RUBISC - a Robust Unification-Based Incremental Semantic Chunker

Michaela Atterer
Department for Linguistics
University of Potsdam
atterer@ling.uni-potsdam.de

David Schlangen
Department for Linguistics
University of Potsdam
das@ling.uni-potsdam.de

Abstract

We present RUBISC, a new incremental chunker that can perform incremental slot filling and revising as it receives a stream of words. Slot values can influence each other via a unification mechanism. Chunks correspond to sense units, and end-of-sentence detection is done incrementally based on a notion of semantic/pragmatic completeness. One of RUBISC's main fields of application is in dialogue systems where it can contribute to responsiveness and hence naturalness, because it can provide a partial or complete semantics of an utterance while the speaker is still speaking. The chunker is evaluated on a German transcribed speech corpus and achieves a concept error rate of 43.3% and an F-Score of 81.5.

1 Introduction

Real-time NLP applications such as dialogue systems can profit considerably from incremental processing of language. When syntactic and semantic structure is built on-line while the speech recognition (ASR) is still working on the speech stream, unnatural silences can be avoided and the system can react in a faster and more user-friendly way. As (Aist et al., 2007) and (Skantze and Schlangen, 2009) show, such incremental systems are typically preferred by users over non-incremental systems.

To achieve incrementality, most dialogue systems employ an incremental chart parser (cf. (Stoness et al., 2004; Seginer, 2007) etc.). However, most existing dialogue systems operate in very limited domains, e.g. moving objects, people, trains etc. from one place to another (cf. (Aist et al., 2007), (Skantze, 2007), (Traum et al., 1996)). The complexity of the semantic representations needed is thus limited. Moreover, user behaviour (ungrammatical sentences, hesitations, false starts) and error-prone ASR require the parsing process to be robust.[1] We argue that obtaining relatively flat semantics in a limited domain while needing exigent robustness calls for investigating shallower incremental chunking approaches as alternatives to CFG or dependency parsing. Previous work that uses a combination of shallow and deep parsing in dialogue systems also indicates that shallow methods can be superior to deep parsing (Lewin et al., 1999).

The question addressed in this paper is how to construct a chunker that works incrementally and robustly and builds the semantics required in a dialogue system. In our framework chunks are built according to the semantic information they contain while syntactic structure itself is less important. This approach is inspired by Selkirk's sense units (Selkirk, 1984). She claims such units to be relevant for prosodic structure and different to syntactic structure. Similarly, (Abney, 1991) describes some characteristics of chunks as follows—properties which also make them seem to be useful units to be considered in spoken dialogue systems:

"when I read a sentence, I read it a chunk at a time. [...] These chunks correspond in some way to prosodic patterns. Chunks also represent a grammatical watershed of sorts. The typical chunk consists of a single content word surrounded by a constellation of function words, matching a fixed template. By contrast, the relationships between chunks are mediated more by lexical selection

[1] cf. The incremental parser in (Skantze, 2007) can jump over a configurable number of words in the input.

Proceedings of EACL 2009 Workshop on Semantic Representation of Spoken Language - SRSL 2009, pages 66–73,
Athens, Greece, 30 March 2009.

than by rigid templates. [...] and the order in which chunks occur is much more flexible than the order of words within chunks."

In our approach chunks are built incrementally (one at a time) and are defined semantically (a sense unit is complete when a slot in our template or frame semantics can be filled). Ideally, in a full system, the definition of their boundaries will also be aided by prosodic information. The current implementation builds the chunks or sense units by identifying a more or less fixed sequence of content and function words, similar to what Abney describes as a fixed template. The relationships between the units are mediated by a unification mechanism which prevents selectional restrictions from being violated. This allows the order of the sense units to be flexible, even as flexible as they appear in ungrammatical utterances. This unification mechanism and the incremental method of operation are also the main difference to Abney's work and other chunkers.

In this paper, we first present our approach of chunking, show our grammar formalism, the main features of the chunker (unification mechanism, incrementality, robustness), and explain how the chunker can cope with certain tasks that are an issue in dialogue systems, such as online utterance endpointing and revising hypotheses. In Section 3, we evaluate the chunker on a German corpus (of transcribed spontaneous speech) in terms of concept error rate and slot filling accuracy. Then we discuss related work, followed by a general discussion and the conclusion.

2 Incremental Chunking

Figure 1 shows a simple example where the chunker segments the input stream incrementally into semantically relevant chunks. The figure also displays how the frame is being filled incrementally. The chunk grammar developed for this work and the dialogue corpus used were German, but we give some examples in English for better readability.

As time passes the chunker receives more and more words from the ASR. It puts the words in a queue and waits until the semantic content of the accumulated words is enough for filling a slot in the frame semantics. When this is the case the chunk is completed and a new chunk is started. At the same time the frame semantics is updated if slot unification (see below) is possible and a check

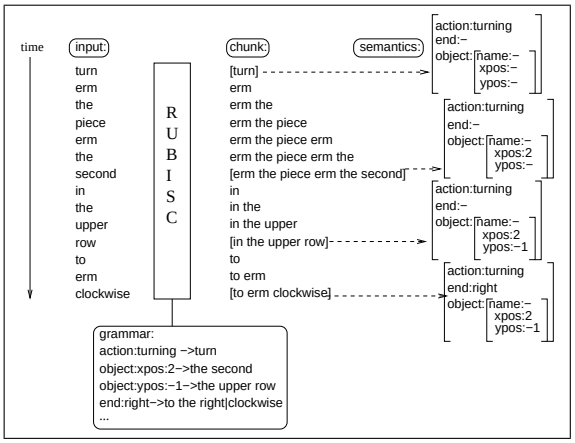

Figure 1: Incremental robust sense unit construction by RUBISC.

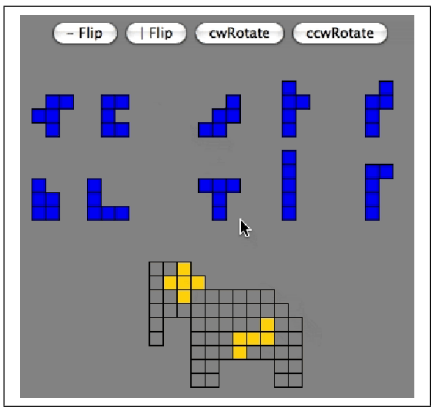

Figure 2: Puzzle-task of the corpus used for grammar building and testing.

whether the utterance is complete is made, so that the chunker can be restarted for the next utterance if necessary.

2.1 A Regular Grammar for Semantics

The grammar we are using for the experiments in this paper was developed using a small corpus of German dialogue (Siebert and Schlangen, 2008), (Siebert, 2007). Figure 2 shows a picture of the task that the subjects completed for this corpus.[2] A number of pentomino pieces were presented. The pieces had to be moved into an animal-shaped figure. The subjects were shown partly completed puzzles and had to give concise and detailed verbal instructions of the next move that had to be done. The locations inside this figure were usually referred to in terms of body parts (*move the x into*

[2]For the corpus used here the difference was that the button labels were German and that the pentomino pieces were not ordered in two rows. For better readability, we show the picture with the English labels.

the head of the elephant).

For such restricted tasks, a simple frame semantics seems sufficient, representing the action (grasping, movement, flipping or turning of an object), the object that is involved, and where or in which position the object will end up. In our current grammar implementation the object can be described with three attributes: `name` is the name of the object. In our domain, the objects are pentomino-pieces (i.e., geometrical forms that can be built out of five squares) which have traditional letter names such as `x` or `w`; the grammar maps other descriptions such as *cross* or *plus* to such canonical names. A piece can also be described by its current position, as in *the lower piece in the third column*. This is covered by the attributes `xpos` and `ypos` demarking the x-position and y-position of a piece. The x- or y-position can be a positive or negative number, depending on whether the description counts from left or right, respectively.

The possible slots must be defined in the grammar file in the following format:

```
@:action
@:entity:name
@:entity:xpos
@:entity:ypos
@:end
```

(That is: `definition marker @:level 1: (optional) level 2`.)

The position where or in which the piece ends up could also be coded as a complex entry, but for simplicity's sake (in the data used for evaluation, we have a very limited set of end positions that would each be described by just one attribute respectively), we restrict ourselves to a simple entry called `end` which takes the value of a body part (*head, back, leg1* etc.) in the case of movement, and the value of a direction or end position *horizontal, vertical, right, left* in the case of a turning or flipping action. It will be (according to our current grammar) set to *empty* in the case of a grasping action, because grasping does not specify an end position. This will also become important later, when unification comes into play. Figure 3 shows a part of the German grammar used with approximate translations (in curly brackets) of the right-hand side into English. The English parts in curly brackets is meta-notation and not part of the grammar file. Note that one surface string can determine the value of more than one semantic slot. The grammar used in the experiments in this paper had 97 rules.

```
action:grasping,end:empty -> nimm|nehme
                                        {take}
action:turning  -> drehe?          {turn}
action:flipping -> spieg(le|el)    {flip}
action:movement -> bewegt          {moved}
action:turning  -> gedreht         {turned}
entity:name:x -> kreuz|plus|((das|ein) x)
                        {cross|pluss|((the|an) x)}
entity:name:w -> treppe|((das|ein) w$)
                              {staircase|(the|a) w}
entity:name:w -> (das|ein) m$
                                   {(the|an) m}
entity:name:z -> (das|ein) z$
                                   {(the|a) z}
end:head      -> (in|an) den kopf
                              {(on|in) the head}
end:leg2 -> ins?  das (hinterbein|hintere
 bein|rechte bein|zweites bein) {in the hindleg|
                        back leg|right leg| second leg}
entity:ypos:lower -> der (unteren|zweiten)
                        reihe {(lower|second) row}
entity:xpos:1  -> das erste     {the first}
entity:ypos:-1 -> das letzte    {the last}
end:horizontal,action:flipping -> horizontal
                                        {horizontally}
```

Figure 3: Fragment of the grammar file used in the experiments (with English translations of the patterns for illustration only).

2.2 Unification

Unification is an important feature of RUBISC for handling aspects of long-distance dependencies and preventing wrong semantic representations. Unification enables a form of 'semantic specification' of verb arguments, avoiding that the wrong arguments are combined with a given verb. It also makes possible that rules can check for the value of other slots and hence possibly become inapplicable. The verb *move*, for instance, ensures that `a`ction is set to *movement*. For the utterance *schieb das äh das horizontal äh liegt ins Vorderbein* (*move that uh which is horizontal into the front leg*). The `action`-slot will be filled with *movement* but the `end`-slot remains empty because *horizontal* as an end fits only with a flipping action, and so is ignored here. Figure 4 illustrates how the slot unification mechanism works.

2.3 Robustness

The chunker meets various robustness requirements to a high degree. First, pronunciation variants can be taken account of in the grammar in a very flexible way, because the surface string or terminal symbols can be expressed through regu-

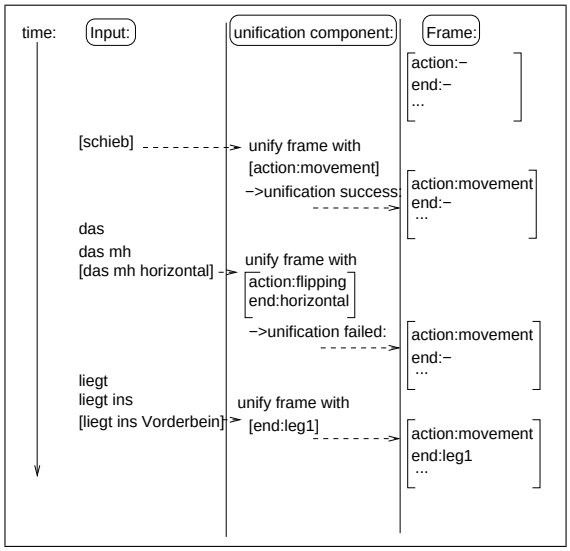

Figure 4: Example of slot unification and failure of unification.

lar expression patterns. *move* in German for instance can be pronounced with or without a final *-e* as *bewege* or *beweg*. *flip* (*spiegle* can be pronounced with or without *-el*-inversion at the end. Note, that this is due to the performance of speakers in our corpus and does not necessarily reflect German grammar rules. A system, however, needs to be able to cope with performance-based variations.

Disfluencies are handled through how the chunker constructs chunks as sense units. First, the chunker only searches for relevant information in a chunk. Irrelevant information such as an initial *uh* in *uh second row* is put in the queue, but ignored as the chunker picks only *second row* as the semantically relevant part. Furthermore the chunker provides a mechanism that allows it to jump over words, so that *second row* will be found in *the second uh row* and *the cross* will be found in *the strange cross*, where *strange* is an unknown word.

2.4 Incrementality

One of the main features of RUBISC is its incrementality. It can receive one word at a time and extract semantic structure from it. Incrementality is not strict here in the sense of (Nivre, 2004), because sometimes more than one word is needed before parts of the frame are constructed and output: *into the right*, for instance, needs to wait for a word like *leg* that completes the chunk. We don't necessarily consider this a disadvantage, though, as our chunks closely correlate to the minimal bits of information that can usefully be reacted to. In our corpus the first slot gets on average filled after 3.5 words (disregarding examples where no slots are filled). The average utterance is 12.4 words long.

2.5 End-of-Sentence Detection

An incremental parser in a dialogue system needs to know when to stop processing a sentence and when to start the next one. This can be done by using prosodic and syntactic information (Atterer et al., 2008) or by checking whether a syntactic S-node is complete. Since RUBISC builds sense units, the completeness of an utterance can be defined as semantic-pragmatic completeness, i.e. by a certain number of slots that must be filled. In our domain, for instance, it makes sense to restart the chunker when the action and end slot and either the name slot or the two position slots are filled.

2.6 History

The chunker keeps a history of the states of the frames. It is able to go back to a previous state when the incremental speech recognition revokes a word hypothesis. As an example consider the current word hypothesis to be *the L*. The slot entity name will be filled with *l*. Then the speech recognition decides to change the hypothesis into *the elephant*. This results in clearing the slot for entity name again.

3 Evaluation

The sense unit chunker was evaluated in terms of how well it performed in slot filling on an unseen part of our corpus. This corpus comes annotated with utterance boundaries. 500 of these utterances were manually labelled in terms of the semantic slots defined in the grammar. The annotators were not involved in the construction of the chunker or grammar. The annotation guidelines detailed the possible values for each slot. The entity names had to be filled in with the letter names of the pieces, the end slot with body parts or *right, left, horizontal* etc., and the position slots with positive and negative numbers.[3] The chunker was then run on 400 of these utterances and the slot values were compared with the annotated frames. 100 of the labelled utterances and 50 additional utter-

[3] In a small fraction (21) of the 500 cases an utterance actually contained 2 statements that were combined with *und/and*. In these cases the second statement was neglected.

ances were used by the author for developing the grammar.

We examined the following evaluation measures:

- the concept error (**concept err**) rate (percentage of wrong frames)
- the percentage of complete frames that were correct (**frames corr**)
- the percentage of **slots** that were **corr**ect
- the percentage of **action** slots **corr**ect
- the percentage of **end** slots **corr**ect
- the percentage of object:**name** slots **corr**ect
- the percentage of object:**xpos** slots **corr**ect
- the percentage of object:**ypos** slots **corr**ect

The results are shown in Table 1. We used a very simple baseline: a system that does not fill any slots. This strategy still gets 17% of the frames right, because some utterances do not contain any real content. For the sentence *Also das ist recht schwer* (Trans: *That's quite difficult.*), for instance, the gold standard semantic representation would be: `{action:None, end:None, object:{xpos:None, name:None, ypos:None}}`. As the baseline 'system' always returns the empty frame, it scores perfectly for this example sentence. We are aware that this appears to be a very easy baseline. However, for some slots, such as the xpos and ypos slots it still turned out to be quite hard to beat this baseline, as wrong entries were common for those slots. The chunker achieves a frame accuracy of 54.5% and an overall slot filling accuracy of 86.80% (compared to 17% and 64.3% baseline). Of the individual slots the action slot was the one that improved the most. The position slots were the only ones to deteriorate. As 17% of our utterances did not contain any relevant content, i.e. the frame was completely empty, we repeated the evaluation without these irrelevant data. The results are shown in brackets in the table.

To check the impact of the unification mechanism, we performed another evaluation with this mechanism turned off, i.e. slots are always filled when they are empty without regarding other slots. In the second step in Figure 4, the end slot would hence be filled. This resulted in a decline in performance as can also be seen in Table 1. We also turned off robustness features to test for their impact. Surprisingly, turning off the skipping of one word within a string specified by a grammar rule (as in *to erm clockwise*), did not have an effect on the results on our corpus. When we also turn off allowing initial material (*erm the piece*), however, performance drops considerably.

We also tested a variant of the system *RUBISC-o* (for *RUBISC-overlap*) which considers overlapping chunks: *Take the third piece* will result in `xpos:3` for the original chunker, even if the utterance is continued with *from the right*. *RUBISC-o* also considers the previous chunk *the third piece* for the search of a surface representation. In this case, it overwrites `3` with `-3`. In general, this behaviour improves the results.[4]

To allow a comparison with other work that reports recall and precision as measures, we also computed those values for RUBISC: for our test corpus recall was 83.47% and precision was 79.69% (F-score 81.54). A direct comparison with other systems is of course not possible, because the tasks and data are different. Nevertheless, the numbers allow an approximate feel of how well the system performs.

To get an even better idea of the performance, we let a second annotator label the data we tested on; inter-annotator agreement is given in Table 1. The accuracy for most slots is around 90% agreement beween annotators. The concept error rate is 32.25%. We also examined 50 utterances of the test corpus for an error analysis. The largest part of the errors was due to vocabulary restrictions or restrictions in the regular expressions: subjects used names for pieces or body parts or even verbs which had not been seen or considered during grammar development. As our rules for end positions contained pronouns like (*into the back*), they were too restricted for some description variants (*such that it touches the back*). Another problem that appears is that descriptions of starting positions can be confounded with descriptions of end positions. Sometimes subjects refer to end positions not with body parts but with *at the right side* etc. In some cases this leads to wrong entries in the object-position slots. In some cases a full parser might be helpful, but not always, because some expressions are syntactically ambiguous: *füge das Teil ganz rechts in das Rechteck ein.* (*put the piece on the right into the square/put the piece into the square on the right.*) A minority of errors was also

[4]Testing significance, there is a significant difference between RUBISC and the baseline, and RUBISC and RIBISC *w/o rob* (for all measures except *xpos* and *ypos*). The other variants show no significance compared with RUBISC but clear tendencies in the directions described above.

	baseline	RUBISC	w/o unif	w/o rob	RUBISC-o	i-annotator
concept err	83.0 (100)	45.5 (44.6)	49.5 (49.7)	73.3 (85.5)	43.3 (42.8)	32.3 (35.5)
frames corr	17.0 (0)	54.5 (55.4)	50.3 (50.3)	26.8 (14.5)	56.8 (57.2)	67.8 (64.5)
slots corr	64.3 (57.0)	86.8 (87.2)	84.6 (84.5)	78.8 (74.9)	87.6 (87.6)	92.1 (91.5)
action corr	27.8 (13.0)	90.3 (92.2)	85.8 (86.7)	64.3 (57.5)	89.8 (90.7)	89.0 (88.6)
end corr	68.0 (61.4)	85.8 (87.3)	81.0 (81.6)	73.8 (69.0)	85.5 (87.0)	95.8 (95.1)
name corr	48.8 (38.3)	86.3 (88.3)	84.5 (86.1)	79.0 (76.2)	86.5 (88.0)	86.8 (85.8)
xpos corr	87.5 (84.9)	83.0 (80.7)	83.0 (80.7)	86.5 (83.7)	85.5 (83.4)	94.5 (94.0)
ypos corr	89.5 (87.3)	88.8 (87.3)	88.8 (87.3)	90.3 (88.3)	90.5 (88.9)	94.5 (94.0)

Table 1: Evaluation results (in %) for RUBISC in comparison with the baseline, RUBISC without unification mechanism (w/o unif), without robustness (w/o rob), RUBISC with overlap (RUBISC-o), and inter-annotator aggreement (i-annotator). See the text for more information.

due to complex descriptions (*the damaged t where the right part has dropped downwards* – referring to the f), transcription errors (*recht* statt *rechts*) etc.

4 Related Work

Slot filling is used in dialogue systems such as the Ravenclaw-Olympus system[5], but the slots are filled by using output from a chart parser (Ward, 2008). The idea is similar in that word strings are mapped onto semantic frames. A filled slot, however, does not influence other slots via unification as in our framework, nor can the system deal with incrementality. This is also the main difference to systems such as Regulus (Rayner et al., 2006). Our unification is carried out on filled slots and in an incremental fashion. It is not directly specified in our grammar formalism. The chunker rather checks whether slot entries suggested by various independent grammar rules are unifiable.

Even though not incremental either, the approach by (Milward, 2000) is similar in that it can pick information from various parts of an utterance; for example, it can extract the arrival time from sentences like *I'd like to arrive at York now let's see yes at 3pm*. It builds a semantic chart using a Categorial grammar. The entries of this chart are then mapped into slots. A number of settings are compared and evaluated using recall and precision measures. The setting with the highest recall (52%) achieves a precision of 79%. The setting with the highest precision (96%) a recall of 22%. These are F-scores of 62.7 and 35.8 respectively.

(Aist, 2006) incrementally identifies what they call 'pragmatic fragments', which resemble the sense units produced in this paper. However, their system is provided with syntactic labels and the idea is to pass those on to a parser (this part appears to not be implemented yet). No evaluation is given.

(Zechner, 1998) also builds frame representations. Contrary to our approach, semantic information is extracted in a second step after syntactic chunks have been defined. The approach does not address the issue of end of sentence-detection, and also differs in that it was designed for use with unrestricted domains and hence requires resources such as WordNet (Miller et al., 1993). Depending on the WordNet output, usually more than one frame representation is built. In an evaluation, in 21.4% of the cases one of the frames found is correct. Other approaches like (Rose, 2000) also need lexicons or similar resources.

(Helbig and Hartrumpf, 1997) developed an incremental word-oriented parser for German that uses the notion of semantic kernels. This idea is similar in that increments correspond to constituents that have already been understood semantically. The parser was later on mainly used for question answering systems and, even though strongly semantically oriented, places more emphasis on syntactic and morphological analysis and less on robustness than our approach. It uses quite complex representations in the form of multi-layered extended semantic networks.

Finally, speech grammars such as JSFG[6] are similar in that recognition patterns for slots like 'action' are defined via regular patterns. The main differences are non-incrementality and that the result of employing the grammar is a legal sequential string for each individual slot, while our grammar

[5] `http://www.ravenclaw-olympus.org/`

[6] `java.sun.com/products/java-media/speech/forDevelopers/JSGF/`

also encodes, what is a legal (distributed) combination of slot entries.

5 Discussion and Future Work

The RUBISC chunker presented here is not the first NLU component that is robust against unknown words or structures, or non-grammaticalities and disfluencies in the input, nor the first that works incrementally, or chunk-based, or focusses predominantly on semantic content instead of syntactic structure. But we believe that it is the first that is all of this combined, and that the combination of these features provides an advantage—at least for the domains that we are working on. The novel combination of unification and incrementality has the potential to handle more phenomena than simple key word spotting. Consider the sentence: *Do not take the piece that looks like an* `s`*, rather the one that looks like a* `w`. The idea is to introduce a negation slot or flag, that will be set when a negation occurs. *nicht das s* (not the s) will trigger the flag to be set while at the same time the name slot is filled with `s`. This negation slot could then trigger a switch of the mode of integration of new semantic information from unification to overwriting. We will test this in future work.

One of the main restrictions of our approach is that the grammar is strongly word-oriented and does not abstract over syntactic categories. Its expressive power is thus limited and some extra coding work might be necessary due to the lack of generalization. However, we feel that this is mediated by the simplicity of the grammar formalism. A grammar for a restricted domain (and the approach is mainly aiming at such domains) like ours can be developed within a short time and its limited size also restricts the extra coding work. Another possible objection to our approach is that handcrafting grammars like ours is costly and to some extent arbitrary. However, for a small specialized vocabulary as is typical for many dialogue systems, we believe that our approach can lead to a good fast-running system in a short developing time due to the simplicity of the grammar formalism and algorithm, which makes it easier to handle than systems that use large lexical resources for complexer domains (e.g. tutoring systems). Other future directions are to expand the unification mechanism and grammar formalism such that alternatives for slots are possible. This feature would allow the grammar writer to specify that `e`nd:right requires a turning action *or* a flipping action.

6 Conclusion

We presented a novel framework for chunking. The main new ideas are that of incremental chunking and chunking by sense units, where the relationship between chunks is established via a unification mechanism instead of syntactic bounds, as in a full parsing approach. This mechanism is shown to have advantages over simple keyword spotting. The approach is suitable for online end-of-sentence detection and can handle revised word hypotheses. It is thus suitable for use in a spoken dialogue system which aims at incrementality and responsiveness. Nevertheless it can also be used for other NLP applications. It can be used in an incremental setting, but also for non-incremental tasks. The grammar format is easy to grasp, and the user can specify the slots he wants to be filled. In an evaluation it achieved a concept error rate of 43.25% compared to a simple baseline of 83%.

7 Acknowledgement

This work was funded by the DFG Emmy-Noether grant SCHL845/3-1. Many thanks to Ewan Klein for valuable comments. All errors are of course ours.

References

Steven Abney. 1991. Parsing by chunks. In *Principle-based Parsing: Computation and Psycholinguistics*, volume 44 of *Studies in Linguistics and Philosophy*. Kluwer.

Gregory Aist, James Allen, Ellen Campana, Carlos Gomez Gallo, Scott Stoness, Mary Swift, and Michael K. Tanenhaus. 2007. Incremental understanding in human-computer dialogue and experimental evidence for advantages over nonincremental methods. In *Decalog 2007*, Trento, Italy.

Gregory S. Aist. 2006. Incrementally segmenting incoming speech into pragmatic fragments. In *The Third Midwest Computational Linguistics Colloquium (MCLC-2006)*, Urbana, USA.

Michaela Atterer, Timo Baumann, and David Schlangen. 2008. Towards incremental end-of-utterance detection in dialogue systems. In *Proceedings of Coling 2008*, Manchester, UK.

Hermann Helbig and Sven Hartrumpf. 1997. Word class functions for syntactic-semantic analysis. In

Proceedings of the 2nd International Conference on Recent Advances in Natural Language Processing (RANLP'97).

I. Lewin, R. Becket, J. Boye, D. Carter, M. Rayner, and M. Wiren. 1999. Language processing for spoken dialogue systems: is shallow parsing enough? In *Accessing Information in Spoken Audio: Proceedings of ESCA ETRW Workshop*, Cambridge, USA.

George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller. 1993. Five papers on wordnet. Technical report, Princeton University.

David Milward. 2000. Distributing representation for robust interpretation of dialogue utterances. In *Proceedings of ACL 2000*, pages 133–141.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain, July. Association for Computational Linguistics.

M. Rayner, B.A. Hockey, and P. Bouillon. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Press, Chicago.

Carolyn P. Rose. 2000. A framework for robust semantic interpretation. In *Procs of NACL*.

Yoav Seginer. 2007. Fast unsupervised incremental parsing. In *Proceedings of ACL*, Prague, Czech Republic.

E. Selkirk. 1984. *Phonology and Syntax. The relation between sound and structure*. MIT Press, Cambridge, USA.

Alexander Siebert and David Schlangen. 2008. A simple method for resolution of definite reference in a shared visual context. In *Procs of SIGdial*, Columbus, Ohio.

Alexander Siebert. 2007. Maschinelles Lernen der Bedeutung referenzierender und relationaler Ausdrücke in einem Brettspieldialog. Diploma Thesis, University of Potsdam.

Gabriel Skantze and David Schlangen. 2009. Incremental dialogue processing in a micro-domain. In *Proceedings of EACL 2009*, Athens, Greece, April.

Gabriel Skantze. 2007. *Error Handling in Spoken Dialogue Systems*. Ph.D. thesis, KTH, Stockholm.

Scott C. Stoness, Joel Tetreault, and James Allen. 2004. Incremental parsing with reference interaction. In Frank Keller, Stephen Clark, Matthew Crocker, and Mark Steedman, editors, *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, Barcelona, Spain, July.

David R. Traum, Lenhart K. Schubert, Massimo Poesio, Nathaniel G. Martin, Marc Light, Chung Hee Hwang, P. Heeman, George Ferguson, and James Allen. 1996. Knowledge representation in the trains-93 conversation system. *International Journal of Expert Systems*, 9(1):173–223.

Wayne H. Ward. 2008. The phoenix parser user manual. http://cslr.colorado.edu/ whw/phoenix/phoenix_manual.htm.

Klaus Zechner. 1998. Automatic construction of frame representations for spontaneous speech in unrestricted domains. In *Proceedings of COLING-ACL 1998*, Montreal, Canada.