UCSG Shallow Parser

G. Bharadwaja Kumar and Kavi Narayana Murthy

Department of Computer and Information Siences,
University of Hyderabad, India
knmuh@yahoo.com, g_vijayabharadwaj@yahoo.com

Abstract. Recently, there is an increasing interest in integrating rule based methods with statistical techniques for developing robust, wide coverage, high performance parsing systems. In this paper[1], we describe an architecture, called UCSG shallow parser architecture, which combines linguistic constraints expressed in the form of finite state grammars with statistical rating using HMMs built from a POS-tagged corpus and an A* search for global optimization for determining the best shallow parse for a given sentence. The primary aim of the design of the UCSG parsing architecture is developing a judicious combination of linguistic and statistical methods to develop wide coverage robust shallow parsing systems, without the need for large scale manually parsed training corpora. The UCSG architecture uses a grammar to specify all valid structures and a statistical component to rate and rank the possible alternatives, so as to produce the best parse first without compromising on the ability to produce all possible parses. The architecture supports bootstrapping with an aim to reduce the need for parsed training corpora. The complete system has been implemented in Perl under Linux. In this paper we first describe the UCSG shallow parsing architecture and then focus on the evaluation of the UCSG finite state grammar for the chunking task for English. Recall of 91.16% and 93.73% have been obtained on the Susanne parsed corpus and CoNLL 2000 chunking task test data set respectively. Extensive experimentation is under way to evaluate the other modules.

Keywords: Chunking, Shallow Parsing, Finite State Grammar, HMM, A* search, UCSG Architecture.

1 Introduction

Although a lot of work has gone into developing full syntactic parsers, high performance, wide coverage syntactic parsing has remained a difficult challenge [1]. In recent times, there has been an increasing interest in wide coverage and robust but partial or shallow parsing systems. Shallow parsing is the task of recovering only a limited amount of syntactic information from natural language sentences. Often shallow parsing is restricted to finding phrases in sentences, in which case it is also called chunking. Steve Abney[2], has described chunking as

[1] The research work reported here was supported in part by the University Grants Commission under the UPE scheme.

A. Gelbukh (Ed.): CICLing 2006, LNCS 3878, pp. 156–167, 2006.

finding syntactically related non-overlapping groups of words. In CoNLL chunking task[3], chunking was defined as *the task of dividing a text into syntactically non-overlapping phrases.* The term *phrase* has come to acquire a very special technical connotation in linguistics and in order to avoid confusion, chunks are also referred to as *word groups.*

As an example, the sentence "He reckons the current account deficit will narrow to only # 1.8 billion in September" could be analyzed as follows by a chunker [3]:

[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only # 1.8 billion] [PP in] [NP September].

Note that prepositional phrases have not yet been built, let alone resolving ambiguities in prepositional phrase attachment. Nor have the thematic roles been assigned to the chunks. Partial parsing systems do a bit more than chunking while still not promising complete syntactic analysis.

Developing computational grammars is a challenging task, even if we restrict to partial parsing. There are broadly two approaches for the development of grammars - the linguistic approach which depends upon hand-crafted rules, and, the machine learning approach where grammars are learned automatically from a parsed training corpus. Developing wide coverage linguistic grammars has proved difficult in practice. Parsed training corpora are also rarely available. Hence the interest in the search for a judicious combination of linguistic and statistical approaches.

In this paper we propose an architecture for shallow parsing, which we call UCSG Shallow Parsing Architecture. The UCSG (Universal Clause Structure Grammar) framework was developed during the early nineties at University of Hyderabad, India. Please see [4, 5] for more on UCSG. In this paper we use only one of the modules of UCSG namely the Finite State Grammar. In the UCSG Shallow Parsing Architecture, the Finite State Grammar is designed to accept *all* valid word groups but not necessarily the *only* those word groups that are appropriate in context for a given sentence. Many additional word groups may also be recognized. The focus in this phase is only on completeness, for, there is a second module consisting of a set of Hidden Markov Models, which will rate and rank the word groups so produced. An A* search algorithm is then used as the final module to obtain globally best chunk sequences for a given sentence. The aim is to produce all possible parses but hopefully in the best first order. The complete system has been implemented in Perl under Linux. The performance of the Finite State Parser has been evaluated on the Susanne parsed corpus as well as CoNLL 2000 chunking task test data set and Recall of 91.16% and 93.73% respectively have been obtained. Extensive experimentation is going on to refine the system through bootstrapping and to evaluate the overall performance.

2 A Brief Survey of Shallow Parsing Systems

Steve Abney [6] proposed finite state cascade models for the chunking task. Grefenstette [7] proposed methods to use finite state transducers for partial

parsing. Parsing with finite state transducers [8] was very popular in the early ninety's. Marc Vilain et al. [9] used rule based sequence processors for the chunking task. Herve Dejean [10] used ALLiS (Architecture for Learning Linguistic Structure), which is a symbolic machine learning system for the chunking task.

Miles Osborne [11] proposed maximum entropy based POS tagger for the chunking task. Veenstra and Bosch [12] used memory based learning for chunking. Zhou et al. [13] proposed error driven HMM based chunk tagger with context dependent lexicon. Rob Koeling [14] applied maximum entropy models for chunking. Christer Johansson [15] proposed context sensitive maximum likelihood approach for chunking task. Tong Zhang et al. [16] proposed generalized winnow algorithm for text chunking. Recently Fei Sha and Pereira [17] used conditional random fields for noun phrase chunking and achieved good performance.

Molina and Pla [18] proposed shallow parsing with specialized HMMs. Carreras et al. [19] used perceptrons for chunking task. Recently, Gondy et al. [20] proposed a shallow parser based on closed-class words to capture relations in biomedical text.

Taku Kudoh et al. [21] proposed SVMs for chunking. This system performed the best in CoNLL-2000 chunking task and achieved an F-measure of 93.48%. Van Halteren [22] proposed Weighted probability distribution voting algorithm (WPDV) for chunking task. Tjong Kim Sang [23] proposed combination of several memory based learning systems for chunking task.

Most of the parsers described in literature have used either only rule based techniques or only machine learning techniques. Hand-crafting rules in the linguistic approach can be very laborious and time consuming. Parsers tend to produce a large number of possible parse outputs and in the absence of suitable rating and ranking mechanisms, selecting the right parse can be very difficult. Statistical learning systems, on the other hand, require large and representative parsed corpora for training. .

Recently, there is an increasing interest on integrating shallow parsers with deep parsing. Berthold Crysmann et al. [24] reported an implemented system called WHITEBOARD which integrates different shallow components with a HPSG based deep parsing system. Ronald M. Kaplan et. al. proposed a hybrid architecture called XLE [25] for combining finite state machine with LFG grammar. In XLE system, first the surface forms are run through the FST morphology to produce the corresponding stems and tags. Stems and tags each have entries in the LFG lexicon. Sub-lexical phrase structure rules produce syntactic nodes covering these stems and tags and standard grammar rules then build larger phrases.

3 UCSG Shallow Parsing Architecture

Purely linguistic approaches have not proved practicable for developing wide coverage grammars and purely machine learning approaches are also impracticable in many cases due to the non-availability of large enough parsed training corpora. Only a judicious combination of the two approaches can perhaps lead to wide coverage grammars and robust parsing systems. In the UCSG Shallow

Parsing Architecture, instead of looking for a grammar that can capture *all and only* valid structures, simultaneous satisfaction of both the requirements having proved very difficult in practice, we employ a Finite State Grammar that is general enough to capture *all* valid word groups without necessarily restricting to *only* those word groups which are appropriate in the context of a given sentence, and a separate statistical component, encoded in HMMs (Hidden Markov Model), to rate and rank the word groups so produced. Note that we are not pruning, we are only rating and ranking the word groups produced. The aim is to produce parse outputs in best first order, without compromising on the ability to produce all possible parses. This system is thus more than a chunker - the word groups produced are often bigger, disambiguated in context to some extent (for example, VVG is disambiguated between a gerund, a present participle and part of a present continuous verb group) and motivated by insights from deeper parsing requirements. This work is part of a larger on-going effort. The UCSG Shallow Parsing Architecture is depicted in Figure 1.

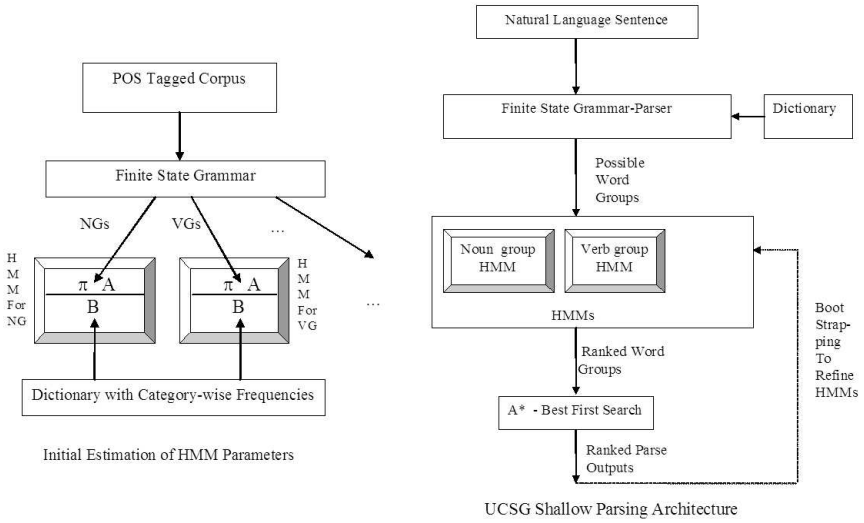

Fig. 1. UCSG Shallow Parsing Architecture

3.1 Finite State Parser

Unlike linguistic grammar formalisms and corresponding full parsers that attempt to capture the full hierarchical and thematic structure within sentences, partial parsing systems and chunkers only need to identify non-overlapping, non-recursive word groups or chunks. Thus the power of Context Free Grammars (also known generally as phrase structure rules in Linguistics) is not required and the simpler Finite State Grammars are sufficient. Finite State Grammars capture simple constraints such as linear precedence, optional items and repetitions but not arbitrarily deep hierarchical nestings or general dependencies

across constituents. Finite State Grammars can be developed with relative ease as compared to developing computational grammars for capturing full syntax. Also word groups can be recognized using these Finite State Machines in linear time [26].

The first module in the UCSG architecture is a Finite State Grammar-Parser. The aim is to develop a general enough grammar that can capture *all* valid word groups. Many additional word groups may be produced due to lexical ambiguities. We do not aim to restrict or prune the various possibilities. Instead we use a separate module to rate and rank these word groups.

The input to the system is one sentence at a time, either plain or POS-tagged. In the former case, the dictionary is consulted to obtain all the possible tags for each word. Our dictionary includes 1,28,000 root words, grammatical categories in the Claws-5 tag-set format, and frequency of each word in each of the possible categories. This dictionary has been developed over the last many years, cross checked against several large corpora including the British National Corpus and Reuters News Corpus. A coverage of 90 to 97 percent plus has been observed on various corpora. Inflectional morphology is handled by the parser and words still not analyzed are taken by default to be proper nouns. We give a running example to illustrate the working of the system through various stages. The input to the system is a sentence in plain text format: *'He was walking along a quiet street when gunmen shot him several times in the head.'* After Dictionary lookup, we have:

<PNPNOM>he <VBD>was <VVG>walking> <AVP_PRP>along <AT0>a
<VVB_AJ0_NN1_VVI>quiet <NN1>street <CJS_AVQ>when
<NN2>gunmen <VVN_VVD_NN1>shot <PNPACC>him
<DT0_PNP_AJ0>several <VVZ_NN2> <times> <PRP_AVP_AJ0>in
<AT0>the <VVB_AJ0_NN1_VVI>head

Note the lexical ambiguities. Sample output from the Finite State Parser is given below. Chunks identified are given one per line, starting with the category of the chunk, followed by the words along with the corresponding tags.

<vg> <VBD><was>
<vg> <VBD><was><VVG><walking>
<vg> <VBD><was><VVG><walking><AVP><along>
<vgs> <VVG><walking>
<vg> <VVB><quiet>
<ng> <VVG><walking>
<ng> <PRP><along><AT0><a><NN1><quiet>
<ng> <PRP><along><AT0><a><NN1><quiet><NN1><street>
<ng> <PRP><along><AT0><a><AJ0><quiet><NN1><street>
<ng> <AT0><a><NN1><quiet>
<ng> <AT0><a><AJ0><quiet><NN1><street>
<ng> <AT0><a><NN1><quiet><NN1><street>

<ng> <NN1><quiet>
<ng> <AJ0><quiet><NN1><street>
<ng> <NN1><quiet><NN1><street>
<ng> <NN1><street>
<ng> <VVG><walking><PRP><along><AT0><a>
<AJ0><quiet><NN1><street>
<ng> <VVG><walking><PRP><along><AT0><a>
<NN1><quiet><NN1><street>
<ng> <VVG><walking><PRP><along><AT0><a><NN1><quiet>
<ajg> <VVG><walking>
<ajg> <AJ0><quiet>
<part> <AVP><along>

3.2 HMM-Module

The second module is a set of Hidden Markov Models (HMM) used for rating and ranking the word groups produced by the Finite State Grammar. A number of word groups would have produced by the first module, the right ones and possibly several additional ones as well. Rating and ranking helps us to prefer the appropriate ones to the others.

One HMM model is built for each major category of word groups. In this work we have used three HMM models, one for noun groups, one for verb groups and one for all other kinds of word groups. Note that in UCSG, prepositional groups are included under noun groups.

States in a HMM correspond to categories. Observation symbols correspond to words. The HMM models $\lambda = (\pi, A, B)$ are defined as follows:

- The number of states of the model N is the number of relevant categories.
- The number of observation symbols M is the number of words.
- The initial state probability $\pi_i = P\{q_i = i\}\ 1 \leq i \leq N$ where q_i is a category (state) starting a particular word group
- State transition probability $a_{ij} = P\{q_{t+1} = j | q_t = i\}\ 1 \leq i, j \leq N\}$ where q_t denotes the current category (state) and q_{t+1} denotes the next state.
- Observation or emission probability, $b_j(k) = P\{o_t = v_k | q_t = j\}\ 1 \leq j \leq N$, $1 \leq k \leq M$ where v_k denotes the k^{th} word, and q_t the current state.

The HMM parameters A, B and π can be obtained from a training corpus. In case a manually checked and certified chunked corpus is available, these parameters can be estimated from such a training corpus. However, chunked/parsed training corpora are difficult to get and we propose a bootstrapping technique to estimate the HMM parameters when only a POS-tagged corpus is available. In the latter case, we first pass the corpus through our Finite State module and obtain the possible chunks. Taking these chunks to be equi-probable, we can estimate the HMM parameters either using Baum-Welch algorithm or by simply taking the ratios of frequency counts.

In the present work, 2,500,000 randomly selected sentences from the British National Corpus [27] were chunked using our Finite State Grammar. It may be noted that this corpus is POS tagged but not parsed/chunked. The π and A matrix values were estimated from these chunks, taking all chunks as equi-probable and the B matrix values were estimated from our dictionary which includes the frequencies for every category for each word. Note that development of the HMMs is a one-time off-line process. However, as we shall see later, the HMM parameters can be further refined later by bootstrapping.

The HMMs are used only for rating and ranking the word groups already obtained by the Finite State Grammar, not for obtaining the word groups per se. We simply estimate the probability of each chunk in the HMM model for the appropriate category:

$$P(O|\lambda) = \sum_{i=1}^{t} \pi_{i_1} b_{i_1}(O_1) a_{i_1,i_2} b_{i_2}(O_2) a_{i_2,i_3} \cdots a_{i_{t-1},i_t} b_{i_t}(O_t)$$

The aim here is to obtain the highest ranks for the correct chunks and to push down other chunks in the ranked order. Contrast this with the more common idea of using of HMMs (say for POS tagging) before parsing. This would require Viterbi search. Also, the HMMs we use in UCSG architecture are specific to different categories of word groups and are local to the neighbourhood of word groups, thereby imposing tighter constraints.

Chunks obtained by the Finite State Grammar can be partitioned into chunk-groups such that chunks across chunk-groups are disjoint. Ranking is performed within chunk-groups, keeping in view the category of the chunks as the example below shows. Performance can be measured in terms of position of the correct chunks in the ranked order. A sample from the running example is given in Table 1 to show the rankings obtained for the various possible word groups. It can be seen that the correct chunks tend to get ranked higher. Extensive experimentation is going on to fine tune the HMMs so that good rankings can be obtained reliably.

3.3 A* Search for Best First Search

It has proved difficult in practice to produce a single parse, or a very small number of parses, and at the same time guarantee correctness of parsing. A large number of possible parse outputs will also be difficult to use if the outputs are not rated and ranked in some way. Our aim is to build parses that can produce parse outputs in best-first order, without compromising on the ability to produce all grammatically valid parses, even when the input sentences are not POS tagged. Depending upon the application, we may either produce all possible parses in ranked order or stop further generation using a suitable thresholding mechanism. In UCSG Architecture, we posit a A* best first search algorithm to select the chunks to produce the best chunk sequence for a given sentence, taking advantage of the ratings provided by the HMM module. Searching involves selecting chunks in best first order to cover the given sentence without overlaps or gaps.

Table 1. Ranking by HMMs

<vg> <VBD><was><VVG><walking>	1
<vg> <VBD><was><VVG><walking><AVP><along>	2
<vg> <VBD><was>	3
<vgs> <VVG><walking>	4
<ng> <VVG><walking><PRP><along><AT0><a>	
<AJ0><quiet><NN1><street>	1
<ng> <VVG><walking><PRP><along><AT0><a>	
<NN1><quiet><NN1><street>	2
<ng> <PRP><along><AT0><a><AJ0><quiet><NN1><street>	3
<ng> <VVG><walking><PRP><along><AT0><a><NN1><quiet>	4
<ng> <PRP><along><AT0><a><NN1><quiet><NN1><street>	5
<ng> <AT0><a><AJ0><quiet><NN1><street>	6
<ng> <PRP><along><AT0><a><NN1><quiet>	7
<ng> <AT0><a><NN1><quiet><NN1><street>	8
<ng> <AT0><a><NN1><quiet>	9
<ng> <AJ0><quiet><NN1><street>	10
<ng> <NN1><quiet><NN1><street>	11
<ng> <NN1><street>	12
<ng> <VVG><walking>	13
<ng> <NN1><quiet>	14

The rating and ranking of the word groups by HMMs is local to the neighbourhood of the word groups, that is, within chunk-groups. Note that while sequences of words within chunks have been considered by the Finite State Parser and HMMs, the sequences of chunks themselves have not been taken into account. Therefore, simply selecting the best rated chunks within each of the chunk-groups will not necessarily form the best parse for the whole sentence.

The A* Best First Search Strategy combines two factors, namely, effort already spent in pursuing the current path (g), and, estimated effort required to reach the goal state (h). A single combined measure of goodness (f) is computed for each node in the search tree and the best node is selected for subsequent expansion: $f(n) = g(n) + h(n)$. The effort already spent is obtained from the probabilities given by the HMM module. The distance to the goal node is estimated in terms of the words yet to be covered in the given sentence. The top parse for our running example is given below:

<ng>[<PNPNOM><he>]</ng>
<vg>[<VBD><was><VVG><walking>]</vg>
<ng>[<PRP><along><AT0><a><AJ0><quiet><NN1><street>]</ng>
_[<CJS><when>]
<ng>[<NN2><gunmen>]</ng>
<vg>[<VVD><shot>]</vg>
<ng>[<PNPACC><him>]</ng>
<ng>[<DT0><several><NN2><times><PRP><in>
<AT0><the><NN1><head>]</ng>

One of the main ideas behind the UCSG architecture is bootstrapping. The final parse outputs produced after A* search will hopefully be more or less in best first order. We can therefore take the top parse, or the top few parses to be correct and re-estimate the HMM parameters using this refined data. We can also manually check the parse outputs and build a dependable partially parsed corpus. We have so far built a manually checked parsed corpus of 2000 plus sentences. Extensive experimentation is on to re-estimate the HMM parameters as also for fine tuning A* search itself.

4 Experiments and Results

The performance of the Finite State module has been evaluated on the Susanne Parsed Corpus as well as CoNLL 2000 Test data set. Since the aim of this module is only completeness, performance is given in terms of Recall.

The Susanne corpus [28] is a manually parsed corpus containing about 130,000 words in more than 6500 sentences. Some preprocessing was necessary. Ambiguities with apostrophes have been resolved. Spelling errors mentioned in the Susanne documentation have been corrected. Since the structure of the parse output in the Susanne corpus differs somewhat from that of UCSG, suitable mapping schemes had to be developed and validated [29]. Plain text sentences were extracted and given as input to the UCSG shallow parser. Results are given below in Table 2 for Noun, Verb, Adjective and Adverb groups.

Table 2. Performance of the Finite State Parser on Susanne Corpus

Word Group Type	No. of Groups in Test Data	No. of Groups Recognized	% Recall
Noun Group	34952	30642	87.67
Verb Group	18134	17975	99.12
Adjective Group	2355	1794	76.18
Adverb Group	5512	5156	93.54
Overall	60953	55567	91.16

Overall, 91.16% of phrases in the Susanne corpus have been correctly identified. 99.12% of all the verb groups could be correctly identified. Failures in the case of verb groups are limited to complex cases such as “have never, or not for a long time, had”.

The CoNLL 2000 test data set consists section 20 of the Wall Street Journal corpus (WSJ) and includes 47377 words and 23852 chunks. In the current evaluation, LST chunks (list items) have been excluded. Also, in the UCSG framework, there are no separate PPs - PPs are included in noun groups. Table 3 gives the performance.

There are a few minor differences in the way chunks are defined in the CoNLL 2000 chunking task and UCSG. Punctuation marks are removed by a pre-processor and handled separately elsewhere in UCSG. Currency symbols

Table 3. Evaluation of Finite State Parser on CoNLL 2000 Test Data Set

CoNLL Chunk Type	UCSG Terms	Chunks in Test Data	Chunks Recognized	% Recall
NP	ng	12422	10588	85.24
VP	vg,infg	4658	3786	81.28
ADVP	avg	866	698	80.60
ADJP	ajg	438	398	90.87
SBAR	sub	535	507	94.77
PRT	part	106	105	99.06
CONJP	sub	9	9	100.00
INTJ	intg	2	1	50.00
Total		19036	16092	84.53

Table 4. Evaluation of the Finite State Parser on CoNLL Data Set after mapping

CoNLL Chunk Type	Chunks in Test Data	Chunks Recognized	% Recall
NP,PP	17233	16158	93.76
VP	4658	4475	96.07

such as $ and # are considered part of numbers in UCSG while they become separate words in CoNLL. CoNLL splits chunks across the apostrophies in genitives as in *Rockwell International Corporation's tulsa unit* while UCSG does not. To-infinitives as in *continue to plummet* are recognized separately in UCSG while they may form part of a VP in CoNLL. Also, in keeping the UCSG philosophy, PPs are not recognized separately in UCSG, they are included in noun groups. In order to get a better feel for the true performance of the UCSG shallow parser, the above differences were discounted for and performance checked again. The results are given in Table 4 for NP, PP and VPs. There is no change in the performance for other groups. Overall, 22351 out of 23847 chunks have been correctly identified, giving a Recall of 93.73%.

Initial results with ranking by HMMs has shown promise and extensive work on bootstrapping to refine the HMM parameters is currently under way. The overall system is also being evaluated in terms of the percentage of correct chunks found in the top ranked parse as also in terms of the ranking of the fully correct parse in the final output.

5 Conclusions

In this paper we have described an architecture for partial parsing called the UCSG shallow parsing architecture. UCSG combines linguistic constraints expressed in the form of finite state grammars with statistical rating using HMMs built from a POS-tagged corpus and a best first search strategy for global optimization. With appropriate bootstrapping, it would hopefully be possible to develop wide coverage and robust partial parsing systems without the need for parsed corpora which are not easily available in many cases. The UCSG Shallow Parsing Architecture is also computationally efficient. Since large scale plain text

and POS-tagged corpora are becoming available in Indian languages, this approach seems to hold promise for developing parsing systems for these languages as well.

The complete system including all the modules in the architecture has been implemented in Perl under Linux. On the Susanne parsed corpus, an overall Recall of 91.16% has been obtained and on the CoNLL 2000 chunking task test data set, a Recall of 93.73% has been obtained for the Finite State Parser. Further work is under way for refining the system through bootstrapping.

References

1. Doran, C., Egedi, D., Hockey, B.A., Srinivas, B., Zaidel, M.: XTAG system – a wide coverage grammar for english. In: Proceedings of the 15th. International Conference on Computational Linguistics (COLING 94). Volume II., Kyoto, Japan (1994) 922–928
2. Abney, S.P.: Parsing by Chunks. Principle-based parsing: Computation and psycholinguistics edn. Kluwer (1991)
3. Tjong Kim Sang, E.F., Buchholz, S.: Introduction to the conll-2000 shared task: Chunking. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 127–132
4. Murthy, K.N.: Universal Clause Structure Grammar. PhD Thesis, University of Hyderabad (1995)
5. Murthy, K.N.: Universal Clause Structure Grammar and the Syntax of Relatively Free Word Order Languages. South Asian Language Review **VII** (1997)
6. Abney, S.: Partial parsing via finite-state cascades. In: Workshop on Robust Parsing, 8th European Summer School in Logic, Language and Information, Prag (1996) 8–15
7. Grefenstette, G.: Light parsing as finite state filtering. In: Workshop on Extended finite state models of language, Budapest, Hungary (1996)
8. Roche, E.: Parsing with finite state transducers. Finite–state language processing edn. MIT Press (1997)
9. Vilain, M., Day, D.: Phrase parsing with rule sequence processors: an application to the shared conll task. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proc. of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 160–162
10. Dejean, H.: Learning rules and their exceptions. In: Journal of Machine Learning Research, volume 2. (2002) 669–693
11. Osborne, M.: Shallow parsing as part-of-speech tagging. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 145–147
12. Veenstra, J., van den Bosch, A.: Single-classifier memory-based phrase chunking. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 157–159
13. Zhou, G., Su, J., Tey, T.: Hybrid text chunking. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 163–166
14. Koeling, R.: Chunking with maximum entropy models. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 139–141

15. Johansson, C.: A context sensitive maximum likelihood approach to chunking. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 136–138
16. Zhang, T., Damerau, F., Johnson, D.: Text chunking based on a generalization of winnow. In: Journal of Machine Learning Research, volume 2. (2002) 615–637
17. Sha, F., Pereira, F.: Shallow parsing with conditional random fields. Technical Report CIS TR MS-CIS-02-35, University of Pennsylvania (2003)
18. Molina, A., Pla, F.: Shallow parsing using specialized hmms. In: Journal of Machine Learning Research, volume 2. (2002) 595–613
19. Carreras, X., Marquez, L.: Phrase recognition by filtering and ranking with perceptrons. In: Proceedings of the International Conference on Recent Advances in Natural Language Processing, RANLP-2003, Borovets, Bulgaria (2003) 127–132
20. Gondy, L., Hsinchun, C., Jesse, M.: A shallow parser based on closed-class words to capture relations in biomedical text. In: Journal of Biomedical Informatics 36. (2003) 145–158
21. Kudoh, T., Matsumoto, Y.: Use of support vector learning for chunk identification. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 142–144
22. van Halteren, H.: Chunking with wpdv models. In Cardie, C., Daelemans, W., Nedellec, C., Tjong Kim Sang, E., eds.: Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal (2000) 154–156
23. Erik F. Tjong Kim Sang: Memory-based shallow parsing. In: Journal of Machine Learning Research, volume 2. (2002) 559–594
24. Berthold Crysmann et al.: An integrated archictecture for shallow and deep processing systems. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), University of Pennsylvania, Philadelphia (2002)
25. Kaplan, R.M., III, J.T.M., King, T.H., Crouch, R.: Integrating finite-state technology with deep lfg grammars1. In: Proceedings of the Workshop on Combining Shallow and Deep Processing for NLP(ESSLLI). (2004)
26. J. Hopcroft and J. Ullman: Introduction to automata theory, languages, and computation. Addison-Wesley (1979)
27. Burnard, L. In: The users reference guide for the British National Corpus. Oxford University Computing Services, Oxford (1995)
28. Sampson, G.: The susanne treebank: Release 5, Univ.of Sussex, England (2000)
29. Nagesh, K.: Towards a robust shallow parser. Masters thesis, Department of Computer and Information Sciences, University of Hyderabad (2004)