Moose: A Robust High-Performance Parser and Generator

Gábor Prószéky
MorphoLogic, Ltd. Budapest
proszeky@morphologic.hu

László Tihanyi
MorphoLogic, Ltd. Budapest
tihanyi@morphologic.hu

Gábor L. Ugray
MorphoLogic, Ltd. Budapest
ugray@morphologic.hu

Keywords: English-Hungarian machine translation, robust bottom-up parsing, incorporated lexicon, immediate transfer, grammar development workbench

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.

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

```
*VP=meet+DOBJ:0401311343
EN.VP[TV.conj] = TV(lex="meet", pass=NO) + DOBJ
HU.VP(focus=NO, EN.DOBJ.reqfocus=YES) = DOBJ[case=INS] + TV[lex="találkozik", VP.tense]
HU.VP = TV[lex="találkozik", VP.tense] + DOBJ[case=INS]
```

Figure 1: A simplified rule illustrating the transitive subcategorization of "meet" with several features omitted for the sake of clarity

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; 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 as 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 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/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.

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 would traditionally be entries in a lexicon are integrated in the form of rules. Such a rule is shown in Figure 1, and Figure 2 below shows the simplified rule for the noun *dog*.

```
*NX=dog:0401311411
EN.NX[N.lex, N.num] = N(lex="dog")
HU.NX = N[lex="kutya", NX.case, NX.ownernum, NX.ownerpers]
```

Figure 2: A simplified rule illustrating the translation of "dog"

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 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 parser implementation. The author's chief contribution to the English-Hungarian MT project is Moose, the parser engine behind the grammar. It is implemented as a set of C++ template headers that can be parametrized according to function (compiler, parser and developer) and grammar type (monolingual, i.e., only parse rules, or bilingual, i.e., transfer rules also). The parser is optimized for grammars reflecting the philosophy outlined above 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 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. Some 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.

A 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.
One instantiation of the parser template, the developer, provides an interface to modify any rule in the grammar on-line without the need to recompile the rest, and to query internal information about the last parse. Around this interface a grammar development workbench with many debugging features has been built to facilitate the work of grammar writers. The developer uses the same code that is run in the parser, while the replacement of the underlying containers makes it possible to alter the grammar and query internal parse results at the cost of increased parse time and memory usage in a separate application.

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 represented by context-free rules and hope to achieve results comparable to state-of-the art approaches in terms of efficiency.

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.

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.