SProUT—Shallow Processing with Unification and Typed Feature Structures

Markus Becker, Witold Drożdżyński, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, Feiyu Xu
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
sprout@dfki.de

Abstract

We present *SProUT*, a platform for the development of multilingual shallow text processing systems. A grammar in *SProUT* consists of a set of rules, where the left-hand side is a regular expression over typed feature structures (TFSs), representing the recognition pattern, and the right-hand side is a sequence of TFSs, specifying how the output structure looks like. The reusable core components of *SProUT* are a finite-state machine toolkit, a regular compiler, a typed feature structure package, and a finite-state machine interpreter.

1 Introduction

In the last decade, an ever-growing trend of deploying lightweight linguistic analysis for solving problems that deal with the conversion of raw textual information into structured and valuable knowledge can be observed. Recent advances in the areas of information extraction, text mining, and textual question answering demonstrate the benefit of applying shallow text processing (STP) techniques, which are assumed to be considerably less time-consuming and more robust than deep processing systems, but are still sufficient to cover a broad range of linguistic phenomena.
This paper centers around *SProUT* (Shallow Processing with Unification and Typed feature structures), a platform for the development of multilingual STP systems. It consists of several linguistic processing resources and provides a grammar development and testing environment. Additionally, it can be used for building higher-level linguistic components by flexibly combining existing resources.
The motivation for developing *SProUT* comes from the need to have a system that (i) allows a flexible integration of different processing modules and (ii) to find a good trade-off between processing efficiency and expressiveness of the formalism. On the one hand, we can find here very efficient *finite state* (FS) devices which have been successfully applied to real-world applications. On the other hand, *unification-based grammars* (UBGs) are designed to capture fine-grained syntactic and semantic constraints, resulting in better descriptions of natural language phenomena. In contrast to FS devices, unification-based grammars are also assumed to be more transparent and more easily modifiable. Our idea now is to take the best of these two worlds, having a FS machine that operates on typed feature structures (TFSs). I.e., transduction rules in *SProUT* do not rely on simple atomic symbols, but

instead on TFSs, where the left-hand side (LHS) of a rule is a regular expression over TFSs, representing the recognition pattern, and the right-hand side (RHS) is a sequence of TFSs, specifying the output structure. Consequently, equality of atomic symbols is replaced by *unifiability* of TFSs and the output is constructed using TFS *unification* w.r.t. a type hierarchy.
This paper is structured as follows. Section 2 presents related work, viz., (extended) FS devices and unification-based grammars. After that, section 3 describes the formalism, starting with the building blocks (TFS, type definition, type hierarchy) and ending in regular expressions over TFSs. We then discuss the architectural framework and core components (section 4). In section 5, we focus on the current system instances and its peculiarities. Finally, section 6 describes the status of our work and addresses future development directions.

2 Related Work

Finite-state devices and unification-based grammars have influenced the shallow TFS formalism presented in section 3.

2.1 Finite-State Devices

The pure finite-state based STP approaches proved to be very efficient in terms of processing speed. [Piskorski and Neumann, 2000] present SPPC, a highly efficient system, which uses cascades of simple finite-state grammars, based on a small number of basic predicates. Complex constraints can not be encoded in the FS device. The idea of using more complex annotations on the transitions of FS automata has been considered in SMES [Neumann *et al.*, 1997] which uses regular grammars with predicates over morphologically analyzed tokens. These predicates inspect arbitrary properties of the input tokens, like part-of-speech or inflectional information. [van Noord and Gerdemann, 2001] introduce arbitrary predicates over symbols and discuss various operations on finite state acceptors and transducers. They observe that automata with predicates generally have fewer states and transitions. However, the discussed automata only operate on symbols of a finite input alphabet.
In the last few years, several cascaded FSM-based systems have been developed for information extraction tasks. The most successful systems provide high-level specification languages for grammar writing. The pioneering FASTUS system [Appelt, 1996; Hobbs *et al.*, 1997] uses CPSL (Common Pattern Specification Language). The more recent GATE system (General Architecture for Text Engineering) [Cunningham *et al.*, 2000] provides JAPE (Java Annotation Patterns Engine), which is similar in spirit to CPSL and admittedly borrows its main features from CPSL. A CPSL/JAPE grammar contains pattern/action rules. The LHS of a rule is a regular expression over atomic feature-value constraints for pattern matching, while the RHS is a so-called *annotation manipulation statement* for output production, which calls native code (e.g., C or Java), making rule writing difficult for non-programmers. Furthermore, even though there is a mechanism for variable binding which is responsible for copying values into the RHS, this mechanism is not capable of declaratively describing coreferences among the rule elements.

2.2 Unification-Based Grammars

Since the late seventies, UBG formalisms have become an important paradigm in NLP and CL. In the beginning, unification was employed as the primary constraint solving mechanism, hence the term *unification-based grammars*. Nowadays, this family of formalisms is often characterized through the

more general notion *constraint-based*.
Their success stems from the fact that they can be seen as a *monotonic*, high-level representation language on which a parser/generator or a uniform type deduction mechanism acts as the inference engine. One of the main advantages of such formalisms is that they provide a *declarative* (as opposed to procedural) representation of linguistic knowledge, i.e., one must only specify the knowledge which participates in the constraint solving process, instead of anticipating the order in which the constraints are applied.
The representation of as much linguistic knowledge as possible through a unique data type called *feature structure* allows the integration of different description levels, spanning phonology, syntax, and semantics. Here, the feature structure itself serves as the abstract interface between the different strata which can thus be accessed and constrained at the same time. Central to feature structures is an operation which *combines* the information from two feature structures into a single structure, but also determines the *satisfiability* of the resulting description: *unification*.
Informally, a feature structure can be seen as a collection of feature-value pairs, where a feature expresses a functional property and the value of a feature might again be a feature structure (or an atom), thus we allow for recursive embeddings. An important characteristic of feature structures is that they allow for *coreference* constraints, meaning that two features share exactly one common value. This concept allows for the transport of information and is exhaustively used in the grammar rules, where features on the LHS share values with other features on the RHS.
Feature structures can also be given a *type* which ultimately leads to a *typed feature structure*. First of all, a type can be seen as a compact abbreviation for a TFS, supporting clarity and easy modifiability of descriptions (*type definition*). Furthermore, types can be arranged in a *type hierarchy*, allowing multiple inheritance of information from all supertypes. The next section will give examples.

3 $\mathcal{XTDL}$—The Formalism

$\mathcal{XTDL}$ combines two well-known frameworks: typed feature structures and regular expressions. $\mathcal{XTDL}$ is defined on top of $\mathcal{TDL}$, a definition language for TFSs [Krieger and Schäfer, 1994] that is used as a descriptive device in several grammar systems (LKB, PAGE, PET). We use the following fragment of $\mathcal{TDL}$, including coreferences and functional application.

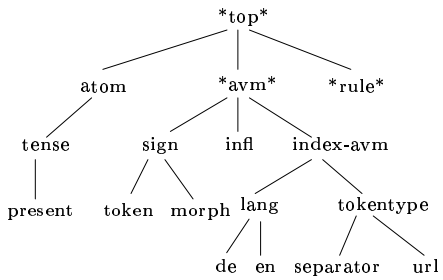
type-def → *type* { *avm-def* | *sub-def* } [*fun-op*] "."
type → *identifier*
sub-def → ":<" *type*
avm-def → ":=" *avm*
avm → *term* { "&" *term* }*
term → *type* | *fterm* | *string* | *coref*
fterm → "[" [*attr-val* {"," *attr-val*}*] "]"
attr-val → *attribute avm*
attribute → *identifier*
coref → "#"*identifier*
fun-op → "where" {*coref* "=" *fun-app*}+
fun-app → *identifier* "(" *term* {"," *term*}* ")"

Apart from the integration into the rule definitions, we also employ this fragment in *SProUT* for the establishment of a type hierarchy of linguistic entities. In the example definition below, the *morph* type inherits from *sign* and introduces three more morphologically motivated attributes with the

corresponding typed values.

```
morph := sign & [ POS  atom,
                  STEM atom,
                  INFL infl ].
```

The next figure depicts a fragment of the type hierarchy used in the example.



A rule in $\mathcal{XTDL}$ is straightforwardly defined as a production part on the LHS, written as a regular expression, and an output description on the RHS.[1] A named label serves as a handle to the rule. Regular expressions over feature structures describe sequential successions of linguistic signs. We provide a couple of standard operators; see the EBNF below. Concatenation is expressed by consecutive items. Disjunction, Kleene star, Kleene plus, and optionality are represented by the operators `|`, `*`, `+`, and `?`, resp. $\{n\}$ after an expression denotes an n-fold repetition. ($\{m,n\}$ repeats at least m times and at most n times.)

rule → *identifier* `":>"` *regexp* `"->"` $\{$*fterm*$\}^*$ `"."`
regexp → *avm* | `"("` *regexp* `")"` | *regexp* $\{$*regexp*$\}^+$ | *regexp* $\{$`"|"` *regexp*$\}^+$ |
regexp $\{$`"*"` | `"+"` | `"?"`$\}$ | *regexp* `"{"` *int* [`","` *int*] `"}"` |

The $\mathcal{XTDL}$ grammar rule below may illustrate the syntax. It describes a sequence of morphologically analyzed tokens (of type *morph*). The first TFS matches one or zero items (`?`) with part-of-speech `Determiner`. Then, zero or more `Adjective` items are matched (`*`). Finally, one or two `Noun` items (`{1,2}`) are consumed. The use of a variable (e.g., `#1`) in different places establishes a coreference between features. This example enforces, e.g., agreement in case, number, and gender for the matched items. I.e., all adjectives must have compatible values for these features. Eventually, the description on the RHS creates a feature structure of type *phrase*, where the category is coreferent with the category `Noun` of the right-most token(s) and the agreement features result from the unification of the agreement features of the *morph* tokens.

(1)
```
np :> morph & [POS  Determiner,
               INFL [CASE #1, NUMBER #2, GENDER #3 ]] ?
     (morph & [POS  Adjective,
               INFL [CASE #1, NUMBER #2, GENDER #3 ]] ) *
      morph & [POS  Noun & #4,
               INFL [CASE #1, NUMBER #2, GENDER #3 ]] {1,2}
```

[1] It is worth noting that $\mathcal{XTDL}$ rules are related to *lexical rules* in UBGs, devices developed for expressing lexical generalizations; see section 4.4.

```
-> phrase & [CAT #4,
             AGR agr & [CASE #1, NUMBER #2, GENDER #3 ]].
```

In the future, we foresee to have weaker, unidirectional coreferences under Kleene star (even under restricted iteration). The idea here is that the values under such coreferences are collected in a list which is given to the RHS of a rule (we indicate this behavior by using the percent sign). Consider the above rule and assume that adjectives also have a relation attribute `RELN`. We would now like to collect all the relations and to have them grouped in a list on the RHS:

```
[POS Det, ...] ([POS Adj, ..., RELN %5])* [POS Noun, ...] -> [..., RELN %5]
```

The choice of *TDL* has a couple of advantages. TFSs as such provide a rich descriptive language over linguistic structures (as opposed to atomic symbols) and allow for a fine-grained inspection of input items. They represent a generalization over pure atomic symbols. Unifiability as a test criterion, whether a transition is viable, can be seen as a generalization over symbol equality. Coreferences in feature structures express structural identity. Their properties are exploited in two ways. They provide a stronger expressiveness since they create dynamic value assignments on the automaton transitions and thus exceed the strict locality of constraints in an atomic symbol approach. Furthermore, coreferences serve as a means of information transport into the output description on the RHS of the rule. Finally, the choice of feature structures as primary citizens of the information domain makes composition of modules very simple, since input and output are all of the same abstract data type. In [Crysmann *et al.*, 2002], we presented an integrated architecture for shallow and deep text processing, which further demonstrates the benefits of using TFSs as a representation and interchange format.
It is worth noting that regular expressions in *SProUT* are quite different from those in LFG which were introduced under the notion of *functional uncertainty* [Kaplan and Maxwell III, 1988] In our framework, regular expressions are defined over typed feature structures; in LFG, they are constructed from features, allowing to specify regular path patterns.
Furthermore, we note here (again) that *SProUT* grammars do *not* have the generative capacity of UBGs. This is due to the fact that output produced by the RHS of a *SProUT* rule is *not* available on the same cascade stage from which the rule was applied. Hence the output, which hopefully has grouped chunks into larger units, can only serve as input to rules on later cascade stages. Since the RHS of a *SProUT* rule might compute a sequence of output structures which is longer or equal than the input, one can even *not* compute *a priori* the number of cascade stages needed to simulate unification-based parsing behavior (viz., finding a completely spanning analysis). Thus, if we would disallow *SProUT* rules to produce such output (and if we would prohibit functional application), the number of cascade stages is bounded by the length of the input. This is the analogue to the *offline parsability constraint* for UBGs, originally formulated for DCG and LFG. However, a single cascade that is fed by its own output will realize such an unbounded number of stages (see also section 6).

4 Architecture

The *SProUT* system has been realized on top of four major core components: a finite state machine toolkit, a regular compiler, a Java typed feature structure package, and a *XTDL* interpreter.

4.1 Finite-State Machine Toolkit

The FS machine toolkit is a generic toolkit for building, combining, and optimizing FS devices [Piskorski, 2002]. In order to cover all STP-relevant types of FS devices and to allow for a parameterizable

weight interpretation, we use the finite-state machine (FSM) as the underlying model for our toolkit. A FSM is a generalization of the more familiar finite-state automaton (FSA), finite-state transducer (FST), and their weighted counterparts [Mohri, 1997]. Contrary to weighted FSTs which are tailored to a specific semiring for weight interpretation, the FSMs are more general in that they admit the use of arbitrary real-valued semirings (e.g., we use the tropical semiring for regular pattern prioritization). The toolkit provides all state-of-the-art operations on FSMs [Mohri, 1997; Mohri *et al.*, 1996; Roche and Schabes, 1995] which are relevant to the realization of the different levels of STP (ranging from tokenization to parsing) in a uniform way. The architecture and functionality of this toolkit is mainly based on the tools developed by AT&T [Mohri *et al.*, 1996]. In contrast to the latter package, we provide some new crucial operations relevant to STP, including weighted local extension [Roche and Schabes, 1995] and an efficient algorithm for incremental construction of minimal, deterministic, and acyclic FSAs from a list of words [Daciuk, 1998]. Furthermore, we improved the general algorithm for removing ϵ-moves [Mohri *et al.*, 1996] in terms of efficiency, which is an essential operation in the process of determinization.

4.2 Regular Compiler

Since regular expressions are regarded as the adequate level of abstraction for thinking about finite-state languages [Karttunen *et al.*, 1996], we developed a flexible XML-based and Unicode-compatible regular compiler for converting regular patterns into their corresponding compressed finite-state representation [Piskorski *et al.*, 2002]. The compiler provides an extendible set of circa 20 standard regular operators. Both the definition and configuration of the transformation process is done via XML which allows for easy and straightforward extensions. The compiler provides an option which allows to decide whether the input data will be interpreted as scanner definitions (e.g., token class definitions) or general regular expressions (e.g., regular expressions over TFSs). The grammar writer may flexibly define the way in which the FS devices are merged together and bias the optimization process. For instance, there are two alternative ways in which ambiguities are handled. The first option is to resolve ambiguities by assigning weights to the patterns which represent their priorities (e.g., in the tokenizer of *SProUT*). In the process of pattern merging, the tropical semiring is applied in order to resolve potential ambiguities [Mohri, 1997]. The second option is to preserve all ambiguities by introducing appropriate final emissions, representing pattern identifiers in the corresponding FS devices (e.g., in shallow grammars in *SProUT*).
The compilation of $\mathcal{XTDL}$ grammars is straightforward. The TFSs of the production part in the LHS of each rule are replaced by symbols representing references to these structures, since FSM arcs may be only labeled with symbolic values. The regular compiler transforms all such modified LHS into a corresponding FS network. Since fully specified TFSs usually do not allow for minimization of the resulting network, the compiler might apply a restrictor [Shieber, 1985] while constructing the mapping from TFSs to symbols. Note that through the use of final emissions mentioned above, the information concerning association of LHSs with their corresponding RHSs and original rules, is preserved in the resulting FS network.

4.3 Typed Feature Structure Package

The JTFS package is a Java implementation of TFSs. JTFS reads in a binary representation of a typed UBG, including type hierarchy and lexicon, and builds up the object in main memory. The lazy-copying unifier is a variant of [Emele, 1991], together with an efficient type unification operation (bit vector unification plus caching). JTFS supports a dynamic extension of the type hierarchy at run

time in order to allow for the incorporation of unknown words. Other operations, such as subsumption, deep copying, path selection, feature iteration, and different printers are available.

4.4 $\mathcal{XTDL}$ Interpreter

The challenge for the *SProUT* interpreter is to combine regular expression matching with unification of TFSs. Since the regular operators such as Kleene star can not be expressed by a TFS, we are faced with the problem of mapping a regular expression to a corresponding sequence of TFS, so that the coreference information among the elements in a rule can be preserved. Our solution is to separate the matching of regular patterns using unifiability (LHS of rules) from the construction of the output structure through unification (RHS). The positive side effect of this decision is that the matching step filters the potential candidates for the space-consuming unification. After a compatible pattern is identified, we embed the sequence of input TFSs (encoded as a list) into a new TFS; see the IN value in Fig. 1. Subsequently, a rule with an instantiated LHS pattern is constructed; see Fig. 2 as an

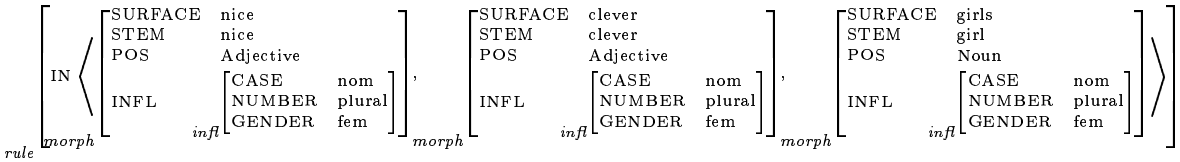


Figure 1: Matched input sequence.

example and note, that the instantiated LHS matches the pattern from the $\mathcal{XTDL}$ rule in example (1). A TFS representation of a rule contains the two attributes IN and OUT. In contrast to the IN value in

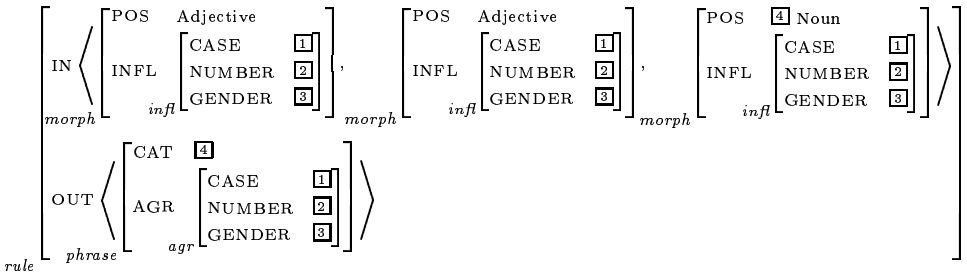


Figure 2: Rule with an instantiated pattern on the LHS.

the matched input TFS representation, the IN value of the rule contains coreference information. The value of OUT is the TFS definition of the RHS of the rule. Given the input TFS and the uninstantiated rule TFS, the unification of the two structures yields the final output result; see Fig. 3.
We note that the use of coreferences between the LHS and the RHS of a *SProUT* rule shares great similarities to lexical rules in PATR-II [Shieber *et al.*, 1983] and HPSG [Pollard and Sag, 1994]. The technique of embedding an instantiated LHS pattern and a RHS via the metafeatures IN and OUT also reminds us of the PATR-II system. The current implementation employs a longest match strategy. In case of match ambiguities, the result is a disjunction of RHSs. Since the output of the interpreter are

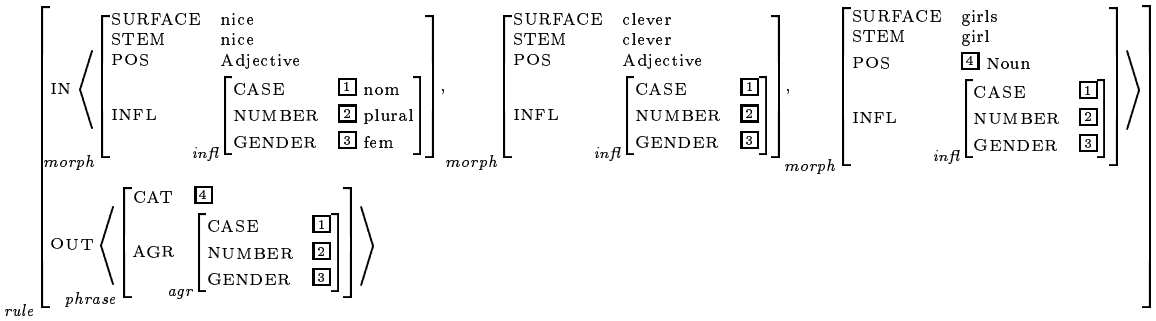


Figure 3: Unified result.

again TFSs, the result can be used as input for further (higher-level) linguistic processing components. In this way, *SProUT* supports cascaded architectures straightforwardly (see section 5).

5 Current System Instance

Currently the pool of linguistic processing resources contains a tokenizer, a gazetteer, and a morphology component. The tokenizer maps character sequences of the input text into word-like units called tokens and performs fine-grained multiple token classification: each token is firstly classified according to the main token type and secondly, depending on its main type, it undergoes additional domain and language specific subclassification. Since we aim at defining clear-cut components of linguistic analysis, the context information is disregarded during token classification. Therefore, sentence boundary detection constitutes a stand-alone module. The task of the gazetteer is recognition of named entities based on a stored list of static named entities. Finally, the morphology component provides lexical resources for English, German, French, Italian, and Spanish which were obtained from the full form lexica of MMorph [Petitpierre and Russell, 1995]. For asian languages, we integrated Chasen [Asahara and Matsumoto, 2000] for Japanese and Shanxi [Liu, 2001] for Chinese. All linguistic processing modules output their results uniformly as TFSs. The following table presents information on the size of grammars for four languages and the resulting automata.

	English	Japanese	Chinese	German
#rules	113	94	125	91
#nodes	387	353	706	230
#edges	2,769	4,831	6,985	1,157

6 Future Work

This section briefly outlines the areas of future work. We intend to conduct some experiments using the restrictor in the process of converting $\mathcal{XTDL}$ grammars into FS representations. As indicated in section 4.2, one should be able to find a reasonable trade-off between optimization of the FS network and the amount of potential pattern candidates for expensive unification.
The current implementation applies a longest match strategy to input tokens. We plan to replace this tie by a priority-driven agenda, allowing us to play with different search strategies, even all-path parsing.

In the current system, components are arranged in a strictly sequential fashion. We like to overcome this inflexible behavior by the following idea: since we use TFSs as the sole data interchange format between processing modules, it is likely that the construction of a concrete system instance can be reduced to a definition of a regular expression of module specifications. We foresee that the following operators needs to be available in a system description language: $\circ$ (concatenation), $*$ (iteration), and $|$ (concurrency).
The use of $\circ$ should be clear, e.g., $A \circ B$ expresses the fact that the output of module A serves as the input to B. This is the usual flow of information in a sequential cascaded shallow architecture. The $*$ operator over a module has the following interpretation: the module feeds its output back into itself until no more changes occur, thus implementing a fixpoint computation. It is clear that such a fixpoint might not be reached in finite time, i.e., the computation must not stop. A possible application was envisaged in [Braun, 1999]. His system was capable of parsing German clause sentential structures, where an iterative application of the base clause module was necessary to model recursive embedding of subordinate clauses. The third operator, $|$, denotes a quasi-parallel computation of independent modules, where the final output of each module serves as the input to a subsequent module.

Acknowledgments

We like to thank our former colleague Oliver Scherf who has contributed several ideas presented in this paper. Thanks are also due to Hans Uszkoreit and especially to Stephan Busemann for their moral support. Finally, we are grateful to the *SProUT* users which have provided us with valuable and constructive comments, helping us to improve the overall approach and performance of the system. This research was supported by the German Federal Ministry for Education, Science, Research, and Technology under grant no. 01 IW 002 (Whiteboard) & 01 IN A01 (Collate) and by an EU grant under no. IST 12179 (Airforce).

References

[Appelt, 1996] D. Appelt. The Common Pattern Specification Language. Technical report, SRI International, 1996.

[Asahara and Matsumoto, 2000] M. Asahara and Y. Matsumoto. Extended models and tools for high-performance part-of-speech tagger. In *Proceedings of the 18th COLING*, 2000.

[Braun, 1999] C. Braun. Flaches und robustes Parsen deutscher Satzgefüge. Master's thesis, Universität des Saarlandes, 1999.

[Crysmann *et al.*, 2002] B. Crysmann, A. Frank, B. Kiefer, S. Müller, G. Neumann, J. Piskorski, U. Schäfer, M. Siegel, H. Uszkoreit, F. Xu, M. Becker, and H.-U. Krieger. An integrated architecture for shallow and deep processing. In *Proceedings of the 40th ACL*, pages 441–448, 2002.

[Cunningham *et al.*, 2000] H. Cunningham, D. Maynard, and V. Tablan. Jape: a Java annotation patterns engine. Research memorandum cs-00-10, Department of Computer Science, University of Sheffield, 2000.

[Daciuk, 1998] J. Daciuk. *Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing.* PhD thesis, Techn. University Gdansk, Poland, 1998.

[Emele, 1991] M. Emele. Unification with lazy non-redundant copying. In *Proceedings of the 29th ACL*, pages 323–330, 1991.

[Hobbs *et al.*, 1997] J. Hobbs, D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson. Fastus: a cascaded finite-state transducer for extracting information from natural-language text. In E. Roche and Y. Schabes, editors, *Finite State Devices for Natural Language Processing*. MIT Press, 1997.

[Kaplan and Maxwell III, 1988] R.M. Kaplan and J.T. Maxwell III. An algorithm for functional uncertainty. In *Proceedings of the 12th COLING*, pages 297–302, 1988.

[Karttunen *et al.*, 1996] L. Karttunen, J.-P. Chanod, G. Grefenstette, and A. Schiller. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):305–328, 1996.

[Krieger and Schäfer, 1994] Hans-Ulrich Krieger and Ulrich Schäfer. *TDL*—a type description language for constraint-based grammars. In *Proceedings of the 15th COLING*, pages 893–899, 1994.

[Liu, 2001] Kaiying Liu. Research of automatic chinese word segmentation. In *International Workshop on Innovative Language Technology and Chinese Information Processing (ILT&CIP-2001)*, 2001.

[Mohri *et al.*, 1996] M. Mohri, F. Pereira, and M. Riley. A rational design for a weighted finite-state transducer library. Technical report, AT&T Labs, 1996.

[Mohri, 1997] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 2(23):269–311, 1997.

[Neumann *et al.*, 1997] G. Neumann, R. Backofen, J. Baur, M. Becker, and C. Braun. An information extraction core system for real world German text processing. In *5th ANLP*, pages 208–215, 1997.

[Petitpierre and Russell, 1995] D. Petitpierre and G. Russell. MMORPH—The Multext Morphology Program, 1995. Multext deliverable report 2.3.1. ISSCO, University of Geneva.

[Piskorski and Neumann, 2000] J. Piskorski and G. Neumann. An intelligent text extraction and navigation system. In *Proceedings of the 6th RIAO*, 2000.

[Piskorski *et al.*, 2002] J. Piskorski, W. Drożdżyński, F. Xu, and O. Scherf. A flexible xml-based regular compiler for creation and converting linguistic resources. In *Proceedings of the 3rd LREC*, 2002.

[Piskorski, 2002] J. Piskorski. The DFKI finite-state machine toolkit. Research Report RR-02-04, German Research Center for Artificial Intelligence (DFKI), 2002.

[Pollard and Sag, 1994] C. Pollard and I.A. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. University of Chicago Press, 1994.

[Roche and Schabes, 1995] E. Roche and Y. Schabes. Deterministic part-of-speech tagging with finite state transducers. *Computational Linguistics*, 21(2):227–253, 1995.

[Shieber *et al.*, 1983] S.M. Shieber, H. Uszkoreit, F. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In B.J. Grosz and M.E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*, pages 39–79. AI Center, SRI International, Menlo Park, 1983.

[Shieber, 1985] S.M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd ACL*, pages 145–152, 1985.

[van Noord and Gerdemann, 2001] G. van Noord and D. Gerdemann. Finite State Transducers with Predicates and Identity. Accepted for Grammars journal.