Moose: A Robust High-Performance Parser and Generator

Gábor Prószéky, László Tihanyi, Gábor L. Ugray

Morphologic, Ltd.
1126 Budapest, Orbánhegyi út 5.
{proszeky,tihanyi,ugray}@morphologic.hu

Abstract. This paper describes Moose, a robust bottom-up parser that implements the MetaMorpho formalism. During the development of the formalism and the parser we have aimed at a compromise between expressive power and performance for MT-related applications. The characteristics of an English-Hungarian grammar are discussed along with the parser.

1. Introduction

The compromise between a natural language grammar's expressive power and the performance of the resulting system is a crucial issue for the developer of any MT application. By opting for a high-performance approach, such as finite state technology, one inevitably has to sacrifice descriptive power, whilst theoretically motivated formalisms such as HPSG or LFG rarely have implementations that are efficient enough for commercial use. The MetaMorpho formalism, and Moose, our parser implementation, is an attempt at a reasonable compromise.

2. The MetaMorpho formalism

2.1 Bottom-up parsing

A MetaMorpho rule is illustrated in Figure 1. The grammar operates with rule pairs that consist of one rewrite rule used during bottom-up parsing and one or more corresponding transfer rules that are applied during top-down generation. The set of parse rules effectively forms a phrase-structure (context free) grammar where each symbol has a list of typed features. Features can either select from a set of symbolic values, i.e., {SG, PL}, or contain a string, such as the lexical form of a structure's head; however, the formalism does not allow for embedded feature structures or functions. The right-hand side of the parse rule can state conditions for any of its symbols' values.

At this level, all conditions and assignments must be stated explicitly: there are no designated head or foot features, nor is there any mechanism to automatically percolate features when a unification takes place.

2.2 Kills

Each rule can state any number of overrides or kills on other rules: if the "killer" rule fires over a specific range of the input, it blocks the "killed" one over the same range and any productions of that rule are removed from the chart. This

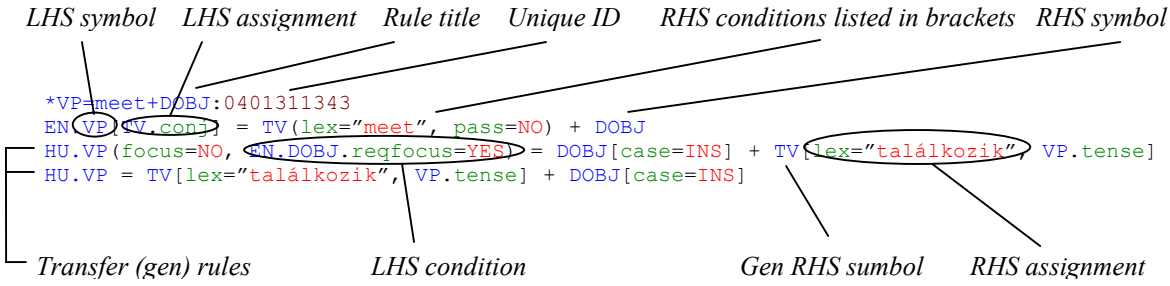

Figure 1. Overview of a MetaMorpho rule illustrating the transitive subcategorization of "meet." Several features have been omitted for the sake of clarity.

mechanism of overrides extends the expressive power of the grammar beyond that of pure context-free formalisms.

Kills are extensively used for syntactic disambiguation: if a PP, for instance, figures in two alternative parses as a free adjunct and as the complement of a VP, respectively, the latter analysis will override the first.

2.3 Top-down generation

When the whole input is processed and no applicable rules remain, generation proceeds top-down from the root symbols by firing the transfer rule corresponding to the parse rule that created the edge at parse time – a solution we term immediate transfer as it uses no separate transfer mechanism or target transformations. Transfer rules can have conditions in the left-hand side, and in the case of multiple transfer rules, the first one whose conditions are satisfied is fired. This mechanism allows for a local rearrangement of the parse tree's subtrees.

In order to handle more complicated word order changes, however, a stronger means of rearrangement is provided also. A subtree can be memorized in a feature when a unification takes places at parse time, and this feature's value can be percolated up the parse tree and down the transfer tree just like any other feature. A phrase swallowed at any level in the source side can thus be expanded at a completely different location in the transfer tree.

The power and simplicity of subtree memorization and random insertion can be demonstrated with the translation of English possessive structures into Hungarian: *the friend of the man* translates into ***a**-DET **férfi**-N-NOM/man **barátja**-N-POS/friend*, i.e., the order of NP's is reversed. Through the interplay of only two rules (the place of memorization at N-bar level and insertion at NP-level), a possessive structure of any length is translated recursively in reverse order into Hungarian. Figure 2 illustrates two such rules and a more complex possessive structure translated in this manner.

3. The English-Hungarian grammar

Our grammar, developed in the MetaMorpho formalism, currently has above 130 thousand rules, the majority of which are lexicalized items. The system uses no separate dictionary: what

```
*NN=NX+of+NP:0109261534-1
EN.NN[lnpf=YES, Left_np<-NP, NX.num] = NX + PREP(lex="of") + NP
HU.NN = NX[NN.case, NN.postp, ownernum=SG, ownerpers=EN.NP.pers]

*NPX=the+NM:0205291509-11
EN.NPX[NM.num]          = DET(dettype=DEF) + NM
HU.NPX(EN.NM.lnpf=NO)   = DET + NM[NPX.case, NPX.postp]
HU.NPX(EN.NM.lnpf=YES) = NP#1{EN.NM.Left_np}[case=DAT] + DET + NM[NPX.case, NPX.postp]
```

```
NP 345
  NPX 344
    DET lex="the"
    NM 341
      NN 339
        NX 4
          N lex="friend"
        PREP lex="of"
        NP 311
          NPX 310
            DET lex="the"
            NM 307
              NN 305
                NX 99
                  N lex="wife"
                PREP lex="of"
                NP 297
                  NPX 293
                    PRONX 270
                      PRON lex="I, case=GEN
                      NM 290
                        NN 288
                          NX 287
                            N lex="neighbour"
```

```
NP 591{345}
  NPX 592{344}
    NP 593{311}
      NPX 596{310}
        NP 597{297}
          NPX 600{293}
            DET dettype=DEF
            NM 602{290}
              NN 603{288}
                NX 604{287}
                  N lex="szomszéd"
        DET dettype=DEF
        NM 599{307}
          NN 606{305}
            NX 607{99}
              N lex="feleség"
    DET dettype=DEF
    NM 595{341}
      NN 609{339}
        NX 610{4}
          N lex="barát"
```

Figure 2. The source and target trees for *"the friend of the wife of my neighbour"*, translated as *"a szomszédomnak a feleségének a barátja,"* and a simplified version of the two rules involved in the reverse order translation

would traditionally be entries in a lexicon are integrated in the form of rules.

As the grammar uses no feature structures, complex lexical information such as the subcats of a verb are not described in terms of features but by creating separate rules, in this case with a VP in their left-hand side, for each subcat frame. The rules in the grammar are much more complex than the ones shown here. Therefore, for the sake of human readability and maintainability, lexical items are coded in a simpler form where all non-lexical information is omitted. The actual rules are then generated off-line from their simplified source, and a large amount of linguistic knowledge is effectively encoded in this conversion. The philosophy behind this is to

Translation	H1	H2	H3	M	M	M	Net	Net	Net
Reference 1	H2	H1	H1	H2	H1	H1	H2	H1	H1
Reference 2	H3	H3	H2	H3	H3	H2	H3	H3	H2
N-gram scores									
1	.7610	.7476	.5789	.5794	.5317	.5556	.4921	.4604	.4842
2	.4375	.5000	.2832	.1935	.1613	.1855	.0720	.0800	.0720
3	.2844	.3333	.1560	.0833	.0500	.0833	.0082	.0082	.0082
4	.1792	.1979	.0857	.0517	.0258	.0517	0	0	0
5	.1068	.1398	.0297	.0268	.0089	.0268			
6	.0700	.0889	.0103	.0093	0	.0093			
7	.0412	.0460	0	0		0			
8	.0213	.0238							
9	.0110	.0123							
10	0	0							

Table 1. N-gram scores for three human translators (H1 through H3), a machine translator available on the Internet (Net), and MetaMorpho (M).

remove the burden of interpreting a complex and linguistically motivated formalism from the parser while representing the same linguistic knowledge in an off-line step. It is a remarkable fact that due to the high proportion of lexicalized rules, 99% of the actual edges created in a parse are the result of a very small subset (about 1%) of the whole grammar. The number of edges generated for a sentence about 30 tokens longs can be in the thousands, and this number is expected to grow significantly as a wider range of gaps is introduced in the grammar.

The grammar has been evaluated on an English text against three human translators and an English-Hungarian machine translator available on the Internet. Table 1 shows n-gram precision results for the five translations, each of them evaluated against two human translations. As can be expected, the performance of our system lags far behind human translations; in comparison to the tested machine translation software, however, the results are very promising.

4. The parser implementation

The parser engine behind the grammar, Moose, is implemented as a set of C++ template headers that can be parametrized according to grammar type (monolingual, i.e., only parse rules, or bilingual, i.e., transfer rules also) and function (compiler, parser and developer).

4.1 Functional versatility

The compiler's task is to create the binary grammar from a type description, which is an XML file containing the grammar symbols and their features, and any number of text files containing the rules. Compiling the grammar rules involves syntactic parsing, semantic checking against the type descriptions, building an index tree for the fast and precise retrieval of lexically specified rules, and saving these structures in a binary format.

The parser is optimized for grammars reflecting the philosophy outlined in the previous section by incorporating the conditions on designated lexical features into an index of rules. Lexicalized rules are stored on disk and loaded

into memory when they are fired; the precision of selecting only the applicable rules is very high.

The parser is also implemented with multi-threading applications in mind. The bulk of the resources is only loaded into memory once, when the parser is started up. Individual parsing requests can be fulfilled simultaneously in separate sessions, which only contain information pertaining to the particular parse in question. The majority of the rules is only loaded into memory, and stored temporarily in the sessions, when their lexical conditions are very likely to be satisfied. The small proportion of very productive rules, at the same time, is kept permanently in the physical memory as a shared resource of the parser.

The third function is the developer: it is really a mixture between the parser and the compiler. It provides the functionality of a parser, but it also has an interface through which even individual grammar rules can be removed, added or modified, and the changes will immediately apply during the next parse. This is very important as long compile times would be a prohibitive factor in the development of a large grammar such as the one described above. Beyond grammar modification, the developer also provides an interface through which detailed information can be queried about the last parse chart: rules that fired, edges that were created from a given symbol, instantiated kills, removed edges etc.

Code duplication is avoided through the extensive use of template programming techniques. While the parsing algorithm is coded only once, very different underlying representations are used in the parser and the developer, as the former is optimized for size and performance, and the latter has to allow quick modification of the stored structures.

4.2 Naïve, robust bottom-up parsing

The motivation for creating a robust bottom-up parser is that the grammar's applications invariably require access to a parse's partial results even in the absence of a full parse tree. The parser invokes a user-defined filter when parsing is complete but before transfer. These filters have access to all parse trees and can select, for instance, a disjunct coverage of the input tokens. Certain features can also be modified at this stage, thereby allowing for a word sense disambiguation tool to make its decisions and inject the results into the parse tree symbols based on syntactic structure.

Another argument for the chosen parsing method is the expressive power of rule overrides. In order to instantiate kills, most possible directions need to be explored; in a top-down framework there is no place for the type of overrides used our grammar is built around.

4.3 Integration with a rule database

A special set of lexicalized rules in the grammar can be freely extended in real-life applications through the parser's integration with a rule database, for instance, an SQL server. This allows the controlled addition of lexical items into an application's grammar – for instance, new words, or sub-sentential structures in a translation memory.

As database rules must be added or removed individually, they cannot participate in the global contest of overrides, and they require a separate indexing mechanism. As the prime objective of database integration is to allow users to flexibly increase the lexical coverage of the precompiled, intransparent grammar, this is a natural limitation.

5. Further applications

Besides being the engine behind the English-Hungarian machine translator, the parser's abstract implementation allows for its use in several other applications. The integration with an SQL database forms the basis of the translation memory (TM) solution currently being developed. For linguistic research purposes, an NP chunker has been built that processes approximately 1500 words per second on a P4 machine. As the input data structure is not wired into the code, we are also experimenting with a Hungarian morphology expressed in context-free rules.

6. Future work

The current bottom-up parsing method is a naïve approach apart from the precise indexing of rules by their lexical feature conditions. We have opted for this solution because of the need for robustness. It has been shown, however, that a certain type of top-down prediction (limited left context constraints) can increase efficiency by an order of magnitude without sacrificing any of the needed robustness by a better treatment of typical natural language phenomena such as gaps. Therefore, within the coming months, we plan to incorporate some type of top-down prediction into the parsing algorithm.

References

Philippe, McLean and R. Nigel, Horspool (1996). 'A Faster Earley Parser', Proceedings of International Conference on Compiler Construction, Linkoping, Sweden, 281–293.

Papineni, Kishore, Roukos, Salim, Ward, Todd, Zhu, Wei-Jing (2001). 'Bleu: A Methord for Automatic Evaluation of Machine Translation', IBM Report RC22176

Gábor, Prószéky (2001). 'Nyelvi technológiák és gépi fordítás', Emberi és gépi nyelv, beszéd és hallás, MTA osztályülés, Budapest

Moore, Robert and Dowding, John (1991). 'Efficient Bottom-Up Parsing', In: Proceedings of the DARPA Speech and Natural Language Workshop, Asilomar, CA, 200–203.

Jean, Senellart, Péter, Dienes and Tamás, Váradi (2001). 'Development of a New Generation of Translation System', EAMT 2001

Gertjan, van Noord (1997). 'An Efficient Implementation of the Head-Corner Parser', Computational Linguistics, volume 23, number 3, 1997.