opts, args) = parser.parse_args()
```

```
convert(root=opts.root, empty=opts.empty, readfrom=opts.file)
```

Listing 6: Source for converting from s-expressions to XML