Overlay Mechanisms for Multi-level Deep Processing Applications

Tracy Holloway King and John T. Maxwell III
PARC

Proceedings of the GEAF 2007 Workshop

Tracy Holloway King and Emily M. Bender (Editors)

CSLI Studies in Computational Linguistics ONLINE

Ann Copestake (Series Editor)

2007

CSLI Publications

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

Abstract

Deep grammars that include tokenization, morphology, syntax, and semantic layers have obtained broad coverage in conjunction with high efficiency. This allows them to play a crucial role in applications. However, these grammars are often developed as a general purpose grammar, expecting "standard" input, and have to be specialized for the application domain. This paper discusses some engineering tools that are used in the XLE grammar development platform to allow for domain specialization. It provides examples of techniques used to allow specialization via overlay grammars at the level of tokenization, morphology, syntax, the lexicon, and semantics. As an example, the paper focuses on the use of the broad coverage, general purpose ParGram English grammar and semantics in the context of an Intelligent Document Security Solutions (IDSS) system. Within this system, the grammar is used to automatically identify sensitive entities and relations among entities, which can then be redacted to protect the content.

1 Introduction

Deep grammars that include tokenization, morphology, syntax, and semantic layers have obtained broad coverage in conjunction with high efficiency (e.g., Kaplan et al., 2004b). This allows them to play a crucial role in applications. Sometimes grammars are developed exclusively for a given application in a given domain. However, a grammar is often developed as a general purpose grammar, expecting "standard" input, and has to be specialized for the application domain. This is done, for example, in MedSLT which is a speech translation system built on top of the Regulus platform (Rayner et al., 2006; Chatzichrisafis et al., 2006).

This paper discusses engineering tools that are used in the XLE grammar development platform (Maxwell and Kaplan, 1996; Crouch et al., 2007) to allow for the domain specialization necessary for applications. Some of the techniques used are similar to those developed for building parallel cross-linguistic grammars (Bender et al., 2002; Butt et al., 2002) but many of them are more fine-grained and involve components that are unlikely to be shared across languages. The focus of this paper is not on how to determine which components to specialize, but instead on what tools have proven useful in allowing the specializations required by the grammar engineers. As an example, the paper focuses on the use of the broad coverage, general purpose ParGram English grammar and semantics in the context of an Intelligent Document Security Solutions (IDSS) system. Within this system, the grammar is used to automatically identify sensitive entities and relations among entities, which can then be redacted via mechanisms such as encryption in order to protect the content.

[†]The IDSS portions of this work were supported in part by Xerox Corporation. We thank the audience of GEAF2007 for comments on the presented version of this paper, and Eric Bier and Jessica Staddon for input on the IDSS application description.

1.1 The IDSS Application

The IDSS application takes document collections, helps a knowledge worker find sensitive entities and relations among entities, and then provides the user to choose mechanisms to protect these entities, including encrypting them so that these sensitive items are only available to those with appropriate keys. The application can be used, for example, to redact documents with sensitive material in them. The documents can simply be printed or produced as a pdf file with the redacted material "black boxed". However, the availability of fine-grained encryption in conjunction with detailed entity and relation analysis allows for documents to be created where each entity type is tied to a particular encryption key. Different end users will have different keys and hence be able to view different parts of the same redacted document. For example, with mortgage documents, some users could see phone number, name, and address information, while others might have access to social security numbers and financial information.

A deep grammar is used to provide an initial list of entities and of relations among entities that the knowledge worker might be interested in. This component is discussed in detail in this paper. There are two other major system components. One is a user interface called Entity Workspace (Bier et al., 2006) which is used to manipulate the document collection and the entities and relations, including adding sensitive entities and relations that were missed by the initial automatic extraction. This component also allows the specification of how much to redact: entities can be redacted at the entity level, the sentence level (any sentence with a sensitive entity is redacted), or the paragraph level (and paragraph with a sensitive entity is redacted).

The second major component is the encryption system that is used to redact entities and relations. This system not only provides the encryption of the sensitive entities, but also allows for fine-grained specification of who can decrypt which sections of the document. This ability to do selective encryption/decription is important in an increasingly electronic workplace where documents are passed from user to user without being printed and where different users of the same document may have much different information needs and rights.

To return to the automatic extraction of entities and relations, as an example, a sentence like (1a) or (1b) would yield the list of facts in (2a) or (2b). These facts are identical except for some byte position information and the word facts for *work* and *employ*. Each fact is designated as being an entity, a relation between entities, or a content word. Entities and relations are typed (e.g. *person*, *location*, *works-for*). Entities can occur with lists of alternative realizations or aliases (e.g. *[Robin, Abramov, Robin Abramov]* in (2a)). Content words can occur with a list of synonyms (e.g. *[hire, use]* in (2b)). Facts are associated with sentence numbers within the document and with byte position of the entity within the sentence.

(1) a. Robin Abramov works for International Business Machines.

b. Robin Abramov is employed by International Business Machines.

(2) a. ENTITY(Abramov, person, sent_num(1), byte_position(7), [Robin, Abramov, Robin Abramov])
ENTITY(International Business Machines, company, sent_num(1), byte_position(25), [International Business Machines, IBM])
ENTITY-REL(cooccuring(1), [International Business Machines, Abramov])
ENTITY-REL(works-for, Abramov, International Business Machines)
WORD(Abramov, sent_num(1), byte_position(7), [male])
WORD(International Business Machines, sent_num(1), byte_position(25), [company])
WORD(work, sent_num(1), byte_position(15), [work, influence, make, cultivate, shape, bring, function, knead, exploit, solve, ferment, sour, exercise])
sentence_num(1)

b. ENTITY(Abramov, person, sent_num(1), byte_position(7), [Robin, Abramov, Robin Abramov])
ENTITY(International Business Machines, company, sent_num(1), byte_position(27), [International Business Machines, IBM])
ENTITY-REL(cooccuring(1), [International Business Machines, Abramov])
ENTITY-REL(works-for, Abramov, International Business Machines)
WORD(Abramov, sent_num(1), byte_position(7), [male])
WORD(International Business Machines, sent_num(1), byte_position(27), [company])
WORD(employ, sent_num(1), byte_position(15), [hire, use])
sentence_num(1)

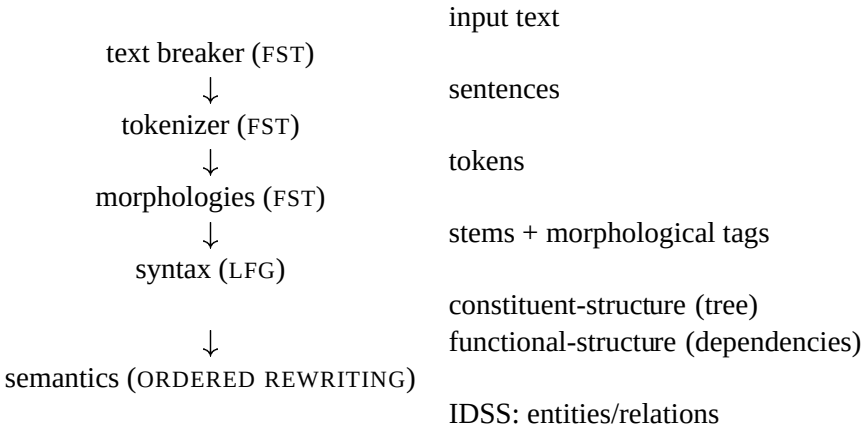
The details of these representations and how they are produced in the IDSS application are discussed in more detail later in the paper.

1.2 The IDSS Natural Language Component

The general XLE parsing pipeline used in the IDSS system is shown in (3).[1]

[1]There is a Makefile which produces a run-time version of the entire pipeline. This depends on XLE's release-grammar mechanisms that allow a single-directory version to be created and frozen for export into the run-time application. In addition to putting the grammar files in a single directory, the release version can include version number information and encrypts the lexicon files (the grammar files themselves are not encrypted).

(3) **XLE Grammar Processing**



This paper focusses on how application-specific extensions were made to the core pipeline, which is used in several different applications and research projects.

Extensions to the finite-state morphologies were needed to allow for additional entities, to the lexicon for Arabic and Russian names, and to the grammar for unusual punctuation and lists. The semantics was extended to pick up the additional entities, to find entities and entity relations, and to delete all other semantic facts. In all cases, we want to ensure that upgrades to the base system can be included in the IDSS application without losing any of the application-specific specialization. We achieve this by allowing for overlay systems at each level. The tools for these overlays, along with examples of how they are applied, are described in this paper.

2 Tokenizers and Morphologies

In the XLE grammars, there is a configuration file for the text breaker, tokenizers, and morphologies. The file specifies which text breaker, tokenizer, and morphology are used by the grammar. When there is more than one tokenizer or morphology, the configuration file specifies how they are combined, e.g., the morphology for recognizing phone numbers may take precendence over the general English morphology which in turn takes precedence over the guesser. The morphology configuration file, called the morphconfig, is called by the syntactic grammar. The XLE ParGram English grammar uses finite-state (FST) text breakers, tokenizers, and morphologies (Kaplan et al., 2004a; Beesley and Karttunen, 2003); these are described in this section.

The input string is first run through the text breaker. The text breaker deterministically breaks the text into sentences. It is a high-precision text breaker: if it is unsure whether something represents a sentence boundary, it will put in a mark (+SB) instead of forcing a sentence break. This way the grammar can be used to provide further information in complex cases. Such cases can occur, for example, when the string *Dr.* appears followed by a form that could be either a common or a proper noun (e.g. *Dr. Bush*); in such cases the text breaker cannot determine whether the

Dr. is a sentence final abbreviation for *Drive* or a sentence internal abbreviation for the title *Doctor*. If this uncertainty is marked and passed through to the syntax, syntactic knowledge can be used to determine whether there should be one sentence or two.

After textbreaking, the tokenizer non-deterministically breaks the string into tokens.[2] The tokenized string is run through an industrial morphology produced by Inxight which in the parsing direction converts inflected forms into lemmas and a set of morphological tags (4a). This morphology covers many proper names, (4b, c), as well as the inflected forms of common nouns, verbs, adjectives, etc.

(4) a. hunts $\longrightarrow$ hunt +Noun +Pl | hunt +Verb +3sg

b. Robin $\longrightarrow$ Robin +Prop +Giv +Fem +Sg | Robin +Prop +Giv +Masc +Sg

c. Detroit $\longrightarrow$ Detroit +Prop +Place +City

In addition, the tokens are run through a set of specialized FSTs to recognise times and dates (5a) and to convert spelled out numbers into digits (5b).

(5) a. April 23rd $\longrightarrow$ month(4) day(23)

b. twenty-four $\longrightarrow$ 24

Items unrecognized by the morphology or one of the specialized FSTs are run through a guesser that uses clues such as capitalization or string ending (e.g. *ing*, *s*) to posit part of speech and other morphological tags. The guesser is currently quite simplistic. An example is shown in (6).

(6) fooing $\longrightarrow$ foo +Noun +VProg +Sg +Guessed
| +Verb +Prog +Guessed
| +Adj +VProg +Guessed

Applications often require special entity recognizers to either supplement or override the morphology. The morphconfig file allows additional FST machines to be called either in an override (USEFIRST) or a supplemental (USEALL) capacity. The override is used when only the analyses in that FST are to be used. For example, the FST that recognizes phone numbers could override any analyses that would recognize the same string as a range of numbers. The supplemental version is used to add in additional analyses. For example, the FST that recognizes years as dates (e.g., *They left in 2000.*) is used in a supplemental capacity in order to allow the analysis of these digits as regular numbers (e.g., *They bought 2000 boxes.*).

[2]The IDSS application did not involve extensions to the tokenizer since the texts parsed followed standard written English punctuation conventions. Other overlays, such as the header/title grammar for parsing technical manuals and web pages, where much of the input has initial upper case or all upper case letters, do use different tokenizer versions.

For IDSS, an FST was added to recognize phone numbers, addresses, and social security numbers. These are provided with unique tags (i.e., *+PhoneNumber*, *+Address*, *+SSNumber*); the syntax and semantics were then extended to recognize these tags and form nouns based on the forms with the tags. An example of the output of the stages of the system for a phone number are shown in (7). The *+PhoneNumber* tag provides the *NE-TYPE phone* feature in the syntax which in turn provides the *ENTITY(. . ., phone, . . .)* feature in the semantics. These all key off of the specialized output of the IDSS FST.

(7) a. Input: They called 123-4567.

b. Tokenizer/morphology: 123-4567 +PhoneNumber +Sg +PreferMorph

c. Syntax:

$$\begin{bmatrix} \text{PRED} & \text{'123-4567'} \\ \text{NTYPE} & \begin{bmatrix} \text{NSYN} & \text{common} \end{bmatrix} \\ \text{NE-TYPE} & \text{phone} \\ \text{NUM} & \text{sg} \\ \text{PERS} & 3 \end{bmatrix}$$

d. Semantics:
ENTITY(123-4567, phone, sent_num(1), byte_position(13))
WORD(123-4567, sent_num(1), byte_position(13), [entity])

The IDSS entity FST was given priority over other FSTs so that only the special named entity analyses would surface. An additional guesser was created to hypothesize certain common person names for nationalities that the standard morphology did not have lists for, namely Arabic and Russian last names; as will be seen in the next section, the grammar was also supplemented with lexicons for the more common of these names. The lexical entries for names provide additional information such as gender and are given higher confidence ratings relative to purely guessed names.

Since the morphology configuration calls both the FSTs for the base grammar and those for the IDSS grammar, any improvements to the base grammar FSTs (e.g., a new time-date FST) can be incorporated into the system by a version update to those files. The morphology configuration allows relative path names so that the FSTs do not need to be copied into the IDSS grammar directory but instead can automatically reference the current version of the base grammar FSTs.

Although not used in the IDSS overlay, XLE also has a command that allows tokenizers to be pushed onto the front of the transducer stack (or popped off of it). That is, the grammar is loaded with the tokenizers specified in the grammar morphconfig, but then an additional tokenizer is run before the ones in the grammar.

This can be used, for example, to have a FST that does spelling correction or named entity markup apply before the regular grammar.[3]

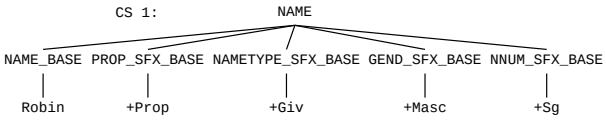
3 Syntax

The output of the tokenizers and morphologies serves as input to the syntax, forming the leaves of a syntactic tree structure. The output of the syntax is a pairing of trees (referred to as c(onstituent)-structures) and dependencies in an attribute value matrix (referred to as f(unctional)-structures). The structures for the sentence in (8a) are shown in (8b,c). The c-structure and f-structure categories are relatively detailed in comparison to most theoretical LFG descriptions (Dalrymple, 2001). XLE's computational approach to syntax and semantics manages ambiguity by combining alternative interpretations into a single packed structure that can be further processed without the typically exponential cost of unpacking (e.g., Robin as a man's name and as a woman's). The XLE syntax and semantics use the same packing mechanism (Maxwell and Kaplan, 1991; Crouch, 2005b).[4]

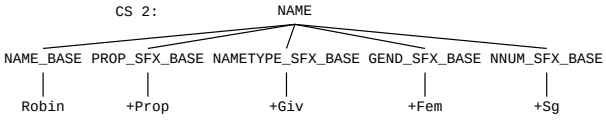
[3]If there is only one grammar being used by the system, then modifying the tokenizer FST via an overlay morphology configuration can be done. However, if multiple versions of the grammar are being run (e.g., one for headers and one for regular text), then using the pop/push facility can save space compared to having two grammars loaded.

[4]There are, in fact, two c-structures for (8a) which differ at the sublexical level due to the two analyses for *Robin* related to the two morphological analyses shown in (4b). Since the display in (8c) does not show the sublexical structure, the difference between the trees is not visible. The two different sublexical trees are shown in (i).

(i) a.

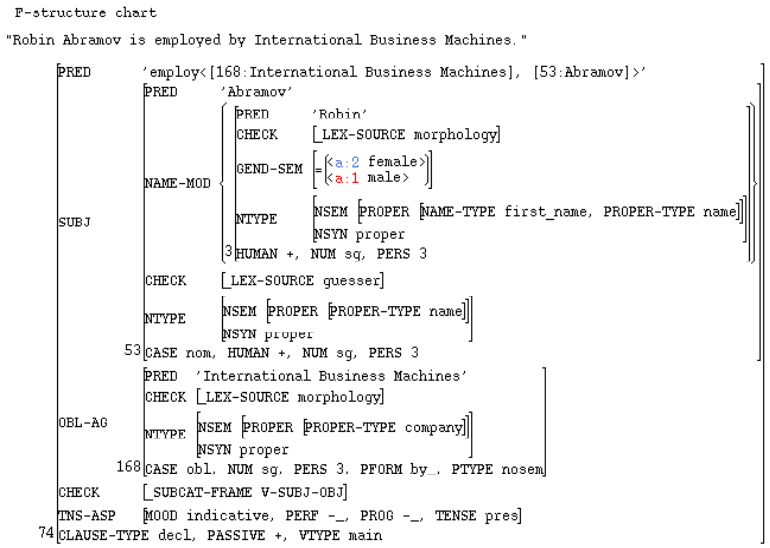


b.

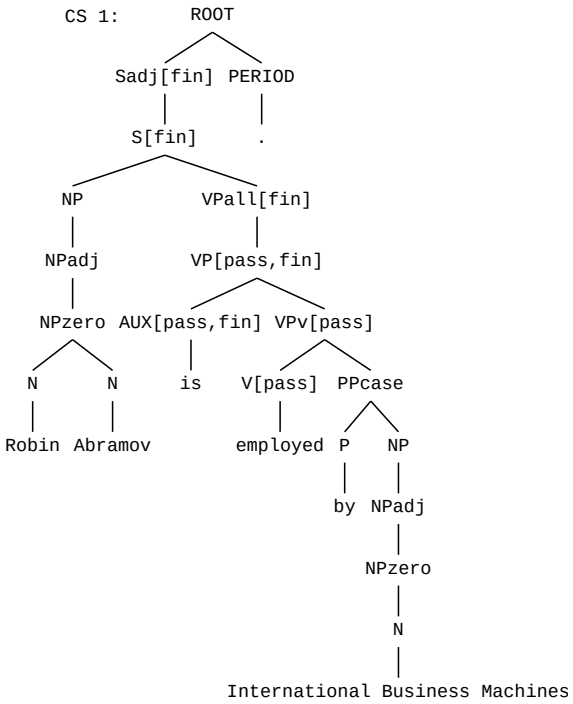


(8) a. Robin Abramov is employed by International Business Machines.

b.



c.



The syntax comprises a configuration file, lexicons, and LFG annotated phrase

structure rules. The lexicons and phrase structure rules can call grammar-defined templates. The configuration file, in conjunction with complex lexicon edit entries (Kaplan and Newman, 1997), is what allows for overlay grammars (Kaplan et al., 2002). The configuration file states which lexicon, rule, template, and system parameter files are used by the grammar. It also states the priority order of these so that application-specific changes take precedence over the more general rules. In addition, a configuration file can state that it is identical to another configuration file except for any stated changes. Such inheritances can be deeply nested, although in practice for this application they only go three levels deep with the standard English grammar as the ultimate base, as is described below for IDSS.

The IDSS syntax overlay is relatively simple; more complex overlays are required for applications used with "non-standard" English, including unedited English or the English used in emails. The IDSS English calls the Aquaint grammar (Bobrow et al., 2005, 2007) which the semantics generally assumes as its input; the Aquaint grammar in turn calls the standard English grammar configuration as its base.[5] In addition, the IDSS grammar calls:

- two lexicon files (one for ∼1900 Arabic names and one for ∼2300 Russian names)
- the morphology configuration as described in the previous section
- a rule file with two sublexical rules for the phone and address entities and a modified version of the sentence final punctuation rule

The configuration file also calls a system parameter file which uses OT mark rankings to effectively remove some unused rules in the standard grammar for efficiency and coverage reasons (e.g., topicalization, initial vocatives) and to set time, memory, and processing limitations for the IDSS system.[6] The IDSS configuration file is shown in (9).

(9) AQUAINT ENGLISH CONFIG (1.0)

```
BASECONFIGFILE ../english-aquaint.lfg
PERFORMANCEVARSFILE
        +idss-performance-vars.txt.
MORPHOLOGY (IDSS ENGLISH).
RULES (STANDARD ENGLISH)
        (AQUAINT ENGLISH)
        (IDSS ENGLISH).
```

[5]Over time, more general solutions are integrated into the standard grammar. Currently, the main overlay in the Aquaint grammar is for certain types of coreference markup used in anaphora resolution.

[6]These could be defined via XLE commands when the system is loaded. However, by including them in the grammar, it is easier to ensure that they are always loaded and always set to the same values. These values can be overridden on the XLE command line to allow for experimentation.

```
FILES +eng-lex-arabic-names.lfg
      +eng-lex-russian-names.lfg
      +english-idss-morphconfig.lfg
      +english-idss-rules.lfg
```

3.1 Overlay Rules

The relative simplicity of the IDSS overlay grammar is due both to the design of the configuration file which allows inheritance and fine-grained modification and to the design of the syntax rules which are divided into subrules to allow for substitution in overlay grammars. The sublexical rules, e.g., the rules used to compose verbs and nouns from combinations of stems and morphological tags (Kaplan et al., 2004a), and the root level rules are particularly finely divided because most applications have required some overlay to these rules. For example, corpora for different applications differ widely as to the type of punctuation allowed sentence finally. As such, there is a rule ROOT-DECL-PUNCT which states the punction options for matrix (root) declarative clauses. In the IDSS grammar, this is redefined to allow colons and, dispreferedly, nothing (as represented in (10) by *e*), in addition to the usual period and exclamation point.

(10) ROOT-DECL-PUNCT –>

```
{   PERIOD
|   EXCL-POINT
|   COLON
|   e: @(OT-MARK NoFinalPunct)
}.
```

This situation highlights the fact that having the proper system tools for overlay grammars is not enough: the grammar developer must design the grammar itself in anticipation of its modification for applications. Fortunately, any changes in modularity to the base grammar benefit all overlay grammars and future applications, and often such changes, such as increased subdivision of rules, are simple to implement. At this point, such subdivisions rarely have to be made; when the standard grammar was first used with overlays, approximately twenty rules were refactored.

3.2 Lexicons

The IDSS grammar calls two lexicon files (one for ∼1900 Arabic names and one for ∼2300 Russian names). These provide information that the forms are person names and indicate whether they are family or given names. When the given name is known as a woman's or a man's name, this information is also included (cf. the discussion of the morphology associated with the English name *Robin* in (4)).

Overlay lexicons can be more complicated. New entries can be added for any part of speech. In addition, entries that exist in the standard grammar can be: (1)

removed, (2) replaced, or (3) altered. This is controlled not just at the level of the stem but also the part of speech and even the possible entries associated with each of these. For example, if the standard grammar had the entry for *push* as in (11a), the overlay grammar could have an enty as in (11b) which would produce the effective entry as in (11c) where the two entries have been merged.

(11) a. push V XLE @(V-SUBJ-OBJ push); ETC.

b. push +V XLE @(V-SUBJ push); ETC.

c. push V XLE { @(V-SUBJ-OBJ push) | @(V-SUBJ push) }; ETC.

The mechanism for lexical edit entries is introduced in Kaplan and Newman (1997) and the current state is described in the XLE documentation (Crouch et al., 2007).

3.3 Performance Variables

The IDSS grammar configuration also calls a system parameter file which effectively removes some unused rules in the standard grammar for efficiency and coverage reasons and sets time, memory, and processing limitations for the IDSS system to allow for effective parsing of large document collections.

The ability to remove and rerank rules takes advantage of the Optimality Theory (OT) mechanism in the XLE system (Frank et al., 2001). The XLE OT system is inspired by theoretical OT (Prince and Smolensky, 1993) but differs from it in crucial respects: in XLE, rules do not need to be ranked, preference as well as dispreference marks are available, and special status marks exist for allowing multiple pass grammars and for declaring rules NOGOOD. Parts of the grammar and lexicon associated with NOGOOD marks are removed from the compiled system.[7] The ability to declare a given OT mark NOGOOD is extremely useful in overlay grammars because both whole rules and specific disjuncts within them can be removed from the grammar in this way. Consider the made-up simple rule in (12).

```
(12)  S –>   (NP:   (^  TOPIC)=!
                    (^  TOPIC)=(^  XCOMP* OBJ)
                    @(OT-MARK TopicMark))
             NP:    (^  SUBJ)=!
             VP:    ^  =!
```

The rule states that an S can consist of an optional NP which will be the topic which also serves as the object somewhere in the structure (e.g. *Bagels, I like.*, *Grammars, I want to write.*), an obligatory NP subject, and a VP that heads the S. The NP topic annotations an OT mark called TopicMark. In the standard grammar, this mark is dispreferred, and so topics will surface only when no more preferrable analysis is

[7] This contrasts with theoretical OT in which constraints can be very lowly ranked but are always violable. NOGOODs could be thought of as inviolable constraints.

possible. However, in many overlay grammars used in applications including IDSS, this mark is declared NOGOOD via the statement in (13) in the overlay performance variables file.

(13) set-OT-rank TopicMark NOGOOD

This effectively creates the rule in (14) without having had to alter the one in (12) in the standard grammar.

(14) S –> NP: (ˆ SUBJ)=!
VP: ˆ =!

The OT marks can similarly be used in the template space to alter the effective behavior of the template. This is often used to control how dispreferred mismatched subject-verb agreement is. In the standard grammar, the OT mark NoVAgr is heavily dispreferred because the grammar expects edited standard written English. However, when used in less formal domains, such as emails, this mark is only slightly dispreferred. This reranking is done in the performance variables file and hence the templates and rules themselves do not need to be altered or have explicit overlay versions.

4 Semantics

The semantics for the ParGram English grammar is written using XLE's ordered rewrite system, referred to as XFR. It takes the f-structure output of the syntax and converts it to a flattened, normalized, skolemized form (Crouch and King, 2006). The output of the semantics is ideal for applications like IDSS because it abstracts away from idiosyncracies of the syntax such as whether the verb was used in the active or the passive.[8] In addition, the semantics provides a mapping to WordNet synsets while also retaining the stemmed word forms from the output of the morphology and syntax. The full semantic structure produced for (15a) is shown in (15b), where the numbers represent WordNet synonym sets (synsets). The output produced from the overlay rules is shown in (15c). In (15c), only relevant entities and relations are kept from the semantics, and the information in these have been rearranged for the application (e.g., the overt marking of sentence and byte position information, the deletion of context information).

(15) a. Robin Abramov is employed by International Business Machines.

b. alias(Abramov:n(7, 1), [Robin, Abramov, Robin Abramov])
alias(International Business Machines:n(30, 1), [International Business

[8]The semantics is a level of linguistic semantics. For greater abstraction, the system can further map into Abstract Knowledge Representation (Crouch, 2005a; Bobrow et al., 2005, 2007). However, this component is not yet as stable and well-developed.

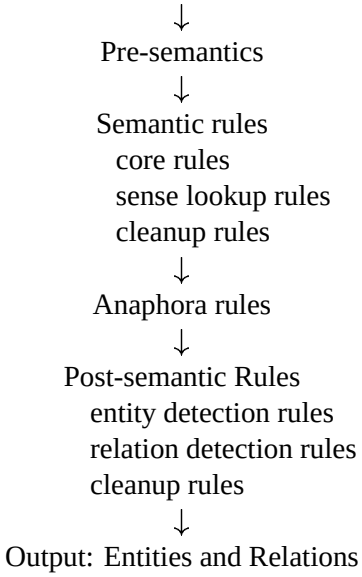
Machines, IBM])
context_head(t, employ:n(18, 1))
in_context(t, pres(employ:n(18, 1)))
in_context(t, cardinality(Abramov:n(7, 1), sg))
in_context(t, cardinality(International Business Machines:n(30, 1), sg))
in_context(t, proper_name(Abramov:n(7, 1), name, Abramov))
in_context(t, proper_name(International Business Machines:n(30, 1), company, International Business Machines))
in_context(t, role(Agent, employ:n(18, 1), International Business Machines:n(30, 1)))
in_context(t, role(Patient, employ:n(18, 1), Abramov:n(7, 1)))
lex_class(employ:n(18, 1), vnclass(unknown))
lex_class(employ:n(18, 1), wnclass(1147708, verb(consumption)))
sortal_restriction(Abramov:n(7, 1), Thing, employ)
sortal_restriction(International Business Machines:n(30, 1), Thing, employ)
word(Abramov:n(7, 1), Abramov, noun, 1, 7, t, [[9487097, 7626, 4576, 4359, 3122, 7127, 1930, 1740]])
word(International Business Machines:n(30, 1), International Business Machines, noun, 1, 30, t, [[7948427, 7943952, 7899136, 7842951, 29714, 2236, 2119, 1740]])
word(employ:n(18, 1), employ, verb, 1, 18, t, [[1147708], [2385846]])

c. ENTITY(Abramov, person, sent_num(1), byte_position(7), [Robin, Abramov, Robin Abramov])
ENTITY(International Business Machines, company, sent_num(1), byte_position(27), [International Business Machines, IBM])
ENTITY-REL(cooccuring(1), [International Business Machines, Abramov])
ENTITY-REL(works-for, Abramov, International Business Machines)
WORD(Abramov, sent_num(1), byte_position(7), [male])
WORD(International Business Machines, sent_num(1), byte_position(27), [company])
WORD(employ, sent_num(1), byte_position(15), [hire, use])
sentence_num(1)

Since the semantics is run on an ordered rewrite system, the overlays take the form of additional rule sets which occur in the stack of ordered semantics rules. To do this effectively, the rules have to be factored so that new rule sets can be interwoven in the stack without having to alter the base files. If the base files have to be altered, then whenever a new version of the base semantics is released, the changes for the overlay will be lost and have to be hand added. In order to overlay the semantics, XLE provides a way to call the new, overlayed stack and to implement application specific commands.

The semantics rules used in IDSS and as the base semantics for the ParGram English grammar are divided into two main sets: semantic rewrites and anaphora resolution. The semantic rewrites are further divided into six sets, including core semantic rules, sense lookup rules, and cleanup rules. For IDSS, two additional rule sets are added before the semantic rules and after the anaphora rules. For other applications, such as consumer search, different sense lookup rules may be overlayed. The basic semantic XFR rule stack used in IDSS is shown in (16). Details of the pre- and post-semantic rules are discussed in this section.

(16) **Semantic XFR Rule Stack**



4.1 Pre-semantic Rewrite Rules

The IDSS pre-semantic rules are very simple (three calls to the same template) and are used to pick up the special entities provided by the IDSS morphology, e.g., the addresses, phone numbers. These convert the entities into a format that resembles that of proper names and other aliased items and hence is recognized by the semantics.

4.2 Post-semantic Rewrite Rules

The post-semantics/anaphora rule set is more complex. These rules operate on the output of the semantics to extract the entities and entity relations needed for IDSS: they identify entities such as proper nouns, time expressions, phone numbers, nouns in certain classes (e.g., currencies and explosives); they identify relations such as who works where, who lives where, and who knows whom; they provide information such as synonyms of each content word.

The entity detection rules are relatively straightforward. They take a subset of the *word* facts already present in the semantics and rewrite them to contain the information needed in the IDSS application. For example, all proper nouns are marked as entities and are included with information as to their type and location in the sentence, as shown in (17).

(17) a. ENTITY(Detroit, location, sent_num(1), byte_position(24), [Detroit])

b. ENTITY(Smith, person, sent_num(1), byte_position(10), [John, Smith, Mister Smith, Mister John Smith])

At the level of the semantics, no lexicon is needed to determine which entities to mark. Instead, it is features from the syntactic f-structure such as the PROPER-TYPE which provide the trigger for the rule.

The rules also allow for words with certain meanings to be extracted. This is done by determining what WordNet (Fellbaum, 1998) synset describes the class of interest and then creating entity facts for any words with this synset somewhere in the hypernyms of the word's semantics. For example, in certain application domains, explosives and weapons may be of interest and hence should be recognized as entities, which can then be highlighted or redacted as appropriate. If this is the case, the extracted entities for a sentence like (18a) will include an entity fact as in (18b) since WordNet knows that dynamite is a type of explosive.

(18) a. The dynamite arrived on Friday.

b. ENTITY(dynamite, explosive, sent_num(1), byte_position(1))

After the entities are identified, relations among them are posited. By identifying the entities first, more general relation rules can be written that look for relations between entities of a particular type, e.g. certain relations hold between person entities and company entities but not between persons and other persons.

The rules to extract relations among entities are more complicated than the entity detection rules. In general, the relations of interest are specific for a given IDSS application. For example, some application domains have rules to extract information as to which people work for which company. Detecting these relations at the semantic level is simpler than at the text string or the syntactic f-structure level. For example, all of the forms in (19) will have the same basic role relations in the semantics.

(19) a. IBM employs Robin Abramov.

b. Robin Abramov is employed by IBM.

c. IBM's employee, Robin Abramov,

d. IBM's employment of Robin Abramov

e. Robin Abramov is an employee of IBM.

f. Robin Abramov's employer is IBM.

These role relations are then used to extract an ENTITY-REL fact as in (15c). However, even at this highly normalized level, several rules can be required to extract a given relation. In the *works-for* relation example in (15), the same relation expressed by the corresponding *work for* phrases have slightly different roles assigned to them by the semantics. As such, for high value relations, there may be several rules to extract the relevant relation facts.

There is a default rule for relation extraction that marks all entities in a sentence as occuring together. This information could be reconstructed from the entity facts because the sentence number is recorded as part of the fact. However, by combining them into a single fact, applications can immediately see co-occurrences. An example is shown in (20).

(20) a. Mary left and John arrived.

b. ENTITY(John, person, sent_num(1), byte_position(15), [John])
ENTITY(Mary, person, sent_num(1), byte_position(1), [Mary])
ENTITY-REL(cooccuring(1), [Mary, John])

Once the entities and relations are identified, the rest of the semantic facts are deleted, leaving just the IDSS specific information, as shown in (15c).

Since the rules operate after all the base semantic rules, improvements to the semantics can be automatically incorporated by updating to the newest version of the base semantics. If there is a change in analysis to the semantics, it may be necessary to change the IDSS rules to be sensitive to these changes. The rules which define the feature space of the base semantics, as well as the svn version control system and the regular use of regression testing whenever changes are incorporated (Chatzichrisafis et al., 2007), make such changes in the base semantics relatively easy to track.

4.3 Flags

The rewrite system also allows flags to be set that can be used to trigger or block rules. The rules check for the setting of the flags and then trigger (or not) based on the setting for the run-time system. These flags are set when loading the rules to produce the desired behavior.

Even in non-overlay grammars, a flag of this type is used to trigger feature checking rules when used in debugging mode. Consider the feature checking rule in (21). The flag *debug(%%)* is set to 1 when the system is being run in debug mode. If it is, then the rule in (21) fires whenever there is a two argument predicate that is not listed as a licensed_feature.

(21) {getp(debug(1))},
qp(%Feat, [%Arg1, %Arg2]), −licensed_feature(%Feat,2)
==>
NOT_LICENSED_FEAT(qp(%Feat, [%Arg1, %Arg2])).

In the run-time system, this flag is turned off by setting *debug(%%)* to 0 in order to avoid the insertion of warning messages in the output structures. The use of flags in the current IDSS overlay system is kept to a minimum, being largely restricted to debugging, but it does offer a feature similar to the syntactic OT marks for removing or inserting (but not ranking) rules without altering the XFR semantic rule files themselves.

5 System Issues and Conclusions

System Issues All the above components are kept under an svn version control system and undergo regular regression testing (Oepen et al., 1998, 2002; Chatzichrisafis et al., 2007). The versioning allows easy access to previous versions of the system. This is useful not only for backing out of changes that turned out not to be improvements, but also for allowing the use of previous versions of the grammar and the semantics until the overlay grammars can catch up to the changes made. In addition, svn makes it possible for multiple developers to work on the system at the same time, helping to merge changes made by different people. The regular regression testing highlights changes, whether improvements or not, to each component and to the system as a whole. Sometimes changes to a given component will have no effect on a specific application while at other times even small changes to components can significantly alter the behavior of the system.

Conclusions Adapting a complex deep processing system to an application requires changes to all levels of the processing pipeline. As such, it is important that easy-to-use overlay mechanisms are provided at each level and that the levels are modular. The form of these mechanisms may vary depending on the type of system component (e.g., overlaying a unification-based grammar requires different techniques than overlaying an ordered rewrite system). Having such mechanisms allows the application to seamlessly incorporate improvements to the base system over time, while maintaining the specialization features. This is particularly important when base components of the system are still undergoing rapid development (e.g., with the semantics in the IDSS application described here), but even relatively stable components will improve over time and applications need to take advantage of these improvements without a major system overhaul.

This paper has outlined a series of tools that are used in XLE to overlay all levels of analysis from tokenization to semantics, using the IDSS application as an example. The XLE overlay mechanisms have been refined over time based on experiences with a number of specialized domains and applications. Even with the overlay mechanisms in place, the base rules of each component have to be designed to

allow overlays through appropriate rule factoring and modularization of rule sets and system components.

References

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

Bender, Emily, Flickinger, Dan and Oepen, Stephan. 2002. The Grammar Matrix: An Open-Source Starter-Kit for the Rapid Development of Cross-linguistically Consistent Broad-Coverage Precision Grammars. In *COLING Workshop on Grammar Engineering and Evaluation.*

Bier, Eric, Ishak, Eddie and Chi, Ed. 2006. Entity Workspace: an evidence file that aids memory, inference, and reading. In *IEEE International Conference on Intelligence and Security Informatics (ISI 2006)*, pages 466–472, Springer Verlag.

Bobrow, Daniel G., Cheslow, Bob, Condoravdi, Cleo, Karttunen, Lauri, King, Tracy Holloway, Nairn, Rowan, de Paiva, Valeria, Price, Charlotte and Zaenen, Annie. 2007. PARC's Bridge and Question Answering System. In *Proceedings of the Grammar Engineering Across Frameworks 2007 Workshop,* CSLI On-line Publications.

Bobrow, Daniel G. Condoravdi, Cleo, Crouch, Richard, Kaplan, Ron, Karttunen, Lauri, King, Tracy Holloway, de Paiva, Valeria and Zaenen, Annie. 2005. A Basic Logic for Textual Inference. In *AAAI Workshop on Inference for Textual Question Answering.*

Butt, Miriam, Dyvik, Helge, King, Tracy Holloway, Masuichi, Hiroshi and Rohrer, Christian. 2002. The Parallel Grammar Project. In *COLING Workshop on Grammar Engineering and Evaluation.*

Chatzichrisafis, Nikos, Bouillon, Pierrette, Rayner, Manny, Santaholma, Marianne, Starlander, Marianne and Hockey, Beth Ann. 2006. Evaluating Task Performance for a Unidirectional Controlled Language Medical Speech Translation System. In *Proceedings of the HLT-NAACL Workshop on Medical Speech Translation.*

Chatzichrisafis, Nikos, Crouch, Dick, King, Tracy Holloway, Nairn, Rowan, Rayner, Manny and Santaholma, Marianne. 2007. Regression Testing For Grammar-Based Systems. In *Proceedings of the Grammar Engineering Across Frameworks 2007 Workshop,* CSLI On-line Publications.

Crouch, Dick. 2005a. Packed Rewriting for Mapping Semantics to KR. In *International Workshop on Computational Semantics.*

Crouch, Dick, Dalrymple, Mary, Kaplan, Ron, King, Tracy Holloway, Maxwell, John T. and Newman, Paula. 2007. XLE Documentation, on-line documentation.

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

Crouch, Richard. 2005b. Packed Rewriting for Mapping Semantics to KR. In *Proceedings Sixth International Workshop on Computational Semantics, Tilburg, The Netherlands.*

Dalrymple, Mary. 2001. *Lexical Functional Grammar*. Academic Press.

Fellbaum, Christiane (ed.). 1998. *WordNet: An Electronic Lexical Database.* The MIT Press.

Frank, Anette, King, Tracy Holloway, Kuhn, Jonas and Maxwell, John T. 2001. Optimality Theory Style Constraint Ranking in Large-scale LFG Grammars. In Peter Sells (ed.), *Formal and Empirical Issues in Optimality Theoretic Syntax*, pages 367–397, CSLI Publications.

Kaplan, Ron, King, Tracy Holloway and Maxwell, John T. 2002. Adapting Existing Grammars: The XLE Experience. In *COLING Workshop on Grammar Engineering and Evaluation.*

Kaplan, Ron, Maxwell, John T., King, Tracy Holloway and Crouch, Richard. 2004a. Integrating Finite-state Technology with Deep LFG Grammars. In *Proceedings of the Workshop on Combining Shallow and Deep Processing for NLP (ESSLLI)*.

Kaplan, Ron and Newman, Paula. 1997. Lexical Resource Concilation in the Xerox Linguistic Environment. In *ACL Workshop on Computational Environments for Grammar Development and Engineering.*

Kaplan, Ron, Riezler, Stefan, King, Tracy Holloway, Maxwell, John T., Vasserman, Alex and Crouch, Richard. 2004b. Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings of HLT-NAACL'04*.

Maxwell, John and Kaplan, Ron. 1991. A Method for Disjunctive Constraint Satisfaction. *Current Issues in Parsing Technologies* .

Maxwell, John and Kaplan, Ron. 1996. An Efficient Parser for LFG. In *Proceedings of the First LFG Conference*, CSLI Publications.

Oepen, Stephan, Flickinger, Dan, Toutanova, Kristina and Manning, Chris. 2002. LinGO Redwoods. A Rich and Dynamic Treebank for HPSG. In *First Workshop on Treebanks and Linguistic Theories.*

Oepen, Stephan, Netter, Klaus and Klein, J.1998. TSNLP — Test Suites for Natural Language Processing. In John Nerbonne (ed.), *Linguistic Databases*, CSLI.

Prince, Alan and Smolensky, Paul. 1993. Optimality Theory: Constraint Interaction in Generative Grammar, ruCSS Technical Report #2, Center for Cognitive Science, Rutgers University.

Rayner, Manny, Hockey, Beth Ann and Bouillon, Pierrette. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI.