Incremental Chart Parsing with Predictive Hints

Rolf Schwitter

Centre for Language Technology Macquarie University Sydney, NSW 2109, Australia schwitt@ics.mq.edu.au

Abstract

This paper describes an incremental chart parser that generates look-ahead categories on the fly for a controlled natural language. These predictive hints tell the author what kind of syntactic (or semantic) structure can follow the current input string and thereby aim at helping the author to reduce the cognitive burden to learn and remember the rules of the controlled language. The parser can handle modifications (insertion, deletion, and replacement) to the input string without the need to reparse the entire string. These modifications are a function of the size of the tokens changed rather than the size of the entire input.

1 Introduction

Over the last decade two types of controlled natural languages have been developed: human-oriented controlled languages and machine-oriented controlled languages (Huijsen, 1998; O'Brien, 2003).

The main goal of human-oriented controlled natural languages is to improve the readability and understandability of technical documents for human readers, especially non-native speakers (AECMA, 2001). Machine-oriented controlled natural languages, on the other hand, try to ease the translation process (Kamprath, 1998) and to facilitate the subsequent inference processes (Fuchs and Schwertel, 2003; Schwitter et al., 2003; Sukkarie, 2003). In general, a controlled natural language can be defined as a subset of a natural language that has been restricted with respect to its grammar and its lexicon. Grammatical restrictions usually result in less complex and less ambiguous texts. Lexical restrictions reduce the size of the vocabulary and the meaning of the lexical entries for a particular application domain.

To allow writing unambiguous and precise specifications, we have developed a machine-oriented controlled natural language (PENG – Processable ENGlish) for knowledge representation (Schwitter, 2002). Specifications written in PENG can be translated unambiguously into first-order predicate logic via discourse representation structures and can be automatically checked for consistency and informativeness with the help of third-party reasoning services (McCune, 2001; Bos, 2001; McCune, 2003; Bos, 2003).

It is well known that writing documents in a controlled natural language is hard and time-consuming without the support of intelligent writing assistance (Goyvaerts, 1996; Power et al., 2003). To ease the writing process and to guarantee wellformed syntactic structures, PENG uses a lookahead editor that displays after each word form a set of syntactic categories that inform the author how the input string can be continued (Schwitter et al., 2003).

The look-ahead editor of PENG communicates with a chart parser that processes a unificationbased grammar. The chart parser generates these predictive hints dynamically while the text is written and thereby enforces the restrictions placed upon the language. Apart from these hints, the chart parser also generates for each input string a discourse representation structure (Kamp and Reyle, 1993) as well as a paraphrase (Schwitter and Ljungberg, 2002) that reflects the interpretation of the machine in controlled natural language.

So far, it has not been possible to modify a substring of the text without having to reparse the entire text and regenerate the output (discourse representation structure and paraphrase) from scratch. This is unsatisfactory and demands for a more sophisticated approach to deal with modifications. It would be desirable to be able to edit the text during the writing process without the need for extensive reparsing. The chart parser should be able to handle modifications (insertion, deletion, and replacement) as efficiently as possible and to do its job in a piecemeal fashion constructing the representation of the text word by word and providing, at the same time, look-ahead categories that are contingent on the current input.

We will call such a chart parser that is capable of constructing a representation bit by bit and handling modifications without the need for exhaustive reparsing and reconstructing of the underlying representation an *incremental chart parser*.

Ideally, the time that the incremental parser spends to process a modification of arbitrary length (without violating the approved rules of the controlled language) should be proportional to the complexity of the change.

The reminder of this paper is organised in the following way: In Section 2, I will set up the requirements for an incremental chart parser. In Section 3, I will discuss the properties and shortcomings of a naïve chart parser for the task at hand. In Section 4, I will present the benefits of an incremental chart parser and show that the intended solution fulfills the requirements that have been specified in Section 2. In Section 5, I will present update handling algorithms for generating look-ahead categories and for dealing with modifications. In Section 6, I will evaluate the introduced algorithms and compare them with naïve reparsing. Finally, in Section 7, I will summarize the advantages of the presented approach and give some indicators for further research.

2 Requirements to an Incremental Chart Parser for PENG

PENG is a machine-oriented controlled natural language designed to write precise specifications and aims at supporting the knowledge acquisition process for various tasks (Schwitter, 2002). PENG

consists of a strict subset of standard English. The restrictions of the language are defined with the help of a controlled grammar and a controlled lexicon, and enforced by a look-ahead editor (Schwitter et al., 2003).

As shown in the dataflow diagram for the PENG system in Figure 1, sentences written in PENG are first sent to the controlled language (CL) processor and then translated into first-order predicate logic via discourse representation structures. This logical representation is subsequently checked for consistency and informativeness with the help of a theorem prover and a model builder.



Figure 1: Dataflow diagram for the PENG system

The look-ahead editor of PENG communicates with the CL processor (chart parser and unification-based grammar) via a socket interface. The CL processor is running as a client process and is connected via a server with a theorem prover (OT-TER; McCune, 2003) and a model builder (MACE; McCune 2001). The theorem prover and the model builder are both running separate client processes.

One of the deciding factors for the acceptability of a controlled language is the availability of automatic writing assistance. Texts in controlled natural language should not only be easy to write but also easy to modify without the need for time-consuming reprocessing.

This is why we need a processing strategy that informs the author about the permissible structure of the text and that supports basic editing operations in an efficient way.

2.1 Look-ahead categories

Given a set of grammar rules that define a controlled natural language such as PENG, the incremental chart parser should be able to generate a set of look-ahead categories after each word form that the author enters. For example, for the sentence

1. The customer inserts a credit card.

the following look-ahead categories should be generated (simplified here):

The [adjective | noun]

- ... customer [relpron | negation | verb]
- ... inserts [determiner]
- ... the [adjective | noun]
- ... credit ['card' | relpron | prep | conj | '. ']
- ... card [relpron | prep | conj |'.']

In this example, all the look-ahead categories are either lexical categories or word forms. However, the algorithm should be easily parameterizable so that predictive categories for various syntactic strata as well as for semantic information can be generated.

2.2 Insertion

The insertion operation should allow the addition of an arbitrary number of words into the input string as long as this modification does not violate the rules of the controlled language. For example, the insertion of the relative pronoun *who* into sentence I should lead to the noun phrase:

2. The customer who inserts the credit card ...

that is part of the controlled natural language PENG.

Note that the insertion results here in a categorial change; instead of a complete sentence we have now to deal with a complex noun phrase. This means that the punctuation mark introduced by the sentence needs to be removed automatically and a new set of look-ahead categories has to be generated and displayed for the last word form of the input string:

... card [relpron | prep | verb | conj]

As this example shows, the noun phrase in 2 could now be continued with a relative clause (3),

a prepositional phrase (4), a verb phrase (5), or a coordinated nominal structure (6):

- 3. ... that is valid owns a password.
- 4. ... into the slot owns a password.
- 5. ... owns a password.
- 6. ... and enters the PIN owns a password.

Each of these sentences constitutes a wellformed structure in the controlled natural language.

2.3 Deletion

The deletion operation should allow the cutting of an arbitrary sequence of words in the input string. For example, the deletion of the prepositional phrase *into the slot* in

6. The customer who inserts the credit card into the slot owns a password.

should result in a well-formed sentence in controlled language:

7. The customer who inserts the credit card owns a password.

The author should be able to delete a substring first by highlighting it and then by cutting it:

8. The customer who inserts the credit card into the slot owns a password.

This design decision reduces the complexity of the algorithm, since it results in one single cutting event that tells the parser when recomputation should be resumed.

2.4 Replacement

If the insertion and deletion operation are in place, then the infrastructure for the replacement operation exists. In essence, the replacement operation is a deletion followed by an insertion operation. For example, the replacement of the compound noun *credit card* with *MasterCard* in sentence *1* results in

9. The customer inserts the MasterCard.

It seems that a replacement operation is more complex than a deletion or an insertion operation, since these two operations need to be applied in sequence. However, as we will discuss in detail in Section 5, this is not the case, since the replacement operation can be implemented in a way that does not demand for extensive reprocessing.

3 Chart Parsing

Parsing is the process of analyzing the syntactic structure of an input string and has traditionally been understood as a batch-mode process.

The problem with any naïve parsing algorithm – independent of the parsing strategy – is the unnecessary repetition of work that will occur for processing any non-trivial grammar.

Suppose a top-down parser is attempting to parse the sentence:

10. The password is valid.

Given the following simple (context-free) phrase structure grammar

```
s → np, vp.
np → det, noun, rc.
np → det, noun.
rc → relpro, vp.
vp → verb, adj.
```

the parser will first attempt using the rule

 $np \rightarrow det$, noun, rc.

and then after failing with that rule, it will try the alternative rule

```
np \rightarrow det, noun.
```

That means the parser will repeat the work of analyzing the determiner (det) and the noun (noun) once for each rule.

An active chart parser (Kay, 1980; Gazdar and Mellish, 1989; Ferro and Pardo, 1995) avoids this repetition of work by storing information about well-formed substrings as well as information about substring hypotheses that it has partially explored in a table (chart). The chart parser can then look up these substrings in the chart and expand them – if necessary – instead of recomputing them.

Given an input string *I* and a grammar *G*, we can define a chart as a set of edges where an edge is a triple of the form $\langle v_i, v_b, R \rangle$. The first two elements v_i and v_t are integers and represent starting and ending vertices of *I* or of a substring of *I* and *R* represents a dotted rule. A dotted rule is a rule of the form $X \rightarrow \alpha \bullet \beta$ and corresponds to an *X* edge containing an analysis of confirmed constituents α that are seeking for constituents β .

For example, if $s \rightarrow np$, vp. is a rule of the grammar and sentence 10 is an input string, then

the first two dotted rules below represent unconfirmed hypotheses while the third rule represents a fully confirmed hypothesis:

<0,0,s \rightarrow • np vp> <0,2,s \rightarrow np • vp> <0,4,s \rightarrow np vp •>

Edges that correspond to unconfirmed hypotheses are known as *active edges* and those that correspond to confirmed hypotheses as *inactive edges*.

The basic operation of a chart parser involves combining an active edge with a completed inactive edge. The result is either a new inactive edge or a new active edge that spans both the active and inactive edges. This fundamental rule cannot be applied to a chart that contains no edges. Before anything can happen, an initialization process needs to set the chart up with inactive word edges and a rule invocation strategy needs to be defined that creates new active edges as a result of the application of the fundamental rule.

A chart parser usually uses an agenda to keep track of the edges that need to be processed. Such an agenda can be thought of as a list of edges. Adding new edges to the front of the agenda leads to a depth-first search strategy and adding them to the end would lead to breath-first search.

In our implementation, edges are stored in the following modified format:

edge(*ID*, *v*_s, *v*_b*LHS*, *RHSL*)

ID is an integer that stands for a sentence identifier. *LHS* represents the category on the left hand side of a dotted rule and *RHSL* represents a list of unconfirmed daughter categories on the right hand side of the rule. If *RHSL* is empty ([]), then the edge is inactive, otherwise active.

For example, a top-down chart parser will produce the following $edges^{1}$ for sentence 10 using our simplified grammar rules introduced above:

edge(0,0,s,[np,vp]) edge(0,0,np,[det,noun]) edge(0,1,np,[noun])

¹ The sentence identifier is not displayed in the edges. The categories of the grammar are atomic and do not contain any additional syntactic or semantic arguments. The grammar is not complete, since preterminal rules such as $det \rightarrow [the]$ are missing. As a consequence inactive edges such as edge(0, 1, det, []) do not appear in the simplified chart.

```
edge(0,2,np,[])
edge(0,2,s,[vp])
edge(2,2,vp,[verb,adj])
edge(2,3,vp,[adj])
edge(2,4,vp,[])
edge(0,4,s,[])
edge(0,0,np,[det,noun,rc])
edge(0,2,np,[noun,rc])
edge(0,2,np,[rc])
edge(2,2,rc,[relpro,vp])
```

Although such a batch-mode chart parser avoids repeating work and keeps active and passive edges in the chart, it cannot deal with modifications to the current input string without reprocessing the entire string.

4 Incremental Chart Parsing

An incremental chart parser, by contrast, can handle modifications to an input string that it has already parsed without having to reprocess the entire string from scratch. The key idea of incremental chart parsing is to use information about edge dependencies for keeping track of edges that have to be updated (Wirén, 1989; Wirén 1994).

Let us explore this idea by an example and then refine it. Suppose we modify sentence 10 by inserting the relative pronoun *that* between the noun phrase and the verb phrase, then we get a complex noun phrase as result:

11. The password that is valid ...

In comparison to the chart for sentence 10 in Section 3, the processing of this noun phrase results in 4 new edges

```
edge(2,3,rc,[vp])
edge(2,5,rc,[])
edge(0,5,np,[])
edge(5,5,vp,[verb,adj])
```

and in 4 modified edges (with modifications in bold face)

```
edge(3,3,vp,[verb,adj])
edge(3,4,vp,[adj])
edge(3,5,vp,[])
edge(0,5,s,[vp])
```

We can make the following observations when we compare the charts for sentence 10 and for the noun phrase 11 in more detail:

- The active edge edge(0,2,np,[rc]) for sentence 10 hypothesizing that the password was the beginning of a noun phrase followed by a relative clause has been expanded to an inactive edge edge(0,5,np,[]) to cover the relative clause in 11.
- All the edges that make up the noun phrase *the password* in sentence *10* remain unaffected by the modification.
- All the edges that make up the verb phrase *is* valid in sentence 10 remain unaffected apart from the indices of the vertices (displayed in bold face) that have been updated.
- The passive edge edge(0,4,s,[]) representing sentence *10* has been replaced by an active edge edge(0,5,s,[**vp**]), because *11* is a noun phrase and not a complete sentence.

In summary, we can state that there is no need to recompute an edge, if that edge does not in any way depend upon the vertices that have been changed or on any edges that were based on those edges.

A closer look into the chart for sentence 10 reveals that the edge edge (0, 4, s, []) is the only one that spans the vertex (insertion point) where the relative pronoun would be inserted. This suggests the following informal solution to process the modification:

- 1) Find all edges on the right hand side of the insertion point, in our case all those edges whose starting vertex is greater than or equal to the insertion point, and create a new subchart *CR* for them.
- 2) Renumber all starting and ending vertices of the edges in *CR* to be v_{s+1} and v_{t+1} .
- 3) Find all edges on the left hand side of the insertion point, in our case all those edges whose ending vertex is smaller than or equal to the insertion point, and create a new subchart *CL* for them.
- 4) Create a new chart *C* by appending the subchart *CR* to the end of the subchart *CL*.
- 5) Create new hypotheses beginning at the insertion point for the word form *that*.

6) Reparse the string, using only the new edges in the agenda and the new chart *C*.

Note that this solution automatically excludes edges such as edge(0, 4, s, []) from the new chart C, since we considered only edges that do not bridge the insertion point. At first glance, it seems that an optimization should be possible, since not all edges in the subcharts are affected by the editing operation. For example, only the modified edge edge(3, 5, vp, []) in the subchart CR spanning the verb phrase on the right hand side of the insertion point takes part in reparsing. Similar observations can be made for the subchart CL where only those edges that end at the insertion point are affected by reparsing. However, it turns out that first filtering the subchart CR and then reconstructing the entire chart after parsing is costly and does not result in a speed-up of parsing in comparison with the unfiltered version of the chart.

5 Update Handling Algorithms

After this informal discussion of the problem, I will give a more formal description of the update handling algorithms for finding look-ahead categories and for dealing with modification (insertion, deletion, and replacement).

5.1 Finding look-ahead categories

Look-ahead categories are generated after each word form that the author enters or whenever an approved modification results in a syntactic structure that needs to be completed by the author.

Formally, a set of look-ahead categories LC for a word w ending at vertex v_i can be calculated in the following way:

- 1) Find all active edges ending at v_i .
- 2) For each active edge:
 - a) Select the *RHSL* of remaining categories.
 - b) For the first category in *RHSL*, check if it is a lexical category:
 - i) If yes, then store the solution in *LC*.
 - ii) If not, find a rule that rewrites the category into further categories, then select the first category and return to 2b.

Apart from lexical categories, it is also possible to collect other categories, for example non-terminal categories by extracting them from the grammar rules, or semantic categories if they are stored in the lexicon and accessible via lexical categories. Collecting also look-ahead categories for non-terminal symbols in Step 2ii results in a list of hierarchically order categories and eases customization of this functionality for the user interface.

5.2 Editing operations

According to our definition, the incremental chart parser should not only be able to handle piecemeal additions to a string but also to handle arbitrary modifications efficiently. Ideally, the time that the incremental algorithm uses for processing a modification should be a function of the size of the modification rather then the size of the entire input. In simple words: a small modification should require less work than a big modification. Note that the algorithms presented below for the editing operations do not explicitly delete bridging edges but rather exclude them by reconstructing the chart.

Insertion

Inserting a word w at a vertex v_i in a string can be calculated in the following way:

- 1) Find all edges for which the index of the starting vertex v_s is greater than or equal to v_i , that is $s \ge i$, and create a new subchart *CR* for them.
- 2) For all the edges in *CR*
 - a) renumber the starting vertex to be v_{s+1} ,
 - b) renumber the ending vertex to be v_{t+1} .
- 3) Find all edges for which the index of the ending vertex v_t is smaller than or equal to v_i and the starting vertex v_s is not equal to v_i , that is $t \le i \land s \ne i$, and create a new subchart *CL*.
- 4) Create a new chart *C* by appending the subchart *CR* to the end of the subchart *CL*.
- 5) Create new hypotheses beginning at v_i for each category that the new word *w* belongs to.
- 6) Reparse the string, using only these new edges as the agenda for the parser, and providing it with the updated chart *C*.

Deletion

Deleting a word w at a vertex v_i in a string is to some extent similar to the reverse of inserting a word. The algorithm looks as follows:

- 1) Find all edges for which the index of the starting vertex v_s is greater than or equal to v_i , that is $s \ge i$, and create a new subchart *CR* for them.
- 2) For all the edges in *CR*
 - a) renumber the starting vertex to be v_{s-1} ,
 - b) renumber the ending vertex to be v_{t-1} .
- Find all edges for which the index of the ending vertex vt is smaller than vi and the starting vertex vs is not equal to vi, that is t < i ∧ s ≠ i, and create a new subchart *CL1*.
- 4) Find all edges for which the index of the ending vertex v_t is equal to v_i , that is t = i, and create a new subchart *CL2*.
- 5) Create a new chart *C* by appending the subchart *CR* to the end of the subchart *CL1*.
- 6) Reparse the string, using the subchart *CL2* as the agenda for the parser, and providing it with the subchart *C* as new chart.

Note that the agenda above consists of the subchart *CL2* in contrast to the insertion operation where the agenda for reparsing consists only of the new word hypothesis.

Replacement

Replacing a word w at a vertex v_i in a string can be described as a deletion followed by an insertion operation. This is what authors do when they replace a word in an interactive text editor. They first delete the word and then insert a new word.

However, simply executing these two operations in sequence would not be very efficient. For example, if the word *customer* is replaced with *client* in sentence 10, then all words lie between the same vertices as they did before the replacement operation. In this case, the chart does not need to be partially recreated twice, since the first recreation will renumber vertices and create edges that will immediately be reset or deleted again.

This observation results in the following optimized algorithm:

- Create new hypotheses beginning at v_i for each category *Cat* that the new word w belongs to.
- Replace the inactive word edge *E* in the chart starting at vertex v_i with the new word edge *E*' so that the categories of *E* and *E*' are identical.

This is a significant improvement over executing the two operations in sequence.

6 Evaluation

The presented algorithms that incrementally update the chart result in a speed-up for all modification operations compared with naïve reparsing of the input string after an editing operation. The average improvement for insertion is of a factor of 1.44, for deletion 1.28, and for replacement 17.64. As the results show, replacement can be implemented very efficiently. As already mentioned, additional filtering of the subchart *CR* does not result in any speed-up. It seems that in our Prolog implementation renumbering of the vertices in the affected edges is the biggest cost factor, since this involves arithmetic operations and not pure unification.

7 Conclusion

In this paper, I discussed an incremental chart parser that generates predictive hints and allows for arbitrary editing operation as long as the result is an approved structure in controlled natural language. The generated look-ahead categories consist of syntactic (or semantic) categories and aim at supporting the writing process of the controlled natural language. These predictive hints ensure that the author follows the rules of the controlled natural language and guarantee unambiguous and precise texts (in our case "seemingly informal" specifications).

The editing operations (insertion, deletion, replacement) are bound to the affected part of the string and require only minimal reparsing. This means that the modifications are a function of the size of the words changed rather than the size of the entire text. The current solution deals only with local updates. In the future, I would like to look into the problem of updating anaphoric references in the text and in the underlying discourse representation structure after a nominal expression has been modified. The goal is to find a solution that does not require extensive reparsing of the input text.

Acknowledgements

I would like to thank the students (Jason Barles, James Cameron, Dennis Chan, Mary Gardiner, Hogan Ho, and Simon Kissane) who were enrolled in the honours class *COMP448: Advanced Topics in Natural Language Processing* in the summer semester 2003 at Macquarie University for their enthusiasm in exploring various directions in incremental chart parsing. I also thank two anonymous reviewers of this paper for their constructive comments.

References

- AECMA. 2001. The European Association of Aerospace Industries. AECMA Simplified English, AECMA Document PSC-85-16598. A Guide for the Preparation of Aircraft Maintenance Documentation in the International Aerospace Maintenance Language. Issue 1, Revision 2, 15 January.
- J. Bos. 2001. DORIS 2001: Underspecification, Resolution and Inference for Discourse Representation Structures, in: Blackburn and Kohlhase (eds.): IcoS-3. Inference in Computational Semantics. Workshop Proceedings, Siena, Italy, June.
- J. Bos. 2003. Exploring Model Building for Natural Language Understanding. Proceedings of the Fourth International Workshop on Inference in Computational Semantics (ICoS-4), September 25-26, INRIA Lorraine, Nancy.
- M. V. Ferro, M. A. A. Pardo. 1995. Exploring interactive chart parsing. Procesamiento del Lenguaje Natural, Bilbalo, Spain, pp. 158-172.
- N. E. Fuchs, U. Schwertel. 2003. Reasoning in Attempto Controlled English. Proceedings of the Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2003), Mumbai, India.
- G. Gazdar, C. Mellish. 1989. Natural Language Processing in Prolog. An Introduction to Computational Linguistics. Addison-Wesley, Wokingham, England.
- P. Goyvaerts. 1996. Controlled English, Curse or Blessing? – A User's Perspective. Proceedings of the First International Workshop on Controlled Language Applications, March 26-27, Leuven.
- W. O. Huijsen. 1998. Controlled Language An Introduction. Proceedings of CLAW 1998, Pittsburgh, pp. 1-15.
- H. Kamp, U. Reyle. 1993. From Discourse to Logic. Kluwer, Dordrecht.
- C. Kamprath, T. Mitamura, E. Nyberg. 1998. Controlled Language for Multilingual Document Production: Experience with Caterpillar Technical English, Proceedings of the Second International Workshop on Controlled Language Applications, Pittsburgh.

- M. Kay 1980. Algorithm Schemata and Data Structures in Syntactic Processing. Report CSL-80-12, Xerox Parc, Palo Alto, California.
- W. McCune. 2001. MACE 2.0 Reference Manual and Guide. ANL/MCS-TM-249. Mathematics and Computer Science Division, Technical Memorandum No. 249, Argonne National Laboratory, Argonne.
- W. McCune. 2003. OTTER 3.3 Reference Manual. ANL/MCS-TM-263, Mathematics and Computer Science Division, Technical Memorandum No. 263, Argonne National Laboratory, Argonne.
- S. O'Brien. 2003. Controlling Controlled English, An Analysis of Several Controlled Language Rule Sets. Proceedings of EAMT-CLAW03, Controlled Language Translation, May 15-17, Dublin City University, pp. 105-114.
- R. Power, D. Scott, A. Hartley. Multilingual Generation of Controlled Languages. Proceedings of EAMT-CLAW03, Controlled Language Translation, May 15-17, Dublin City University, pp. 115-123.
- R. Schwitter. 2002. English as a Formal Specification Language. Proceedings of the Thirteenth International Workshop on Database and Expert System Applications (DEXA 2002), Aix-en-Provence, France, pp. 228-232.
- R. Schwitter, A. Ljungberg. 2002. How to write a document in controlled natural language. Proceedings of the Seventh Australasian Document Computing Symposium, Sydney, Australia, December, pp. 133-136.
- R. Schwitter, A. Ljungberg, D. Hood. 2003. ECOLE: A Look-ahead Editor for a Controlled Language. Proceedings of EAMT-CLAW03, Controlled Language Translation, May 15-17, Dublin City University, pp. 141-150.
- J. Z. Sukkarieh. 2003. Mind your Language! Controlled Language for Inference Purposes. Proceedings of EAMT-CLAW03, Controlled Language Translation, May 15-17, Dublin City University, pp. 160-169.
- M. Wirén. 1989. Interactive incremental chart parsing. Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics, Manchester, England, pp. 241-248.
- M. Wirén. 1994. Bounded incremental chart parsing. Clause Report 36. Universität des Saarlandes, Saarbrücken, Germany.