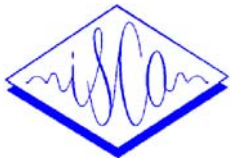

6th European Conference on
Speech Communication and Technology
(EUROSPEECH'99)
Budapest, Hungary, September 5-9, 1999

THE CPK NLP SUITE FOR SPOKEN LANGUAGE UNDERSTANDING

Tom Brøndsted

Center for PersonKommunikation, Fredrik Bajers Vej 7A-5
Institute for Electronic Systems, Aalborg University, DK-9220, Aalborg Ø, Denmark
tb@cpk.auc.dk

ABSTRACT

This paper describes a number of freely available tools for implementing and running spoken language understanding systems. Unlike other free tools (e.g. the CSLU toolkit), the main emphasis is on spoken language understanding (syntactic/semantic parsing, generation of language models for recognition etc.). The suite supports (reads and/or writes) a number of grammar formats defined for speech recognisers like Entropic's GrapHvite (HTK standard lattice) as well as common unification grammar formats used in NLP: The American PATRII and the European EUROTRA (Augmented Phrase Structure Grammar) format. The open architecture is completed by a general API allowing simple interfacing to dialogue management and speech recognition. The suite, implemented in C++ (partly C, Lex, Yacc), can compile and run on any machine under any OS having a 32 bit C++ compiler, Flex/Bison or Lex/Yacc. The suite can be downloaded at http://www.cpk.auc.dk/~tb/nlpsuite.

10.21437/Eurospeech.1999-668

1. INTRODUCTION

The suite of tools described in this paper focuses on the problem of developing, testing, and running the NLP modules of spoken dialogue systems. The suite is rooted in a number of EU-projects (REWARD, LTR "Language and Image Data Fusion") as well as in Danish National Projects (Chameleon, "Spoken Language Dialogue Systems") [1][3][4][5][7]. The main concepts are: (1) Support for generally used NLP and recognition grammar formats rather than inventing new formats. (2) An open architecture allowing designers to interface to their own speech recognisers and their own dialogue managers. (3) Addressing NLP-needs in both simple system-directed dialogue systems and more sophisticated needs in user-directed systems. The paper gives an overview of the suite (section 2) together with some concrete examples illustrating some of the ideas behind the suite (section 3).

2. OVERVIEW

Currently, the suite encompasses 25 modules (fig. 1). Each module has a name consisting of a three-letter code denoting the grammar format to be read and a four-letter code denoting what is done to or with the grammar. The five grammar formats are described in section 2.1 (A-E) and the five program types in section 2.2 (A-E).

	APS	**PTR**	**PSG**	**ICM**	**VOC**
PARS	apspars	ptrpars	psgpars	icmpars	vocpars
CONV	apsconv	ptrconv	psgconv	icmconv	vocconv
TREC	apstrec	ptrtrec	psgtrec	icmtrec	voctrec
TSLU	apstslu	ptrtslu	psgtslu	icmtslu	voctslu
SGEN	apssgen	ptrsgen	psgsgen	icmsgen	vocsgen

Figure 1: Programs included in the suite

The internal grammar representation is the same for all 25 programs. This implies an (in terms of "source code") extensive overlap between the programs. E.g., the only difference between **apspars** and **ptrpars** is the (lex/yacc) code reading the grammar file into the internal representation (fig. 2). For technical reasons they are separated in different programs.

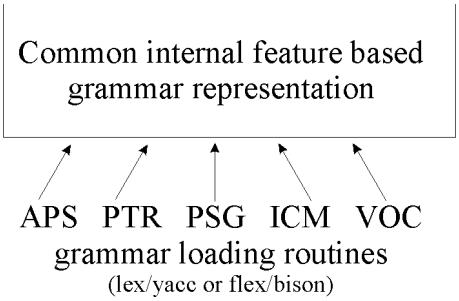

Figure 2: Internal grammar representation

Further grammar formats, e.g. the HTK standard lattice format [8], can be *written* by the converters (cf. 2.2 B below). However, to add the HTK format to the grammars being *read*, one only has to implement a corresponding (lex/yacc) reading routine. The result would be a column of five new modules: **htkpars** (checking if input sentences are accepted by the HTK network), **htkconv** (converting the HTK network into e.g. a Vocalis recognition grammar or into a word pair grammar in the same HTK lattice format) etc.

Grammars may be organised in named sub-task specific sub-grammars that can be activated and deactivated by a dialogue manager. This presupposes a speech recogniser that supports multiple grammar networks.

2.1 Grammar formats

The grammar formats being read by the suite can be grouped into two categories: (1) Formats intended for assigning semantics to sentences (APS, PTR) and (2) formats intended only for recognition (PSG, VOC). ICM (see below) belongs to both categories. When parsing with a grammar of the first category, the output is a (complex) semantic frame structure. Otherwise, the output is a structure equivalent to a simple Boolean value indicating if the sentence was accepted by the grammar or not. The suite includes grammar samples where the format is indicated by a three-letter code extension (**number.aps, dates.aps**, etc.).

A. **APS** (Augmented Phrase Structure Grammar): A compound feature based unification grammar based on a subset of the EUROTRA User Language developed originally for machine translation, however sufficiently general to meet other NLP demands as well [2]. The APS format was used in the Danish national research programme "Spoken Language Dialogue Systems" [1] but

has meanwhile been subject to several improvements [5]. Semantic frames can be generated in two ways: (1) With semantic mapping rules consisting of a condition in terms of a sub-tree to look for, and an action to carry out if the condition is met. (2) Extraction of frames from the top-node of the parse tree built during syntactic parsing. This presupposes extensive percolation of features up through the tree. Further details appear from the examples in section 3.

B. **PTR**: A unification formalism adapted from the PC_PATR parser by S. McConnel (distributed through the Summer Institute of Linguistics, Arlington in Texas, at http://www.sil.org/pcpatr) and based on PATR-II by Shieber [10]. Unlike the APS format, PTR supports complex (nested) features but does not include mapping rules. Hence, the extraction of semantic frames is always based on the feature structures created at the top node of the parse tree.

C. **PSG** (Phrase Structure Grammar): A trivial symbol-based (non-unification based) context free grammar or "BNF" format that primarily serves teaching purposes.

D. **ICM**: "Interpretation and Control Module" grammar format. A symbol-based recursive transition network format developed at CPK and used in the EU-funded Esprit project SUNSTAR [1]. Semantic actions can be associated with lexical rules. When used for speech recognition, semantic actions are ignored (and may be omitted).

E. **VOC**: A recognition grammar network defined for the speech recogniser developed by the CPK project partner Vocalis Inc. (EU-project REWARD). Formally, VOC can be compared to HTK (cf. 2.2.Ba). The network may include non-terminals (corresponding to "Sub-lattices" in HTK) and may (in contrast to HTK) involve recursions.

2.2 Program Types

The five program types included in the suite (cf. Fig 1) are:

A. **PARS**: A unification-based left-corner chart parser which processes input bottom up with top-down filtering (left corner dependencies) and left-to-right. The syntactic parsing strategy largely corresponds to the enhancement of Earley's algorithm [6] presented in [9] or to the "Earley algorithm" described in [7]. The result of syntactic parsing is one or (in case of ambiguity) more parse trees. If the grammar involves mapping rules, mapping is carried out by scanning the tree bottom-up and creating the defined semantic predicate-argument structures at each node meeting the condition of the rule. The semantics of a node may be inserted as an argument in other nodes through this process.

The NLP Application Programming Interface (NLPAPI) provides the interface to PARS and is available as a C library. The core functions with the self-explanatory names are:

```
int LoadGrammarFile(char *filename);
int ActivateSubGrammar(char *subgrammarid);
int DeactivateSubGrammar(char *subgrammarid);
int ActivateAllSubGrammars();
int DeactivateAllSubGrammars();
APISemFrameList *Parse_1Best(char *Sentence);
```

The C structure APISemFrameList returned by the parsing function contains a list of semantic frame structures (0 if parsing failed, 1 in case of unambiguous input, otherwise 2 or more). If parsing partly failed, the list may consist of fragmentary frame structures. A simple dialogue manager may reject any fragmentary and ambiguous structures. The uppermost node of each frame structure is always the name of the sub-grammar accepting input, e.g. *date*(day(10),month(10),year(1999)). In system-directed dialogues, sub-grammar names may be used for branching the dialogue flow.

B. **CONV**: Grammar converter that derives syntactic speech recognition networks from the common *internal* feature based grammar representation (Fig. 2). The algorithm is entirely based on the fact that the two feature based grammar formalisms supported by the suite, APS and PTR, are unification grammars where features take values drawn from a *finite* set. Hence, a feature set can always be rewritten as a finite set of symbols. The converter changes the internal representation incrementally into four different states each of which can be written in a number of external file formats defined for specific speech recognisers:

- RTN: Recursive Transition Network grammar network. This corresponds to a HTK "main" sub lattice with included sub lattices. However HTK does not accept RTNs involving real recursions. An internal RTN representation is always fully equivalent to the external APS/PTR/PSG/ICM/VOC grammar definition.
- FSN: Finite State Network grammar, in speech technology often denoted a "fullgram". Recursions of the RTN are rewritten as iterations. Unless the RTN involves certain (rare) recursive problems, in formal language theory known as the A^nB^n problem, the FSN is fully equivalent to the RTN. Otherwise FSN accepts a larger set of sentences. The algorithm changing the RTN state into FSN can be compared to the HTK tool Hsbuild [11].
- WPG: Word Pair Grammar, a state which is more compact than FSN, but the grammar perplexity is higher. If FSN accepts sentences longer than two words, WPG accepts a larger set of sentences.
- NOG: A "nogram" network, more compact than WPG, but the grammar perplexity is higher. If WPG accepts sentences longer than one word, NOG accepts a larger set of sentences.

Each state of the internal grammar representation (RTN/FSN/WPG/NOG) can be written in three formats:

a. HTK: The HTK standard lattice format [8][11].

b. ICM (see section 2.1 D)

c. VOC (see section 2.1 E)

Files written by the converter are named using the input file name, an extension denoting the network type and an extension denoting the format, e.g. **number.aps.fsn.htk** (a fullgram HTK lattice derived from the APS grammar **number.aps)**. Dictionaries have the extension dic (**number.aps.dic.htk**). The function writing the network (RTN/FSN/WPG/NOG) in a specific format consists of only ca. 80 lines of c code. *Hence, it should be relatively simple to extend the suite to support other speech recognition systems* (e.g. ViaVoice by IBM etc.).

C. **TREC**: A grammar constrained typed-text recogniser based on a standard linear Viterbi pattern-matching algorithm. Input is typed text and output sentences "similar" to input and acceptable to the grammar. TREC uses (by default) an FSN derivation of the internal grammar representation (see above). It supports "garbage models" (in the suite defined as the string "*") as it always produces a medium distance between garbage words and input. By default, it returns linguistic symbols also for recognised garbage and silence. The main purpose of the text recogniser is to simulate speech recognition during the design phase and to assists in wizard experiments with incomplete dialogues. An API similar to the NLPAPI of PARS is available.

D. **TSLU**: "Textual" spoken language understanding: TREC and PARS combined.

E. **SGEN**: A random sentence generator used for exploring the coverage of grammars. The program corresponds to the HTK tool HSgen [11] and uses an RTN derivation of the internal grammar representation (see above).

3. EXAMPLES

The examples in this section are inspired by the voice driven email application in the GrapHvite tutorial [8]. We show how a similar network can be implemented in the APS format in a way that generates usable semantic frames to be passed on to the application manager. We avoid linguistic jargon, "np"s, "vp"s, that in speech technology has given NLP the reputation of being "not sexy".

3.1 Locating Email Messages

Figure 3 from the GrapHvite tutorial shows a sub-network "msg_spec" generating phrases like "message number three", "previous message", "next" etc. As the phrases are objects to operations like "delete", "forward" etc., the semantic role of "msg_spec" obviously is to generate an absolute number (e.g. "message number three") or a position number relative to the current one (e.g. "previous message"). GrapHvite presupposes that this is done via a parsing mechanism separated from the recognition network designed with the Netbuilder tool.

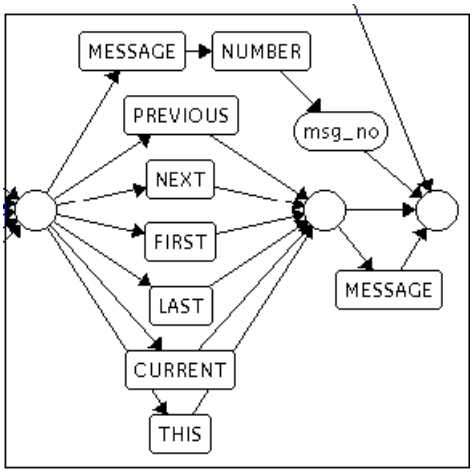

Figure 3: Sub-network "msg_spec" in [8]

The corresponding APS lexicon may look:

```
{lex=previous,cat=spec, relpos=-1}.
{lex=current, cat=spec, relpos=0}.
{lex=this, cat=spec, relpos=0}.
{lex=next, cat=spec, relpos=1}.
{lex=first, cat=spec, abspos=0}.
{lex=last, cat=spec, abspos=10000}.
{lex=message,cat=msg}.
{lex=email,cat=msg}.
<number>.
```

where <number> is an "ad-hoc feature set" internally expanded into {lex=number, cat=number}. The actual network in Fig. 3 can be implemented in a single structure building rule using various operators of the APS format such as '^' (optional), ',' (and), ';' (or) and brackets '(', ')' to group feature sets object to the operators. Variables, i.e. pointers to shared structures ('$' + identifier), are used to percolate values bottom up during unification:

```
{cat=msg_spec, relpos=$A,abspos=$B}
[
    (  {cat=spec, relpos=$A, abspos=$B},
       ^{cat=msg}  /* optional */
    );     /* or */
    (  {cat=msg},
       <number>,
       {cat=msg_no, val=$B} /* "nonterminal" */
    )
].
```

The sub-network "msg_no" is not shown here. When "msg_spec" is used as main-network, `apspars` generates frame structures like:

```
previous message :          $X(relpos(-1))
message number three:       $X(abspos(3))
first email:                $X(abspos(0))
this:                       $X(relpos(0))
```

where $X is an uninstantiated variable indicating that the sentences were accepted by a global grammar (otherwise $X would contain a sub-grammar name, cf. section 2.2). Further, `apsconv` can be used for deriving an FSN in the HTK standard lattice format.

3.2 Email Address Book

One of the powerful aspects of unification grammars traditionally emphasised in the literature is their ability to express the grammatical relation "agreement" through variables. Again, avoiding linguistic jargon, we may explore this ability in an application where we want "first names" to agree with "last names" so that they only express existing persons according to some database, e.g. an email address book:

```
Tom Brøndsted tb@cpk.auc.dk
Paul Dalsgaard pd@cpk.auc.dk
Paul McKevitt pmck@cpk.auc.dk
```

The corresponding grammar sample in the APS format is:

```
{lex=Tom, cat=firstname, email="tb@cpk.auc.dk"}.
{lex=Brøndsted, cat=lastname, email="tb@cpk.auc.dk"}.
{lex=Paul, cat=firstname, email="pd@cpk.auc.dk"}.
{lex=Dalsgaard, cat=lastname, email="pd@cpk.auc.dk"}.
{lex=Paul, cat=firstname, email="pmck@cpk.auc.dk"}.
{lex=McKevitt, cat=lastname, email="pmck@cpk.auc.dk"}.
```

```
{cat=person,emailadr=$A}
    [
        {cat=firstname,email=$A},
        ^{cat=lastname,email=$A}
    ].
```

Examples:

Tom Brøndsted:	$X(emailadr(tb@cpk.auc.dk))
Paul:	$X(emailadr(pd@cpk.auc.dk))
	$X(emailadr(pmck@cpk.auc.dk))
Paul Dalsgaard:	$X(emailadr(pd@cpk.auc.dk))
Paul Brøndsted:	<null>

3.3 The Meaning of Garbage and Silence

We may explicitly add two garbage words to the APS in 3.2 enabling the grammar to reject unknown first names and last names without rejecting the entire utterance:

```
{lex="*", cat=firstname, email=UNKNOWN }.
{lex="*", cat=lastname, email=UNKNOWN }.
```

For PARS, "*" is just an ordinary word. TREC however will always produce a medium distance score when matching "*" with input. CONV sets up the garbage models as defined for the recognition grammar formats. In general, it is useful to define garbage rules for open word classes. Also problems with "meaningful" short speech pauses (e.g. "twenty seven"= 27, as opposed to "twenty silence seven" = 20, 7), can be solved if such "words" are explicitly defined in the APS. Finally, we may use phonemic representations of words in stead of orthographic ones. This solves problems arising from ambiguous pronunciations. For instance, the Danish first name "Paul" has two alternative spelling forms, "Povl" and "Poul" and (at least) two pronunciation variations: [p ao l sp] and [p oh l sp]. All problems arising from this circumstance can be solved by implementing the lexical rules as

```
{phon="p ao l sp|p oh l sp", cat=firstname,email=pd}.
{phon="p ao l sp|p oh l sp", cat=firstname,email=pmck}.
```

and subsequently instruct the suite to separate phon-values at the character '|' and use them for lexical lookup. By default, the suite uses the lex-values and separates input at the space character.

3.4 Mapping rules and Post processing of Frames

The examples in section 3.1-3.3 use simple percolation of values to extract semantic frames. An alternative and more powerful way is the use of mapping rules (section 2.1. A and 2.2 A). Further, post processing functions can be applied through the API. A post processing function takes a semantic frame as argument, with access to all daughters of this frame, and modifies it. Typically, post processing functions may involve arithmetical operations for calculation of numerals, dates, and so on, e.g.,

```
year(hundreds(19),tens(9),ones(9)) >
year(1900,90,9) >
year(1999)
```

The suite comes with functions to handle numbers based on the decimal system.

4. CONCLUSION

The NLP suite represents work in progress and is made available for non-commercial use, research and discussion, "as it is". There are some "known bugs" and many "in-conveniences". We welcome users to contribute with code that allows further grammar formats to be read (section 2) or written (section 2.2). However, supporting new grammar formats may imply that the common internal grammar representation must be modified. This is only possible in a limited scale because it interferes with every single module in the suite.

5. REFERENCES

[1] A. Baekgaard, et al. The Danish spoken language dialogue project - a general overview. *ESCA WS on Spoken Dialogue Systems: Theory and applications, Vigsø, Denmark 1995, pp. 89-92.*

[2] A. Bech: Description of the Eurotra Framework. *C. Copeland, J. Durand, S. Krauwer, B. Maegaard (eds). The Eurotra Formal Specifications. Studies in Machine Translation and Natural Language Processing, Volume 2, 1991*.

[3] T. Brøndsted, B. Bai, J.Ø. Olsen: "The REWARD Service Creation Environment. An Overview". *ICSLP. Sydney 1998, pp. 1175-78*.

[4] T. Brøndsted, P. Dalsgaard, L.B. Larsen, M. Manthey, P. Mc Kevitt, T. Moeslund, K.G. Olesen: "A Platform for developing Intelligent MultiMedia Applications". *Technical Report R-98-1004. Institute of Electronic Systems, Aalborg University 1998.*

[5] T. Brøndsted: "The Natural Language Parsing Modules in REWARD and IntelliMedia 2000+". *S. Kirchmeier-Andersen, H.E. Thomsen (eds.): LAMBDA 25, Copenhagen Business School, Dep. of Computational Linguistics, 1999.*

[6] J. Earley: An efficient context-free parsing algorithm. *Communications of the ACM 1970, Vol. 13*

[7] B. Music, C. Povlsen: The NLP Module of a spoken Language Dialogue System for Danish Flight Reservations. *3rd European Conference on Speech Communication and Technology (EUROSPEECH), Berlin 1993, pp. 1859-62.*

[8] K. Power, C. Matheson, D. Ollason, R. Morton: The grapHvite Book. For grapHvite v. 1.1. Entropic, Cambridge Research Laboratory, Cambridge 1997

[9] S.M. Shieber: Using Restriction to Extend Parsing Algorithms for Complex-feature-based Formalims". *Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics, Chicago 1985, pp.145-152.*

[10] S.M. Shieber: An Introduction to Unification based Approaches to Grammar. *CSLI Lecture Notes Series, Number 4, Stanford CA 1986.*

[11] S.Young, J. Odell, D. Ollason, P. Woodland: The hTk Book. Version 2.2. Entropic Ltd. 1999.