Parsing with PCFGs and Automatic F-Structure Annotation

Aoife Cahill   Mairéad McCarthy   Josef van Genabith   Andy Way
Computer Applications, Dublin City University, Ireland
{ acahill, mccarthy, josef, away }@computing.dcu.ie

Proceedings of the LFG02 Conference

National Technical University of Athens, Athens

Miriam Butt and Tracy Holloway King (Editors)

2002

CSLI Publications

http://csli-publications.stanford.edu/

Abstract

The development of large coverage, rich unification- (constraint-) based grammar resources is very time consuming, expensive and requires lots of linguistic expertise. In this paper we report initial results on a new methodology that attempts to partially automate the development of substantial parts of large coverage, rich unification- (constraint-) based grammar resources. The method is based on a treebank resource (in our case Penn-II) and an automatic f-structure annotation algorithm that annotates treebank trees with proto-f-structure information. Based on these, we present two parsing architectures: in our pipeline architecture we first extract a PCFG from the treebank following the method of [Charniak, 1993; Charniak, 1996], use the PCFG to parse new text, automatically annotate the resulting trees with our f-structure annotation algorithm and generate proto-f-structures. By contrast, in the integrated architecture we first automatically annotate the treebank trees with f-structure information and then extract an annotated PCFG (A-PCFG) from the treebank. We then use the A-PCFG to parse new text to generate proto-f-structures. Currently our best parsers achieve more than 81% f-score on the 2400 trees in section 23 of the Penn-II treebank and more than 60% f-score on gold-standard proto-f-structures for 105 randomly selected trees from section 23.

1 Introduction

The development of large coverage, rich unification- (constraint-) based grammar resources is very time consuming, expensive and requires considerable linguistic expertise [Butt et al, 1999; Riezler et al, 2002].

In this paper we report initial results on a new methodology that attempts to partially automate the development of substantial parts of large coverage, rich unification- (constraint-) based grammar resources.

A large number of researchers (cf. [Charniak, 1996; Collins, 1999; Hockenmaier and Steedman, 2002]) have developed parsing systems based on the Penn-II [Marcus et al, 1994] treebank resource but to the best of our knowledge, to date, none of them have attempted to semi-automatically derive large coverage, rich unification (constraint) grammar resources.

Our method is based on the Penn-II treebank resource and an automatic Lexical-Functional Grammar [Kaplan and Bresnan, 1982; Bresnan, 2001; Dalrymple, 2001] f-structure annotation algorithm that annotates treebank trees with proto-f-structure information.

Based on these, we present two parsing architectures: in our *pipeline* architecture we first extract a PCFG from the treebank following the method of [Charniak, 1993; Charniak, 1996], use the PCFG to parse new text, automatically annotate the resulting trees with our f-structure annotation algorithm and generate proto-f-structures. By contrast, in the *integrated* architecture we first automatically annotate the treebank trees with f-structure information and then extract an annotated PCFG (A-PCFG) from the treebank. We then use the A-PCFG to parse new text to generate proto-f-structures.

The paper is structured as follows: first, we briefly describe the automatic f-structure annotation algorithm and proto-f-structure representations. Second, we introduce our two parsing architectures: the pipeline model and the integrated model. Third, we evaluate our approaches quantitatively and qualitatively in terms of f-structure fragmentation and precision and recall results against trees and manually encoded gold standard f-structures. We also compare our results with a PCFG resulting from a parent transformation as discussed in [Johnson, 1999] and we conduct a number of thresholding grammar compaction experiments [Krotov et al, 1998]. We briefly compare our approach with that of [Riezler et al, 2002]. Fourth, we outline further work on "proper" (rather than "proto-") f-structures and finally, we summarise and conclude.

2 An Automatic F-Structure Annotation Algorithm

In this section we describe our automatic f-structure annotation algorithm by means of an example. Consider the following Penn-II treebank tree:

```
(S (NP-SBJ (DT The)
           (NN investment)
           (NN community))
   (, ,)
   (PP (IN for)
       (NP (CD one)))
   (, ,)
   (VP (VBZ has)
       (VP (VBN been)
           (VP (VBG anticipating)
               (NP (DT a)
                   (JJ speedy)
                   (NN resolution)))))
   (. .))
```

In LFG this tree would be associated with an f-structure of the following form representing abstract syntactic information approximating to predicate-argument-modifier (or: rich dependency-) structure:[1]

[1]Note that our f-structures are more hierarchical with XCOMP functions for temporal and aspectual auxiliaries than e.g. the f-structures given in [Butt et al, 1999]. The repetition of the subject `NP`

```
subj : spec : det : pred : the
       headmod : 1 : num : sg
                     pers : 3
                     pred : investment
       num : sg
       pers : 3
       pred : community
adjunct : 1 : obj : pred : one
              pred : for
xcomp : subj : spec : det : pred : the
               headmod : 1 : num : sg
                             pers : 3
                             pred : investment
               num : sg
               pers : 3
               pred : community
        xcomp : subj : spec : det : pred : the
                       headmod : 1 : num : sg
                                     pers : 3
                                     pred : investment
                       num : sg
                       pers : 3
                       pred : community
                obj : spec : det : pred : a
                      adjunct : 2 : pred : speedy
                      pred : resolution
                      num : sg
                      pers : 3
                participle : pres
                pred : anticipate
        pred : be
        tense : past
pred : have
tense : pres
```

F-structures are associated with strings and their parse trees (c-structures) in terms of annotating nodes in parse trees (and hence the corresponding CFG rules) with f-structure annotations (in simple cases, attribute value structure equations; more generally, expressions in a full equality logic including disjunction, negation etc. [Johnson, 1988]). The f-structure in our example would be induced by the following annotated CFG rules

is due to a reentrancy ($\uparrow$ SUBJ = $\uparrow$ XCOMP SUBJ) annotation in the VP rule.

$$
\begin{array}{llccc}
\text{S} & \rightarrow & \text{NP-SBJ} & \text{PP} & \text{VP} \\
 & & \uparrow\texttt{subj}=\downarrow & \downarrow\in\uparrow\texttt{adjn} & \uparrow=\downarrow \\
 & & & & \\
\text{NP-SBJ} & \rightarrow & \text{DT} & \text{NN} & \text{NN} \\
 & & \uparrow\texttt{spec}=\downarrow & \downarrow\in\uparrow\texttt{headmod} & \uparrow=\downarrow \\
 & & & & \\
\text{PP} & \rightarrow & \text{IN} & \text{NP} & \\
 & & \uparrow=\downarrow & \uparrow\texttt{obj}=\downarrow & \\
 & & & & \\
\text{NP} & \rightarrow & \text{CD} & & \\
 & & \uparrow=\downarrow & & \\
 & & & & \\
\text{VP} & \rightarrow & \text{VBZ} & \text{VP} & \\
 & & \uparrow=\downarrow & \uparrow\texttt{xcomp}=\downarrow & \\
 & & & \uparrow\texttt{subj}=\downarrow\texttt{subj} & \\
 & & & & \\
\text{VP} & \rightarrow & \text{VBN} & \text{VP} & \\
 & & \uparrow=\downarrow & \uparrow\texttt{xcomp}=\downarrow & \\
 & & & \uparrow\texttt{subj}=\downarrow\texttt{subj} & \\
 & & & & \\
\text{VP} & \rightarrow & \text{VBG} & \text{NP} & \\
 & & \uparrow=\downarrow & \uparrow\texttt{obj}=\downarrow & \\
 & & & & \\
\text{NP} & \rightarrow & \text{DT} & \text{JJ} & \text{NN} \\
 & & \uparrow\texttt{spec}=\downarrow & \downarrow\in\uparrow\texttt{adjn} & \uparrow=\downarrow
\end{array}
$$

together with equations resulting from lexical entries.

The question is how can we construct such an LFG grammar? Traditionally, LFG (and HPSG [Pollard and Sag, 1994]) grammatical resources have been constructed manually. For large coverage and rich grammars that scale to the Penn-II treebank data (about 1 million words of WSJ text) [Butt et al, 1999; Riezler et al, 2002], this can easily accumulate to a person decade.

Is there any way of (at least partially) automating the construction of large coverage, rich, unification-based grammatical resources? From the large literature on probabilistic parsing it is clear that given a treebank we can easily extract a probabilistic CFG (following the method of [Charniak, 1993; Charniak, 1996]. Indeed, such grammars are at the heart of many probabilistic parsing approaches such as [Charniak, 1993; Charniak, 1996; Collins, 1999; Hockenmaier and Steedman, 2002], to mention but a few. However, to the best of our knowledge, to date, there have been no attempts at semi-automatically deriving large coverage, rich unification- (constraint-) based grammar resources.

In order to obtain a unification grammar, we need the functional annotations (the f-structure constraints) in addition to the CFG rules. Of course, theoretically, we could annotate the CFG rules extracted from a treebank manually. For the Penn-II treebank, however, we would be required to manually annotate more than 19,000 extracted rule types resulting in at least 2 person years of annotation work (on a very conservative estimate with 10 minutes annotation time for each rule type and no time budgeted for testing, comparison, improvement and verification cycles).

2.1 Previous Work

A number of researchers have addressed automatic functional structure identification or annotation in CFG trees or, alternatively, direct transformation of trees into functional structures.

To date, we can distinguish three different types of automatic f-structure annotation architectures:[2]

- annotation algorithms,
- regular expression based annotation,
- flat, set-based tree description rewriting.

All approaches are based on exploiting categorial and configurational information encoded in trees. Some also exploit the Penn-II functional annotations.[3]

Annotation algorithms come in one of two forms. They may

- *directly* (recursively) transduce a treebank tree into an f-structure – such an algorithm would more appropriately be referred to as a tree to f-structure transduction algorithm;
- *indirectly* (recursively) annotate CFG treebank trees with f-structure annotations from which an f-structure can be computed by a constraint solver.

As was recently pointed out to us by Shalom Lappin (p.c.), the earliest approach to automatically identify SUBJ, OBJ etc. nodes in CFG trees structures is probably [Lappin et al, 1989]. Their algorithm *identifies* nodes in CFG trees (output of the PET parser) corresponding to grammatical functions to facilitate the statement of

[2]These have all been developed within an LFG framework and although we refer to them as automatic f-structure annotation architectures, they could equally well be used to annotate treebanks with e.g. HPSG typed feature-structures [Pollard and Sag, 1994] or Quasi-Logical Form (QLF) [Liakata and Pulman, 2002] annotations.

[3]Note that apart from SBJ and LGS, functional annotations or tags in the Penn-II treebank do not provide LFG type predicate-argument style annotations but semantically classify e.g. modifying PP constituents as TMP (temporal), LOC (locative) etc. modifiers.

transfer rules in a machine translation project. It does not construct f-structures (or other attribute-value structures) although it could easily form the basis of such an algorithm for trees generated by the PET parser.

The first *direct* automatic f-structure annotation algorithm we are aware of is unpublished work by Ron Kaplan (p.c.) from 1996. Kaplan worked on automatically generating f-structures from the ATIS corpus to generate data for LFG-DOP applications. The approach implements a direct tree to f-structure transduction algorithm, which walks through the tree looking for different configurations (e.g. `NP` under `S`, 2nd `NP` under `VP`, etc.) and "folds" or "bends" the tree into the corresponding f-structure.

A regular expression-based, *indirect* automatic f-structure annotation methodology is described in [Sadler et al, 2000]. The idea is simple: first, the CFG rule set is extracted from the treebank (fragment); second, regular-expression based annotation principles are defined; third, the principles are automatically applied to the rule set to generate an annotated rule set; fourth, the annotated rules are automatically matched against the original treebank trees and thereby f-structures are generated for these trees. Since the annotation principles factor out linguistic generalisations, their number is much smaller than the number of CFG treebank rules. In fact, the regular expression-based f-structure annotation principles constitute a principle-based LFG c-structure/f-structure interface.

In a companion paper, [Frank, 2000] develops an automatic annotation method that in many ways is a generalisation of the regular expression-based annotation method. The idea is again simple: trees are translated into a flat set representation format in a tree description language and annotation principles are defined in terms of rules employing a rewriting system originally developed for transfer-based machine translation architectures. In contrast to [Sadler et al, 2000] which applies only to "local" CFG rule contexts, [Frank, 2000] can consider arbitrary tree fragments. Secondly, it can be used to define both order-dependent cascaded and order-independent annotation systems. [Liakata and Pulman, 2002] have recently developed a similar approach to map Penn-II trees to QLFs.

The approaches detailed in [Sadler et al, 2000; Frank, 2000] and compared in [Frank et al, 2002] are proof-of concept and operate on small subsets of the AP and Susanne corpora.[4]

2.2 A New Automatic Annotation Algorithm

In our more recent research [Cahill et al, 2002a; Cahill et al, 2002b], we have developed an algorithmic indirect annotation method for the $>$ 49.000 parse annotated

[4]This is not to say that these approaches cannot be scaled to a complete treebank!

strings in the WSJ section of the Penn-II treebank.

The algorithm is implemented as a recursive procedure (in Java) which annotates Penn-II treebank tree nodes with f-structure information. The annotations describe what we call "proto-f-structures". Proto-f-structures

- encode basic predicate-argument-modifier structures;
- interpret constituents locally (i.e. do not resolve long-distance dependencies or "movement" phenomena encoded as traces in the Penn-II trees);
- may be partial or unconnected (the method is robust: in case of missing annotations a sentence may be associated with two or more unconnected f-structure fragments rather than a single complete f-structure).

Even though the method is encoded in the form of an annotation algorithm, we did not want to completely hardwire the linguistic basis for the annotation into the procedure. In order to support maintainability and reusability of the annotation algorithm and the linguistic information encoded within, the algorithm is designed in terms of three main components thatare applied in sequence:

$$\boxed{\text{L/R Context APs}} \quad \Rightarrow \quad \boxed{\text{Coordination APs}} \quad \Rightarrow \quad \boxed{\text{Catch-All APs}}$$

L/R Context Annotation Principles are based on a tripartition of the daughters of each local tree (of depth one, i.e. of CFG rules) into a prefix, head and suffix sequence. We automatically transform the Penn-II trees into head-lexicalised trees by adapting the rules of [Magerman, 1994; Collins, 1999]. For each LHS in the Penn-II CFG rule types we construct an annotation matrix. The matrix encodes information on how to annotate CFG node types in the left (prefix) and right (suffix) context. Table 1 gives a simplified matrix for NP rules.

NP	left context	head	right context
subcat	DT,CD: $\uparrow$spec=$\downarrow$	NN,NNS,NP: $\uparrow$=$\downarrow$	…
non-sub	ADJP: $\downarrow\in\uparrow$adjn NN,NNS,NP: $\downarrow\in\uparrow$headmod …		SBAR,VP: $\uparrow$relmod=$\downarrow$ PP: $\downarrow\in\uparrow$adjn NN,NNS,NP: $\uparrow$app=$\downarrow$

Table 1: Simplified, partial annotation matrix for NP rules

For each LHS category, the annotation matrices are populated by analysing the most frequent rules types such that the token occurrence of the rule types in the corpus covers at least 85%. To give an example, this means that instead of looking at $>$ 6,000 NP rule types in the Penn-II corpus, we only look at the 102 most frequent ones to populate the NP annotation matrix.

To keep L/R context annotation principles simple and perspicuous, they only apply if the local tree does not contain coordination. Like and unlike coordinate structures are treated by the second component of our annotation algorithm, Finally, the algorithm has a catch-all and clean-up component. Lexical information is supplied via macros associated with each pre-terminal tag type.

The automatic annotation algorithm generates the following annotations. The annotations are collected, sent to a constraint solver and the f-structure shown before is generated.

```
(S
   (NP-SBJ[up-subj=down]
      (DT[up-spec:det=down] The[up-pred='the'])
      (NN[down-elem=up:headmod]
         investment[up-pred='investment',up-num=sg,up-
pers=3])
      (NN[up=down]
         community[up-pred='community',up-num=sg,up-
pers=3]))
   (, ,)
   (PP[down-elem=up:adjunct]
      (IN[up=down] for[up-pred='for'])
      (NP[up-obj=down]
         (CD[up=down] one[up-pred='one'])))
   (, ,)
   (VP[up=down]
      (VBZ[up=down]
          has[up-pred='have',up-tense=pres])
      (VP[up-xcomp=down,up-subj=down:subj]
         (VBN[up=down] been[up-pred='been',up-tense=past])
         (VP[up-xcomp=down,up-subj=down:subj]
            (VBG[up=down]
                anticipating[up-pred='anticipate',up-
participle=pres])
            (NP[up-obj=down]
               (DT[up-spec:det=down] a[up-pred='a'])
               (JJ[down-elem=up:adjunct] speedy[up-
pred='speedy'])
               (NN[up=down]
                  resolution[up-pred='resolution',up-
num=sg,up-pers=3])))))
   (. .))
```

Annotation coverage is measured in terms of f-structure fragmentation. The method is robust and in case of missing annotations may deliver unconnected f-structure fragments for a tree. Annotation accuracy is measured against a manually constructed gold-standard set f-structures for 105 trees randomly selected form Section 23 of the Penn-II treebank.[5]

# f-str.	[Cahill et al, 2002a]		[Cahill et al, 2002b]		current	
frags	# sent	percent	# sent	percent	# sent	percent
0	2701	5.576	166	0.343	120	0.25
1	38188	78.836	46802	96.648	48304	99.75
2	4954	10.227	387	0.799	0	0
3	1616	3.336	503	1.039	0	0
4	616	1.271	465	0.960	0	0
5	197	0.407	70	0.145	0	0
6	111	0.229	17	0.035	0	0
7	34	0.070	8	0.017	0	0
8	12	0.024	6	0.012	0	0
9	6	0.012	0	0	0	0
10	4	0.008	0	0	0	0
11	1	0.002	0	0	0	0

Table 2: Automatic proto-f-structure annotation fragmentation results

Table 2 shows the progress we have made over the last 6 months. Initially, 78.836% of the trees in the Penn-II treebank were associated with a single complete proto-f-structure with quite a number of trees having more than one proto-f-structure fragments and 2701 trees failing to get an f-structure because of inconsistent annotations. Currently our automatic annotation algorithm associates 99.75% of the trees with a complete (unfragmented) proto-f-structure while 120 trees do not receive any proto-f-structure.

Table 3 reports the quality of the f-structures generated in terms of precision and recall against our manually encoded gold-standard f-structures. We currently achieve P&R results of 0.94 and 0.87 for preds-only proto-f-structures.[6]

[5] Our gold-standard f-structures are available for inspection at `http://www.computing.dcu.ie/~away/Treebank/treebank.html`.

[6] Preds-only f-structures only show paths ending in a PRED feature.

	[Cahill et al, 2002b]		current	
	All annotations	Preds-only	All annotations	Preds-only
Precision	0.95	0.94	0.93	0.94
Recall	0.94	0.89	0.90	0.87

Table 3: Precision and Recall on descriptions of proto-f-structures

3 Two Parsing Architectures

Once we have the annotation algorithm and the annotated version of the Penn-II treebank, we can parse new text into trees and f-structures in two ways:

In our *pipeline* architecture we first extract a PCFG (probabilistic context free grammar) from the unannotated version of the Penn-II treebank and use this to parse new text. We then take the most probable tree associated with a string and send it to our automatic annotation algorithm. The algorithm annotates the tree with f-structure equations. We collect the equations and send them to a constraint solver to generate an f-structure:

$\boxed{\text{Treebank}} \rightarrow \boxed{\text{PCFG}} \rightarrow \text{text} \rightarrow \boxed{\text{Trees}} \rightarrow \text{f-str ann.} \rightarrow \boxed{\text{F-Str}}$

In our *integrated* architecture we first annotate the Penn-II treebank trees with f-structure information using our automatic annotation algorithm and then extract an annotated PCFG (A-PCFG) from the annotated treebank. We treat strings consisting of CFG categories followed by one or more f-structure annotations, e.g. `NP[up-subj=down]`, as monadic categories. The effect of this is that the rules extracted in the A-PCFG will be different and will be associated with different probabilities compared to the simple PCFG rules extracted for the pipeline model. For instance, the A-PCFG distinguishes between subject and object NPs whereas the PCFG does not. We then parse text with these annotated rules and pick again the tree with the highest probability. We then collect the f-structure annotations from this tree and send them to a constraint solver to generate an f-structure:

$\boxed{\text{Treebank}} \rightarrow \text{f-str ann.} \rightarrow \boxed{\text{A-PCFG}} \rightarrow \text{text} \rightarrow \boxed{\text{A-Trees}} \rightarrow \boxed{\text{F-Str}}$

We employ the following pre-processing steps to extract both the PCFG and the A-PCFG from the treebank: every tree is associated with a `Root` category node; we eliminate empty productions; we remove unary branches percolating daughter category information up in the tree and finally we annotate auxiliary verbs with an `Aux` category but only in the case where there is a sister `VP` node somewhere to the right of the `Aux` node.

Our grammars are extracted from sections 01 – 21 in the WSJ part of the Penn-II treebank and we measure our results against the held out section 23. All results are for sentences with length less than or equal to 40. We assume correct tagging of the roughly 2400 test sentences in section 23, i.e. we parse the tag sequences for those sentences given by the Penn-II treebank.

Our parsing experiments are based on a Java implementation of a CYK parser. For parsing, the grammar has to be transformed into Chomsky Normal Form (with binary branching productions). After parsing, the output trees are retransformed into the possibly n-ary branching treebank trees without loss of information. The parser is efficient ($O(n^3)$) and returns the single most probable tree.

We deliberately decided to use simple PCFG-based parsing technology in our experiments. The reason is that we want to be able to construct a system suitable for on-line parsing (this is not always possible with log-linear models as they may require unpacking of ambiguities). Mathematically speaking, however, this means that we do not use proper probability models. The reason is simple: PCFG technology is based on independence assumptions. The probability of a tree is the product of the probabilities of the productions in the tree (as each CFG production is assumed to be independent). What can happen in our model is that the parser returns the most probable tree but the f-structure equations generated for this tree (in the pipeline or the integrated architecture) are inconsistent and the constraint resolver cannot generate an f-structure. In such a case the probability mass associated with this tree is lost. Based on our experiments this case is extremely rare (less than 0.1%) so that the advantages of our engineering approach (speed and the ability to construct on-line applications) outweigh the disadvantages.

In order to evaluate our parsing results, we measure precision and recall using `evalb` on the 2400 trees in section 23. In order to evaluate the f-structures we measure f-structure fragmentation and precision and recall against manually encoded f-structures for 105 randomly extracted sentences from section 23.

We conducted a number of experiments. We extracted a simple PCFG for the pipeline model. We extracted an annotated A-PCFG for the integrated model. The integrated model with its f-structure annotations on CFG categories allows us to distinguish between subject and object `NPs`, for example, and associates different probabilities with rules expanding such `NPs`. A similar effect can be achieved in terms of a simple parent transformation on treebank trees: every daughter node receives an additional annotation involving its mother CFG category. An `NP-S`, e.g., is a an `NP` under `S` (i.e. a subject `NP` in English) while an `NP-VP` is a an `NP` under `VP` (i.e. an object `NP`). This approach (attributed to Charniak) has been studied extensively in [Johnson, 1999]. In our experiments we compare the parent transformation PCFG+P with our f-structure annotation pipeline and integrated approach. Finally, we conducted a number of experiments with simple thresholding grammar compaction techniques [Krotov et al, 1998]. Our results are summarised in the following tables.

		Pipeline		Integrated	
		PCFG	PCFG+P	A-PCFG	A-PCFG+P
Full Grammar	# Rules	19439	30026	29216	35815
	F-Score Labelled	77.37	80.49	81.26	81.18
	F-Score Unlabelled	79.89	82.56	83.16	83.08
	F-Str Fragm.	95.80	95.64	94.69	94.93
	F-Score Gold Stnd.	51.98	56.28	60.66	60.21
	# Parses	2240	2227	2223	2207

All fractions are percentages. F-scores are calculated as $2 \times$precision$\times$recall/(precision$+$recall). The first row shows the number of rules extracted for each grammar. PCFG+P is the standard PCFG with the parent transformation and A-PCFG+P is the annotated PCFG with the parent transformation (here the functional annotations of the mother nodes do not carry over to daughter nodes). Interestingly, PCFG+P and the automatically annotated A-PCFG are of roughly equal size. The next two rows show labelled and unlabelled f-score (composite precision and recall) on trees against the section 23 reference trees as computed by `evalb`. The parent transfom shows a 3% advantage over the PCFG while our A-PCFG performs slightly better with an almost 4% advantage for labelled f-score. Interestingly, combining the parent transform with our automatic f-structure annotation does not seem to improve results. The next row measures f-structure fragmentation. Our current results seem to indicate that (surprisingly perhaps) the pipeline parsing architecture outperforms the integrated model with respect to f-structure fragmentation. The next row reports f-score results against the hand-coded, gold-standard reference f-structures for the 105 randomly selected trees in section 23. The results show clearly that currently the f-structures generated by the integrated model are of higher quality than the ones generated by the pipeline model.

		Pipeline		Integrated	
		PCFG	PCFG+P	A-PCFG	A-PCFG+P
Threshold 1	# Rules	7906	12117	11545	13970
	F-Score Labelled	76.81	79.79	80.08	80.39
	F-Score Unlabelled	79.41	81.92	82.09	82.30
	F-Str Fragm.	96.14	93.92	94.79	95.41
	F-Score Gold Stnd.	51.99	56.07	59.15	60.42
	# Parses	2227	2173	2189	2115

This table reports on a simple thresholding grammar compaction experiment with a threshold set to 1. This means that every rule that occurs only once in the

Penn-II training set is discarded from the grammar. The first row in the table shows that this corresponds to a reduction in size of 60–70% against the original grammars with a very small loss in coverage as indicated in the last row in the table. The reduction in size results in a corresponding increase in parsing speed. According to our experiments, a threshold of 1 results in an increase in speed by a factor of 6–12. To give just one example, parsing section 23 (about 2400 sentences of length average around 20 words) takes 48.71 CPU hours with the full PCFG but "only" 7.91 CPU hours with the compacted PCFG under threshold 1. The table shows that labelled and unlabelled recall on trees as well as f-structure fragmentation and the quality of the f-structures measured as f-score against the gold standard f-structures suffer surprisingly little from this level of thresholding.

		Pipeline		Integrated	
		PCFG	PCFG+P	A-PCFG	A-PCFG+P
Threshold 2	# Rules	5433	8400	7924	9538
	F-Score Labelled	76.18	79.64	79.96	79.90
	F-Score Unlabelled	78.75	81.74	82.01	81.92
	F-Str Fragm.	96.38	96.34	95.10	95.75
	F-Score Gold Stnd.	50.95	55.65	59.07	57.46
	# Parses	2210	2104	2143	2025

Setting the threshold to 2 (i.e. discarding rules that are used less than or equal to 2), again shows remarkably little overall effect apart from a slight decrease in coverage and slightly worse overall f-score results. F-structure fragmentation, by contrast, even decreases in some cases. Parsing speed increases by a factor of 15–20 compared to the full grammar. To give an example, parsing section 23 takes 3.08 CPU hours with the compacted PCFG under a threshold of 2.

		Pipeline		Integrated	
		PCFG	PCFG+P	A-PCFG	A-PCFG+P
Threshold 5	# Rules	3246	4899	4520	5447
	F-Score Labelled	75.07	79.16	78.66	79.24
	F-Score Unlabelled	77.75	81.23	80.77	81.21
	F-Str Fragm.	96.19	96.36	95.77	96.30
	F-Score Gold Stnd.	48.99	54.63	56.26	58.80
	# Parses	2125	1922	1963	1784

A more severe threshold of 5 shows more marked results. On the one hand, all grammars now parse the 2400 sentences in section 23 in less than 1 CPU hour in our Java implementation. However, there is a marked decrease in both coverage and quality. In terms of coverage, the table shows that the more fine-grained grammars PCFG+P, A-PCFG and A-PCFG+P suffer more under severe compaction than the simple PCFG.

It is difficult to compare our approach with that of [Riezler et al, 2002]. The proto-f-structures generated in our approach are much more coarse-grained than the detailed proper f-structures delivered by their carefully handcrafted and optimised LFG grammar. Riezler et. al use an off-line exponential discriminative disambiguation method and achieve f-structure f-scores close to 80% (as against about 60% in our semi-automatic grammar development approach). They use partial bracketing derived from the Penn-II treebank in section 23 to guide their parses whereas we report our results for free (unguided) parses of tagged strings in section 23.

4 Current Work

In our work to date, we have shown how the development of large coverage, rich unification-based grammar resources can be partially automated. However, our research is only a first step in that direction. The reason is that the attribute-value structures we parse into are proto-f-structures. Proto-f-structures interpret linguistic material locally where it occurs in the tree and not where it should be interpreted semantically. Examples of such "non-local" phenomena are extraposition, topicalisation, wh-dependencies, distribution of subjects into VP-coordinate structures to mention but a few. Penn-II employs a rich arsenal of traces and empty productions (nodes that do not realise any lexical material) to coindex "displaced material" (and partly to indicate passive constructions) with positions where the material "originated" (or, to put it in more neutral terms, positions where it should be interpreted semantically). The proto-f-structure annotation algorithm ignores all such traces and empty productions. In our current work we have extended our automatic annotation algorithm to exploit this information. We do this in terms of a new fourth component (Traces) to our annotation algorithm:

$\boxed{\text{L/R Context APs}} \Rightarrow \boxed{\text{Coord APs}} \Rightarrow \boxed{\text{Catch-All APs}} \Rightarrow \boxed{\text{Trace APs}}$

So far we have incorporated traces for A and A$'$ movement (movement to argument and non-argument positions) including traces for wh-questions, relative clauses, fronted elements and subjects of participle clauses, gerunds and infinitival clauses (including both controlled and arbitrary PRO) as reentracies in our f-structures. Null constituents are now treated as full nodes in the annotation (except passive empty object NP) and traces are recorded in terms of INDEX = n f-structure annotations. Traces without indices are translated into arbitrary PRO. The encoding of passive is important as LFG "surface-syntactic" grammatical functions such as SUBJ and OBJ differ from "logical" grammatical functions: surface-syntactic grammatical functions are identified in terms of e.g. agreement phenomena while logical grammatical functions are more akin to thematic roles. The surface-syntactic subject of a passive sentence is usually a logical object, while a surface grammatical object of an optional by-prepositional phrase is usually the logical subject.

In order to evaluate the new "proper-" (rather than "proto-"), f-structures we have updated our manually encoded 105 gold-standard reference f-structures from section 23 with traces encoding reentrancies reflecting locations where linguistic material is encountered and where it should be interpreted. Our current results for proper f-structures (fragmentation and precision and recall) are summarised in the following tables:

# frags.	# sent.
0	507
1	47916
2	1

preds only	
Precision	0.93
Recall	0.87

Proper f-structure annotation precision and recall results suffer very little compared to the best (in terms of coverage) proto-f-structure annotation (preds only: precision 0.93 against 0.94; recall 0.87 against 0.87), indicating that the extended annotation algorithm can reliably determine traces for wh-questions, relative clauses, fronted elements and subjects of participle clauses, gerunds and infinitival clauses as well as passives, and reflect them accurately in terms of indices in the f-structure representations. At this stage, however, fragmentation goes up considerably. We are confident of being able to reduce the number of sentences that do not receive a proper f-structure significantly in further work.

5 Future Work

Penn-II trees annotated with proper f-structures reflecting non-local dependencies are an important ingredient in automatically deriving large coverage unification grammar resources. Recall, however, that the f-structure reentrancies were induced from traces and empty productions in full Penn-II trees relating linguistic material to where it should be interpreted semantically. Full Penn-II style trees with detailed coindexation traces and empty productions are not the standard fare in probabilistic parsing. Indeed, empty productions are usually eliminated in PCFGs and similar approaches to parsing.

In view of this, where do we now get our CFG backbone to parse into proper f-structures? Fortunately, LFG comes to the rescue: standardly, LFG assumes a very surface-oriented approach to its CFG backbone. Non-local phenomena are dealt with in terms of functional uncertainty equations on the level of f-structure representations. Given our annotated Penn-II treebank resource we can automatically compute shortest paths through proper f-structure representations relating material coindexed in f-structure. For each of the long-distance phenomena we collect the paths and compact them into a regular expression to obtain e.g. the functional uncertainty equation for sentential TOPIC etc. During parsing (or annotation), our automatic annotation algorithm then associates every sentential TOPIC function with this equation and the reentrancy is resolved by the constraint solver. In order to make this work properly, we will need to enforce completeness and coherence constraints on our f-structures. Completeness and coherence constraints rely on semantic form values of PRED features. These semantic forms provide subcategorisation information in the form of the syntactic functions required by the predicate governing that level of f-structure. Semantic forms are supplied lexically. Our approach to date, however, is mostly non-lexical. Annotation is driven by categorial and configurational information in Penn-II trees, templates for pre-terminal tags and occasionally by functional annotations (-TMP, -LOC, -CLR etc.) in the Penn-II treebank. How are we going to obtain the semantic forms? One possibility is to use a lexical resource such as COMLEX. Given our annotated version of the Penn-II treebank another option is available to us: if the quality of the automatically generated f-structures is good we can automatically read off semantic forms from these f-structures. Following [van Genabith et al, 1999], for each level of embedding in an f-structure we determine the value of the PRED function and collect all subcategorisable grammatical functions present at that level of f-structure. From these we construct a semantic form. We will explore this in the next stage of our research.

6 Conclusions

In this paper we have developed the first steps and presented initial results of a new methodology to partially automate the development of large coverage, rich unification-based grammar resources. The method is based on an automatic f-structure annotation algorithm that annotates trees in the Penn-II treebank with proto-f-structure information. We have presented two parsing architectures based on this resource: a pipeline model and an integrated model. Currently our best parsers achieve more than 81% f-score on the 2400 trees from section 23 and more than 60% f-score on gold-standard f-structures for 105 randomly selected trees from the same section. We have compared our results with the parent transform approach investigated in [Johnson, 1999] and have conducted a number of thresholding grammar compaction experiments [Krotov et al, 1998]. We have briefly compared our approach with that of [Riezler et al, 2002]. We have presented results of current work on automatic annotation of “proper-” (as opposed to “proto-”) f-structures and outlined future research involving functional uncertainty equations and semantic forms to semi-automatically develop grammatical resources that parse new text into full f-structures.

Acknowledgements

Anette Frank has provided us with root forms for the words in the Penn-II treebank. Ron Kaplan, Stefan Riezler, Mary Dalrymple, John Maxwell, Tracy King and Hiroshi Masuichi have provided comments and support. The research reported in the present paper is supported by Basic Research Grant SC/2001/186 from Enterprise Ireland.

References

[Bresnan, 2001] J. Bresnan 2001. *Lexical-Functional Syntax*. Blackwell, Oxford.

[Butt et al, 1999] Miriam Butt, T.H. King and F. Second, 1999. *A grammar writer's cookbook*. Stanford, Calif. : CSLI Publications

[Cahill et al, 2002a] Cahill, A., M. McCarthy, J. van Genabith and A. Way (2002). Automatic Annotation of the Penn Treebank with LFG F-Structure Information. in *Proceedings of the LREC Workshop on Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data*, Las Palmas, Canary Islands, Spain, pp.8–15.

[Cahill et al, 2002b] Cahill, A., M. McCarthy, J. van Genabith and A. Way (2002). Evaluating Automatic F-Structure Annotation for the Penn-II Treebank in *Proceedings of the Treebanks and Linguistic Theories (TLT'02) Workshop*, Sozopol. Bulgaria, Sept.19th-20th, 2002 to appear (2002)

[Charniak, 1993] Eugene Charniak, 1996. *Statistical language learning*. Cambridge, Mass : MIT Press, 1993.

[Charniak, 1996] Eugene Charniak, 1993. *Tree-bank Grammars*. in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park, pp.1031–1036

[Collins, 1999] M. Collins 1999. *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

[Dalrymple, 2001] Mary Dalrymple 2001. *Lexical-Functional Grammar*. San Diego, Calif.; London : Academic Press

[Frank, 2000] A. Frank. 2000. Automatic F-Structure Annotation of Treebank Trees. In: (eds.) M. Butt and T. H. King, *The fifth International Conference on Lexical-Functional Grammar*, The University of California at Berkeley, 19 July - 20 July 2000, CSLI Publications, Stanford, CA.

[Frank et al, 2002] A. Frank, L. Sadler, J. van Genabith and A. Way 2002. From Treebank Resources to LFG F-Structures. In: (ed.) Anne Abeille, *Treebanks: Building and Using Syntactically Annotated Corpora*, Kluwer Academic Publishers, Dordrecht/Boston/London, to appear (2002)

[Hockenmaier and Steedman, 2002] Hockenmaier, Julia and Mark Steedman, 2002. Generative Models for Statistical Parsing with Combinatory Categorial Grammar Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02) Philadelphia, PA.

[Johnson, 1999] Mark Johnson, 1999. PCFG models of linguistic tree representations Computational Linguistics

[Johnson, 1988] Mark Johnson, 1988. Attribute Value Logic and Theory of Grammar. CSLI Lecture Notes Series, Chicago University Press.

[Kaplan and Bresnan, 1982] R. Kaplan and J. Bresnan 1982. Lexical-functional grammar: a formal system for grammatical representation. In Bresnan, J., editor 1982, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge Mass. 173–281.

[Krotov et al, 1998] Krotov, A., M. Hepple, R.Gaizauskas and Y. Wilks 1998. Compacting the Penn Treebank Grammar, in *Proceedings of COLING/ACL'98* pp.699–703

[Lappin et al, 1989] S. Lappin, I. Golan and M. Rimon 1989. *Computing Grammatical Functions from Configurational Parse Trees* Technical Report 88.268, IBM Israel, Haifa, Israel

[Liakata and Pulman, 2002] M. Liakata and S. Pulman. 2002. *From trees to predicate-argument structures*. COLING'02, Proceedings of the Conference, Taipei, 24 August – 1 September, 2002

[Magerman, 1994] Magerman, D. 1994. *Natural Language Parsing as Statistical Pattern Recognition* PhD Thesis, Stanford University, CA.

[Marcus et al, 1994] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, M. Ferguson, K. Katz and B. Schasberger 1994. The Penn Treebank: Annotating

Predicate Argument Structure. In: *Proceedings of the ARPA Human Language Technology Workshop*.

[Pollard and Sag, 1994] Pollard, C. and I. Sag. 1994. Head-Driven Phrase Structure Grammar, Chicago: University of Chicago Press, and Stanford: CSLI Publications.

[Riezler et al, 2002] Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and Discriminative Estimation Techniques Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02) Philadelphia, PA.

[Sadler et al, 2000] L. Sadler, J. van Genabith and A. Way. 2000. Automatic F-Structure Annotation from the AP Treebank. In: (eds) M. Butt and T. H. King, *The fifth International Conference on Lexical-Functional Grammar*, The University of California at Berkeley, 19 July - 20 July 2000, CSLI Publications, Stanford, CA.

[van Genabith et al, 1999] J. van Genabith, L. Sadler and A. Way. 1999. Data-Driven Compilation of LFG Semantic Forms EACL 99, Workshop on Linguistically Interpreted Corpora (LINC-99), Bergen, Norway, June 12th, 1999, pp.69-76