# Realization of long sentences using chunking

**Ewa Muszyńska**
Computer Laboratory
University of Cambridge
emm68@cam.ac.uk

**Ann Copestake**
Computer Laboratory
University of Cambridge
aac10@cam.ac.uk

## Abstract

We propose sentence chunking as a way to reduce the time and memory costs of realization of long sentences. During chunking we divide the semantic representation of a sentence into smaller components which can be processed and recombined without loss of information. Our meaning representation of choice is Dependency Minimal Recursion Semantics (DMRS). We show that realizing chunks of a sentence and combining the results of such realizations increases the coverage for long sentences, significantly reduces the resources required and does not affect the quality of the realization.

## 1 Introduction

Surface realization, or generation, is a task of producing sentences from a representation. Realization can be thought of as the inverse of parsing and constraint-based approaches, implemented, for instance, using chart algorithms, make it possible to use the same reversible grammar for both parsing and realization.

Large semantic representations present a challenge to generators. In the worst-case scenario chart generation has exponential complexity with respect to the size of the representation, although the algorithm can be modified to improve the performance (Carroll et al., 1999; White, 2004).

In this paper we propose chunking (Muszyńska, 2016) as a way to reduce memory and time cost of realization. The general idea of chunking is that strings and semantic representations can be divided into smaller parts which can be processed independently and then recombined without a loss of information. For realization we chunk input semantic representations. There may be multiple chunking points in one sentence. Here we show that the efficient realization of a full sentence is possible through a principled composition of realizations of the chunks.

We show that the technique noticeably reduces the cost of realization and in some cases it allows for realization where no result was found using the standard approach. This effect is achieved without significantly degrading realization quality.

## 2 DELPH-IN framework

The semantic representation we use in our experiments is Dependency Minimal Recursion Semantics (DMRS) (Copestake, 2009), developed as part of the DELPH-IN initiative[1], together with several wide-coverage HPSG-based grammars, notably the English Resource Grammar (ERG) (Flickinger, 2000; Flickinger et al., 2014). The ERG is a broad-coverage, symbolic, bidirectional grammar of English. The DELPH-IN realization systems have been used successfully in a number of applications, such as question generation (Yao et al., 2012), paraphrasing logic forms for teaching purposes (Flickinger, 2016) and abstractive summarisation (Fang et al., 2016).

An example of a DMRS graph is shown in Fig. 1. Nodes correspond to predicates, edges (links) represent relations between them. It is inter-convertible

[1]Deep Linguistic Processing with HPSG, www.delph-in.net

with Minimal Recursion Semantics (MRS) format (Copestake et al., 2005).

DMRS graphs can be manipulated using two existing Python libraries. The `pyDelphin` library[2] is a more general MRS-dedicated library which we use for conversions between MRS and DMRS. The `pydmrs` library[3] (Copestake et al., 2016) is dedicated solely to DMRS.

In these experiments, we work with DMRS graphs which are the output of parsing with the 1214 version of the ERG. The realizer we use, ACE[4], is one of the processors designed to work with DELPH-IN grammars. It is a more efficient re-implementation of the chart parser and generator of the LKB (Carroll et al., 1999; Copestake, 2002; Flickinger, 2016). Parsing and realization results are ranked by a maximum entropy language model (Velldal, 2008).

In this paper we refer to the realization using default ACE settings as the standard realization. We introduce two adjustments to this set-up: a fixed time-out of 30s after which a realization attempt is abandoned even if it did not produce a result, and a mechanism to deal with unknown words. The time-out chosen is quite high and does not affect most realizations.

The ACE generator does not currently have a mechanism to cope with unknown predicates, i.e. predicates which do not appear in the grammar's lexicon. They can be parsed, however, and assigned a part-of-speech tag. Based on this information, we substitute each unknown predicate with a known predicate with the same part-of-speech tag before the semantic representation is input to the generator. Afterwards, the known surface form of the substitute predicate is replaced in the realization string with the surface form of the original unknown predicate (Horvat, 2017).

We retrieve all possible realizations for the given semantic representation together with their ranking scores. We also note the amount of memory and time needed for the realization, and the number of edges produced in the chart generation process.

[2]`https://github.com/delph-in/pydelphin`

[3]`https://github.com/delph-in/pydmrs`

[4]The Answer Constraint Engine 0.9.24, `http://sweaglesw.org/linguistics/ace/`, by Woodley Packard.

We do not expect full realization coverage, even though the generator uses the same grammar as the parser which produced the representations. Some lexical items, such as infinitival *to*, are semantically empty according to the ERG analysis, i.e. they are not assigned their own predicates (Carroll et al., 1999). During realization with ERG/ACE, hand-written rukes are used to signal that particular semantically empty lexical items may be required. Missing rules sometimes cause realization failure.

## 3 Realization with chunking

Realization from chunks consists of four phases. After chunking a sentence (§ 3.1), we convert each chunk into a well-formed DMRS, introducing small place-holder graphs where necessary (§ 3.2). During the realization phase we generate from each of the chunk DMRSs separately. Finally, we use the information about how chunks are related to combine the chunk realizations into a full sentence realization (§ 3.4).

### 3.1 Chunking

Here we use an approach to chunking based on DMRS graphs. We chunk a semantic representation by dividing it into subgraphs, without access to any information about the surface form of the represented sentence. The link structure of the DMRS graph reveals appropriate chunk boundaries. Currently chunking is based on three grammatical constructions: clausal coordination, subordinating conjunctions and clausal complements.

For each chunking decision, we identify a functional chunk which plays the role of a trigger for chunking, i.e. its presence indicates the chunking possibility. For example, if a semantic representation contains a subordinating conjunction, it can be chunked as shown in Fig. 1. A functional chunk in this case consists of a single node with `_since_x_subord` predicate representing the subordinating lexeme. Each chunking decision also identifies two clauses. In Fig. 1 they are simple main and subordinate clauses, but in more complicated sentences these clauses could contain further chunking triggers, forming a tree-like hierarchy of chunks.

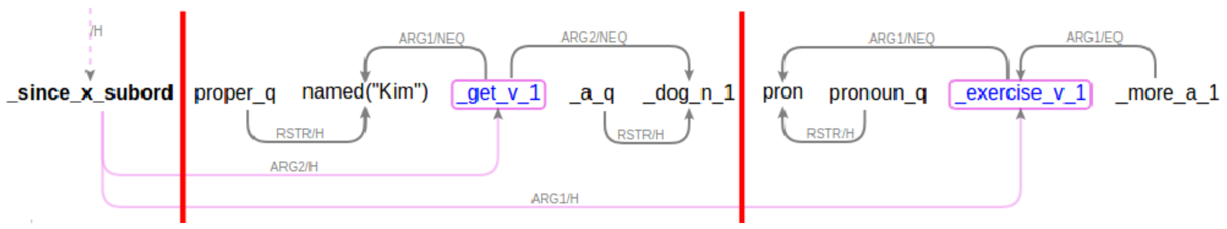

**Figure 1:** A DMRS graph for the sentence *Since Kim got a dog, she exercises more.* Chunk boundaries are marked in red.

### 3.2 Substitution

Functional chunks are not well-formed DMRSs – they typically consist of a single node. During chunking we preserve information about severed links between the chunks (trigger links, highlighted in Fig. 1) and in the substitution phase we introduce small pre-defined DMRSs at the end of the trigger links, where we would originally find other chunks. This ensures the well-formedness of the functional chunk.

In the experiment we use two minimalistic substitution DMRSs corresponding to clauses *It was snowing* and *it was raining*. Their DMRS graphs consist of single nodes and have one possible realization, which is important for the assembly phase. For coordination and subordinating conjunction we substitute both clauses and for clausal complements we substitute the complement. The resulting DMRS for the functional chunk of the example in Fig. 1 would consist of three nodes: `_since_x_subord`, `_snow_v_1` at the end of the highlighted ARG1 link and `_rain_v_1` at the end of the ARG2 link.

### 3.3 Realization

In this phase we simply feed chunk DMRSs into the ACE generator. In the case of functional chunks these are the DMRSs obtained through substitution. Collected data and realization settings are the same as for the standard realization.

### 3.4 Surface assembly

After all possible realizations are collected for all chunks, we replace realizations of the substitute DMRSs with realizations of appropriate chunks. Based on the chunk hierarchy preserved during chunking, we know which chunk was originally at the end of each trigger link and following this information we can assemble the full sentence recursively.

| | Percentage | Count |
|---|---|---|
| Both | 40.6 | 128 |
| Only chunking | 17.5 | 55 |
| Only full | 8.9 | 28 |
| Neither | 33.0 | 104 |

**Table 1:** The percentage and absolute counts of examples for which the standard realization and/or realization with chunking were successful or not.

## 4 Dataset

We use the 1214 release of WeScience (Ytrestøl et al., 2009), a fragment of a 2008 Wikipedia snapshot. It is a part of the Redwoods treebank (Oepen et al., 2004), so the analyses it contains are verified by humans as optimal for the original sentence.

Out of the entire dataset, we took 315 sentences which have DMRSs with more than 40 nodes and which can be chunked. There are on average 3.6 chunks per sentence (st. dev. 1.3, max. 12). We do not have space to illustrate the sentences here, but see `tinyurl.com/y9ghd35x`.

## 5 Coverage and performance

In the experiment we compare the results of the standard realization from a full sentence and realization from a chunked sentence (Table 1).

Realization with chunking allowed realization from some semantic graphs which do not produce a sentence using the standard ACE set-up. The coverage is about 9% higher overall. Some sentences cannot be realized with the new method even though the standard system works. This is because of the presence of grammatical structures not covered by the chunking algorithm, which lead to incorrect subgraphs. Limitations of the chunking algorithm are discussed in detail elsewhere (Muszyńska, 2016).

We investigated the performance of the two approaches in terms of time and memory usage. Fig. 2

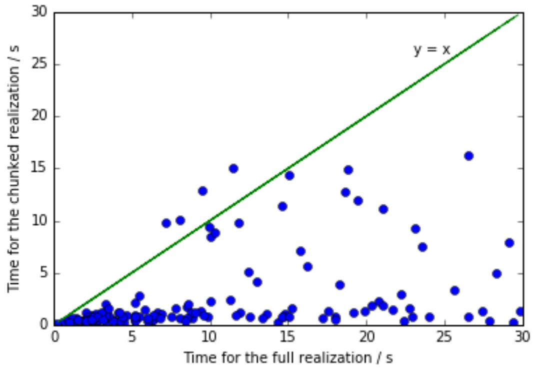

**Figure 2:** The time needed for realization with chunking against the time taken by the standard realization.

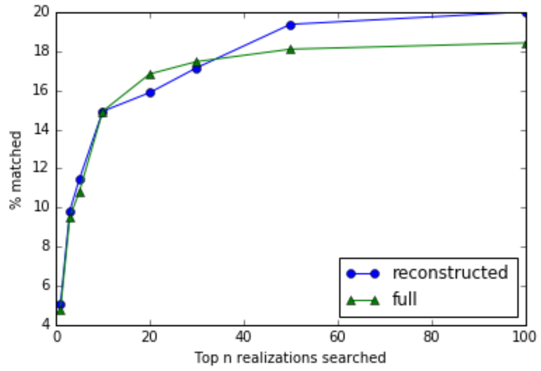

**Figure 3:** Percentage of exact matches in top $n$ realizations.

shows CPU time for all examples where sentences were successfully realized with both methods or where the standard realization failed without a time-out. The time measured for realization with chunking was the sum across all chunks. The maximum time needed for realization with chunking was 16s. The outliers in the upper half of the graph correspond to sentences where the standard realization failed or chunking was incorrect.

We also recorded the maximum memory used. For space reasons, we do not show the graph but its shape matches that for time, including the outliers. The maximum number of passive edges produced in chart generation also follows a similar pattern and is consistently smaller for chunking than for the standard realization.

The ACE generator ranks its results with a maximum entropy language model and assigns a score to each result. We assign a score to a result of realization with chunking by adding logarithms of scores of the constituent chunk realizations.

Following the original work on the ERG generator by Velldal (2008), we evaluate the ranking quality by comparing the top-ranked realization result with the original sentence on which the semantic representation was based. We use two metrics: the exact match percentage and the BLEU score.

We report an exact match in top n realizations if the original and realized surface strings are identical after removing capitalization and punctuation. Realization with chunking yields comparable results for all n for the examples where both methods produced results (Fig 3). In fact, it slightly overtakes the standard approach for $n \approx 40$ as some lower ranked realizations are produced only with chunking.

The BLEU score is evaluated only for the top ranked realizations, again after removing punctuation and capitalization. The average score for the standard realization is $0.79 \pm 0.14$ (st. dev.), and $0.77 \pm 0.15$ (st. dev.) for realization with chunking. The standard approach achieved a higher score for 17.1% examples, while realization with chunking scored higher for 12.4%. However, there is no statistically significant difference between the two approaches.

## 6 Conclusions

Chunking noticeably reduces the realization cost for long sentences without affecting the quality of results. In fact, some sentences can be realized only after applying chunking (given time-out). We expect that refinements in chunking will further improve the realization coverage. In future we will also investigate whether the chunking information can be used to improve realization ranking.

## Acknowledgements

The authors would like to thank Dan Flickinger for helpful discussions and Matic Horvat for assistance with resolving unknown words. The research is funded by EPSRC Doctoral Research Studentship (EP/M508007/1).

## References

John Carroll, Ann Copestake, Dan Flickinger, and Victor Poznanski. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the 7th European Workshop in Natural Language Generation (ENLG)*, pages 86–95.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal Recursion Semantics: An introduction. *Research on Language and Computation*, 3(2):281–332.

Ann Copestake, Guy Emerson, Michael Wayne Goodman, Matic Horvat, Alexander Kuhnle, and Ewa Muszyńska. 2016. Resources for building applications with Dependency Minimal Recursion Semantics. In *Proceedings of the Tenth Language Resources and Evaluation Conference (LREC '16)*.

Ann Copestake. 2002. *Implementing typed feature structure grammars*, volume 110 of *CSLI Lecture Notes*. Center for the Study of Language and Information, Stanford, CA.

Ann Copestake. 2009. Slacker semantics: Why superficiality, dependency and avoidance of commitment can be the right way to go. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 1–9, Athens, Greece.

Yimai Fang, Haoyue Zhu, Ewa Muszyńska, Alexander Kuhnle, and Simone Teufel. 2016. A proposition-based abstractive summariser. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics*, pages 567–578, Osaka, Japan.

Dan Flickinger, Emily M. Bender, and Stephan Oepen. 2014. Towards an encyclopedia of compositional semantics. Documenting the interface of the English Resource Grammar. In *Proceedings of the Ninth Language Resources and Evaluation Conference (LREC '14)*, pages 875–881, Reykjavik, Iceland.

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(1):15–28.

Dan Flickinger. 2016. Generating English paraphrases from logic. In Martijn Wieling, Martin Kroon, Gertjan Van Noord, and Gosse Bouma, editors, *From Semantics to Dialectometry*, chapter 11. College Publications.

Matic Horvat. 2017. *Hierarchical statistical semantic translation and realization*. Ph.D. thesis, University of Cambridge, Computer Laboratory.

Ewa Muszyńska. 2016. Graph- and surface-level sentence chunking. In *Proceedings of the ACL 2016 Student Research Workshop*, pages 93–99, Berlin, Germany.

Stephan Oepen, Dan Flickinger, Kristina Toutanova, and Christopher D. Manning. 2004. LinGO Redwoods. *Research on Language and Computation*, 2(4):575–596.

Erik Velldal. 2008. *Empirical Realization Ranking*. Ph.D. thesis, University of Oslo, Department of Informatics.

Michael White. 2004. Reining in CCG chart realization. In Anja Belz, Roger Evans, and Paul Piwek, editors, *Natural Language Generation. Lecture Notes in Computer Science*, volume 3123. Springer, Berlin, Heidelberg.

Xuchen Yao, Gosse Bouma, Yi Zhang, Paul Piwek, and Kristy Elizabeth Boyer. 2012. Semantics-based question generation and implementation. *Dialogue and Discourse*, 3(2):11–42.

Gisle Ytrestøl, Dan Flickinger, and Stephan Oepen. 2009. Extracting and annotating Wikipedia subdomains - towards a new eScience community resource. In *Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theories (TLT 7)*, pages 185–197, Groningen, The Netherlands.