The Kalashnikov 691 dependency bank

Tomas By

Centro de Linguística da Universidade Nova de Lisboa
Avenida de Berna 26-C, 1069-61 Lisboa, Portugal
tomas.by@fcsh.unl.pt

Abstract

The PARC 700 dependency bank has a number of features that would seem to make it less than optimally suited for its intended purpose, parser evaluation. However, it is difficult to know precisely what impact these problems have on the evaluation results, and as a first step towards making comparison possible, a subset of the same sentences is presented here, marked up using a different format that avoids them. In this new representation, the tokens contain exactly the same sequence of characters as the original text, word order is encoded explicitly, and there is no artificial distinction between full tokens and attribute tokens. There is also a clear division between word tokens and empty nodes, and the token attributes are stored together with the word, instead of being spread out individually in the file. A standard programming language syntax is used for the data, so there is little room for markup errors. Finally, the dependency links are closer to standard grammatical terms, which presumably makes it easier to understand what they mean and to convert any particular parser output format to the Kalashnikov 691 representation. The data is provided both in machine-readable format and as graphical dependency trees.

All that is complicated is unnecessary;
all that is necessary is simple.
Michail Kalashnikov

1. Introduction

This work complements the criticism in By (2007) of the PARC 700 dependency bank (King et al., 2003) by offering a concrete proposal for a better dependency bank format, with semi-automatically created markup of the same sentences. Nine of the sentences in the PARC 700 consist of one single token, and therefore have no syntactic structure, so they have not been included.[1] Although most of the work of creating this data involved linguistic decisions on how exactly to represent particular constructions, and precisely which set of attributes and link labels are needed, the rationale for these decisions are not explained here. This would require much more space, and will, hopefully, be provided in a later publication. Rather, this paper is mainly concerned with presenting the format in its final[2] form, to make it possible for people to use it.

2. Problems with the PARC 700 format

The PARC 700 is meant to be used for evaluating parsers using the method suggested in Carroll et al. (1998): converting the parser output to a set of dependency relations between strings representing the base forms of some of the words in the sentence, and comparing them with the correct ones in the dependency bank. Multiple occurrences of the same word in a sentence are distinguished, in the PARC 700, by a numerical index. But the order of the words in the sentence is not indicated, so when matching against the parser output there is a risk of using the wrong token. The problem is aggravated by the fact that the words are lemmatised, so that even if two occurences of a word have different surface form they will have the exact same representation in the dependency bank. About 15% of the tokens in the PARC 700 are ambiguous in this way (By, 2007, pp. 275–7).

Since the PARC 700 also includes a large amount of attributes of single tokens,a user might be inclined to not just compare the dependency links but also the attributes, or try to use this data for some further, semantic, processing. Very quickly one then runs into the problem of tokenisation. Less than half of the tokens in the PARC 700 are identical to a corresponding token in the Penn Treebank, and about 12% of the PTB tokensin the seven hundred sentences do not occur in the PARC 700 at all (By, 2007, pp. 277–8). Automatic mapping of the tokens in the PARC 700 and the PTB is described in By (2007, pp. 268–9), but it is not a trivial problem.

There is also a certain amount of inconsistency in the PARC 700 treatment of hypenation (By, 2007, p. 273), tokens that contain spaces (By, 2007, pp. 273–4), comparative constructions (By, 2007, pp. 266, 274–5), and the technical distinction between indexed tokens and ‘attribute tokens’ (By, 2007, p. 263). Finally, the few, but completely unnecessary, markup errors (By, 2007, p. 272) could have very easily been avoided by using an established programming language syntax instead of a specially made-up format.

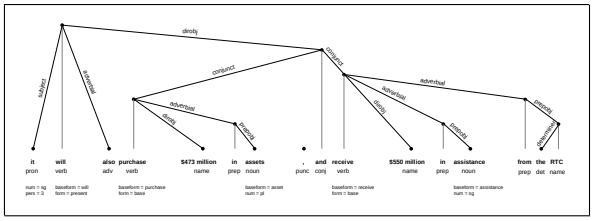

Figure 1: Sentence two as a graphical tree

[1] Sentences 151, 170, 171, 190, 217, 228, 256, 334, and 337.

[2] This means of course only the final form of the 691 sentences in the dependency bank. If other corpora are marked up using this format, it will presumably eventually need to be extended, or modified.

Category	Class	Attributes Obligatory	Optional	Possible attribute values
Verb	verb	baseform		(any)
		form		base, present, past, pres_part, past_part
Auxiliary verb	verb	baseform		(any)
		form		base, present, past, pres_part, past_part
		aux		be, have, do, modal
Infinitive marker	verb	baseform		to
		form		base
		aux		to
Particle	prt			—
'that'	that			—
Pronoun	pron		pers	1, 2, 3
			num	sg, pl
			case	common, subject, object, genitive
Determiner	det			—
Numeral	number	type		cardinal, ordinal
Common noun	noun	baseform		(any)
			num	sg, pl
			case	common, genitive
Proper noun	name		case	common, genitive
Adjective	adj		form	base, comparative, superlative
Adverb	adv		form	base, comparative, superlative
Preposition	prep			—
Conjunction	conj			—
Punctuation	punc			—
Interjection	intj			—

Table 1: All the word classes and their attributes

Label	Modifier (word class)	Direction	Head (word class)	Number of links
modal	verb[form=base][1]	↖	verb[aux=modal]	One
perfect	verb[form=past_part]	↖	verb[baseform=have]	One
progressive	verb[form=pres_part]	↖	verb[baseform=be]	One
passive	verb[form=past_part]	↖	verb[baseform=be]	One
do	verb	↖	verb[aux=do]	One
infinitive	verb	↖	verb[aux=to]	One
that	verb	↖	that	One
particle	part	↖	verb[¬aux][2]	One
subject	pron / noun / name	↗	verb	One
dirobj	pron / noun / name	↖	verb[¬aux]	One
indobj	pron / noun / name	↖	verb[¬aux]	One
complement	adj / prep	↖	verb[¬aux]	Many
prepobj	pron / noun / name	↖	prep	One
determiner	det	↗	noun / name	Many
premod	adj / noun / name / verb	↗	noun	Many
postmod	verb / prep	↖	noun	Many
adverbial	adv / prep / verb	(either)	verb	Many

[a] The node must have this attribute and value.
[b] The node cannot have this attribute.

Table 2: Syntactic dependency types (grammatical functions)

3. The Kalashnikov 691 format

The main requirements that have guided the design of this format is that it should be unambiguously formally specified, and also easy for a human to read and understand. The first requirement has been met by using Prolog syntax, specifying the data format in Backus-Naur Form (By, 2007, pp. 269–71), and also listing all possible attribute names, attribute values, and dependency links together with their meanings (tables 1 and 2).

The second requirement is of course more difficult, and the first step towards satisfying it has been to use traditional, well established names for the word categories and dependency links. Figure 1 shows an example of how this looks, in the form of a classic dependency tree (Matthews, 1981, p. 81). The finite verb is the root of the sentence, except if sentence is complex (Quirk et al., 1985, p. 719), in which case the conjunction is the root. The tokens are not modified, so concatenating them (and adding appropriate whitespace) will produce the original sentence string. Instead, all verbs and nouns have the baseform stored in an attribute.[3] Names (and numerical expressions) are one single token. The word classes are those suggested by Dionysius Thrax two thousand years ago, and which are still being taught in grade school, with some minor modifications.[4] Table 1 lists all of these, with the obligatory and optional attributes, together with all possible attribute values, except for the 'baseform' attribute which has an unlimited set of possible values.

The link types used in the Kalashnikov 691 are shown in table 2, together with the types of the nodes (tokens), and the link direction and maximal number of links per head word. There cannot be more than one direct and indirect object per verb, for instance. In the PARC 700, the set of link types is less traditional.[5] While it is not clear, at least to the present author, whether either of these approaches is technically preferable, and, if so, which one it is, it seems beyond doubt that the standard grammatical terms will be easier for the average user to understand, and the more so the less linguistic expertise he has. In addition to the combinations listed in table 2, the `conj` word class is allowed in any position, and it can only have `conjunct` child links, as in figure 1.

Figure 2 shows the Prolog format used for the data. The tokens are numbered consecutively (but the word order is also encoded by the list in the `sentence`/4-clause). Words and empty nodes are numbered separately, both starting from zero. Normally, all tokens should be connected and form a tree, but punctuation characters are not included, so those tokens will be unconnected. There are no cycles in the dependency graphs.

Clause ::=	`sentence(`$SentNum$`,`Id`,`$WordNums$`,`$NodeNums$`)`
	| `word(`$SentNum$`,`$WordNum$`,`$Word$`,`$Attributes$`)`
	| `node(`$SentNum$`,`$NodeNum$`,`$Word$`,`$Attributes$`)`
	| `dependency(`$SentNum$`,`$Dnode$`,`$DepRel$`,`$Dnode_{to}$`)`
SentNum ::=	Number (Identifying the sentence)
WordNum ::=	Number (Identifying the word)
NodeNum ::=	Number (Identifying the empty node)
Id ::=	Atom (Arbitrary string identifying the sentence)
WordNums ::=	List of numbers (In the right sentence order)
NodeNums ::=	List of numbers (In arbitrary order)
Word ::=	Atom
Attributes ::=	List of pairs of atoms
Dnode ::=	`w(` $WordNum$ `)` | `n(` $NodeNum$ `)`
DepRel ::=	Atom (Name of the grammatical relation)

Figure 2: The format of the Prolog representation

The Kalashnikov 691 dependency bank can be downloaded from the following web page.

`http://www.basun.net/nlp/kalashnikov691/`

There are two files: '`kalashnikov691.pl`' is the machine readable Prolog data and '`kalashnikov691.pdf`' contains the graphical 'trees' of all the six hundred and ninety one sentences.

4. Partially automated quality assurance

Since the Kalashnikov 691 dependency bank uses Prolog syntax it is a simple matter to collect, for example, all the tokens,[6] and find those that contain hyphens or spaces. As can be verified by the reader at the URL given above, the automatically generated depdendency trees file includes an index of all the tokens, indicating the sentences where they occur. With these facilities, it is reasonably straightforward to ensure the consistency of the tokenisation. But a dependency representation also allows a more powerful type of automatic verification, namely checking that the trees are projective (By, 2007, pp. 267–8). While it is probably not possible to represent all constructions in the language using fully projective dependency trees (Mel'čuk and Pertsov, 1987, pp. 184–6); (Mel'čuk, 1988, pp. 36–8), it seems likely that for most constructions it is, and it also seems likely that errors in the graph structure will typically violate projectivity. This means that automatic projectivity checking is a useful means of controlling the quality of the data. The tree-drawing tool does this, and the only non-projective constructions in the Kalashnikov 691 dependency bank are relative clauses (and the unconnected punctuation tokens).

5. Conclusions

The Kalashnikov 691 dependency bank is superior to the PARC 700 for the following reasons. The tokens contain exactly the same sequence of characters as the original text. It is not always the same tokens as in the Penn Treebank, but the mapping can be done relatively easily by looping

[3] The attribute '`baseform`' (table 1).

[4] Originally there were eight classes: noun, verb, participle, article, pronoun, preposition, adverb, and conjunction (Davidson, 1874, p. 8). Here, participles are considered verbs, and the noun class has been split into proper nouns, common nouns, and adjectives.Winograd (1983, pp. 51–3), Quirk et al. (1985, pp. 67–8), and Halliday and Matthiessen (2004, pp. 52, 362) all use similar classifications.

[5] For example, there are seven different link types for non-adverbial verb arguments (`obj`, `obj_theta`, `comp`, `xcomp`, `obl`, `obl_ag`, `obl_compar`). The three used in the Kalashnikov 691 (`dirobj`, `indirobj`, `complement`) are the normal grammatical terms (Quirk et al., 1985, p. 54).

[6] There are 3795 token types, about 25% of which are names or numerical expressions.

from left to right. There is no need for any disambiguation (By, 2007, pp. 268–9). Word order is encoded explicitly, so multiple occurrences of the same word in one sentence are not ambiguous.[7] There is no artificial distinction between full tokens and attribute tokens, and there is a clear division between word tokens and empty nodes (By, 2007, p. 265).[8] The token attributes are stored together with the word in Kalashnikov 691. In the PARC 700 files they are spread out individually, with a format that is quite similar to the dependencies. Since a standard programming language syntax is used for the data, there is little room for markup errors (By, 2007, p. 272), and because the automatic projectivity verification would detect dependency link problems such as misattached modifiers (By, 2007, p. 267), some confidence might be felt about the quality of the encoding of the syntactic structures. Finally, the set of dependency link types are closer to normal grammatic terms in the Kalashnikov 691 than in the PARC 700, which ought to make the data more accessible.

6. References

Tomas By. 2007. Some notes on the PARC 700 dependency bank. *Natural Language Engineering*, 13(3):261–282.

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the first International Conference on Language Resources and Evaluation*.

Thomas Davidson. 1874. The grammar of Dionysios Thrax. *The Journal of Speculative Philosophy*.

M.A.K. Halliday and Christian M.I.M. Matthiessen. 2004. *An Introduction to Functional Grammar*. Hodder Arnold, London, third edition.

Richard Hudson. 1990. *English Word Grammar*. Basil Blackwell Ltd.

Richard Hudson. 2007. *Language Networks*. Oxford University Press.

Tracy Holloway King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 Dependency Bank. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora, held at the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL'03)*, Budapest.

P.H. Matthews. 1981. *Syntax*. Cambridge University Press.

Igor A. Mel'čuk and Nikolaj V. Pertsov. 1987. *Surface Syntax of English*. John Benjamins.

Igor A. Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

R. Quirk, S. Greenbaum, G. Leech, and J. Svartvik. 1985. *A Comprehensive Grammar of the English Language*. Longman, London.

Terry Winograd. 1983. *Syntax*, volume 1 of *Language as a Cognitive Process*. Addison-Wesley.

[7] 15% in PARC 700 (By, 2007, pp. 275–7).

[8] The string '`pro`' is both a word and an empty node label in PARC 700 (By, 2007, p. 278).