Adapting Stochastic Output for Rule-Based Semantics

Wissenschaftliche Arbeit
zur Erlangung des Grades eines Diplom-Handelslehrers
im Fachbereich Wirtschaftswissenschaften
der Universität Konstanz

Februar 2009

Verfasst von:
Annette Hautli
Im Baumgarten 1
78465 Konstanz
01/549505

Bearbeitungszeit: 6. Dezember 2008 - 13. Februar 2009

1. Gutachter: Prof. Dr. Miriam Butt, FB Sprachwissenschaft
2. Gutachter: Prof. Dr. Maribel Romero, FB Sprachwissenschaft

Konstanz, den 13. Februar 2009

Contents

List of Figures

Acknowledgements

First of all I want to thank Tracy Holloway King, Powerset Inc. (formerly at Palo Alto Research Center) for her extremely valuable help and the time she spent answering my questions and suggesting new ways to pursue.
Thanks also go to the whole NLTT group at Palo Alto Research Center for a truly inspiring and motivating atmosphere during my time there.
I express my deep gratitude to Miriam Butt, my adviser in Konstanz, who made this cooperation possible, supported me whenever she could and partly released me from my duties in Konstanz.
A big thank you also goes to Josef van Genabith from Dublin City University who agreed in cooperating with PARC and offered me to spend some time at DCU to intensify the work on the experiment.
Thanks also go to Jennifer Foster from DCU, who provided the initial data and offered help whenever she could.

Without my friends, I wouldn't have had the fun I enjoyed over the last couple of years. It's great to know that I can count on every one of you. Many thanks to those of you who proof-read this thesis or contributed in any other way.

Very importantly, I want to thank my family, especially my parents, who always supported me and believed in me. Without your effort I wouldn't be where I am now.

Abstract

The current tendency in Natural Language Processing is to use statistical methods in order to build NLP applications. In this context I explore whether a stochastic LFG-like grammar for English can be used as the input to a rule-based semantic system, in the place of the original rule-based English LFG grammar. Integrating the stochastic grammar requires creating a set of ordered rewrite rules to augment and reconfigure the output of the stochastic grammar. The results are promising in that the missing features can be reconstructed to provide sufficiently rich input to the semantic component. As a result, the advantages of both sides are combined. On the one hand, one can make use of the significant time-saving effects of a stochastic grammar; on the other hand, the combined approach does not lack any of the information compared to the rule-based system.

Chapter 1

Introduction

In this thesis I report on an experiment to explore whether a stochastic LFG-like grammar for English (Cahill et al. (2008)) could be used as the input to a rule-based semantic system (Crouch and King (2006), Bobrow et al. (2007)) in the place of the original rule-based English LFG grammar, which is being developed at Palo Alto Research Center. This experiment follows the current tendency in Natural Language Processing to intensify the usage of statistics in NLP applications.

Integrating the new grammar involves hybridizing the original rule-based English grammar of Palo Alto Research Center in a way that the strictly rule-based system is mixed with a stochastic component (Hautli (2008)). The core of the experiment and this thesis is a set of ordered rewrite rules that augment and reconfigure the output of the stochastic grammar in order to add more information to the stochastic output. The results are promising in that the missing information can be reconstructed to provide sufficiently rich input to the semantic representation.

The reasons for using such a grammar are two-fold. In the case of English, the language used in the experiment, the stochastic grammar can be used in the place of the rule-based grammar for out-of-coverage sentences (e.g. fragmented sentences), thereby supplying more connected input to the semantics. In the case of other languages, if no rule-based grammar is available, but a treebank of the target language is, it can be faster to create a stochastic grammar instead of a rule-based one, thereby reducing the necessary time to create a system for the new language (Cahill et al. (2005)).

In chapter 2, I introduce the framework that is involved in this project, the syntax theory Lexical-Functional Grammar (2.1.), and also present the tools which are used in this experiment, namely XLE (2.2.), developed by PARC, and the f-structure annotation algorithm using treebanks, provided by Dublin City University (2.3.). The way these tools interact is explained in section 2.4.

In chapter 3, I present the overall layout of the experiment and explain each step, starting with the stochastic output of DCU and ending with its usage as input to the rule-based semantics. Core of this chapter is the set of ordered rewrite rules I wrote for the transfer from DCU to PARC. I also concentrate on some of the problems that arose, namely with interrogative and imperative clauses.

Chapter 4 deals with the evaluation of the transfer results, as to how high the matching figures (precision, recall and f-score) are between the transferred DCU output and the original PARC output. I also take the experiment a step further and compare the semantic output if the rule-based input and the transferred stochastic input is used.

Chapter 5 discusses the results of the experiment and answers the question how a truly integrated system would have to be built up in order to benefit from stochastic input. I also focus on some important aspects like ambiguity management and efficiency.

The conclusion in chapter 6 summarizes the experiment and also gives an outlook as to how the project could be extended.

Chapter 2

Framework and Tools

2.1 Lexical-Functional Grammar

Lexical-Functional Grammar (LFG) (Bresnan and Kaplan (1982), Dalrymple (2001)) is an early member of the family of constraint-based grammar formalisms. Others are Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag (1994)) and Generalized Phrase Structure Grammar (GPSG). LFG enjoys continued popularity in theoretical and computational linguistics and in natural language processing applications and research.

At its most basic, LFG assigns two levels of syntactic description to every sentence of a language. Phrase structure configurations are represented in a *constituent structure*. A constituent structure (or 'c-structure') is a conventional phrase structure tree, a well-formed labeled bracketing that indicates the surface arrangement of words and phrases in the sentence. Grammatical functions are represented explicitly at the other level of description, called *functional structure*. The functional structure (or 'f-structure') provides a precise characterization of traditional syntactic notions such as subject, object, complement and adjunct. It is the basis for the semantic component, which is a flat representation of the sentence's predicate argument structure and the semantic contexts in which those predications hold (Crouch and

King (2006)). The semantic representation will be discussed in more detail in 2.2.3.

C-structure The c-structure example in Figure 2.1 is the product of a context-free grammar, which means that the formalism doesn't look to the left or the right context of a constituent in order to determine what category it belongs to, but works on the basis of rules which determine what nodes can make up a constituent. In the case of Figure 2.1 the following rules apply:

$$\begin{array}{rcll} \text{S} & \rightarrow & \text{NP} & \text{VP.} \\ \text{NP} & \rightarrow & \text{D} & \text{N.} \\ \text{VP} & \rightarrow & \text{V} & \text{(PP).} \\ \text{PP} & \rightarrow & \text{P} & \text{NP.} \end{array}$$

This is a very simple rule example, but it suffices as the basis for the c-structure for *Mary hops in the hay.*

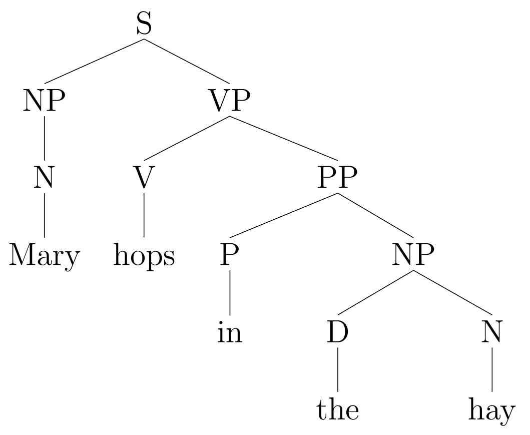


Figure 2.1: C-structure for *Mary hops in the hay*

F-structure The f-structure reflects the collection of constraints imposed on the context-free skeleton (Butt et al. (1999)) and thus contains attributes, such as PRED, SUBJ, and OBJ, whose values can be other f-structures, as in Figure 2.2. In contrast to other syntactic theories, e.g. Minimalism (Chomsky (1995)), LFG encodes predicate-argument structure in the f-structure

$$\begin{bmatrix} \textsc{pred} & \text{`hop}\left\langle\textsc{subj}\right\rangle\text{'} \\ \textsc{tense} & \text{pres} \\ \textsc{subj} & \begin{bmatrix}\textsc{pred} & \text{`Mary'}\end{bmatrix} \\ \textsc{adjunct} & \begin{bmatrix} \textsc{pred} & \text{`in}\left\langle\textsc{obj}\right\rangle\text{'} \\ \textsc{obj} & \begin{bmatrix} \textsc{pred} & \text{`hay'} \\ \textsc{def} & + \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 2.2: F-structure for *Mary hops in the hay*

and not in a Deep-Structure (D-Structure), which is the basis for all movement in the tree.
By formally distinguishing these two levels of representation, the theory separates those grammatical phenomena that are purely syntactic (involving only c-structures and f-structures) from those that are purely lexical (involving lexical entries before they are inserted into c-structures and f-structures).

But where do the lexical items itself come from and how does a c-structure relate to an f-structure? Due to pursuing the goal of psycholinguistic research, the aim of LFG is to give an account of the mental operations that underlie linguistic abilities. In the course it is assumed that lexical items are stored away in a mental lexicon in addition to information about the lexical entry, e.g. word class, etc. A lexical entry according to LFG looks like the following:

$$\begin{array}{lllc} \text{boys} & \text{N} & (\uparrow \text{PRED}) = & \text{`boy'} \\ & & (\uparrow \text{NUM}) = & \text{pl} \\ & & (\uparrow \text{PERS}) = & 3. \end{array}$$

Figure 2.3: Lexical entry for *boys*

The lexeme is on the left hand side of the entry (boys), followed by the word class it belongs to (N). After that, the features of the lexeme are listed. In this case, *boy* is the underlying form and has the features that it is third person and plural. The arrows are a core component of LFG, they are needed to create a c-structure where the information of nodes is transported upwards in the tree to guarantee correct unification. “The intuition behind this notation comes from the way trees are usually represented: the up arrow $\uparrow$ points to the mother node, while $\downarrow$ points to the node itself” (Dalrymple (2001)). Sometimes, the f-structure annotations are written above the node labels of a constituent structure, making the intuition behind the $\uparrow$ and $\downarrow$ annotation clearer. An example can be seen in Figure 2.4:

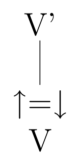
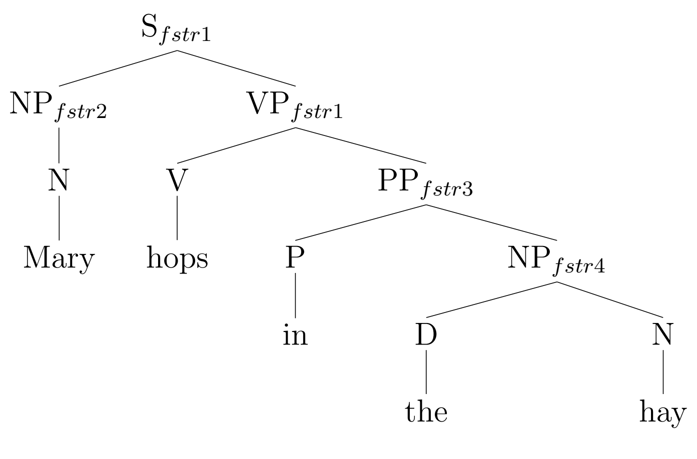


Figure 2.4: C-structure annotated with functional equations

The relationship between c- and f-structure is given by a *functional projection function* from c-structure nodes to f-structure attribute-value matrices (Dalrymple (2001)). Figure 2.5 shows the functional projection from c-structure to f-structure by adding variables to each node and corresponding f-structure.



$$
{}_{\textit{fstr1}}\begin{bmatrix}
\textsc{pred} & \text{‘hop}\left\langle\textsc{subj}\right\rangle\text{’} \\
\textsc{tense} & \text{pres} \\
\textsc{subj} & {}_{\textit{fstr2}}\begin{bmatrix}\textsc{pred} & \text{‘Mary’}\end{bmatrix} \\
\textsc{adjunct} & {}_{\textit{fstr3}}\begin{bmatrix}
\textsc{pred} & \text{‘in}\left\langle\textsc{obj}\right\rangle\text{’} \\
\textsc{obj} & {}_{\textit{fstr4}}\begin{bmatrix}\textsc{pred} & \text{‘hay’} \\ \textsc{def} & + \end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

Figure 2.5: C- and f-structure relation

The next question is: How can it be guaranteed that an f-structure is coherent and complete? There are three well-formedness conditions on the f-structure: functional uniqueness, completeness, and coherence (see Bresnan and Kaplan (1982) for the original definitions) that rule out “false” f-structures.

Functional uniqueness guarantees that an attribute does not have more than one value. This, for example, rules out an f-structure in which the DEF attribute does have the values + and - at the same time (value for the definiteness of the noun is plus and minus). An example for such an f-structure is given below:

$$
\begin{bmatrix}
\textsc{pred} & \text{’boy’} \\
\textsc{num} & \text{sg} \\
\textsc{pers} & 3 \\
\textsc{det} & \begin{bmatrix}\textsc{def} & +/- \end{bmatrix}
\end{bmatrix}
$$

Figure 2.6: Example for violation of the *Uniqueness* condition

The second condition is called the *Completeness* condition. It states that all grammatical functions for which the sentence predicate subcategorizes for must be assigned values. This rules out clauses such as **John likes*, which lack the argument that is liked by John, namely the object of the sentence. The f-structure for such an incomplete sentence is shown in Figure 2.7.

$$\begin{bmatrix} \textsc{pred} & \text{`like}\left\langle \textsc{subj},\ \textsc{obj} \right\rangle\text{'} \\ \textsc{tense} & \text{pres} \\ \textsc{subj} & \begin{bmatrix} \textsc{pred} & \text{`Mary'} \end{bmatrix} \end{bmatrix}$$

Figure 2.7: Example for violation of the *Completeness* condition

Coherence requires all arguments in the argument structure of the sentence predicate to be a grammatical function in the f-structure. This results in clauses like **Mary appears the cat* to be ill-formed. *Appear* only needs a subject, therefore adding an object to the f-structure makes the sentence ungrammatical. This can be seen in the f-structure in Figure 2.8.

$$\begin{bmatrix} \textsc{pred} & \text{`appear}\left\langle \textsc{subj} \right\rangle\text{'} \\ \textsc{tense} & \text{pres} \\ \textsc{subj} & \begin{bmatrix} \textsc{pred} & \text{`Mary'} \end{bmatrix} \\ \textsc{obj} & \begin{bmatrix} \textsc{pred} & \text{'cat'} \\ \textsc{det} & + \end{bmatrix} \end{bmatrix}$$

Figure 2.8: Example for violation of the *Coherence* condition

2.2 XLE

One platform that has been used in grammar development efforts within Lexical Functional Grammar is XLE. It consists of cutting-edge algorithms for parsing and generating Lexical-Functional Grammars along with a user interface for writing and debugging such grammars (Crouch et al. (2008)). XLE is written in C, uses Tcl/Tk for the user interface; the transfer component uses prolog and is being ported to C++. Both currently run on Solaris Unix, Linux and Mac OS X.

XLE has been developed and maintained by Palo Alto Research Center in California and provides the basis for the Parallel Grammar Project (Par-

Gram) (Butt et al. (1999, 2002)) which develops industrial-strength grammars for different languages, among them English, French, German, Norwegian, Japanese and Urdu. Recent efforts to present the achievements of XLE to a wider public have resulted in the start-up company Powerset, part of Microsoft Inc., which licensed PARC technology. Powerset's first product is a search engine for Wikipedia which returns precise results on questions and queries, often answering questions directly. Basis to all this is XLE.

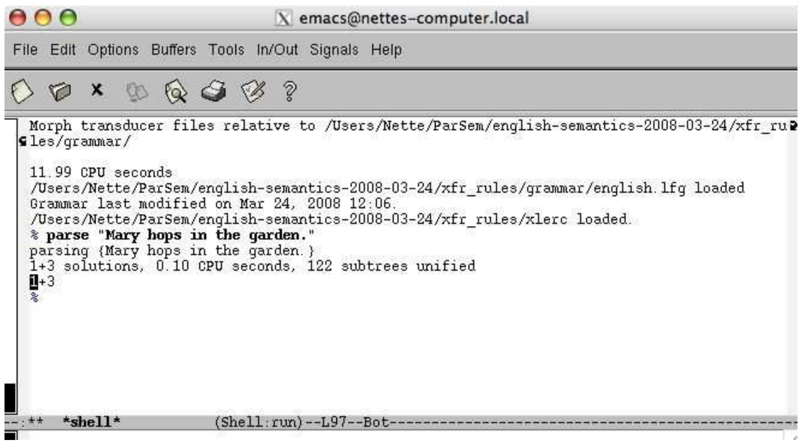
There are three key ideas that XLE uses to make its parser efficient:

- "The first idea is to pay careful attention to the interface between the phrasal and functional constraints. In particular, XLE processes all of the phrasal constraints first using a chart, and then using the results to decide which functional constraints to process.
- The second key idea is to use contexted unification to merge multiple feature structures together into a single, packed feature structure.
- The third key idea is to use lazy contexted copying during unification. Lazy contexted unification only copies up as much of the two daughter feature structures of a subtree as is needed to determine whether the feature structures are unifiable (Crouch et al. (2008))"

2.2.1 The User Interface

The XLE platform currently runs on Solaris Unix, Linux and Mac OS X and makes use of freely accessible software such as *emacs* (text editor) and TCL. The user can interface with XLE by means of an emacs lfg-mode designed by Mary Dalrymple. This mode gives the user an easy mechanism of invoking XLE and provides automatic formatting for rules, templates and lexical entries (Butt et al. (1999)).

An example of how XLE starts is shown in Figure 2.9. A configuration file in the top directory of the grammar automatically uploads the grammar and all its components when typing `xle` in the command line of the shell. At first, the semantics of the English grammar are loaded. After that, XLE


emacs@nettes-computer.local
File Edit Options Buffers Tools In/Out Signals Help

```
Morph transducer files relative to /Users/Nette/ParSem/english-semantics-2008-03-24/xfr_ru
les/grammar/

11.99 CPU seconds
/Users/Nette/ParSem/english-semantics-2008-03-24/xfr_rules/grammar/english.lfg loaded
Grammar last modified on Mar 24, 2008 12:06.
/Users/Nette/ParSem/english-semantics-2008-03-24/xfr_rules/xlerc loaded.
% parse "Mary hops in the garden."
parsing {Mary hops in the garden.}
1+3 solutions, 0.10 CPU seconds, 122 subtrees unified
1+3
%
```

```
--:**  *shell*          (Shell:run)--L97--Bot-----------------------------------------------
```

Figure 2.9: XLE User Interface

reports how many rules, states, arcs and disjuncts the grammar has and then loads the morphology and the tokenizer in the next step. Finally, the system loads the syntax rules and reports whether the system is ready to parse a sentence.
If the syntax of a sentence needs to be analyzed, the command `parse ''Mary hops in the garden.''` is typed in the XLE window (as shown above). XLE returns that it is now parsing and then returns the following information about the parse:

- “1+3” means that there was one optimal solution and three unoptimal solutions. The unoptimal solutions are filtered out by the optimality operator.

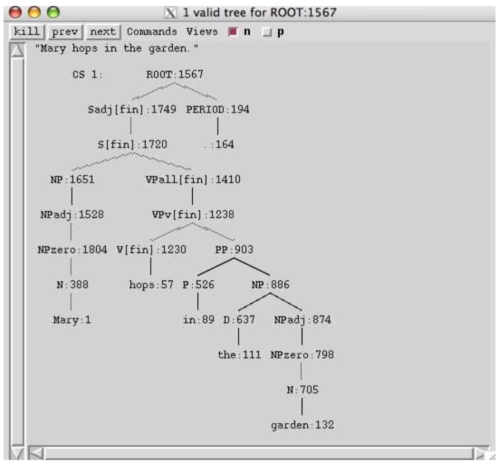
- “0.10 CPU seconds” indicates how many CPU seconds it took to parse the sentence.

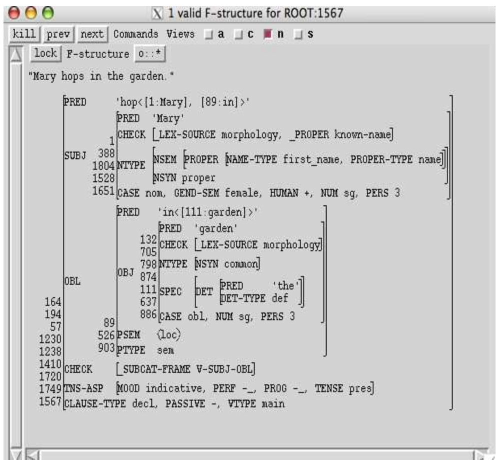
- “122 subtrees unified” shows the number of subtrees that were explored. This number gives the grammar writer an indication of the complexity of the system.

2.2.2 The XLE Output

Once a sentence is parsed, XLE returns the syntactic analyses in four windows. We get one window for the c-structure (tree structure) and another one for the f-structure of the parsed sentence (Figure 2.10). The other two show two different packed representations of the valid solution (Figure 2.11) (Butt et al. (1999)).

It is very useful for the grammar writer to be able to choose between different analyses for a sentence in order to decide which one is the most optimal solution. The c- and f-structures change according to the solution which is chosen out of the set of packed representations. In the example given here, there is only one grammatical solution for the sentence, which is why the fourth window in 2.11 stays empty.





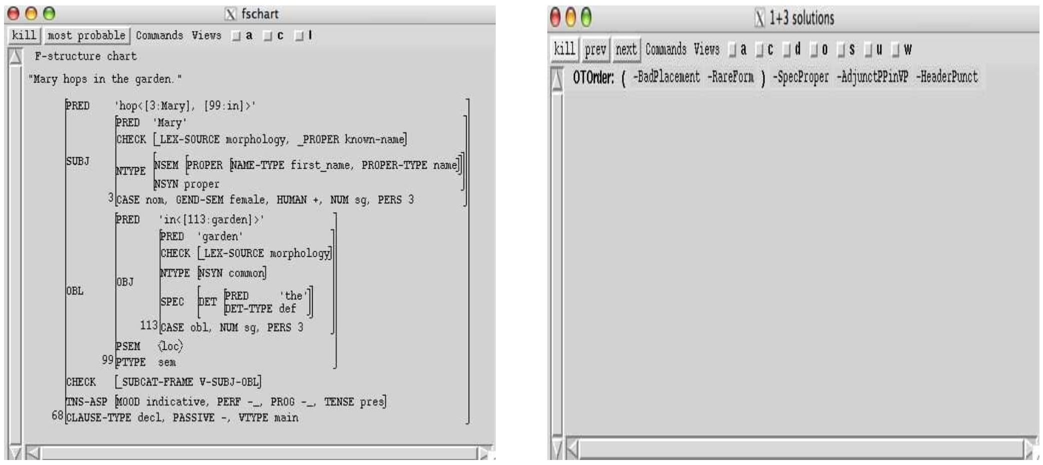
```
"Mary hops in the garden."

       PRED       'hop<[1:Mary], [89:in]>'
                  PRED   'Mary'
                1 CHECK  [LEX-SOURCE morphology, _PROPER known-name]
       SUBJ   388        NSEM [PROPER [NAME-TYPE first_name, PROPER-TYPE name]]
             1804 NTYPE
             1528        NSYN proper
             1651 CASE nom, GEND-SEM female, HUMAN +, NUM sg, PERS 3
                  PRED   'in<[111:garden]>'
                         PRED   'garden'
                     132 CHECK  [LEX-SOURCE morphology]
                     705
                     798 NTYPE  [NSYN common]
                  OBJ 874
       OBL           111 SPEC   [DET [PRED      'the']]
   164               637                [DET-TYPE def  ]
   194               886 CASE obl, NUM sg, PERS 3
    57             89
  1230            526 PSEM   (loc)
  1238            903 PTYPE  sem
  1410 CHECK      [SUBCAT-FRAME V-SUBJ-OBL]
  1720
  1749 TNS-ASP    [MOOD indicative, PERF -_, PROG -_, TENSE pres]
  1567 CLAUSE-TYPE decl, PASSIVE -, VTYPE main
```

Figure 2.10: XLE output: c-and f-structure

The `prev` and `next` buttons allow the user to navigate between the different representations, regardless of the parses being valid or invalid. To get morphological information the user has to right-klick on a terminal node and then go to `Show Morphemes`. The tags displayed there are generated in the finite-state morphology and are fed into the system via sublexical rules. This will be described in more detail in 2.2.3.

The nodes in the c-structure have corresponding numbers in the f-structure, indicating which part of the f-structure a given c-structure node maps to (this



```
F-structure chart
"Mary hops in the garden."

  PRED     'hop<[3:Mary], [99:in]>'
           PRED   'Mary'
           CHECK  [LEX-SOURCE morphology, _PROPER known-name]
  SUBJ     NTYPE  [NSEM [PROPER [NAME-TYPE first_name, PROPER-TYPE name]]]
                  [NSYN proper                                          ]
         3 CASE nom, GEND-SEM female, HUMAN +, NUM sg, PERS 3
           PRED   'in<[113:garden]>'
                  PRED   'garden'
                  CHECK  [LEX-SOURCE morphology]
                  NTYPE  [NSYN common]
  OBL      OBJ    SPEC   [DET [PRED      'the']]
                              [DET-TYPE  def  ]
              113 CASE obl, NUM sg, PERS 3
           PSEM   <loc>
        99 PTYPE  sem
  CHECK    [SUBCAT-FRAME V-SUBJ-OBL]
  TNS-ASP  [MOOD indicative, PERF -_, PROG -_, TENSE pres]
68 CLAUSE-TYPE decl, PASSIVE -, VTYPE main
```

```
OTOrder: ( -BadPlacement -RareForm ) -SpecProper -AdjunctPPinVP -HeaderPunct
```

Figure 2.11: XLE output: fschart and OT marks

is equal to the *functional projection function* which ensures that c-structure and f-structure fit together).

There is also a Prolog format of the f-structure in the XLE grammar. It lists all the facts of an f-structure. Below is the example of the f-structure and its Prolog format of the NP *the girls*:

```
"the girls"

 [PRED   'girl'                                ]
 [CHECK  [_LEX-SOURCE countnoun-lex]           ]
 [NTYPE  [NSEM [COMMON count]]                 ]
 [       [NSYN common        ]                 ]
 [SPEC   [DET [PRED      'the']]               ]
 [       [    [DET-TYPE  def  ]]               ]
1[NUM pl, PERS 3                               ]
```

Figure 2.12: XLE f-structure for the NP *the girls*

```
fstructure('the girls',
% Properties:
[],
% Choices:
[],
% Equivalences:
[],
% Constraints:
[
cf(1,eq(attr(var(0),'PRED'),semform('girl',1,[],[]))),
cf(1,eq(attr(var(0),'CHECK'),var(1))),
cf(1,eq(attr(var(0),'NTYPE'),var(2))),
cf(1,eq(attr(var(0),'SPEC'),var(4))),
cf(1,eq(attr(var(0),'NUM'),'pl')),
cf(1,eq(attr(var(0),'PERS'),'3')),
cf(1,eq(attr(var(1),'_LEX-SOURCE'),'countnoun-lex')),
cf(1,eq(attr(var(2),'NSEM'),var(3))),
cf(1,eq(attr(var(2),'NSYN'),'common')),
cf(1,eq(attr(var(3),'COMMON'),'count')),
cf(1,eq(attr(var(4),'DET'),var(5))),
cf(1,eq(attr(var(5),'PRED'),semform('the',0,[],[]))),
cf(1,eq(attr(var(5),'DET-TYPE'),'def'))
].
```

The convention behind the `var(n)` arguments is that they are interpreted as standing for f-structure nodes/indices. The outermost node is always labeled 0 in an f-structure (`var(0)`). The `PRED` value of the main f-structure (`var(0)`) is `girl`, the `CHECK`-attribute of `var(0)` opens another f-structure (`var(1)`) and so on.

Since transfer rules operate on the Prolog format of f-structures, each `cf` can be seen as a transfer fact. These facts provide the input to the transfer rules. The input facts are then converted to output transfer facts by the ordered rewrite system. The output facts in Prolog provide the basis for the transferred f-structure (Crouch et al. (2008)). This procedure happens with every f-structure transfer that is done in this experiment.

XLE parses and generates sentences on the basis of grammar rules, one or more LFG lexicons, a tokenizer which segments an input stream into an ordered sequence of tokens, and a finite-state morphological analyzer which encodes morphological alternations. The English XLE LFG grammar is one of the most highly developed grammars and is designed to handle well-edited English text (e.g. newspaper text, manuals). Powerset built additional semantic rules on top of the original LFG grammar in order to be able to deal with Wikipedia. The original English grammar developed by PARC is built up of morphology and tokenizer, followed by syntax which is the basis for the semantic representation and in the last step follows the Abstract Knowledge Representation (AKR) (Bobrow et al. (2007)) (the AKR is not used by Powerset and is solely built at PARC). The outlay of the English XLE grammar is explained in the following.

2.2.3 The English XLE Grammar

Tokenizer and Morphology

First of all, the text is broken into sentences and each sentence is tokenized. The tokenized sentences are then processed by an efficient, broad-coverage LFG grammar run on the XLE system (Crouch et al. (2008)). To get a correct analysis from the syntax, locations like *New York* or dates like *the fifth of January* are processed in a way that they are not split up into several tokens, but are dealt with as one word.

The morphology is built as a finite-state transducer which is used in order to specify natural-language lexicons. It facilitates the definition of morphotactic structure, the treatment of gross irregularities, and the addition of tens of thousands of baseforms typically encountered in natural language. These morphological analyzers are generally built as finite-state transducers with the Xerox finite-state technology tools and follow the methodology established by Beesley and Karttunen (2003). Morphological information is encoded via tags that are attached to the base form of the lexeme, as is illustrated below:

```
hop+Verb+Pres+3Pers+Sg
hops
```

The upper side of the transducer consists of strings showing baseforms and tags and the lower-side language consists of valid words in English (Beesley and Karttunen (2003)). Two-sided networks like these are also called lexical transducers.

The finite-state transducer interfaces with the syntax via the morphology-syntax interface and provides information which is needed in the f-structure and for unification in the c-structure.

Syntax

Sublexical rules on the syntax side pick up the morphological tags and use them for unification in the tree and for features in the f-structure. The lexemes are fed into the right-hand side of the syntax rules (as shown above in the introductory section on LFG). The output is a tree-structure (c(onstituent)-structure), encoding linear order and constituency and an attribute value matrix (f(unctional)-structure) encoding predicate argument structure and semantically important features such as number and tense. The XLE structures are much more articulated than those usually found in LFG textbooks and papers because they contain all the features needed by subsequent processing and applications.

The English XLE grammar produces a packed representation of all possible solutions as its output and also uses a form of Optimality Theory (OT) (Frank et al. (1998)) that allows the grammar writer to indicate that certain constructions are dispreferred. In addition, XLE has the capability of producing well-formed fragments if the grammar does not cover the entire input. The combination of these capabilities makes XLE robust in the face of ill-formed inputs and shortfalls in the coverage of the grammar (Crouch et al. (2008)).

Semantics

In order to get a semantic representation, the syntactic output is processed by a set of ordered rewriting rules — also called the transfer system XFR. The rewrite system applies rewrite rules to a set of packed input terms/facts to produce a set of packed output terms/facts (Crouch et al. (2008)). "The semantics gives a flat representation of the sentence's predicate argument structure and the semantic contexts in which those predications hold." (Crouch and King (2006)). Figures 2.13 and 2.14 show f-structure and semantics for *Mary did not hop.* Figure 2.15 presents a transfer rule for the semantics.

```
"Mary did not hop."

    PRED     'hop<[1:Mary]>'
             PRED  'Mary'
             CHECK [_LEX-SOURCE morphology, _PROPER known-name]
    SUBJ     NTYPE  NSEM [PROPER [NAME-TYPE first_name, PROPER-TYPE name]]
                    NSYN proper
            1CASE nom, GEND-SEM female, HUMAN +, NUM sg, PERS 3
    ADJUNCT {  PRED         'not'  }
            {84ADJUNCT-TYPE neg    }
    CHECK    [_SUBCAT-FRAME V-SUBJ]
    TNS-ASP  [MOOD indicative, PERF -_, PROG -_, TENSE past]
  57CLAUSE-TYPE decl, PASSIVE -, VTYPE main
```

Figure 2.13: F-structure for *Mary did not hop.*

```
cf(1, context_head(t,hop:n(14,'**'))),
cf(1, in_context(t,past(hop:n(14,'**')))),
cf(1, in_context(t,cardinality('Mary':n(1,'**'),sg))),
cf(1, in_context(t,proper_name('Mary':n(1,'**'),name,'Mary'))),
cf(1, in_context(t,role(adeg,not:n(10,'**'),normal))),
cf(1, in_context(t,role(amod,hop:n(14,'**'),not:n(10,'**')))),
cf(1, in_context(t,role(sem_subj,hop:n(14,'**'),'Mary':n(1,'**')))),
cf(1, original_fsattr('ADJUNCT',hop:n(14,'**'),not:n(10,'**'))),
cf(1, original_fsattr('SUBJ',hop:n(14,'**'),'Mary':n(1,'**'))),
cf(1, original_fsattr(gender,'Mary':n(1,'**'),female)),
cf(1, original_fsattr(human,'Mary':n(1,'**'),'+')),
cf(1, original_fsattr(subcat,hop:n(14,'**'),'V-SUBJ')),
cf(1, skolem_byte_position('Mary':n(1,'**'),1,4)),
cf(1, skolem_byte_position(hop:n(14,'**'),14,16)),
cf(1, skolem_byte_position(not:n(10,'**'),10,13)),
cf(1, skolem_info('Mary':n(1,'**'),'Mary',name,name,n(1,'**'),t)),
cf(1, skolem_info(hop:n(14,'**'),hop,verb,verb,n(14,'**'),t)),
cf(1, skolem_info(not:n(10,'**'),not,adv,adv,n(10,'**'),t))
```

Figure 2.14: Semantic representation for *Mary did not hop.*

Each clause of the core of the Prolog representation "is set within a context (`in_context`)" (Fig. 2.14) (Crouch and King (2006)). They can be introduced by clausal complements like COMPs and XCOMPs in the f-structure, but can also be introduced lexically, in this case by the sentential adverb *not.*

The transfer system applies an ordered set of rewrite rules, "which progressively consume the input f-structure replacing it by the output semantic representation" (Crouch and King (2006)). Figure 2.15 shows a transfer

```
PRED(%V, hop), SUBJ(%V, %S), -OBJ(%V, %%), -OBL(%V, %%)
==>
word(%V, hop, verb), role(Agent, %V, %S).
```

Figure 2.15: Transfer rule to insert thematic information

rule that would insert thematic information for the subject in *Mary did not hop.* in the semantic representation. This transfer rule runs through the f-structure, if it can find a node `%V` (the `%` is used to indicate a variable), which in this case is the verb *hop*, and a subject `%S`, the rule fires. If the left-hand side of the rule is matched, the matching facts PRED and SUBJ are removed from the description and are replaced by the content on the right-hand side of the rule.[1]

On the basis of all the information on the XLE system one can say that the more information is included in the f-structure, the more precise is the semantic analysis. This poses the challenge for my transfer algorithm, because the more features can be added to the stochastic DCU f-structures, the better are the matching results between the PARC output and the transferred DCU output. If it is possible to add enough information, then the approach of using the stochastic syntax output could prove to be much quicker considering developing time and existing resources could be used.

Abstract Knowledge Representation (AKR)

To get to an Abstract Knowledge Representation (AKR) (Bobrow et al. (2007)), natural language sentences are mapped into a logical abstract knowledge representation language. Using this mapping, the application supports high-precision question-answering of natural language queries from large document collections. For example, if a collection includes the sentence *The man killed the President in January.*, the system could answer the queries *Did anyone die in January?* and *Did the President die?* with YES and negate the

[1]The "-" on the left-hand side of the rule indicates that the rule is only allowed to fire, if no object or oblique is being found in the argument structure of the verb. If a "+" is put in front of a transfer fact, then this fact is not consumed by the rule but is still available for later application.

query *Did anyone die in February?* Also, the phrase in the document where this information is found, could be highlighted (Bobrow et al. (2007)).

I will not go into further detail on the AKR, as it is not of significant importance for the experiment conducted here.

2.2.4 ParGram

Within a given linguistic theory (e.g. LFG), there are often several possible analyses for syntactic constructions. In any language, there might be two or three possible solutions for one construction, probably one solution being the most obvious and elegant, also taking into account that this solution might be the most elegant for other languages as well (Butt et al. (1999)). This effort of keeping grammars as parallel as possible with respect to syntactic analyses has been the aim of the ParGram (Parallel Grammar) project. Having started out with three languages (English, German and French), the cooperation has attracted many new languages, among them Japanese, Turkish, Indonesian and Urdu (developed here in Konstanz).

The loose connection of researchers from California, Europe, Japan and Turkey meets twice a year to keep the grammar development as parallel as possible. To keep up with the development of parallel semantics on top of the syntax grammar, a new project namely ParSem is being planned, which projects the aims of ParGram on the development of parallel semantics.

2.2.5 Interim Summary

After having explained the necessary details on the English XLE grammar and the syntax theory behind it (LFG), I would now like to present the counterpart to the rule-based XLE system, the annotation algorithm on top of Penn-II treebanks of Dublin City University (DCU).

The output of the stochastic parser is being used as input to the rule-based XLE grammar and therefore hybridizes the XLE system. Basis of the stochastic parser is the Penn-II treebank (Marcus et al. (1994)), which is annotated with f-structure information. The annotation process is the focus of the coming section on the LFG treebank annotation algorithm of Dublin City University.

2.3 DCU Annotation Algorithm

Traditionally, deep unification- or constraint-based grammars (for instance the English XLE grammar) have been manually constructed, which is time-consuming and expensive. The availability of treebank resources has facilitated "a new approach to grammar development: the automatic extraction of probabilistic context-free grammars (PCFGs) from treebanks" (Burke (2006)).

Treebanks are a corpus of parsed sentences; parsed in the sense that the sentences are annotated with syntactic information. Syntactic information has traditionally been represented in a tree structure, hence the name treebank. It is possible to annotate a corpus with simple labelled brackets which represent constituency and allow the extraction of simple predicate-argument structures (Marcus et al. (1993)). Most of the time, the corpus has been additionally annotated with part-of-speech tags, providing every word in the corpus with its wordclass.

Dublin City University (DCU) has developed an automatic treebank annotation algorithm which annotates the Penn-II treebank with LFG f-structure information (Cahill (2004)). The annotated treebank can be used as a training resource for stochastic versions of unification and constraint-based grammars and for the automatic extraction of such resources (Cahill and Mccarthy (2002)). The treebank is annotated in a way that by solving the annotated functional equations, LFG-like f-structures can be produced. The annotations describe what are called "proto-f-structures", which

- "enocde basic predicate-argument-modifier structures;
- may be partial or unconnected (i.e. in some cases a sentence may be associated with two or more unconnected f-structure fragments rather than a single f-structure);

- may not encode some reentrancies, e.g. in the case of wh- or other movement or distribution phenomena (of subjects into VP coordinate structures etc.)" (Cahill and Mccarthy (2002))

Figure 2.16 shows an annotated tree for the noun phrase *the mouldy hay*, with the resulting f-structure in Figure 2.17.

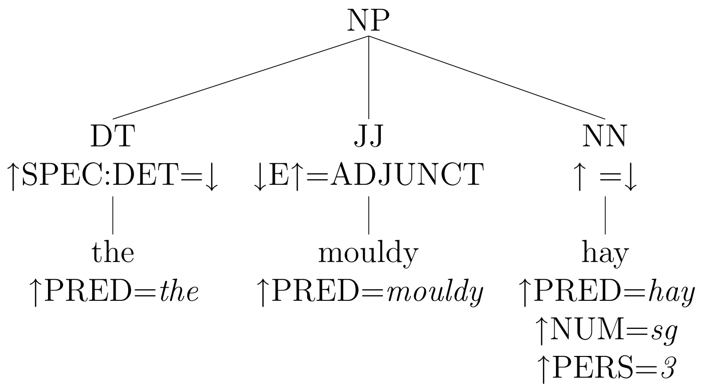


Figure 2.16: Automatically annotated Penn-II tree for *the mouldy hay*

$$\begin{bmatrix} \textsc{spec} & \begin{bmatrix}\textsc{det} & \begin{bmatrix}\textsc{pred} & \text{the}\end{bmatrix}\end{bmatrix} \\ \textsc{adjunct} & \begin{bmatrix}\textsc{pred} & \text{mouldy}\end{bmatrix} \\ \textsc{pred} & \text{hay} \\ \textsc{num} & \text{sg} \\ \textsc{pers} & 3 \end{bmatrix}$$

Figure 2.17: Resulting f-structure for *the mouldy hay*

The annotation algorithm is implemented in Java as a recursive procedure and proceeds in a top-down, left-to-right manner. The annotation of a subtree begins with the identification of the head node. For each Penn-II parent category, the rules list the most likely head categories in rank order and indicate the direction from which the search for the head category should begin. E.g. a rule indicates that the head of an S subtree is identified by traversing the daugther nodes from right to left and a VP is the most likely head. The annotation algorithm marks the rightmost VP in an S subtree as head using the f-structure equation: `^=!`. If the S subtree does not contain a VP node, it is searched from right to left for the next most likely head candidate. In the unlikely event that none of the listed candidates occur in the subtree, the rightmost non-punctuation node is marked as head.

In *the mouldy hay*, the NP node is annotated `^=!` as the NP head rules indicate that the rightmost nominal node is the head. The nodes `DT` (for *the*) and `JJ` (for *mouldy*) lie in the left context. Consulting the NP annotation matrix provides the annotations `^SPEC: DET=!` and `!E^ADJUNCT` for `D` and `ADJUNCT`, respectively. Lexical macros for each Penn-II POS tag provide annotations for word nodes, e.g. verbal categories are annotated with `TENSE` features while nouns receive number and person features. The annotation algorithm and the automatically-generated f-structures are the basis for the automatic acquisition of wide-coverage and robust probabilistic approximations of LFG grammars.

This approach, like previous shallow automatic grammar acquisition techniques, is quick, inexpensive and achieves wide coverage (Burke (2006)). Evaluation against gold standards, especially dependency-based gold standards such as the PARC700[2] (King et al. (2003)) and PropBank (Palmer et al. (2005)) have shown that the results of this LFG-like parser are of high quality (e.g. an f-score of 82.73% against the PARC700). Foster (2007) shows in addition that stochastic grammars, such as those used by the DCU parser, can be trained to have improved coverage of ungrammatical sentences.

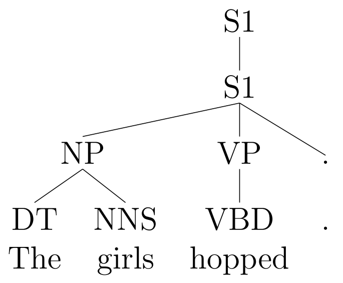
DCU's efforts have resulted in a robust parser (Cahill et al. (2008)) that saves a lot of time in creating f-structures compared to the rule-based system of PARC. However, a lot of information has to be added in order to create f-structures as precise as those generated by PARC. Therefore it's worthwhile to conduct an experiment where probabilisitic f-structures are augmented and the resulting f-structures are evaluated to see if they can be used as input to a rule-based semantic system. Two DCU structures out of my own training data are provided in section 2.4 in order to illustrate what was the basis of the transfer process and how much work needed to be done.

[2]PARC700 consists of 700 sentences extracted from section 23 of the UPenn Wall Street Journal treebank. It contains predicate-argument relations and other features.

Part of my job at Dublin City University in 2009 will be to work on the annotation algorithm, trying to optimize it in a way that the initial output is closer to the PARC f-structures in order to optimize the transfer process.

2.4 Hybridization of the XLE pipeline

This thesis reports on an experiment to use the DCU LFG-like output as input to the PARC semantics. The main issue was whether the DCU structures could be augmented and changed to closely enough match the XLE output. In general, the issue was in adding additional features since the features in the DCU output were already highly parallel to that of the XLE output due to the DCU's participation in the Parallel Grammar (ParGram) project (Butt et al. (1999, 2002)). The ParGram project aims to produce similar f-structures cross-linguistically for similar syntactic constructions; in the case of the English DCU and XLE systems, the parallelism was within one language but across two systems.



```
subj :   spec :    det : pred : the
         pred :    boy
         num :     pl
         pers :    3
pred :   hopped
tense :  past
```

Figure 2.18: DCU c- and f-structure for *The girls hopped*

One sample of DCU structures is shown in Figure 2.18. Comparing it to f-structures shown in the LFG introduction reveals that the core predicate-argument structure and semantic features are available in the the DCU structure, however some information is left unspecified (e.g., case, determiner type, noun type, negative values for features). The terminal nodes have different names than the nodes in the XLE grammar, however this is not relevant in this experiment as only the f-structures matter for the transfer system.

```
"The girls hopped."

   PRED      'hop<[21:girl]>'
             PRED   'girl'
             CHECK  [_LEX-SOURCE countnoun-lex]
                    NSEM [COMMON count]
             NTYPE
   SUBJ             NSYN common

             SPEC   DET  PRED      'the'
                         DET-TYPE def
           21 CASE nom, NUM pl, PERS 3
   CHECK     [_SUBCAT-FRAME V-SUBJ]
   TNS-ASP   [MOOD indicative, PERF -_, PROG -_, TENSE past]
64 CLAUSE-TYPE decl, PASSIVE -, VTYPE main
```

Figure 2.19: PARC's output for *The girls hopped*

To give a quick account of what PARC would produce for this sentence, I show their f-structure for *The girls hopped.* (Figure 2.19). Despite the fact that the DCU f-structure lacks the brackets that a "normal" f-structure has, it also lacks a lot of features. For instance, almost all information on tense and aspect is missing in the DCU structure. Also, many features on the noun *girls* is missing, e.g. that it is a proper count noun in the nominative. In addition, clause type features are missing. The sequence of ordered rewrite rules that I wrote ensures the inclusion of these features.

The following section describes the process of altering the DCU output to make it as similar to the PARC output as possible so that it can serve as input to the PARC semantics. I will give a brief overview of the basics of packed rewriting and then focus on the explanation of the transfer algorithm, therefore coming to the heart of this thesis and the experiment.

Chapter 3

Adapting the Stochastic DCU Output

The system of Dublin City University provides a probabilistic treebank-based parser (PTBP) that uses Penn-II Treebank trees (Marcus et al. (1994)), which are then annotated with functional equations that are solved to produce f-structures.[1] This is a quick, inexpensive approach in order to create a wide-coverage grammar. DCU then augments their generated f-structures with additional features they insert so that they are able to evaluate their stochastic results against dependency banks, e.g. PARC700 (King et al. (2003)). This brings the f-structures significantly closer to those used by the PARC system (see the section on future work for discussion of this step). The structures are then reformatted by a short Prolog script written at PARC to serve as input to the PARC XLE ordered rewriting system.

The issue explored in this experiment was whether the DCU output contained sufficient information after the application of the ordered rewrite rules (core component of this thesis) so that the semantics can process them and extract the information needed for a semantic representation[2]. The processing pipeline in Fig 3.1 shows the outlay of the experiment.

[1]The DCU grammars use two parsing architectures (Cahill et al. (2002)). The details are unimportant for this experiment since the output is identical for both architectures.

[2]C-structure information plays a minor role here. Although the semantics uses the c-structure to determine the position of the words in the sentence (useful in applications for highlighting the original text), the c-structure was ignored in this experiment.

DCU-XLE Processing Pipeline

text breaker (FST)

↓

DCU syntax output

(PTBP + ANNOTATION ALGORITHM)

↓

DCU feature augmentation

↓

reformatting (PROLOG SCRIPT)

↓

main feature augmentation

(XFR ORDERED REWRITING)

↓

semantics (XFR ORDERED REWRITING)

↓

AKR (XFR ORDERED REWRITING)

Figure 3.1: Processing Pipeline from DCU to PARC

In the following sections I will go through the experiment; step by step from the DCU syntax output to the ordered rewrite rules (XFR) and special rules that changed the overall structure of DCU output. I will give examples of the code for each step and also focus on some of the problems that arose during the transfer process.

3.1 DCU Syntax Output

Thanks to the help of Jennifer Foster from Dublin City University, the hundreds of test sentences I used as training data for the transfer were batch-parsed at DCU. Batch-parsing means that the the parser parses every sentence of a testfile one after another and puts the result for each sentence in a single file. This file contains the Prolog format for each f-structure. Nevertheless, there is also an online-version of the parser available on the DCU webpage (http://lfg-demo.computing.dcu.ie/parc_lfgparser.html), which can parse a whole set of sentences but puts the result for all sentences in one file.

The output of the DCU parser is an f-structure in Prolog format, similarly built up like the XLE Prolog output for an f-structure. As an example, Figure 3.2 shows the Prolog output for sentence number 126 of the training data; *He has a tractor.* Figure 3.3. shows the corresponding f-structure.

```
fstr(fstructure_126,
[subj:[pred:pro,pron_form:he,num:sg_6707],
stmt_type:declarative,
tense:pres,
pred:have,
obj:[spec:[det:[pred:a|_6672]|_6677],
pred:tractor,num:sg,pers:3|_6657]|_6687]).
```

Figure 3.2: DCU Prolog file

```
   [obj   [spec [det [pred a]]                    ]
   [      [num sg, pers 3, pred tractor]          ]
   [subj  [num sg, pred pro, pron_form he]        ]
-1 [pred have, stmt_type declarative, tense pres  ]
```

Figure 3.3: DCU f-structure for *He has a tractor.*

This output needed to be reformatted in order to be loaded into XLE.

3.2 Reformatting the DCU output

The initial output of DCU cannot be used in the XLE system due to the different Prolog formatting used by DCU. Therefore, a reformatting program was written in Prolog by Rowan Nairn from PARC, to convert the DCU output into a format that can be loaded into XLE. It modifies the syntax of the file in a way that the transfer rules can apply. An exemplary reformatted DCU output can be seen in Figure 3.4. One can see that in the original DCU Prolog output, no contexted facts (`cf`) appear. Contexted facts show in which context facts are true. In the example below, there is only one context, namely context 1. The reformatted output for *He has a tractor.* can be seen in Figure 3.4.

```
fstructure(dcu2xle,
[],
[],
[],
[cf(1,eq(attr(var(0),subj),var(1))),
cf(1,eq(attr(var(1),pred),pro)),
cf(1,eq(attr(var(1),pron_form),he)),
cf(1,eq(attr(var(1),num),sg)),
cf(1,eq(attr(var(0),stmt_type),declarative)),
cf(1,eq(attr(var(0),tense),pres)),
cf(1,eq(attr(var(0),pred),have)),
cf(1,eq(attr(var(0),obj),var(2))),
cf(1,eq(attr(var(2),spec),var(3))),
cf(1,eq(attr(var(3),det),var(4))),
cf(1,eq(attr(var(4),pred),a)),
cf(1,eq(attr(var(2),pred),tractor)),
cf(1,eq(attr(var(2),num),sg)),
cf(1,eq(attr(var(2),pers),3))],
[]).
```

Figure 3.4: Reformatted DCU f-structure prolog file

The top f-structure has the variable 0 (`var(0)`) and contains the predicate `have`. The `SUBJ` of the sentence is stored under variable 1 (`var(1)`), which contains a pronominal predicate with the pron_form `he`. The `OBJ` of variable 0 is variable 2, `the tractor`, which is third person singular.

3.3 Ordered Rewrite Rules (XFR)

The input to the experiment is a set of Prolog facts representing the f-structures obtained by the DCU parser and the output is a set of transferred Prolog facts representing the f-structures that are fed into the PARC semantic system. The transfer system operates on a source f-structure and transforms it incrementally into a target structure. The operation controlled by a transfer grammar consists of a list of rules whose order is important because each rule has the potential of changing the situation that the subsequent rules will encounter. In particular, rules can prevent following rules from applying by removing facts that they would otherwise have applied to.

They can also enable the application of later rules by introducing material that these rules need.

The rewriting works as follows: if a set of f-structure features (or part of an f-structure) is recognized by the left-hand side of a rule, then the rule applies to produce the features on the right-hand side of the rule. A simple transfer rule which changes `Mary` to `Marie` (in the case of an English to French translation) is shown in the following figure:

$$\begin{bmatrix} \text{PRED} & \text{'Mary'} \\ \text{GEND-SEM} & \text{female} \end{bmatrix}$$

```
PRED(%2, Mary), GEND-SEM(%2, female)
==>
PRED(%2, Marie), GEND-SEM(%2, female).
```

$$\begin{bmatrix} \text{PRED} & \text{'Marie'} \\ \text{GEND-SEM} & \text{female} \end{bmatrix}$$

Figure 3.5: Transfer process from *Mary* to *Marie*

The left-hand side of the rule goes through the list of transfer facts and matches with the `PRED` argument that has the value `Mary` and also picks up the `GEND-SEM` attribute with the `female` value. As soon as both components are found, the rule transfers these facts into what is on the right-hand side of the rule. This is a very simple example of how the transfer between DCU and PARC f-structures works. In the following section, I will focus on my system, present the overall composition of the transfer system and explain certain rules.

3.4 The Algorithm

The XFR transfer algorithm is the heart of the experiment. It is the link between the time-saving DCU f-structure parser which does not assign much information and the time-consuming rule-based XLE system of PARC, whose f-structures are rich with information in order to get a detailed semantic representation. The transfer algorithm is a set of 162 rewrite rules and an additionally included file with all verbs in English together with their subcategorization frames. The top lines of the file look like the following:

```
"PRS (1.0)"

grammar = transfer_new.

"*******************************TRANSFER NEW***********************"

include(verb_subcats_nette2‘.pl).
"verb subcatframes from the English grammar"

"******************************************************************"
```

The first thing that has to be done in an XFR transfer system, is to declare which rule syntax is used. This is specified in the first non-blank line in the rule file with the comment `PRS (1.0)`, which stands for Packed Rewrite Syntax, Version 1.0. Once the rule syntax is specified, the rule set must be given a name, in my case the algorithm is called `transfer_new`.

In advanced transfer systems, other files are included in the process with the Prolog command `include(filename'.pl)`. Here, a list of all English verbs with subcategorization frame (`verb_subcats.pl`) is included in the transfer system. Especially for large rule sets it is convenient to split rules across multiple files (Crouch et al. (2008)). Most of the time it is sensible to include these additional files on the top, otherwise the system gets less and less transparent.

3.4.1 Verbs

The addition of features for verbs is one of the most important tasks of the transfer system, as many features specify TNS-ASP and the subcategorization frame. In the following sections I discuss the initial problems and present solutions as to how these problems were solved.

Subcategorization Frames

The system starts out by adding the missing subcategorization frames to the DCU output; these are essential for mapping the verb's arguments to thematic roles in the semantics. At first, ditransitives are filtered out, after that sentences with transitive and intransitive verbs. That way, no ditransitive sentence is consumed by the rule for intransitive declaratives. The problem is that the original DCU f-structures do not contain any information concerning the subcategorization frames of verbs. To overcome this major lack of information I included the verb lexicon of the English XLE grammar (`verb_subcats.pl`) which contains all English verbs together with their subcategorization frames. The XLE grammar's verb lexicon contains almost 9800 verb stems with an average of 2.8 subcategorization frames each. Most of the frames were obtained from electronic dictionaries or by hand. For ways to bootstrap creation of such lexical resources from treebanks see O'Donovan et al. (2005). To extract the subcategorization information, I connected the verb in the DCU f-structure with the lexicon entry in the XLE verb lexicon via a transfer rule.

Figure 3.6 shows the original DCU f-structure for the sentence *He flashes it.* Underneath is the lexical entry for *flash* in the verb lexicon and the corresponding transfer rule, which feeds the missing subcategorization information into the f-structure.

```
   [obj  [num sg, pred pro, pron_form it]          ]
   [subj [num sg, pred pro, pron_form he]          ]
 -1[pred flash, stmt_type declarative, tense pres  ]
```

```
|- verb_subcat(flash,V-SUBJ-OBJ).

+pred(%1, %2), +subj(%1, %%), +obj(%1, %%),
-obj2(%1, %%), -adjunct(%1, %%), -xcomp(%1, %%),
+tense(%1, %%), +stmt_type(%1, %%),
(verb_subcat(%2, %Subcat), {%Subcat \=%%:%%})
==>
CHECK(%1, %Check), _SUBCAT-FRAME(%Check, %Subcat).
```

```
  [PRED     'flash<[-1-SUBJ:he], [-1-OBJ:it]>'                                          ]
  |         [PRED  'he'                                                          ]     |
  |SUBJ     |NTYPE [NSYN pronoun]                                                |     |
  |         [CASE nom, GEND-SEM male, HUMAN +, NUM sg, PERS 3, PRON-TYPE pers]     |
  |CHECK    [_SUBCAT-FRAME V-SUBJ-OBJ]                                                |
  |         [PRED  'it'                                                               ]|
  |OBJ      |NTYPE [NSYN pronoun]                                                     ||
  |         [CASE obl, GEND-SEM nonhuman, HUMAN -, NUM sg, PERS 3, PRON-TYPE pers]|
  |TNS-ASP  [MOOD indicative, PERF -_, PROG -_, TENSE pres]                          |
-1[CLAUSE-TYPE decl, PASSIVE -, VTYPE main                                             ]
```

Figure 3.6: Insertion of subcategorization features

The left-hand side of the rule picks up all the information it gets from the DCU f-structure. The variable `%2` stands for every transitive verb in a declarative sentence. If we apply the rule to this sentence, the `%2` stands for the verb *flash*. The placeholder `%%` states that any subject can be involved in the sentence, as well as any object (`obj(%1, %%)`). So, if a sentence like *He flashes it.* has to be transferred, the rule in 3.6 fires. The sentence contains a predicate which needs a subject and an object, but has neither an indirect object (`obj2`), nor an adjunct or an xcomp. The `verb_subcat(%2, %Subcat)` pattern automatically gets the verb from `verb_subcats_nette2.pl` and its information about the subcategorization frame. The variable `%Subcat` assigns whatever argument structure is assigned in `verb_subcat_nette2.pl`.

A plus in front of a instantiation fact (as in the first and third line of the rule example) means that this fact is not allowed to be consumed by the rewrite rule, but has to be available for the application of rules later on in the set. The minus in front of a transfer fact (second line in the example) is called a negated pattern, which can only be included on the left-hand side of rules. A negated pattern means that the rule only applies if there are no f-structure facts that match the negated pattern.

On the right-hand side of the rule, all consumed facts are transfered into transfer facts similar to the PARC f-structure for the argument structure of verbs. The information about the argument structure of the verb is encoded in the feature `_SUBCAT-FRAME` which is inside a `CHECK` feature. In the transferred f-structure shown in 3.6 various other things have been changed, among them the features for nouns and pronouns, which will be discussed later. Other verb-related features such as `tense`, `vtype`, `mood` and `passive` were added or reconfigured by a separate rule for all clause types (declarative, imperative, interrogative). This is discussed in the following section.

Tense and Aspect

Other verb-related features such as `TENSE`, `VTYPE`, `MOOD` and `PASSIVE` posed a challenge to the system at some stages, because it was quite difficult to get the right tense features assigned. It is still a problem in the sense that imperatives and interrogatives clauses sometimes do not get the right `TNS-ASP` features. Apart from that, the right features are assigned in almost all cases, apart from occurrences when verbs aren't correctly stemmed on the DCU side.

```
   [obj  [num sg, pred pro, pron_form it]                  ]
   [subj [num sg, pred pro, pron_form he]                  ]
-1 [pred flash, stmt_type declarative, tense pres          ]
```

```
+stmt_type(%1, declarative), -pred(%1, be), tense(%1, pres),
-perf(%1, +), -prog(%1, +), -modal(%1, +)
==>
CLAUSE-TYPE(%1, decl), PASSIVE(%1, -), VTYPE(%1, main),
TNS-ASP(%1, %2), MOOD(%2, indicative),
PERF(%2, -_), PROG(%2, -_), TENSE(%2, pres).
```

```
  [PRED     'flash<[-1-SUBJ:he], [-1-OBJ:it]>'
  [         [PRED   'he'
  [SUBJ     [NTYPE  [NSYN pronoun]
  [         [CASE nom, GEND-SEM male, HUMAN +, NUM sg, PERS 3, PRON-TYPE pers]
  [CHECK    [_SUBCAT-FRAME V-SUBJ-OBJ]
  [         [PRED   'it'
  [OBJ      [NTYPE  [NSYN pronoun]
  [         [CASE obl, GEND-SEM nonhuman, HUMAN -, NUM sg, PERS 3, PRON-TYPE pers]
  [TNS-ASP  [MOOD indicative, PERF -_, PROG -_, TENSE pres]
-1[CLAUSE-TYPE decl, PASSIVE -, VTYPE main                                          ]
```

Figure 3.7: Insertion of tense and aspect features

A rule example for the assignment of the correct features for an indicative clause in the present is shown in Figure 3.7. In order to make clear which features were changed, the original and the transferred f-structure are also included. One can say that all clauses in the present tense get the necessary features, unless there is some feature irregularity on the DCU side, which hardly happens.

It is very important to define on the left-hand side of the rule that no `pred` attribute with the value `be` is allowed for the rule to fire. Also, no `perf`, `prog`, `modal` attributes are allowed in the original DCU f-structure. Sentences with the verb `to be` have a different `TNS-ASP` structure which is why I have to write two separate rules. This rule is shown in Figure 3.8.

```
+stmt_type(%1, declarative), +pred(%1, be),
tense(%1, pres), -perf(%1, +), -prog(%1, +), -modal(%1, +)
==>
CLAUSE-TYPE(%1, decl), PASSIVE(%1, -), VTYPE(%1, main),
TNS-ASP(%1, %2), MOOD(%2, indicative), PERF(%2, -_), PROG(%2, -_),
TENSE(%2, pres).
```

Figure 3.8: Rule to assign tense and aspect features for the verb *to be*

Another rule shows the transfer for clauses in the future tense as in the example *Mary will hop.*:

```
stmt_type(%1, declarative), tense(%1, fut), -perf(%1, +),
-prog(%1, +), modal(%1, +)
==>
CLAUSE-TYPE(%1, decl), PASSIVE(%1, -), VTYPE(%1, main),
TNS-ASP(%1, %2), MOOD(%2, indicative), PERF(%2, -_), PROG(%2, -_),
TENSE(%2, fut).
```

Figure 3.9: Rule to assign tense and aspect features for the future tense

The `tense` and `stmt_type` facts are typical for declarative sentences in the DCU structures, they don't appear in f-structures for interrogative and imperative sentences, therefore they are used as "anchors" to find out very early in the rule set, whether a sentence is declarative, interrogative or imperative.

The two facts are included as negated patterns in the verb subcategorization rules for interrogative and imperative sentences, and therefore prevent declaratives to be misanalyzed. This is explained in detail later on, when the change of the overall structure of f-structures is discussed.

3.4.2 Nouns and Pronouns

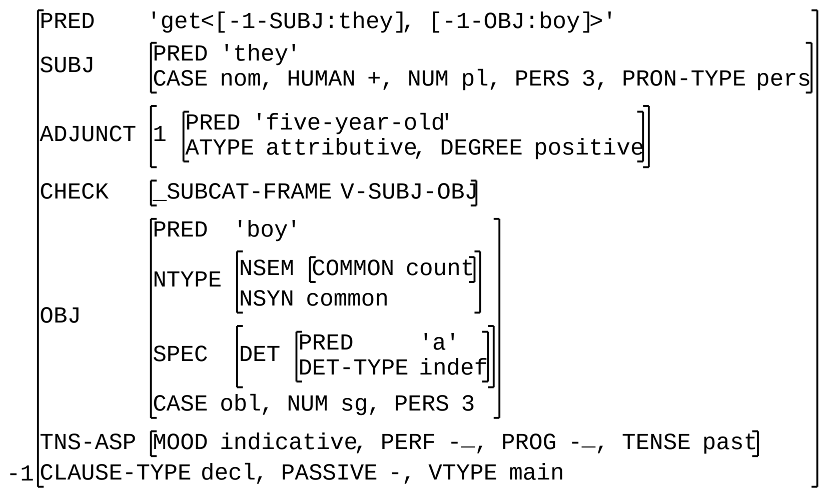
Nouns

Due to their special features in XLE f-structures, proper nouns, including person names (*Mary*, *John*, etc.) and locative nouns like city names and place names need to minimally be identified as proper nouns and, ideally, be provided with features indicating their type since these provide more accurate concept lookup in the semantics. In the following example (Figure 3.10) we see the transfer from DCU to PARC for a simple f-structure.

The top part of Figure 3.10 shows the original DCU f-structure for sentence 181 of my training data (*They got a five year old boy*). The rewrite rule below checks all the attributes of the f-structure for *boy* and states that no `proper` feature with the values `misc`, `location` or `date` is allowed. It adds information on the type of the noun (`NTYPE` f-structure) and rewrites the number and person information.

```
  [      [adjunct [1 [adegree positive, pred five-year-old]]]]
  [obj   [spec    [det [pred a]]                           ]]
  [      [num sg, pers 3, pred boy                         ]]
  [subj [num pl, pred pro, pron_form they]                  ]
-1[pred get, stmt_type declarative, tense past              ]
```

```
pred(%1,%pred), num(%1,sg), pers(%1,3),
-proper(%1, misc), -proper(%1, location), -proper(%1, date)
==>
PRED(%1,%pred), NUM(%1,sg), PERS(%1,3),
NTYPE(%1, %2), NSEM(%2, %3), COMMON(%3, count), NSYN(%2, common).
```



```
  [PRED      'get<[-1-SUBJ:they], [-1-OBJ:boy]>'                              ]
  |SUBJ      [PRED 'they'                                                  ]  |
  |          [CASE nom, HUMAN +, NUM pl, PERS 3, PRON-TYPE pers            ]  |
  |ADJUNCT   [1 [PRED 'five-year-old'                    ]]                   |
  |             [ATYPE attributive, DEGREE positive      ]]                   |
  |CHECK     [_SUBCAT-FRAME V-SUBJ-OBJ]                                       |
  |          [PRED   'boy'                           ]                        |
  |          |NTYPE  [NSEM [COMMON count]]           |                        |
  |OBJ       |       [NSYN common        ]           |                        |
  |          |SPEC   [DET [PRED     'a'   ]]         |                        |
  |          |            [DET-TYPE indef ]]         |                        |
  |          [CASE obl, NUM sg, PERS 3               ]                        |
  |TNS-ASP   [MOOD indicative, PERF -_, PROG -_, TENSE past]                  |
-1[CLAUSE-TYPE decl, PASSIVE -, VTYPE main                                    ]
```

Figure 3.10: Transfer process for *They got a five year old boy*

Many time-related nouns, such as months, days, and seasons were lexicalized, as the DCU output did not provide enough information to distinguish them from other nouns. In Figure 3.11 we see the syntax of a transfer template. Templates allow for time-saving lexicalization, because one and the same “rule-skeleton” is used and different lexical forms are fed into the template instead of writing an individual rule for each lexical entry.

```
date(%DCU, %xle)::
pred(%1, %DCU), num(%1, sg), pers(%1, 3), proper(%1, date)
==>
PRED(%1, %xle), NTYPE(%1, %2), NSEM(%2, %3), TIME(%3, date),
NE-TYPE(%1, time_date), NUM(%1, sg), PERS(%1, 3).

@date(january, month‘(1‘)).
@date(february, month‘(2‘)).
@date(march, month‘(3‘)).
@date(april, month‘(4‘)).
@date(may, month‘(5‘)).
@date(june, month‘(6‘)).
@date(july, month‘(7‘)).
@date(august, month‘(8‘)).
```

Figure 3.11: Transfer of months with a template

`date` is the name of the template, followed by the declaration of the variables used (`%DCU` and `%xle`) and a `::` which is required in the the rule syntax for templates. Then the usual rewrite rule follows with the designated variables. Below follows the lists of lexical items that are supposed to be rewritten, the template is called with `@date`, followed by the item to be transferred and the item it is transferred to.

Pronouns

Personal, as well as possessive, demonstrative, interrogative and relative pronouns have to be listed individually due to the lack of relevant features on the DCU side. The example below shows a rewrite rule for the personal pronoun *he.* The subject in this sentence is changed, all the other parts are unaffected by this rule.

```
    obj   [adjunct [1 [adegree positive, pred last]]
          [num sg, pers 3, pred winter              ]
    subj  [num sg, pred pro, pron_form he]
 -1 pred laugh, stmt_type declarative, tense past
```

```
pred(%1,pro), pron_form(%1,he), num(%1, sg)
==>
PRED(%1,he), PRON-TYPE(%1, pers), PERS(%1,3), NUM(%1,sg),
HUMAN(%1, +), GEND-SEM(%1,male), NTYPE(%1, %2), NSYN(%2, pronoun).
```

```
   PRED      'laugh<[-1-SUBJ:he], [-1-OBJ:winter]>'
             [PRED  'he'
   SUBJ      [NTYPE [NSYN pronoun]
             [CASE nom, GEND-SEM male, HUMAN +, NUM sg, PERS 3, PRON-TYPE pers]
   ADJUNCT   [1 [PRED 'last'
                [ATYPE attributive, DEGREE positive]]
   CHECK     [_SUBCAT-FRAME V-SUBJ-OBJ]
             [PRED  'winter'
   OBJ       [NTYPE [NSEM [TIME season]]
             [CASE  obl
   TNS-ASP   [MOOD indicative, PERF -_, PROG -_, TENSE past]
-1 CLAUSE-TYPE decl, PASSIVE -, VTYPE main
```

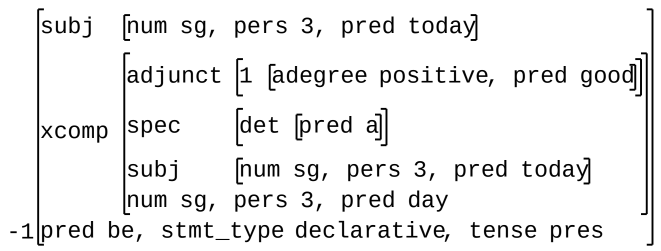
Figure 3.12: Transfer process for *He laughed last winter*

Unfortunately, it is not possible to use templates for pronouns, e.g. every personal pronoun has different `NUM` and `PERS` values and therefore needs its own rule. In addition, case features were added to pronominal forms as well as nouns in case-marked positions which is a second problem when trying to put every personal pronoun in one template (e.g. consider *he* vs. *him*). However, relative pronouns are the only pronouns that could be put in a template. Below is the template for the three pronouns *who*, *that* and *which*.

```
pronrel2pronrel(%DCU, %xle) ::
pron_rel(%1, %2), pred(%2, pro), +pron_form(%2, %DCU), num(%2, sg)
==>
PRON-REL(%1, %2), PRED(%2, %xle), NTYPE(%2, %3), NSYN(%3, pronoun),
PRON-TYPE(%2, rel), TOPIC-TYPE(%2, relative).
@pronrel2pronrel(who, who).
@pronrel2pronrel(that, that).
@pronrel2pronrel(which, which).
```

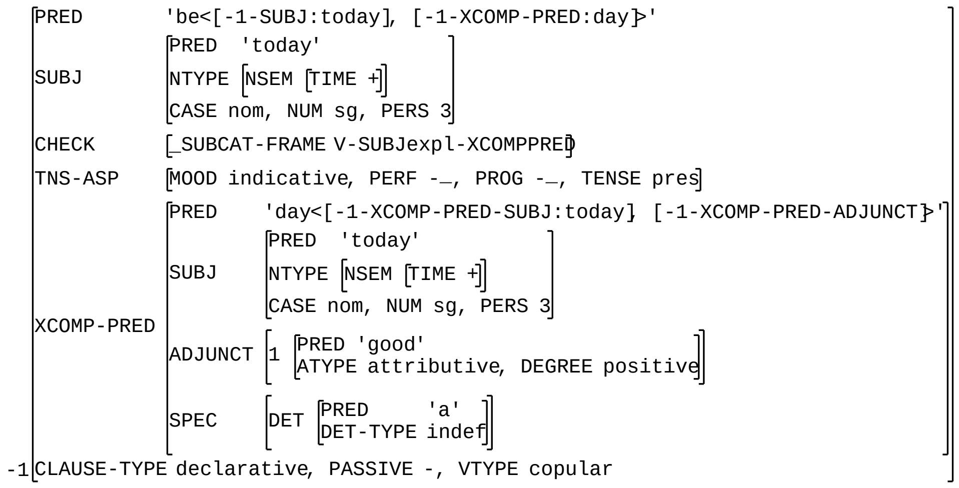
3.4.3 Adjectives and Adverbs

Adjectives and adverbs do not have to be lexicalized. Additional features are added based entirely on their part of speech. This means that for adverbs and adjectives, only one rule is needed in order to catch and transfer each one of them. The transfer process for an adverb can be seen in Figure 3.13.



```
  [subj   [num sg, pers 3, pred today]                            ]
  [       [adjunct [1 [adegree positive, pred good]]]             ]
  [xcomp  [spec    [det [pred a]]                   ]             ]
  [       [subj    [num sg, pers 3, pred today]     ]             ]
  [       [num sg, pers 3, pred day                 ]             ]
-1[pred be, stmt_type declarative, tense pres                     ]
```

```
pred(%1, %DCU), adegree(%1, positive)
==>
PRED(%1, %DCU), ATYPE(%1, attributive), DEGREE(%1, positive).
```



```
   PRED        'be<[-1-SUBJ:today], [-1-XCOMP-PRED:day]>'
               PRED  'today'
   SUBJ        NTYPE [NSEM [TIME +]]
               CASE nom, NUM sg, PERS 3
   CHECK       [_SUBCAT-FRAME V-SUBJexpl-XCOMPPRED]
   TNS-ASP     [MOOD indicative, PERF -_, PROG -_, TENSE pres]
               PRED     'day<[-1-XCOMP-PRED-SUBJ:today]  [-1-XCOMP-PRED-ADJUNCT]>'
                        PRED  'today'
               SUBJ     NTYPE [NSEM [TIME +]]
                        CASE nom, NUM sg, PERS 3
   XCOMP-PRED
               ADJUNCT  1 [PRED 'good'
                           ATYPE attributive, DEGREE positive]

               SPEC     DET [PRED     'a'
                             DET-TYPE indef]
-1 CLAUSE-TYPE declarative, PASSIVE -, VTYPE copular
```

Figure 3.13: Transfer process for *Today is a good day.*

3.4.4 Determiners and other Specifiers

Articles have to be lexicalized, because they do not share the same features and there was no point in splitting up into common features and special features.

```
spec(%1, %2), det(%2, %3), pred(%3,the)
==>
SPEC(%1, %2), DET(%2,%3), PRED(%3, the), DET-TYPE(%3, def).

spec(%1, %2), det(%2, %3), pred(%3, a)
==>
SPEC(%1, %2), DET(%2, %3), PRED(%3, a), DET-TYPE(%3, indef).

spec(%1, %2), det(%2, %3), pred(%3, an)
==>
SPEC(%1, %2), DET(%2, %3), PRED(%3, an), DET-TYPE(%3, indef).
```

The result of the transfer of articles can be seen in one of the transfer structures above. Similar rules exist for other quantifiers, e.g. *every* and *some.*

3.4.5 Some Issues

There were a few places where the overall structure of the analysis was altered by the rewrite rules. One problem was the lack of more detailed features for proper nouns. That is, proper nouns are correctly identified and marked as such, but are not categorized by type. The morphology used in the XLE parser types many proper nouns (e.g., locations (*Detroit*, *France*), organizations (*IBM*, *Congress*), people (*Mary*, *Smith*), and gender for first names (*Mary* vs. *John*)). Such information is valuable for the semantic interpretation, especially for anaphora resolution. This is one reason why the matching between the semantics output of the transferred DCU f-structures and the PARC f-structures is less good in comparison to the f-structure matching. One task for DCU could be to spend some time on inserting more of those features in order for the analysis to supply more precise results. For this experiment, we extracted this information from the morphology (or called the morphology directly) in order to incorporate proper noun typing into the ordered rewriting rules. However, in a run-time system (see chapter 5) we would want to include this information in an automatic fashion, such as invoking the morphology used by the XLE system.

Another difference is that the DCU parser always treats hyphenated forms as single units. For example, in the noun phrase *high-interest loan*, the head noun *loan* is modified by a single adjunct *high-interest*. This loses certain semantic relationships, which would be needed for the semantic matching. Such situations are easy to spot in the DCU structures due to the hyphen in the predicate. It was difficult to systematically reanalyze these within the rewrite rules. However, it might be possible to do so within the semantics where the word meaning lookup could be used to guide the reanalysis. This has not been explored any further, but could be a task in the future in order to optimize the system.

In addition to that, a word would occasionally not be properly lemmatized by the DCU grammar (e.g., *hopped* was not stemmed to *hop*). This was relatively rare, but the semantics depends on lemmatized forms and therefore more investigation is needed on how to avoid this problem.

I have reported on some issues that occurred during the process of writing the transfer rules, although these problems could be spotted fairly easily. In this following section I report on some of the more salient issues.

Imperatives

One systematic problem arose with imperative sentences. The training data for Burke's annotation algorithm (Burke (2006)) consist of sentences of the Wall Street Journal Penn Treebank corpus (Marcus et al. (1994)). On the one hand this results in a good coverage of indicative sentences; on the other hand, interrogatives and imperatives are analyzed very differently from the PARC analyses due to lack of training data (Judge et al. (2005)). This poses a serious problem to the rewriting system. Nevertheless, the ordered rewriting rules solve the problem in the following way:

The DCU structures for imperatives and certain other structures lack subjects. These constructions were identified and the appropriate subject information was provided. This occurs because the f-structures produced by the DCU parser are not subject to the LFG completeness requirement whereby all the arguments of a predicate must be present in the f-structure. An example for the imperative sentence *Take either box* (sentence 116 of the training data) as analyzed by DCU is given in the top part of Figure 3.14. The second f-structure is the transferred f-structure and the third one is the original PARC f-structure.

```
   obj  spec [det [pred either]]
        num sg, pers 3, pred box
-1 pred take, stmt_type declarative
```

```
   PRED     'take'
   CHECK    [INF-TYPE bare, _SUBCAT-FRAME V-SUBJ-OBJ]
            PRED  'box'
            NTYPE  NSEM [COMMON count]
                   NSYN common
   OBJ
            SPEC   DET [PRED 'either']
            NUM sg, PERS 3
   TNS-ASP  [MOOD imperative, PERF -_, PROG -_]
-1 CLAUSE-TYPE imp, PASSIVE -, VTYPE main
```

```
  PRED     'take<[-1-SUBJ:null_pro], [-1-OBJ:box]>'
           PRED  'null_pro'
  SUBJ     NTYPE [NSYN pronoun]
           PERS 2, PRON-TYPE null
  CHECK    [_SUBCAT-FRAME V-SUBJ-OBJ]
           PRED  'box'
           NTYPE [NSYN common]
  OBJ
           SPEC  [QUANT [PRED 'either']]
           CASE obl, NUM sg, PERS 3
  TNS-ASP [MOOD imperative, PERF -_, PROG -_]
-1CLAUSE-TYPE imp, PASSIVE -, VTYPE main
```

Figure 3.14: Transfer process for *Take either box*

The transferred f-structure differs concerning tense and aspect features as well as subcategorization frames. It is much more parallel to the original PARC-generated f-structure. We see at the beginning of the transfer process that a wrong `stmt_type` value is assigned (should be `imperative` instead of `declarative`). Nevertheless, this mistake can be used in a way that, as soon as the system finds an f-structure which has a `pred` and a `stmt_type` but no `tense` feature, the sentence is automatically analyzed as an imperative clause and is assigned all necessary features. These features include `PASSIVE`, `VTYPE` and `TNS-ASP`.

Very importantly, a null-subject was included in every imperative sentence, otherwise the subcategorization frame could not have been assigned. This proved to be quite difficult to implement but in the end it guarantees that a lot of additional information is added which helps the semantics to "understand" what the content of the imperative clause is.

Interrogatives

Another major problem is the transfer of interrogative clauses. As mentioned above, the training data of DCU is a corpus of Wall Street Journal, which does not contain many questions. Due to the lack of training data, questions are analyzed very differently than they are analyzed at PARC. Judge et al. (2006) propose a method to add more interrogatives to the training data to alleviate problems like this by building a QuestionBank. This bank consists of a corpus of 4000 annotated questions used to train parsers in Question Answering Technology and the evaluation of question parsing. As this option was not available in this experiment, another solution to the problem had to be found.

```
  [focus     [adjunct [1 [adegree positive, pred often]]]]
  [          [pred pro, pron_form how                 ]]
  [obj       [num sg, pred pro, pron_form it]          ]
  [pron_int  [adjunct [1 [adegree positive, pred often]]]]
  [          [pred pro, pron_form how                 ]]
-1[pred      appear                                    ]
```

Figure 3.15: DCU f-structure for *How often did it appear?*

```
  [PRED        'appear'                                                          ]
  [            [PRED     'how'                                  ]                ]
  [            [ADJUNCT  [1  [PRED 'often'                    ]]]                ]
  [FOCUS-INT   [             [ADV-TYPE vpadv, DEGREE positive]]]                ]
  [            [NTYPE    [NSYN pronoun]                         ]                ]
  [            [PRON-TYPE int, PSEM temp, PTYPE sem             ]                ]
  [            [PRED  'it'                                                      ]]
  [OBJ         [NTYPE [NSYN pronoun]                                            ]]
  [            [GEND-SEM nonhuman, HUMAN -, NUM sg, PERS 3, PRON-TYPE pers      ]]
  [            [PRED     'how'                                  ]                ]
  [            [ADJUNCT  [1  [PRED 'often'                    ]]]                ]
  [PRON-INT    [             [ADV-TYPE vpadv, DEGREE positive]]]                ]
  [            [NTYPE    [NSYN pronoun]                         ]                ]
  [            [PRON-TYPE int, PSEM temp, PTYPE sem             ]                ]
  [TNS-ASP     [MOOD indicative, PERF -_, PROG -_, TENSE past]                   ]
-1[CLAUSE-TYPE int, PASSIVE -, VTYPE main                                        ]
```

Figure 3.16: Transferred DCU f-structure for *How often did it appear?*

Generally speaking, f-structures for interrogative clauses lack a good deal of information on the DCU side, which was partly good, because then this missing information served as an anchor for the transfer rules. The general problem of missing subjects can be seen in Figures 3.15 and 3.16. The DCU parser does not analyze the subject of the sentence as the subject, but the object. This poses a big problem to the matching later on, because the existence of a subject is crucial to the overall matching of the f-structure. Even more importantly, any mismatch in grammatical functions is a major issue for the semantic processing.

DCU does not assign a stmt_type feature to interrogative clauses (Figure 3.15). This information is included in the transfer rule in a way that, as soon as an f-structure doesn't have a stmt_type and a tense feature but a pred, then the rule for transferring interrogative sentences fires and assigns all the missing features like PASSIVE, VTYPE and TNS-ASP. To compare transferred DCU output and original PARC output, Figure 3.17 is included.

```
"How often did it appear?"

   [PRED        'appear<[76:it]>'                                                           ]
   |            [PRED  'it'                                                                ]|
   |SUBJ        |NTYPE [NSYN pronoun]                                                      ||
   |          76[CASE nom, GEND-SEM nonhuman, HUMAN -, NUM sg, PERS 3, PRON-TYPE pers]      ||
   |            {  [PRED    'often'                          ]}                             |
   |ADJUNCT     {  |ADJUNCT {  [PRED 'how'                  ]}|}                            |
   |            {  |        {21[DEGREE positive, PRON-TYPE int]}|}                          |
   |            {30[DEGREE positive, TIME +                  ]}                             |
   |PRON-INT    [21:how]                                                                    |
   |FOCUS-INT   [30:often]                                                                  |
   |CHECK       [_SUBCAT-FRAME V-SUBJ]                                                      |
   |TNS-ASP     [MOOD indicative, PERF -_, PROG -_, TENSE past]                             |
 49[CLAUSE-TYPE int, PASSIVE -, VTYPE main                                                  ]
```

Figure 3.17: Original PARC f-structure for *How often did it appear?*

The subcategorization frame for the verb *to appear* requires a subject. Other features like CASE also do not get the right value as soon as subject and object are confused. In addition, the assignment of the ADJUNCT feature for *often* is misattached in the DCU f-structure and is included in the f-structures for PRON-INT and FOCUS-INT. Although this phenomenon is a structural problem, it still can not be regulated via a transfer rule.

Due to the facts mentioned above, interrogative and imperative sentences were only partly taken into account in the matching process and the focus was put on indicative sentences. After having described the transfer process of different syntactic phenomena, I will concentrate on the actual transfer in XLE in the following section.

3.5 Transfer process

The transfer was done in batch mode, which means that the rewrite rules were applied to the whole testfile, instead of transferring each sentence one by one. To transfer the whole file, the following command has to be typed in the shell:

```
transfer
--inStem /out/fstr
--outStem /out/final
--from 1
--to 429
--inMode fs_file
--outMode fs_file
--rules transfer_new.pl
```

The transfer command above passes the original f-structure through transfer by

- running the current set of transfer rules (`transfer_new.pl`)
- on the input file (`--inStem`)
- and writing the results to the output file (`--outStem`)

The mode defaults to `fs` and specifies the formats of the input and output files, in this case f-structures, therefore `fs`. If one wants to transfer just one f-structure and is already in the right directory, the command can be reduced to

```
transfer fstr145.pl final145.pl.
```

This also requires that XLE and the grammar rules are already loaded with the command `load-transfer-rules transfer_new.pl`. To display a transferred f-structure, the command `read-prolog-chart-graph final145.pl` is typed into the command line of the shell. As a result, four windows open (as has been explained in the XLE introduction), one among them displaying the transferred f-structure. Another very helpful command is `tdbg`, which allows the debugger to see which rules were applied for each sentence in the transfer process.

After explaining all steps through the experiment, starting with the DCU syntax output, going through the transfer process for different phenomena, I come to the discussion of the experiment results in the following chapter.

Chapter 4

Evaluation

To determine what features needed to be added or changed to create XLE-style f-structures, I used a testsuite of 430 short sentences. Some might argue that the number of test sentences is not sufficient to evaluate the system,

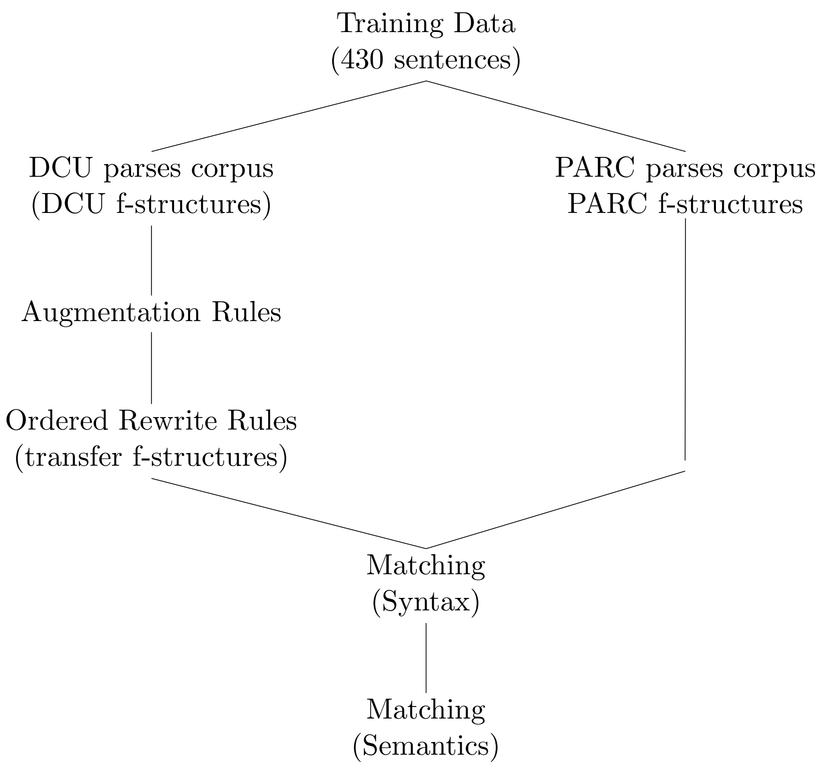


Figure 4.1: Outlay of the experiment

however, they cover different syntactic phenomena, among them extraposition, extraction, gerunds, and sentential subjects and different clause types (indicative, interrogative and imperative sentences) and therefore can be considered as being representative.

A schematic outlay of the experiment is provided in Figure 4.1. The test sentences in the training data were parsed by the XLE parser to obtain full f-structures. The same sentences were also run through the DCU parser, creating the f-structures to be transferred. In the next step, augmentation rules and ordered rewriting rules reformatted and transferred the DCU f-structures. The resulting f-structures were compared to those produced by the XLE parser and any difference was used to modify the ordered rewriting rules (therefore the XLE f-structures are used as a gold standard).

4.1 Evaluation Measures

The benefits of objective evaluation have encouraged many researches involved in machine translation (MT). The aim is to find reliable methods to evaluate MT systems, most of the methods proposed involve some kind of similarity score between the output of the MT system and a "reference" translation (Melamed et al. (2003)). Since the invention of BLEU (Papineni et al. (2002)) and the NIST metric (Doddington (2002)), a lot of attention has been given to automatic evaluation metrics for MT (Owczarzak et al. (2007)). Whereas BLEU and the NIST metric are limited to superficial comparison of word sequences between translated sentence and one or more reference sentences, there are other evaluation measures that are more sensible for this kind of experiment. To match the different f-structures and semantic representations here, I will use the evaluation measures *f-score, precision and recall.* They are most useful for this sort of experiment as they compare the features of a test set in relation to the features of a reference set.

Precision, Recall and F-Score of MT Precision and recall are widely used to evaluate the output of an MT system. When a set of "test" items Y (transferred DCU f-structures) is compared to a set of "reference" items

X (original PARC f-structures), precision (X|Y) = $\frac{|X \cap Y|}{Y}$ and recall (Y|X)= $\frac{|X \cap Y|}{X}$ (Melamed et al. (2003)) are measures to compare the output. Precision takes the absolute value of the feature matches between original PARC f-structures (reference set) and transferred DCU f-structures (test set) and divides this figure through the total number of features on the DCU side. Recall takes the absolute value of the matching features between between original PARC f-structures (reference set) and transferred DCU f-structures (test set) and divides it by the number of features in the PARC f-structures. The f-score is a measure for the accuracy of a test and is a weighted average of precision and recall.

Precision in this experiment is usually higher than recall: the number of features on the transferred DCU side is lower, therefore the denominator is lower and the fraction becomes higher. The other way around, the number of features on the PARC side is high, the denominator is higher and precision gets lower. The way it works for f-structure, it also works for the semantic representation, comparing semantic facts instead of f-structure features.

The matching figures are calculated for each pair of f-structure (test f-structure against reference f-structure), but can also be calculated over the whole testfile or parts of it. This is the task of the following section where I will focus on the actual matching figures for the different clause types and for the whole testfile. In 4.2 I will only match f-structures. To see if the transferred f-structures provide enough information for the PARC semantics, I will match the semantic output in section 4.3.

4.2 F-structure Matching

The matching between transferred DCU f-structures and original PARC f-structures can be done in batch-mode, similar to the transfer process. The command for this is:

```
triples match --matchMode best
--sourceMode fs_file
--sourceStem /out/final
--targetMode fs_file
```

```
--targetStem /parcout/final
--from 1 --to 429
```

The triple match command can be used in order to compare f-structures to each other. Source and target f-structure are converted into triples and then matched against each other.

If the command `matchMode` is set to `best`, then all source analyses are mapped to find the best match, instead of mapping only the selected ones. This is not of great importance here because both the DCU and the XLE side contain only one parse for each sentence, nevertheless it is the correct way. Other match options only calculate the average match of the different analyses or just map against the selected one.

The command above is the command for the batch-mode of the matching (all files with the beginning `final` of the `/out` directory are matched against the `final`-files in the `/parcout` directory, which fall in the rage from sentence number 1 to 429. To see which features in the f-structure are matched and which are not, one can attach the command `--diff` to the matching command. This can be of great help but provides too much information if over 400 f-structures are matched. Once the matching runs, XLE returns with information for every matched f-structure. This looks like the following:

```
(425 1 (33 34 42) 87 97 79)
(426 1 (28 34 35) 81 82 80)
(427 1 (15 15 15) 100 100 100)
(428 1 (15 15 15) 100 100 100)
(429 1 (8 11 15) 62 73 53)
```

The number on the left-hand side is the sentence number which is matched. The `1` means that there is only one optimal parse for this sentence. The last three figures in the row are the numbers for precision, recall and f-score. They are the focus of the following part, when the matching results for the transfer algorithm are presented.

F-structure matching of indicatives The matching of indicatives varies quite a bit, depending on how many proper nouns are included in the sentence. In my training data, a lot of expressions like *Eiffel Tower*, *President*

George W. Bush were included, which is why precision and recall are relatively low.

precision	recall	f-score
72.63	65.61	68.94

Figure 4.2: Matching results for indicatives with proper nouns

If we consider sentences that do not contain proper nouns (e.g. *I'll go* or *He laughed every third year*) we get the following numbers:

precision	recall	f-score
87.13	82.67	84.84

Figure 4.3: Matching results for indicatives without proper nouns

Quite a lot of clauses get a precision, recall and an f-score of 100, but this success is then outdone by issues concerning coordination and the right assignment of adjuncts.

F-structure matching of interrogatives The first 31 sentences of the training data are interrogative sentences. Matching the transferred interrogatives against the original PARC interrogatives gives the following results:

precision	recall	f-score
45.17	43.15	44.13

Figure 4.4: Matching results for interrogatives

I explained the reasons for the relatively low matching figures in detail above. Due to the lack of interrogative sentences in the DCU training data (WSJ corpus), these clauses get incorrect analyses. To get the correct ones, DCU would have to spend some time on making the structures initially closer to the PARC ones.

F-structure matching of imperatives The figures for the matching of imperatives are higher than those for interrogatives, as more features could be added and the analyses in general are closer to the PARC f-structures. The reason for the higher matching figures than for interrogatives is most likely the possibility to add subcategorization features.

precision	recall	f-score
61.83	49.10	54.74

Figure 4.5: Matching results for imperatives

If all this suffices for the semantics to understand the content of the transferred f-structures, we will match the semantic output for the f-structures in the following section.

4.3 Matching of the Semantic Representation

To quickly repeat the basic XLE system, the basic pipeline looks as follows:

text breaker (FST)
↓
tokenizer & morphologies (FST)
↓
syntax (XLE LFG)
↓
semantics (XFR ORDERED REWRITING)
↓
AKR (XFR ORDERED REWRITING)

Figure 4.6: Standard XLE pipeline

In order to get a feedback from the semantics if enough information is included in the f-structures for them to "understand" the content, I prepared a separate testfile with 66 queries and answers. These were parsed by both DCU and PARC parser, then the DCU f-structures were transferred by the ordered rewrite rules. To match the semantic output, both f-structures were fed into the semantic system and the semantic representation of transferred DCU output and original PARC output was then matched against each other. This step was done in order to see if a question-answering system would work on the basis of the stochastic output. An example for query and answer is shown below:

```
Although Mary likes vegetables she eats them raw.
Does Mary like vegetables?
```

The matching figures are the following:

precision	recall	f-score
64.04	60.27	62.10

Figure 4.7: Matching results for the semantic representation

These figures look quite good, especially because a lot of questions are involved here, and they naturally get lower matching results. Also, some of the sentences are also longer than those in the testsuite which proves that the transfer system also works on other test sets than the training data. The final step after matching the semantic output would be to see if the AKR can still "understand" the semantic representation of the stochastic input and could still answer queries like the one above. This hasn't been tested yet and could be part of the future work on the experiment.

4.4 Interim Summary

In this section I have shown that with the help of using the popular MT evaluation measures precision, recall and f-score, the hybridization of the XLE system by using stochastic DCU output is worth the effort. The matching results for indicative sentences depend on the occurrence of proper nouns like city names etc. These are not being dealt with on the DCU side, therefore the analysis is very different from the PARC analysis. However, this could be improved on the DCU side. If no proper noun is included in the sentence, the results look significantly better. Interrogatives and imperatives get lower matching results due to the structure of the training data (Wall Street Journal). Some effort has been put into building a QuestionBank which would be very beneficial to the results of this experiment.

I have then shown the matching results if transferred DCU f-structures are being fed into to the rule-based PARC semantics. The results are promising, especially taking into account the fact that a lot of questions are used here and the matching results are still quite high. Whether this system could be used for question-answering in a truly integrated system needs to be explored. In the following chapter I want to discuss the system in a little more detail, with a special focus on ambiguity and efficiency.

Chapter 5

Discussion

Given the initial success of the experiment, the next step is to build a truly integrated hybrid DCU-XLE system that can be run over large corpora and the results need to be compared with that of the standard XLE system. Of particular importance is the behavior of the DCU-XLE system in application contexts.

In this section, I first discuss the issues arising from the different treatment of ambiguity in the two systems. I then discuss efficiency: back-of-the-envelope calculations show that the two systems should be roughly similar in efficiency, but this remains to be tested empirically. Finally, I discuss how I would wish to deploy the DCU-XLE system once an integrated version exists.

5.1 Ambiguity

The XLE LFG grammar can efficiently produce multiple analyses, sometimes thousands of analyses, for a given sentence (Maxwell and Kaplan (1991)). A maximum entropy model is applied to the output of the grammar to rank the parses (Riezler et al. (2002)). An n-best subset of the parses is then passed to the semantics. The more parses that are passed forward, the more processing that the semantics and AKR rules must perform, although the impact of this is mitigated by the ability of the ordered rewrite system to operate on the packed structures produced by the XLE grammar (Crouch (2005)). In fact, the ordered rewrite system uses the same packing mechanism and code that the XLE parser does. For meaning sensitive applications, the n-best, instead of the single best parses are used in order to increase the chances that the correct parse is available.

For this experiment, I use the single parse produced by the DCU system. In theory, it would be possible to obtain ranked output from the DCU parser, e.g. by taking the n-best trees produced by the PTBG. In order for the semantics to operate on them efficiently, these parses would have to be packed. However, packing unpacked input can be difficult and inefficient. As such, the hybrid DCU-XLE approach seems best suited for applications and situations where a single parse provides sufficient information. Search, as opposed to question-answering, is one possible application of this type. In addition, as will be discussed below, even in heavily meaning sensitive applications, the DCU-XLE approach may be superior when the XLE grammar produces fragment parses.

5.2 Efficiency

The efficiency of the hybrid DCU-XLE approach was not systematically explored. The XLE system can process sentences in documents with an average of ∼20 words per sentence (e.g. Penn Treebank WSJ sentences) at less than a second from text to semantic output. Half of the time is spent on the syntax (i.e. creating the f-structure). Within the XLE LFG parser, the syntax time is roughly divided as: morphology (textbreaker and tokenizer) (4%), lexicon (6%), chart (25%), unifier (55%), completer (4%), solver (6%). The exact percentage of time depends on how many parses are passed forward to the semantics rules: when more parses are passed forward, the processing by the ordered rewriting slows.

XLE has a number of performance variables that can be set to trade speed for accuracy (Crouch et al. (2008)). The one-second-a-sentence results use relative aggressive settings with the result that ∼1.1% of the sentences time out or run out of memory. This version of the XLE grammar uses c-structure chart pruning to trim the context-free c-structure forest before unification.

C-structure pruning eliminates unlikely subtrees if (1) there is another subtree analysis available and (2) the subtree is significantly less probable than the most-probable subtree. The chart pruner uses a simple stochastic CFG model where the probability of a tree is the product of the probabilities of each of the rules used to form the tree, including the rules that lead to lexical items. The probability of a rule is basically the number of times that that particular form of the rule occurs in the training data divided by the number of times the rule's category occurs in the training data, plus a smoothing term. If a subtree's probability is lower than the best probability by a given factor, then the subtree is pruned. This approach ensures that there is always at least one tree and that only highly improbable subtrees are eliminated. The resulting c-structure forest is often still very large, but can be significantly smaller than the original one. Using c-structure pruning speeds the XLE parser by ∼40% for English, while maintaining accuracy.

The DCU parser runs with a similar level of efficiency and hence should not significantly change the speed of the system. In parsing the British National Corpus (BNC) (Wagner et al. (2007)), which has an average sentence length of 18. words, the PTBG, annotation, and unification took an average of 1.48 seconds per sentence (extremely long sentences take much longer to parse, as is also the case for the XLE parser). This longer per-sentence parse time is somewhat misleading because the parser in the DCU experiment in Wagner et al. (2007) was configured to provide analyses for all sentences, no matter how long, complex, or grammatical; if the occasional missed analysis is acceptable for a given application, more efficient processing settings can be used. Regardless, the longer parse time could be balanced out by the more connected f-structures produced for out-of-coverage sentences for the XLE parser.

The ordered rewrite rules used to map from the DCU output to the semantics input are relatively few in number compared to those used in the semantics and AKR stack. As a result, they should add a negligible amount of time to the processing.

5.3 An Integrated System

In order to truly test efficiency and the results of the loss of ambiguity, an integrated system would have to be built. For the purposes of this initial experiment, DCU provided the structures for the sentences which we used in the experiment. They also ran their augmentation rules over their structures in order to provide more complete output.

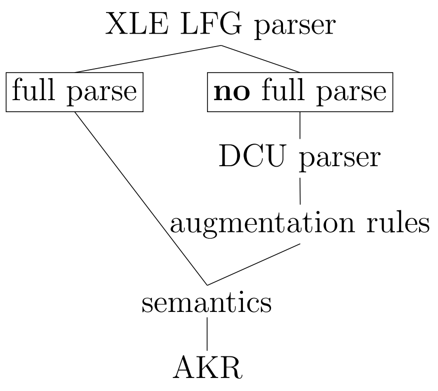


Figure 5.1: Coverage-sensitive DCU-XLE system

An integrated system would need to call the DCU parser directly. The DCU f-structure output would then be reformatted to serve as input to the ordered rewrite system (probably through a call to prolog, similar to that used in the experimental system). The ordered rewrite augmentation rules described in this thesis could then be added in order to take into account the initial augmentation step currently done by DCU. It is likely that a single stack of ordered rewrite rules would be more efficient than the current DCU+XLE augmentation system, largely because some changes could be done in one step instead of two.

Having an integrated DCU-XLE system would allow for experiments on efficiency and accuracy that are currently only estimated. Of particular interest is building a system that would use the current XLE system for in-coverage sentences but the DCU-XLE system for out-of-coverage, fragmenting sentences (for most domains ∼15% of the input sentences fragment). The fragment structures have well-formed sub-f-structures, but these sub-f-structures are knitted together in a FIRST-REST structure[1], thereby often loosing the overall coherence of the sentence. The more gradual degradation of the DCU parser allows for complete and coherent parses of sentences that are out of coverage of the XLE LFG grammar. Connected structures are inherently more suited for semantic interpretation and hence meaning-sensitive applications. The XLE LFG grammar uses a two pass system to create fragment parses: first it tries with the regular grammar and if no spanning analysis can be created, then the fragment grammar is used. With the DCU-XLE hybrid system, instead of calling on the fragment grammar for out-of-coverage sentences, the DCU grammar would be used, as shown in Figure 5.1.

I examined a small set of newspaper sentences that fragmented in the XLE grammar to see whether the connected DCU f-structures were more accurate from the perspective of predicate argument structure.[2] In each case, the DCU structure had a more connected structure. The XLE parser averaged 2.1 f-structure chunks per sentence and .7 tokens.[3] In a third of the sentences, at least one verb was incorrectly identified by the XLE parser; the DCU parser never incorrectly identified a verb. As verbs are key for the core meaning of the sentence, this is a significant advantage. The XLE analyses missed an average of 3.7 core dependencies per fragmented sentence, where core dependencies are ones that occur between two f-structures containing PREDs, e.g. SUBJ, OBJ, ADJUNCT, SPEC. The DCU parser missed an average of 1.3 core depedencies per sentence; most of these were due to the fact that hyphenated forms are always treated as a single PRED, although there were also mis-attachments of adjuncts and mis-analyses of arguments.

[1]If something like *Mary ! hops ! skips.* is parsed in the English grammar, it will fragment. The fragments are in a first-rest structure in the f-structure (these are fairly efficient to compute and record the linear order of fragments). The terminology comes from computer science where lists can divided into the first element and the rest of the elements; then the rest of the elements is divided into the first element and the rest of the elements, and so on.

[2]In extremely rare cases (Cahill et al. (2008)), the DCU system cannot produce a connected f-structure and the result is multiple f-structure chunks, similar to the XLE parser fragment analyses.

[3]Tokens occur when a word cannot be part of any well-formed f-structure chunk; instead, the f-structure just records the surface form of the word.

This small fragmentation analysis was biased in favor of the DCU parser in that it used Penn Treebank WSJ data, which the DCU parser is optimized for. However, a few samples of non-financial blog sentences that fragment in the XLE parser also show better connected structures with the DCU parser. As such, I'm optimistic that for fragmented sentences, the DCU parser could provide better connected input to the semantics than the XLE parser.

Chapter 6

Conclusion

This thesis reported on an experiment to use a stochastic parser (the probabilistic DCU parser) that produces f-structures in the place of a rule-based LFG parser (the XLE parser) as the input to a semantic parser. The f-structures were augmented by a set of ordered rewrite rules that use the same mechanism as the rules that create the semantic structure. When matching DCU and PARC output on the f-structure level, the results are promising in that the transferred f-structures can be used by the semantics to produce well-formed structures. This success provides the opportunity to build hybrid systems using different grammar versions depending on their ability to parse the input data. The disadvantages of the DCU parser, which assigns less features to the f-structures, can be overcome by the ordered rewrite rules presented in this thesis. The benefits like the time-consuming effects and the use of already existing resources make the probabilistic parser an attractive extension to the rule-based system the way it is used at PARC.

Although there are some drawbacks, especially with interrogative and imperative clauses, the system as a whole achieves promising results, especially since the lack of training data for these sentences might be smoothed out, e.g. by using a QuestionBank.

Our primary interest is in using the DCU parser for out-of-coverage sentences. As more researchers wish to build semantic processing on top of ParGram-style grammars, hybrid systems can be built using DCU grammars exclusively for the syntactic processing step (e.g. for Spanish for which there is a DCU ParGram grammar but no XLE one) and save a lot of time building a rule-based system for the syntax. Therefore this thesis provides another justification for the use of statistics in natural language applications, however one side cannot exist without the other and interaction is needed in order to achieve the best results.

Bibliography

Kenneth Beesley and Lauri Karttunen. *Finite State Morphology.* CSLI Publications, Stanford, CA, 2003.

Daniel G. Bobrow, Bob Cheslow, Cleo Condoravdi, Lauri Karttunen, Tracy Holloway King, Rowan Nairn, Valeria de Paiva, Charlotte Price, and Annie Zaenen. PARC's Bridge and Question Answering System. In *Grammar Engineering Across Frameworks*, pages 46–66. CSLI Publications, 2007.

Joan Bresnan and Ronald M. Kaplan. *The Mental Representation of Grammatical Relations.* Cambridge, MA: The MIT Press, 1982.

Michael Burke. *Automatic Treebank Annotation for the Acquisition of LFG Resources.* PhD thesis, Dublin City University, 2006.

Miriam Butt, Tracy Holloway King, Maria-Eugenia Niño, and Frédérique Segond. *A Grammar Writer's Cookbook.* CSLI Publications, 1999.

Miriam Butt, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. The Parallel Grammar Project. In *Proceedings of COLING2002, Workshop on Grammar Engineering and Evaluation*, pages 1–7, 2002.

Aoife Cahill. *Parsing with Automatically Acquired, Wide-Coverage, Robust, Probabilistic LFG Approximations.* PhD thesis, School of Computing, Dublin City University, Dublin 9, Ireland, 2004.

Aoife Cahill and Mairead Mccarthy. Automatic Annotation of the Penn Treebank with LFG F-Structure Information. In *Proceedings of the LREC Workshop on Linguistic Knowledge Acquistion and Representation: Bootstrapping Annotated Language Data*. LREC Workshop on Linguistic Knowledge Acquisiton and Representation, Bootstrapping, Las Palmas, Canary Islands, 2002.

Aoife Cahill, Mairéad McCarthy, Josef van Genabith, and Andy Way. Parsing with PCFGs and Automatic F-Structure Annotation. In *LFG02 Proceedings*, pages 76–95, Stanford, CA, 2002. CSLI Publications.

Aoife Cahill, Martin Forst, Michael Burke, Mairéad McCarthy, Ruth O'Donovan, Christian Rohrer, Josef van Genabith, and Andy Way. Treebank-Based Acquisition of Multilingual Unification Grammar Resources. *Journal of Research on Language and Computation; Special Issue on "Shared Representations in Multilingual Grammar Engineering"*, pages 247–279, 2005.

Aoife Cahill, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith, and Andy Way. Wide-Coverage Deep Statistical Parsing Using Automatic Dependency Structure Annotation. *Computational Linguistics*, 34(1):81–124, 2008.

Noam Chomsky. *The Minimalist Program*. MIT Press, 1995.

Dick Crouch. Packed rewriting for mapping semantics to KR. In *Proceedings of the International Workshop on Computational Semantics*, 2005.

Dick Crouch and Tracy Holloway King. Semantics via F-structure Rewriting. In *LFG06 Proceedings*. CSLI Publications, 2006.

Dick Crouch, Mary Dalrymple, Ron Kaplan, Tracy King, John Maxwell, and Paula Newman. XLE Documentation. http://www2.parc.com/isl/groups/nltt/xle/doc/, 2008.

Mary Dalrymple. *Lexical functional grammar*. Academic Press, 2001.

George Doddington. Automatic Evaluation of MT Quality using N-gram Co-occurence Statistics. In *Proceedings of Human Laanguage Technology Conference*, pages 138–145, 2002.

Jennifer Foster. Treebanks gone bad: Parser evaluation and retraining using a treebank of ungrammatical sentences. *International Journal on Document Analysis and Recognition*, 10:129–145, 2007.

Anette Frank, Tracy Holloway King, Jonas Kuhn, and John Maxwell. Optimality Theory Style Constraint Ranking in Large-scale LFG Grammars. In *LFG98 Proceedings*. CSLI Publications, 1998.

Annette Hautli. Hybridization of the XLE pipeline - Die Umwandlung von DCU F-Strukturen in PARC F-Strukturen. DGfS-CL Postersession 2008, Bamberg, 2008.

John Judge, Michael Burke, Aoife Cahill, Ruth O'Donovan, Josef van Genabith, and Andy Way. Strong Domain Variation and Treebank-Induced LFG Resources. In *LFG05 Proceedings*, pages 186–204. CSLI Publications, 2005.

John Judge, Aoife Cahill, and Josef van Genabith. QuestionBank: Creating a Corpus of Parse-Annotated Questions. In *Proceedings of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics*, pages 497–504, 2006.

Tracy Holloway King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ron Kaplan. The PARC700 Dependency Bank. In *Proceedings of the EACL03: 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*, 2003.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotative predicate argument structure. In *ARPA Human Language Technology Workshop*, 1994.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June 1993.

John Maxwell and Ron Kaplan. A method for disjunctive constraint satisfaction. *Current Issues in Parsing Technologies*, 1991.

I. Dan Melamed, Ryan Green, and Joseph P. Turian. Precision and recall of machine translation. In *Proceedings of HLT/NAACL*, 2003.

Ruth O'Donovan, Michael Burke, Aoife Cahill, Josef van Genabith, and Andy Way. Large-Scale Induction and Evaluation of Lexical Resources from the Penn-II and Penn-III Treebanks. *Computational Linguistics*, 31, 2005.

Karolina Owczarzak, Josef van Genabith, and Andy Way. Labelled dependencies in Machine translation Evaluation. In *Proceedings of the Second Wokrshop on Statistical Machine Translation*, pages 104–111, 2007.

Martha Palmer, Dan Gildea, and Paul Kingsbury. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31: 71–106, 2005.

Kishore Papineni, Salim Roukos, Todd Ward, and WeiJing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Preceedings of Association for Computational Linguistics Conference*, pages 311–318, 2002.

Carl Pollard and Ivan Sag. *Head-driven Phrase Structure Grammar*. CSLI Publications, Stanford, CA, 1994.

Stefan Riezler, Tracy Holloway King, Ron Kaplan, Dick Crouch, John Maxwell, and Mark Johnson. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 271–278, 2002.

Joachim Wagner, Djamé Seddah, Jennifer Foster, and Josef van Genabith. C-Structures and F-Structures for the British National Corpus. In *LFG07 Proceedings*. CSLI On-line Publications, 2007.