1

TSNLP — Test Suites for Natural Language Processing

STEPHAN OEPEN,
KLAUS NETTER, AND JUDITH KLEIN

1.1 Introduction

Given the growing number of natural language applications, there is an increasing demand for reference data other than the type of data found in large collections of text corpora . Test suites have long been accepted in the NLP context, because they provide for controlled data that is systematically organized and documented. However, with increasing and realistic applications of NLP technology a generation of general-purpose test suites will be required that (i) have broad coverage, (ii) are multilingual , and (iii) incorporate consistent and highly structured linguistic annotations . Linguistic databases of this kind will not only facilitate the deployment of test suites as diagnostic tools, but as well support different kinds of evaluation and serve as repositories for developers.

The objective of the TSNLP project[1] is to construct test suites in

[1]The project was started in December 1993 and completed in March 1996; the consortium is comprised of three academic partners — viz. the University of Essex (UK), the Istituto Dalle Molle per gli Studii Semantici e Cognitivi ISSCO (Geneva, Switzerland), and DFKI Saarbrücken (Germany) who have strong backgrounds in machine translation, evaluation, and natural language processing respectively — and the Common Research Center of Aerospatiale (Suresnes, France) as an industrial partner.

Most of the project results (documents, bibliography, test data, and software) as well as on-line access to the test suite database (see section 1.2.3) can be obtained through the world-wide web from the TSNLP home page `http://tsnlp.dfki.uni-sb.de/tsnlp/`.

The TSNLP project was funded within the Linguistic Research Engineering (LRE) programme of the European Commission under research grant LRE-62-089; special

three different languages building on a common basis and methodology. Specifically, TSNLP addresses a range of issues related to the construction and use of test suites. The main goals of the project are to:

- provide guidelines for the construction of test suites to facilitate a coherent and systematic data collection ;
- develop a rich annotation schema that is maximally neutral with respect to particular linguistic theories or specific evaluation and application types (section 1.2.1);
- construct substantial test fragments in English, German, and French that are applicable to different NLP application types (viz. parsers, grammar and controlled language checkers) (section 1.2.2);
- design and implement an extensible linguistic database to store, access, and manipulate the test data (section 1.2.3); and to
- investigate methods and tools for (semi-) automated test suite construction.

Both the methodology and test data developed currently are validated in a testing and application phase (section 1.2.4).

To ensure the wide distribution and dissemination of TSNLP test suites, the results of the project will be made available to the public domain. Ideally, the TSNLP annotation schema and database technology will serve as a starting and reference point for future test suite efforts, thus for the first time allowing the interchange of test data and the comparison of test and diagnostic results.

In the present paper the authors take the opportunity to present some of the recent outcome of TSNLP to the community of language technology developers as well as to potential users of NLP systems. Accordingly, the presentation puts emphasis on practical aspects of applicability and plausibility rather than on theoretically demanding research topics; the TSNLP results presented are of both methodological and technological interest.

1.1.1 Test Suites vs. Text Corpora

Test suites consisting of systematically constructed individual test items are not the only source of reference data for testing and evaluating NLP applications. In particular they clearly cannot replace test corpora drawn from naturally occurring texts but rather have to be seen as serving different and complementary purposes. Among the key differences

thanks to Dan Flickinger and John Nerbonne for the constructive peer reviews and to Roger Havenith of the CEC.

between test suites and corpora are the following (see Balkan et al. 1994 for more details):

- **Control over test data:** While test suites allow the construction of focussed test data using small vocabulary and illustrating specific phenomena in isolation or in controlled interaction, corpora will always comprise arbitrary combinations of different phenomena. Although the combinations of phenomena found in corpora may be empirically relevant, they do not allow for a focussed and fine-grained diagnosis of system performance.
- **Systematic coverage:** Test suites can provide test items which illustrate systematic variations over a specific phenomenon. In a corpus such variations are likely to occur only accidentally and they will not be related to each other.
- **Non-redundant representation:** While well-constructed test suites will list every phenomenon as concisely as possible, corpora are clearly redundant, in that one and the same phenomenon can occur over and over again.
- **Inclusion of negative data:** Ungrammatical examples do not occur naturally (or occur unsystematically) in corpora. However, for diagnostic purposes testing for negative data may be relevant (especially, for example, with grammar and controlled language checkers).
- **Coherent annotation:** As is exemplified in section 1.2.2, manually constructed test data can be associated with different kinds of systematic and coherent annotations . With few exceptions such annotations are hard to find in combination with corpora.

For most of these aspects it could be argued that, for example, the selection and combination of phenomena and the degree of variation within some particular phenomenon as they are found in corpora reflect what is actually relevant in the real world. However, corpora — regardless of their size — will only give a more or less representative sample and do not allow for a systematic diagnosis. Ideally, test suites and corpora should therefore stand in a complementary relation, with the former building on the latter wherever possible and necessary (section 1.3 briefly sketches the plans for future research on relating test suites and text corpora to each other).

1.1.2 TSNLP Objectives

The TSNLP project aims to construct test suites that address all of the desiderata listed in section 1.1. There is a special emphasis on the collection of minimal sets of test items reflecting the systematic variation of a single parameter while others remain controlled.

In order to encourage the distribution and assessment of TSNLP results, the annotation schema and selection of test data has to be maximally neutral with respect to a particular theory of grammar or individual languages. The test data construction and database technology developed shall encourage developers and users of NLP technology to tailor the test suites to their needs and to extend the database with user or application specific information.

1.2 Results of TSNLP

In both test data construction and database design issues TSNLP could — among others — build on experiences from the DiTo project previously carried out at DFKI (Nerbonne et al. 1993). The DiTo effort produced some 1500 systematically constructed and annotated sentences for four syntactic phenomena of German; the test data were organized into a simple relational database.[2]

The annotation schema and parts of the data developed in DiTo (for German at least) served as a starting point for work done in TSNLP. At the same time, however, the DiTo project had revealed two important problems that prompted the TSNLP consortium to pursue more elaborate approaches:

(i) Data construction in DiTo was directly carried out in the internal format defined by the database: the test data authors had to know and understand some of the database internals and produce the appropriate format; the method, obviously, is inefficient, laborious, and error-prone.

(ii) Although the interpreted pattern matching language `awk(1)` as the DiTo query engine provided for high-level string manipulation facilities, the hybrid combination with Un*x scripts and `yacc(1)` programs imposed severe limitations on efficiency as well as interfaces, portability, and simplicity.

Building on the experiences from the DiTo project, TSNLP put strong emphasis on tool building — to support test data construction independent of the database internal representation — as well as on scalable database technology. Sections 1.2.1, 1.2.2, and 1.2.3 present the TSNLP strategy in test data construction and database design respectively. Fi-

[2]Basically, the database software was implemented through the Un*x commands `sort(1)`, `uniq(1)` and `join(1)` combined with a simple query engine (an `awk(1)` script) and a query processor (realized in `yacc(1)` and `lex(1)`).

nally, in section 1.2.4 some initial results of the ongoing testing phase of test data and methodology are reported.

1.2.1 The TSNLP Annotation Schema

Based on a survey of existing test suites (Estival et al. 1994) and the types of annotations (if any) found, a detailed annotation schema has been designed which neither presupposes some token linguistic theory nor a particular evaluation type nor a limited range of suitable applications (see Estival et al. 1994 for a complete presentation).

Test data and annotations in TSNLP test suites, basically, are organized at four distinct representational levels:

- **Core data:** The core of the data collection are the individual *test items* (sentences, phrases et al.) together with all general, categorial, and structural information that is independent of phenomenon, application, and user parameters. Besides the actual string of words, annotations at this level include (i) bookkeeping records (date, author, origin, and item id), (ii) the item format, length, category, and wellformedness code, (iii) the (morpho-)syntactic categories and string positions of lexical and — where uncontroversial — phrasal constituents, and (iv) an abstract dependency structure. Encoding a dependency or functor-argument graph rather than a phrase structure tree avoids the need to impose a specific constituent structure but still can be mapped onto one (see section 1.2.4.3).
- **Phenomenon related data:** Based on a hierarchical classification of *phenomena* (e.g. verb valency being a subtype to general complementation) each phenomenon description is identified by its phenomenon identifier, supertype(s), interaction with other phenomena, and presupposition of such. Moreover, the set of (syntactic) parameters that is relevant in the analysis of a token phenomenon (e.g. the number and type of complements in the case of verb valency) is determined. Individual test items can be assigned to one or several phenomena and annotated according to the respective parameters.
- **Test sets:** As an optional descriptive level in between the test item and phenomenon levels, pairs or sets of grammatical and ill-formed items can be grouped into *test sets*. Thus, it can be encoded how, for example, test items that originate from the systematic variation of a single phenomenon parameter relate to each other.
- **User and application parameters:** Information that will typically correlate with different evaluation and application types in

Test Item		
item id: *24033*	author: *dfki-klein*	date: *jan-94*
register: *formal*	format: *none*	origin: *invented*
difficulty: *1*	wellformedness: *1*	category: *S_v2*
input: *Der Manager sieht den Präsidenten*		length: *5*
comment:		

position	instance	category	function	domain
0:2	*Der Manager*	*NP_nom-sg*	*subj*	*2:3*
2:3	*sieht*	*V*	*func*	*0:5*
3:5	*den Präsidenten*	*NP_acc-sg*	*obj*	*2:3*

Phenomenon		
phenomenon id: *24*	author: *dfki-klein*	date: *jan-94*
name: *C_Complementation*		
supertypes: *Complementation*		
presupposition: *C_Agreement, NP_Agreement*		
restrictions: *neutral*	interactions: *none*	purpose: *test*
comment:		

FIGURE 1 Sample instance of the TSNLP annotation schema for one test item: annotations are given in tabular form for the *test item*, *analysis*, and *phenomenon* levels. The *function* and *domain* attributes encode the abstract dependency structure using (zero-based) substring positions to refer to parts of the test item.

the use of a test suite is factored from the remainder of the data into *user & application profiles*. Currently, there are foreseen (i) a centrality measure on a per phenomenon, test set, or even test item basis (e.g. user x may in general consider phenomenon y to be central but test set z to be marginal within y) and (ii) judgements of relevance with respect to a specific application. To match the results obtained from a token NLP system (a parser, say) against a test suite, a formal or informal output specification can be added at the profile level.

In addition to the parts of the annotation schema that follow a more or less formal specification, there is room for textual comments at the various levels in the above hierarchy to accommodate information that cannot be formalized.

1.2.2 Test Data Construction

Following the TSNLP test suite guidelines (Balkan et al. 1996) and using the annotation schema sketched in section 1.2.1, the construction of

Phenomenon	English	French	German	Total
C_Complementation	117 | 845	225 | 639	218 | 246	560 | 1730
C_Agreement	68 | 55	104 | 183	224 | 175	396 | 413
C_Modification		329 | 63		329 | 63
NP_Complementation		12 | 28		12 | 28
NP_Agreement	201 | 993	272 | 1082	299 | 1732	772 | 3807
NP_Modification	302 | 509		53 | 60	355 | 569
Diathesis	201 | 61	176 | 118	147 | 148	524 | 327
Tense Aspect Modality	157 | 39	77 | 275	186 | 134	420 | 448
Sentence Types	80 | 100	322 | 454	105 | 14	507 | 568
Coordination	147 | 186	379 | 320	105 | 429	631 | 935
Negation	289 | 129	62 | 106	82 | 210	433 | 445
Word Order		7 | 7	60 | 160	67 | 167
Extragrammatical	24 | 34		253 | 0	277 | 34
Total	1586 | 2951	1965 | 3275	1732 | 3308	5283 | 9534

FIGURE 2 Status of the data construction as of March 1996: relevance and breadth of individual phenomena differ for the three languages (the figures are grammatical vs. ungrammatical items respectively).

test data was based on a classification of (syntactic) phenomena to be covered. From judgements on linguistic relevance and frequency for the individual languages[3] the following list of *core phenomena* for TSNLP was compiled:

- complementation;
- agreement;
- modification;
- diathesis;
- modality, tense, and aspect;
- sentence and clause types;
- topology and word order;
- coordination;
- negation; and
- extragrammatical (e.g. parentheticals and temporal expressions).

[3]Given the very limited resources for corpora studies in TSNLP (see Dauphin et al. 1995a), the choice of syntactic phenomena for the three languages was primarily based on subjective assessments obtained from experienced linguists and grammar engineers. Naturally, the current list is incomplete and cannot be considered exhaustive.

A further sub-classification of phenomena relates to the relevant *syntactic domains* in which a phenomenon occurs; for the above list these are sentences (S), clauses (C), noun phrases (NP), adjectival phrases (AP), prepositional phrases (PP), and adverbial phrases (AdvP). Cross-classified phenomena names are composed by attaching the syntactic domain as a prefix to the phenomenon name (e.g. *C_Complementation, NP_Agreement* et al.) and can be further sub-classified according to phenomenon-internal dimensions.

Figure 2 gives a survey of the test material constructed in TSNLP (as of March 1996); although — because of language-specific differences and the parallel construction of the data — coverage varies across the three languages, all three test suites have a comparable breadth of analysis for each of the phenomena (each totaling some 4500 test items). The vocabulary used in the TSNLP test suites in general is drawn from the business domain and aims for a minimally-sized lexicon (complete vocabulary lists for the three languages are given in Lehmann et al. 1996); current lexicon sizes range between 550 and 900 inflected forms of (on average) some 400 lexemes.

As the identification of relevant parameters for each of the phenomena turned out to be difficult and arguable in several cases, not all of the test data are annotated to the same depth on the phenomenon-specific level. Nevertheless, for the phenomena agreement, complementation, and topology and word order (and in part for sentence and clause types as well) the set of relevant parameters has been determined and used in annotating the test data. The ungrammatical test items for each phenomenon are coarsely classified according to the source of ungrammaticality (ideally resulting from the variation of a single parameter).

Following is an example for the C_Complementation (verbal government) phenomenon which is characterized by two parameters: (i) the total number of (obligatory) complements; and (ii) a summary of the actual valency frame.[4] For the sentence *Der Manager sieht den Präsidenten.* ('the manager sees the president.'), for example, the parameter values are:

[4]Obviously, both of the C_Complementation parameters are redundant with respect to the information that is already encoded in the phenomenon-independent *analysis* part of the *test item* annotations. However, making the valency information explicit as two separate per phenomenon parameters allows one to explain the source of ungrammaticality in ill-formed test items quite straightforwardly. Additionally, the parameters (at least intuitively) correspond to the procedures that can be applied to grammatical test items in order to derive ill-formed examples: in the case of verbal government ungrammatical items either result from the deletion of complements or from a violation of the governed properties.

Category	**German Example**
S_imp-v1	*Denke an den Test!*
S_non-imp-v1	*Denkt der Manager an den Test?*
S_wh-v2	*Wer kommt?*
S_v2	*Der Manager denkt an den Test.*
S_cpl(dass)-vfinal	*Daß der Student nur ja an den Test denkt.*
C_cpl(dass)-vfinal	Der Manager glaubt, *daß der Student arbeitet.*
C_cpl(ob)-vfinal	Es ist fraglich, *ob der Student arbeitet.*
C_v2	Der Manager sagt, *der Chef halte den Vortrag.*
C_wh-v2	Der Manager fragt: *Wer hält den Vortrag?*
C_wh-vfinal	Der Manager fragt, *wer den Vortrag hält.*
C_rel-vfinal	Der Manager, *der kompetent ist*, arbeitet.

FIGURE 3 Excerpt from the list of categories used in annotations at the sentential and clausal levels: lexical properties (e.g. the type of complementizer) appear in parentheses; language-specific dimensions (e.g. the verb-initial vs. verb-second vs. verb-final distinction for German) are cast into suffixes.

number of complements: *2*
valency frame: ⟨ *subj (NP_nom)* , *obj (NP_acc)* ⟩

The corrsponding ill-formed test items derived from this example

**Der Manager sieht.*
**Der Manager sieht [dem Präsidenten]$_{dat}$.*

are classified according to the respective parameter violated and linked to the positive example into a *test set* (see section 1.2.1).

In order to enforce consistency of annotations across the three languages — even though the test data construction is carried out in parallel — a canonical list of categories and functions used in the description of categorial and dependency structure had to be established; figures 3 and 4 present a (small) excerpt from the two lists. The dimensions chosen in the classification attempt to avoid properties that presuppose very specific assumptions of a particular theory of grammar (or language), and rather try to capture those distinctions that evidently are relevant across the set of TSNLP core phenomena; thus, the subclassification of German clause types (see figure 3), for example, mostly corresponds to what is actually realized as clausal complements for the verbal government phenomenon.

To ease the laborious test data construction and to reduce erratic variations in filling in the TSNLP annotation schema, a graphical test suite construction tool (tsct) has been implemented. The tool instan-

Function	German Example
func	Der Manager *arbeitet.*
func	Es liegt *daran*, daß der Manager arbeitet.
subj	*Der Manager* arbeitet.
p-obj(um)	Der Manager bittet *um den Test.*
p-obj-loc	Der Manager wohnt *in der Stadt.*
subj-pred-adj	Der Manager ist *krank.*
mod-loc-place	Der Manager arbeitet *in der Stadt.*
mod-temp-duration	Der Manager arbeitet *den ganzen Tag.*
mod-manner	Das *alte* Projekt
spec	Alle *diese* Manager arbeiten.

FIGURE 4 Excerpt from the list of functions used in dependency structure annotations: examples of functors, arguments, modifiers, and specifiers.

tiates the annotation schema as a form-based input mask and provides for (limited) consistency checking of the field values. Additionally, tsct allows the reuse of previously constructed and annotated data, as often when constructing a series of test items it will be easier to duplicate and adapt a similar item rather than producing annotations from scratch.

1.2.3 The Test Suite Database

One of the primary technological objectives of TSNLP was to design and implement a linguistic database that can ease the storage, maintenance and retrieval of natural language test data.

1.2.3.1 Motivation

According to the specific TSNLP requirements (see section 1.1.2), the key desiderata of the database design are:

- **usability**: to facilitate the application of the methodology, technology and test data developed in TSNLP to a wide variety of diagnosis and evaluation purposes of different applications by developers or users with varied backgrounds;
- **suitability**: to meet the specific necessities of storing and maintaining natural language test data (e.g. in string processing) and to provide maximally flexible interfaces;
- **extensibility**: to enable and encourage users of the database to add test data and annotations according to their needs without changes to the underlying data model;
- **portability and simplicity**: to make the results of TSNLP available on several different hard- and software platforms free of royalties and easy to use.

Although the hierarchical organization of phenomena and the concept of per-phenomenon parameters (see section 1.2.1) seem to suggest a data model incorporating inheritance relations (as they are known from object-oriented or taxonomical databases), hierarchical relations are of only limited significance in the annotation schema. Therefore — and also to most closely approximate the five desiderata — the TSNLP database takes a conservative approach building on plain relational database technology.

Because it is expected that the applicability and acceptance of the tools (i.e. the technology) produced by the project will be one of the keys to the wide distribution and deployment of the methodology and test data developed, a special emphasis in the database design was put on aspects of flexibility and interfaces (to both people and programs). As a matter of fact, however, the technological requirements of developers or users from an academic background typically differ substantially from those in an industrial environment. To account for the two different areas of application, the TSNLP database was implemented taking a dual strategy.

Firstly, to produce a tool that is well tailored to the purpose of storing and retrieving TSNLP test data, highly flexible in its interfaces and portable and free of royalties at the same time, a small and simple relational database engine in plain ANSI C (which we presently call tsdb_1 using the subscript to distinguish between the two parallel implementations of tsdb; see section 1.2.3.4) was implemented. Since the expected size of the database as well as major parts of the database schema are known in advance, it seemed plausible to impose a few restrictions on the full generality of the relational algebra that allow fine-tuning and simplification of both the program and especially the query language.

Secondly, an alternative implementation was carried out through the exploitation of a commercial database product (called tsdb_2; see section 1.2.3.5) to produce a tool that meets the demands of potential users, especially in a non-academic context. Because commercially available database systems have different assets (and typically many more features) than the home-grown implementation tsdb_1, the dual realizations of the TSNLP database carried out in parallel nicely complement each other and provide for valuable feedback and comparison.[5]

[5]Recently, in an extension to the original TSNLP work plan, a third implementation of the test suite database has been started at the University of Essex; building on the popular Microsoft Access database package, this version of tsdb has very similar design goals and features to the tsdb_2 realization.

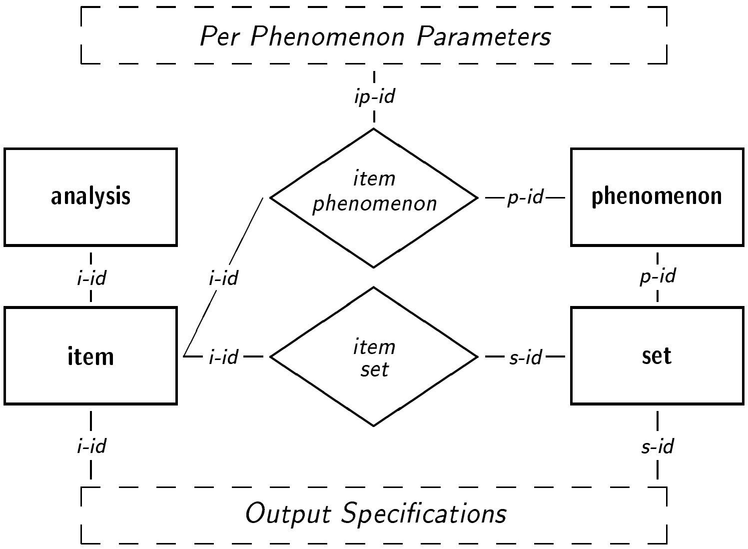

FIGURE 5 Graph representation of the TSNLP database schema. Distinct relationship types (viz. *item-set* and *item-phenomenon* are postulated for $m \times n$ relations where necessary.

1.2.3.2 The Database Schema

While the data format deployed in the test data construction tool tsct mirrors the rough four-level classification given in section 1.2.1 and is primarily motivated by data construction considerations, the requirements for the abstract TSNLP data model (and analogously for the tsdb database schema) were somewhat different. Although the classification into four basic tables (corresponding to the four tsct windows) serves very well for data input purposes, it falls short from a data storage and maintenance perspective in that (because of the tsct reuse and duplicate facilities) the resulting data files are highly redundant (i.e. repetitive information is stored over and over with the individual items).

Accordingly, during the database import the four-level classification given in section 1.2.1 had to be refined and cast into more specific relations that allow for the elimination of redundancy and (i) factor the basic entity (or object) types from the relationship types and (ii) make this distinction explicit, in order to allow for variable logical views on the data and to enable the retrieval engine to compute the appropriate combined (or virtual) relations required to process complex queries. As another interesting facet of the import procedure, list- or multi-valued attributes which typically are not foreseen in plain relational databases are normalized as multiple records in a separate relation.[6]

Figure 5 presents the core of the database schema (in an abstract entity-relationship graph representation) as it is currently in use; still missing in the presentation is an account of how *user & application profiles* are linked to the core data (see section 1.2.4 for an example).

1.2.3.3 Query and Retrieval: Some Examples

For the TSNLP test data the basic purpose of the relational database retrieval facilities are:

- to dynamically construct test suite instances according to an arbitrary selection from the data; and
- to employ the query engine to compute appropriate virtual (join) relations that contain everything relevant for a particular task, but at the same time hide all internal, bookkeeping or irrelevant data.

— Following are a few retrieval examples — formulated in the simplified SQL-like query language that is interpreted by the home-grown implementation tsdb_1 (see section 1.2.3.4) — that demonstrate some of the flexibility of the annotations and retrieval engine (see Oepen et al. 1996 for more examples and complete documentation on the database and query language):[7]

- find all grammatical verb-second sentences relevant to the phenomenon of clausal (i.e. subject verb) agreement:

```
retrieve i-input
  where i-wf = 1 && i-category = "S_v2" &&
        p-name = "C_agreement"
```

- collect all grammatical sentences with pronominal subjects together with their identifiers and categories:

[6]For example the tsct representation of test sets — relating a comma-separated list of positive test items (or rather their identifiers) to a set of negative items — is transformed into multiple data records that relate the individual items to one and the same test set (again, its identifier). Because the lists of positive and negative items may not match up pairwise, the polarity (positive or negative) for each of the items is cast as an attribute of its own (the new *polarity* attribute rather than the *i-wf* grammaticality judgement) in order to allow for test sets in which well-formed items can serve as negative examples or vice versa; in testing a controlled language checker, for example, it may be desirable to construct test sets containing grammatical items that still are assigned negative polarity because they are not part of the controlled target language (e.g. based on a distinction between active vs. passive voice).

[7]Attribute names incorporate the first character of the defining (or base) relation as a prefix: thus, *i-input*, for example, corresponds to the *input* field of the test *item* relation (see figures 1 and 5).

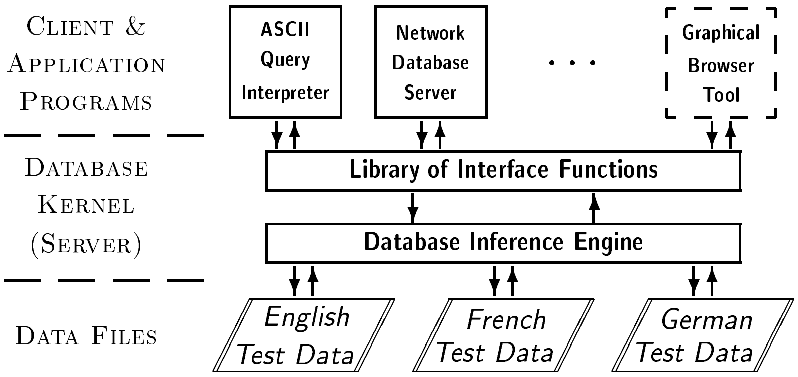

FIGURE 6 Rough sketch of the modular tsdb_1 design: the database kernel is separated from client programs through a layer of interface functions.

```
retrieve i-id i-input i-category
  where i-wf = 1 &&
        a-function = "subj" && a-category ~ "PRON"
```

1.2.3.4 Implementing tsdb_1: Building it Ourselves

Although TSNLP had no intention to compete with commercial database products in terms of efficiency and generality, there are good reasons speaking in favour of the proprietary implementation of tsdb_1 within the project:

+ the database software and query language can be fine-tuned to the test suite extraction task; because the focus of natural language test data retrieval is on string manipulation, the tsdb_1 implementation is enriched with regular expression matching (a concept that is often unavailable in standard relational database products);
+ both data *and* software can be made available free of royalties;
+ the database can be given a flexible and well-documented interface allowing the connection to arbitrary (client) applications within or outside of TSNLP; the applicability of the interface so far has been exemplified in three cases (viz. the ASCII query interpreter, the network server, and the DFKI HPSG system);
+ the data format can be made transparent (similar to the tsct ASCII files) and thus accessible to standard Un∗x text processing tools.

Figure 6 gives an overview of the tsdb_1 architecture: the database kernel is constituted by the inference engine and a separate layer of interface functions that are organized into a library of C functions.

Any application that is capable of interfacing to a library of external functions (e.g. programs that can be statically linked with the library or Common-Lisp or Prolog applications by use of the foreign function interface) can be given full bidirectional access to the database. Within TSNLP, two of the client applications have been developed: (i) an ASCII-based query interpreter that allows the retrieval of data in a subset of the SQL query language (figure 7 shows a screen dump from a retrieval session using tsdb_1) and (ii) a network database server that allows (read-only) access to the database from remote machines in a TCP/IP network. The tsdb_1 query language provides for the standard numeric and string comparison operations plus regular expression matching, boolean connectives (conjunction, disjunction, and negation of conditions), and a dynamic query result storage.

Other applications to connect to the TSNLP database include a lexical replacement tool built by Essex (see Arnold et al. 1994) which so far reads and writes data files in the tsct format and possibly a graphical browser for test data derived from tsct itself.[8]

The implementation of tsdb_1 is in ANSI C and supports (among others) Un∗x workstations as well as Macintosh and Intel x86-based personal computers. The interface specification is as a C header file to be included by client programs.

1.2.3.5 Implementing tsdb_2: Commercial Database Software

Even though the ANSI C implementation of tsdb_1 allows one to deploy the tool in a standard personal computer environment, an alternative implementation of tsdb based on a commercial database product was carried out in parallel in order to produce a tool that has a *look + feel* familiar to personal computer users from typical office software; additionally, building on common standard software the TSNLP results will be better accessible to users who have to rely on the availability of on-site technical support.

Based on the evaluation of several low- and medium-priced commercially available database systems for Macintosh and Intel x86-based personal computers running under Microsoft Windows, the product of choice was Microsoft FoxPro (in its current version 2.6) because it (i) is available for both platforms, (ii) comes with high-level interface building support for a graphical front end to the TSNLP database, and (iii) is reasonably priced. The parallel implementation of two tsdb versions

[8] However, it rather seems that the implementation of tsdb based on the commercial database software Microsoft FoxPro (see section 1.2.3.5 for details) will naturally be superior in terms of convenient graphical browsing of the test data, as FoxPro comes with high-level support for graphical interfaces.

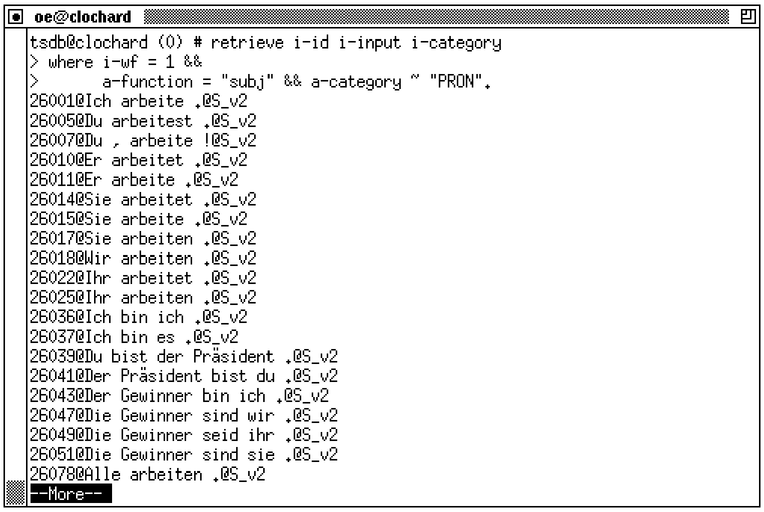

FIGURE 7 Screen dump of a tsdb_1 session: the query interpreter (topmost three lines) has command line editing, input history and relation and attribute name completion facilities.

is possible because both tsdb_1 and tsdb_2 use exactly the same database schema (see section 1.2.3.2); thus the two resulting databases come out as one-to-one replicas of each other such that data can be freely exchanged between them at any time.

While the home-grown implementation of tsdb_1 scores high for its flexible functional interface (to client software), the FoxPro version tsdb_2 has a more advanced user interface (see figure 8 for a screen dump of one of the windows in the current development version). Since FoxPro encourages the implementation of graphical, mouse- and menu-based forms that hide away technical details of the underlying database, tsdb_2 comes with a sophisticated graphical browser and editor for the TSNLP test data that is (at least) equivalent in its functionality to the X11-based test data construction tool.

1.2.4 Putting it to the Test

To validate the TSNLP annotation schema and test data, the project results have been tested against three different application types: a grammar checker (*Le Correcteur* for French), a controlled language checker

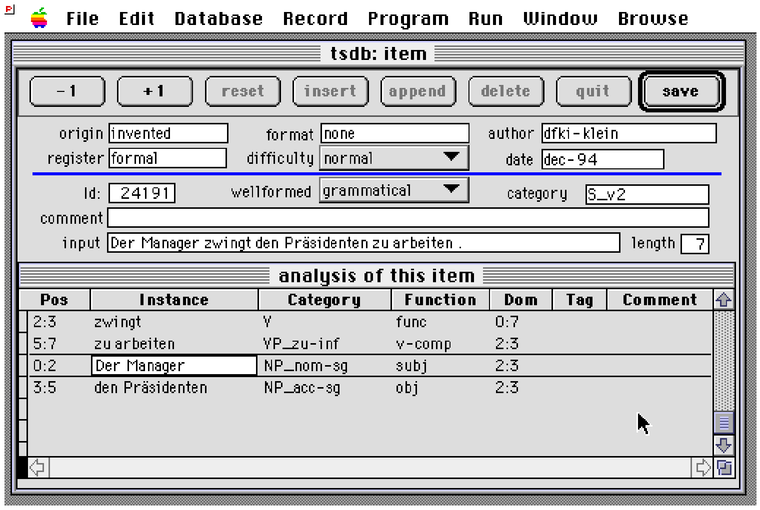

FIGURE 8 Screen dump of the development version of tsdb_2: the FoxPro database software has high-level support for graphical browsing of the data.

(*SECC* for English) and a parser (the HPSG system developed at DFKI[9] for German). Following is a brief summary of some of the testing results obtained for the German test suite (for testing results for the other two test suites and applications see Dauphin et al. 1995b).

1.2.4.1 A Glass Box: Connecting TSNLP to an HPSG System

Whereas for the grammar and controlled language checkers there was no or only limited access to internal datas tructures or the control flow within the application (i.e. the application as a whole seen from the outside is a *black box*), the evaluation situation was different for the PAGE

[9]The PAGE (Platform for Advanced Grammar Engineering) system has continuously been developed over more than five years by the DFKI Computational Linguistics group. The current development version combines

- a powerful state-of-the-art grammar development system including a rich typed feature formalism and unifier, a parameterizable parser and generator, a command shell, and several development tools;
- linguistic competence for German (morphology, syntax and, semantics);
- new methods for linguistic performance modelling; and
- an interesting sample application for appointment and calendar management.

The system has been delivered to several international research institutions (including CSLI Stanford, Simon Fraser, Ohio State, and Brandeis Universities, and German sites taking part in the international VerbMobil machine translation project); see Uszkoreit et al. 1994 for more details.

parser for German: since the test suite and the parser have been developed by the same group, there was full access to the internal structures of individual modules as well as to the flow of control. Testing TSNLP results in this setup amounted to an ideal *glass box* evaluation.

In order to take full advantage of this favourable evaluation situation when connecting the PAGE system to the TSNLP test suites and diagnosing the German HPSG grammar, individual steps were to

- identify and tailor relevant test data (including lexical replacement where necessary);
- establish a bidirectional interface to the TSNLP database allowing the application to retrieve *and* store data (through the Common-Lisp foreign function interface and the tsdb_1 interface library);
- supplementarily: add output specifications (the number of readings, phrase structural or semantic representations et al.) to a PAGE *user & application profile* (see section 1.2.4.2);
- at the application side: investigate a mapping between structures derived from the grammar and the TSNLP morpho-syntactic and relational annotations (see section 1.2.4.3);
- run automated *retrieve, process, and compare* cycles in grammar diagnosis and progress evaluation; and to
- deploy the TSNLP database to store additional application-specific data (e.g. performance measures).

Given this bidirectional connection to the test suite, the DFKI PAGE system presumably is one of the few grammar development platforms seamlessly integrated with a multi-purpose , multi-lingual test suite[10], thus enriching the grammar engineering environment with a validated and comparable evaluation facility and highly valuable coverage and performance measures for the ongoing grammar development for (currently) German and English.[11]

Additionally, initial work done on the automated mapping of HPSG feature structures as they are derived by the PAGE grammar into the TSNLP annotations at lexical and phrasal levels (see section 1.2.4.3 be-

[10] Although in the Hewlett-Packard NL system already there was a close connection to the Hewlett-Packard test suite (see Flickinger et al. 1987), the link was unidirectional rather than bidirectional: since the HP test suite (similar to most first generation test suites) is organized as a set of ASCII files, the NL system would retrieve a set of test items, process them in a batch job, and compare the results to the (shallow) annotations available.

[11] A similar degree of integration is currently aimed for in a cooperation with the developers of the ALEP (Advanced Linguistic Engineering Platform) grammar development system funded by the European Commission.

i-id	system	version	readings	first	total	user	date	error
11001	page	4711	1	0.9	2.0	oe	23-sep-1995	
11006	page	4711	2	3.6	8.8	oe	23-sep-1995	
11178	page	4711	-1			oe	23-sep-1995	bus error

i-id	nll (normalized meaning representation)
11001	direct(theme:^(∃ ?x ?y addressee(instance:?x) work(instance:?y agent:?x)))
11006	direct(theme:^(∃ ?x ?y addressee(instance:?x) work(instance:?y agent:?x)))
11006	ask(theme:^(∃ ?x ?y addressee(instance:?x) work(instance:?y agent:?x)))

FIGURE 9 Information added in customizing the German test suite for testing the PAGE HPSG grammar: the PAGE *user & application profile* records processing measures (e.g. the grammar version and date of testing, the number of readings obtained, the processing times for finding the first reading and overall total, and the type of processing error where applicable) for each of the test items. Rather than bracketed phrase structure trees, a normalized semantic representation serves as an output specification allowing judgements on the quality of analyses and to identify spurious ambiguities.

low) into allowed for the validation of the type and depth of annotations chosen.

1.2.4.2 Diagnostic Results Obtained

In order to facilitate a continuous progress evaluation (i.e. the automated comparison of actual system performance in terms of coverage and efficiency after changing the software or grammar to the expected or former behaviour) for grammar development in PAGE, the German test suite was customized and extended following the *user & application profile* approach (see section 1.2.1): all information specific to PAGE and the German HPSG grammar is stored into a designated place in the database (viz. two specific and separate relations) such that the generality of the overall test suite is not affected; figure 9 gives an excerpt from the resulting PAGE *user & application profile*.

Using the customized German test suite and the test data available at the beginning of the testing phase (3674 test items for 9 phenomena) three complete test cycles (each of more than a full day in processing time[12]) were carried out allowing to (i) debug the interface and database technology, (ii) determine the system coverage and precision and to

[12] In order to obtain comparable efficiency measures, the system is run in development mode and reset to its initial state for each of the test items; hence, even though the actual parsing time totals less than two hours, the batch processing mode adds a substantial time overhead.

elimininate data flaws, and (iii) compute reference data for future diagnosis (including manual inspection and correction of the $\mathcal{NLL}$ output specifications).

Feedback for the Test Suite The overall feedback from testing the German test suite with the PAGE system was very positive: both, the test data and the database technology have proven highly adequate and sufficiently flexible to provide for a valuable diagnostic tool in the (progress) evaluation of a constraint-based parser. The test suite, annotations, and software could be taken as-is, yet allowing the smooth and seamless integration into a complex grammar development system.

Besides the feedback for the application, comparing the German test data to the analyses derived by the PAGE HPSG grammar revealed some encoding mistakes in the test data (often resulting from human failures — typing errors and misclassifications — that were not detected during the import and consistency checking), concerning above all unexpected and unintended ambiguities detected by the system as well as other inconsistencies. However, even without this additional validation and feedback the test data already turned out to be of very high quality.

Feedback for the Application Although the testing within TSNLP proper had its focus on the validation of the project methodology, test data, and technology, naturally, several diagnostic results for the application were obtained at the same time.

Fundamental in both judging the current status of the application as well as for the continuous progress evaluation while the system evolves is the classification of *output mismatches*, i.e. test items for which the expected performance of the application is different from the actual result obtained. Using the information stored in the PAGE *user & application profile* and the results obtained in the second and third test cycles, the database retrieval engine allows one to extract and count classes of output mismatches:

mismatches of expected vs. actual output	**2$^{\text{nd}}$ cycle**	**3$^{\text{rd}}$ cycle**
wellformed items without analysis	868	751
illformed items with analysis	235	43
spurious ambiguities ($\geq$ 3 readings)	27	19
fatal system errors	7	0
total reduction of mismatches	**29 %**	

While the first class of output mismatches (grammatical test items that could not be analyzed by PAGE) is mostly determined by lexical gaps in the grammar[13] and restrictions on the treatment of coordination, the

[13]Since the testing for different complementation patterns (e.g. verbal complemen-

second class (supposedly ill-formed test items that nevertheless have one or more analyses) predominantly indicates overgeneration in the application and allowed the grammar writer to identify two systematic sources of failure (viz. spurious rule application and lexical misclassification) Besides, two serious software failures were identified that in several years of (unsystematic) manual testing had not manifested.

1.2.4.3 Mapping HPSG structures into TSNLP Categories

In order to maximally profit from the TSNLP annotations when testing the HPSG analyses produced by the PAGE parser, it becomes necessary to map the HPSG typed feature structures onto the atomic category names used in the TSNLP database and to compute appropriate dependency relations from a derivation tree (a parsing result).

Building on a typed feature structure rewrite engine (the zebra system (see Konrad 1995) developed at DFKI) that allows the compilation and recursive application of individual or groups of rewrite rules over typed feature structures, an incomplete version of a suitable mapping procedure for PAGE result structures has been implemented. Without caring for the details of HPSG (see Pollard and Sag 1994), in the following the method and technology are illustrated by a few simplified examples.

In general, by partitioning rewrite rules into disjoint sets that are applied sequentially, it is possible to process heavily nested feature structures inside-out; for example for the mapping into TSNLP annotations it seems plausible (i) to rewrite embedded categories (i.e. lexical and phrasal nodes) first, then (ii) to deploy another set of rewrite rules to collect the results from step (i), and (iii) finally to compute the dependency relations in a third stage.

Example (1) is a simplified version of the rule that rewrites the HPSG representation of a (dative singular) saturated noun phrase into the TSNLP equivalent.[14] The context for an application of rule (1) (i.e. the pattern on the left hand side) is strictly local: it is restricted to the token phrase being rewritten.

tation in the *C_Complementation* phenomenon) is partly a matter of lexical rather than of syntactic overage, obviously further test suite or grammar customization will be required.

[14]Although in the rewrite rule examples for expository reasons type names are omitted where they are not strongly required to constrain the context, in the zebra system each of the embedded feature structures is assumed to bear an appropriate type. Because the types used in the specification of rewrite rules can be taken from exactly the same hierarchy that constitutes the parsing grammar itself, the set of TSNLP rewrite rules can be interpreted as a proper and declarative extension to the PAGE grammar.

$$(1)\quad \underset{sign}{\begin{bmatrix} \texttt{CAT}\,|\,\texttt{LOC}\,|\,\texttt{SYN} & \begin{bmatrix} \texttt{SPEC} & \langle\rangle \\ \texttt{COMPS} & \langle\rangle \\ \underset{noun}{\texttt{HEAD}} & \begin{bmatrix} \texttt{AGR} & \begin{bmatrix} \texttt{CASE} & dat \\ \texttt{NUM} & sg \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}} \longrightarrow \mathit{NP\_dat\text{-}sg}$$

For the extraction of dependecy structures from larger phrases and sentences, typically a larger context is required. In order to identify the subject and object(s) of a verbal head (a functor in the TSNLP sense), for example, the entire phrase structure tree headed by the verb (the domain of the functor) has to be taken into account: even though the grammatical functions can be determined from the valency features of the verbal head (structure $\boxed{1}$ in example (2)), the remainder of the derivation tree (represented in (2) as a HPSG DTRS structure) contains the information about the actual realization of the subcategorized complements and the corresponding string positions.[15]

$$(2)\quad \boxed{4}\ \underset{sign}{\begin{bmatrix} \texttt{DTRS} & \begin{bmatrix} \texttt{H-DTR}\,|\,\texttt{DTRS} & \begin{bmatrix} \texttt{H-DTR} & \boxed{1} \begin{bmatrix} \texttt{CAT}\,|\,\texttt{LOC}\,|\,\texttt{SYN} & \begin{bmatrix} \texttt{SUBJ} & \langle \boxed{2} \rangle \\ \texttt{COMPS} & \langle \boxed{3} \rangle \end{bmatrix} \end{bmatrix} \\ \texttt{C-DTR} & \boxed{3} \end{bmatrix} \\ \texttt{S-DTR} & \boxed{2} \end{bmatrix} \end{bmatrix}}$$

$\longrightarrow$

a-position	a-category	a-function	a-domain
$\lvert\boxed{1}\rvert$	$\boxed{1}$	*func*	$\lvert\boxed{4}\rvert$
$\lvert\boxed{2}\rvert$	$\boxed{2}$	*subj*	$\lvert\boxed{1}\rvert$
$\lvert\boxed{3}\rvert$	$\boxed{3}$	*obj*	$\lvert\boxed{1}\rvert$

As of March 1996 a categorial mapping for root categories of TSNLP test items (i.e. the *category* field in the *item* relation) has been established and used to automatically compare the results obtained from PAGE with the annotations found in the TSNLP test suite. Analogously to the grammaticality results, the categorial information in several cases has proven to be valuable in testing the PAGE application. Regarding feedback for the test suite, the mapping of HPSG feature structures into TSNLP categories made it possible to detect two systematic error sources in the data: firstly, it greatly helped in spotting incorrect or inconsi-

[15] In example (2) the notation '$\lvert\boxed{1}\rvert$' is used to indicate the position of the substring corresponding to the feature structure $\boxed{1}$. String positions in HPSG can in general be read off the PHON feature (a glossed list-valued representation); however, since the PAGE system incorporates a chart-based parsing algorithm, a more straightforward option is to extract the position values immediately from the chart.

stent categorial annotations (e.g. two different test data authors had used slightly distinct categories for the same lexical item and type of construction); secondly, it provided the basis for the identification of categorial ambiguities that test data authors had not been aware of (e.g. the use of the form *arbeiten* ('work') as both a verb and a noun).

However, unless the similar exercise is completed for TSNLP functional annotations, it remains difficult to judge to what extent an exhaustive mapping onto the TSNLP annotations can be established and inhowfar the granularity of test data and level of annotations suffice for the thorough and automated matching against a highly formalized theory of grammar such as HPSG.

1.3 Future Work: Beyond TSNLP

An issue that has been discussed several times is that general-purpose test suites normally list items on a par without rating them in terms of frequency or even relevance with respect to an application. Clearly, test suites without this information cannot serve as the basis of an evaluation in the same way as application-, task- or domain-specific corpora do, unless a human evaluator assigns a personal ranking to the test data.

As to relevance, presumably a manual linguistic judgement will always be required, since a relevant phenomenon need not necessarily be a frequent phenomenon.[16]

As to frequency, one of the major challenges in constructing and applying test suites is the combination of the controlled test suite environment with information about the frequency of individual constructions and classes of phenomena to be found in text corpora. One approach envisaged to extend the TSNLP approach into this direction is to match tagged test items against analogously tagged corpora, such that the matching abstracts away from additional material in the corpus, taking into account only the relevant parts of a sentence. Quite obviously, relating test suites to text corpora along these lines is a long-term scientific goal that will require future research.

Additional straightforward extensions of TSNLP results will be to increase the coverage of the test data and to apply the methodology to more languages.

Acknowledgements

Most of the TSNLP results presented originate from joint research of the TSNLP consortium; besides, many of the underlying ideas — especially

[16]The correct analysis of negated constructions, for example, can be crucial to an application even though the type of construction may be relatively infrequent.

relating to the kind of linguistic resource and type of annotations required — and the initial project stimulus are due to Hans Uszkoreit.

The authors are most grateful for the constructive cooperation and acknowledge the contributions of the partners Doug Arnold, Lorna Balkan, Frederik Fouvry, and Siety Meijer (University of Essex, UK), Dominique Estival, Kirsten Falkedal, and Sabine Lehmann (ISSCO, Geneva, Switzerland) and Eva Dauphin, Veronika Lux, and Sylvie Régnier-Prost (Aerospatiale, France).

Major parts of the test data construction and implementation work at DFKI have been and still continue to be carried out by diligent research assistants; the authors gratefully appreciate the enthusiasm of Judith Baur, Tom Fettig, Fred Oberhauser, and Andrew P. White.

Bibliography

Arnold, Doug, Martin Rondell, and Frederik Fouvry. 1994. Design and Implementation of Test Suite Tools. Report to LRE 62-089 D-WP5.1. University of Essex, UK.

Balkan, Lorna, Frederik Fouvry, and Sylvie Regnier-Prost (editors). 1996. TSNLP User Manual. Volume 1: Background, Methodology, Customization, and Testing. Technical report. University of Essex, UK.

Balkan, Lorna, Klaus Netter, Doug Arnold, and Siety Meijer. 1994. Test Suites for Natural Language Processing. In *Proceedings of the Language Engineering Convention.* Paris.

Dauphin, Eva, Veronika Lux, Sylvie Regnier-Prost (principal authors), Doug Arnold, Lorna Balkan, Frederik Fouvry, Judith Klein, Klaus Netter, Stephan Oepen, Dominique Estival, Kirsten Falkedal, and Sabine Lehmann. 1995a. Checking Coverage against Corpora. Report to LRE 62-089 D-WP3.2. University of Essex, UK.

Dauphin, Eva, Veronika Lux, Sylvie Regnier-Prost, Lorna Balkan, Frederik Fouvry, Kirsten Falkedal, Stephan Oepen (principal authors), Doug Arnold, Judith Klein, Klaus Netter, Dominique Estival, Kirsten Falkedal, and Sabine Lehmann. 1995b. Testing and Customisation of Test Items. Report to LRE 62-089 D-WP4. University of Essex, UK.

Estival, Dominique (principal author), Kirsten Falkedal, Lorna Balkan, Siety Meijer, Sylvie Regnier-Prost, Klaus Netter, and Stephan Oepen. 1994. Survey of Existing Test Suites. Report to LRE 62-089 D-WP1. Istituto Dalle Molle per gli Studii Semantici e Cognitivi (ISSCO) Geneva, Switzerland.

Estival, Dominique, Kirsten Falkedal, Sabine Lehmann (principal authors), Lorna Balkan, Siety Meijer, Doug Arnold, Sylvie Regnier-Prost, Eva Dauphin, Klaus Netter, and Stephan Oepen. 1994. Test Suite Design — Annotation Scheme. Report to LRE 62-089 D-WP2.2. University of Essex, UK.

Flickinger, Daniel, John Nerbonne, Ivan A. Sag, and Thomas Wassow. 1987. Toward Evaluation of NLP Systems. Technical report. Hewlett-Packard Laboratories. Distributed at the 24^{th} Annual Meeting of the Association for Computational Linguistics (ACL).

Konrad, Karsten. 1995. Abstrakte Syntaxtransformation mit getypten Merkmalstermen. Technical report. Deutsches Forschungszentrum für Künstliche Intelligenz. forthcoming.

Lehmann, Sabine, Dominique Estival, Kirsten Falkedal, Hervé Compagnion, Lorna Balkan, Frederik Fouvry, Judith Baur, and Judith Klein. 1996. TSNLP User Manual. Volume 3: Test Data Documentation. Technical report. Istituto Dalle Molle per gli Studii Semantici e Cognitivi (ISSCO) Geneva, Switzerland.

Nerbonne, John, Klaus Netter, Kader Diagne, Ludwig Dickmann, and Judith Klein. 1993. A Diagnostic Tool for German Syntax. *Machine Translation* 8:85–107.

Oepen, Stephan, Frederik Fouvry, Klaus Netter, Tom Fettig, and Fred Oberhauser. 1996. TSNLP User Manual. Volume 2: Core Test Suite Technology. Technical report. Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI) Saarbrücken, Germany.

Pollard, Carl, and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar.* Studies in Contemporary Linguistics. Stanford, California: The University of Chicago Press.

Uszkoreit, Hans, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne, Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger, Klaus Netter, Günter Neumann, Stephan Oepen, and Stephen P. Spackman. 1994. DISCO — An HPSG-based NLP System and its Application for Appointment Scheduling. In *Proceedings of the 15th Conference on Computational Linguistics (COLING)*. Kyoto.