# Integrating Finite-state Technology with Deep LFG Grammars[1]

**Ronald M. Kaplan, John T. Maxwell III, Tracy Holloway King, and Richard Crouch**
Palo Alto Research Center
{kaplan|maxwell|thking|crouch}@parc.com

## 1 Introduction

Researchers at PARC were pioneers in developing finite-state methods for applications in computational linguistics, and one of the original motivations was to provide a coherent architecture for the integration of lower-level lexical processing with higher-level syntactic analysis (Kaplan and Kay, 1981; Karttunen et al., 1992; Kaplan and Kay, 1994). Finite-state methods for tokenizing and morphological analysis were initially incorporated into the Grammar-writer's Workbench for Lexical Functional Grammar (LFG) (Dalrymple, 2001; Kaplan and Bresnan, 1982), a relatively complete LFG parsing system but for relatively small-scale grammars (Kaplan and Maxwell, 1996). Finite-state transducers, also of a relatively small scale, were constructed automatically in this system from specifications provided by the LFG grammar writer, using the compilation techniques described by (Kaplan and Kay, 1994).

The algorithms and interfaces developed in the Grammar-writer's Workbench were re-implemented and extended to create a hardened, industrial-scale parsing and generation environment for broad-coverage LFG grammars, called XLE. By the time the new system was being designed, the finite-state approach to lexical processing had firmly taken hold and substantial morphological transducers were being produced for many computational applications in many languages. Thus, the XLE system was organized to make it easy to combine LFG syntactic specifications with externally developed lexical resources embodied in finite-state transducers. Our experience over the last decade has demonstrated that existing FST morphologies are invaluable in creating broad-coverage grammars and using them for efficient deep LFG parsing and generation.

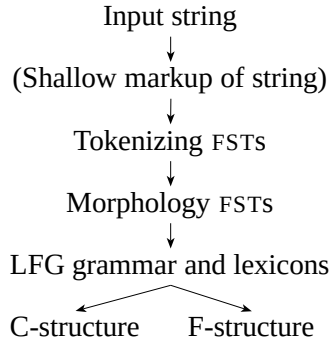
In this paper we describe two uses of FSTs in the XLE ParGram grammars (Butt et al., 1999; Butt et al., 2002).[2] The examples discussed come from the English grammar, but FSTs are used in this way in the majority of the grammars (there are grammars for English, French, German, Japanese, Norwegian, and Urdu; grammars for Danish, Malagasy, and Welsh are also under development). The basic organization of the XLE system is diagramed in (1). So, a string like *Boys appeared.* will pass through the system as in (2). Note that all components of the system are non-deterministic and can pass multiple output to the next component; this is necessary to avoid pruning potentially good parses too early as there is no back-tracking among the components. This non-deterministic cascading can be done efficiently in XLE due to its ambiguity packing technology (Maxwell and Kaplan, 1989).

Section 2 discusses the intergration of morphological FSTs into the grammars. This integration has resulted in greatly decreasing the lexicon development task for the grammars. Section 3 discusses using FSTs for tokenization and the intergration of shallow markup such as part of speech tagging.

[1]This work was supported in part by the Advanced Research and Development Activity (ARDA)'s Novel Intelligence from Massive Data (NIMD) program under contract # MDA904-03-C-0404.

[2]The ParGram project develops parallel deep grammars for a number of languages. The grammar produce similar output structures to aid in multi-lingual applications, such as machine translation, and in porting applications from one language to another.

(1)




(2)

| | |
|---|---|
| Input string: | Boys appeared. |
| Tokenizing: | boys TB appeared TB . TB |
| Morphology: | boy +Noun +Pl |
| | appear +Verb +PastBoth +123SP . |
| C-structure: | Figure 1 |
| F-structure: | FIgure 2 |

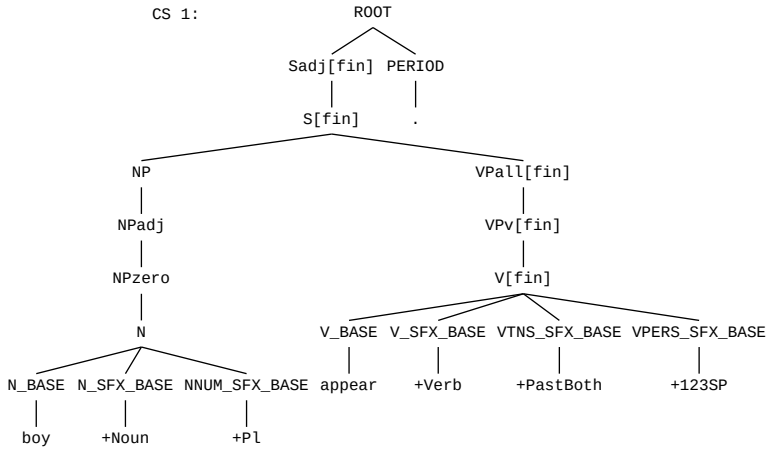



Figure 1: Constituent-structure for *Boys appeared.*

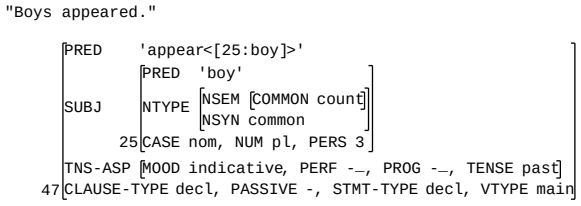



Figure 2: Functional-structure for *Boys appeared.*

## 2 Morphologies

In this section we describe how large FST morphologies can be incorporated into the XLE LFG grammars. These morphologies can either have been created for other purposes and used "off the shelf" or can be created as part of the grammar development process. Fortunately, large FST morphologies already exist for many languages (e.g., produced by companies like Inxight).

Before demonstrating how the morphologies are integrated into the deep grammars, it is important to explain what they do. FST morphologies associate a surface form with a canonical form (stem or lemma) and a set of tags that encode relevant morphological features. Some English examples are shown in (3).

(3) a. turnips $\Longleftrightarrow$ turnip +Noun +Pl

b. Mary $\Longleftrightarrow$ Mary +Prop +Giv +Fem +Sg

c. wound $\Longleftrightarrow$ wound +Noun +Sg
wound +Verb +Non3sg
wind +Verb +PastBoth +123SP

(3a) states that the surface form *turnips* can be analyzed as a stem *turnip* and two morphological tags. The first tag *+Noun* indicates the part of speech. The second tag *+Pl* indicates that the word is plural in number. Note that the surface form *turnip* would have corresponded to the same stem and part of speech tag, but with the number tag *+Sg* corresponding to its singular status. (3b) states that the surface form *Mary* can be analyzed as a stem *Mary* and four tags which indicate that it is a proper noun, is a given femine name, and is singular. (3c) demonstrates how a single surface form can correspond to more than one analysis. The surface form *wound* has a noun and two verb analyses. That is, it can be a singular noun, in which case it stems to *wound* and is associated with the tags *+Noun* and *+Sg*. Or, it can be either the base form of a verb, in which case it stems to *wound* and is associated with the tags *+Verb* and *+Non3sg*, or it can be a past tense verb, in which case it stems to *wind* and is associated with the tags *+Verb*, *+PastBoth*, and *+123SP*. All three forms are passed to the grammar for parsing in order to avoid early pruning of potentially correct analyses (but see section 3 on using POS tagging as a filter).

The exact set of tags depends on the language and the desired application. In general, the FST morphologies used in the ParGram project were not designed for deep grammars. Instead, their primary intended application was information retrieval. Because of this, the analysis of proper nouns is extremely detailed (witness the four tags for the form *Mary* in (3b)), while from a syntactic standpoint this level of detail is probably not necessary. However, as long as the tag set and tag order is known, it is possible to integrate the FST into the grammar. In addition, it is possible to use FST tools to modify existing FSTs so that their output tags are more suited for deep grammars (see (Kaplan and Newman, 1997) for a full discussion of this issue). The Norwegian FST, for example, was modified because the original representation of the Norwegian gender system did not allow the grammar writers to capture some needed distinctions. The English FST was also restricted to prevent the generator from generating both British and American spellings while still permitting alternative spellings to be recognized in parsing.

The association between surface forms and stems plus tags is extremely useful even in a language like English which has relatively impoverished morphology. The more inflectional morphology that a language has, the more useful such FST morphologies are. This is because the integration of the FST morphology into the grammar allows there to be a single lexical entry for the stem form of the word, avoiding having to list all the surface forms. In addition, as discussed in the next section, words with predictable subcategorization frames do not need to have an explicit lexical entry at all.

### 2.1 Basic Integration

This section describes how the FST morphologies are integrated into the deep grammars (again, see (Kaplan and Newman, 1997) for more details). The basic idea is as follows. First the surface forms are run through the FST morphology to produce the corresponding stems and tags. Stems and tags each have entries in the LFG lexicon. Sublexical phrase-structure rules produce syntactic nodes covering these stems and tags and standard grammar rules then build larger phrases.

In order to do this, XLE must know which morphological transducers to use with a particular grammar. This is specified in the MORPHCONFIG associated with each grammar. A MORPHCONFIG might look like (4) (the tokenizers are discussed in the section 3).

(4) STANDARD ENGLISH MORPHOLOGY (1.0)
TOKENIZE: ../common/english.tok.parse.fst
ANALYZE: ../common/english.infl.fst
MULTIWORD: english.standard.mw.fst

(4) states that the tokenizer for the grammar is the file english.tok.parse.fst (with the path name where this FST is located). The FST morphological analyzer is the file english.infl.fst. The multiword analyzer, which is not discussed here, is the file english.standard.mw.fst.

When XLE parses a string, it first passes it through the tokenizer to non-deterministically divide the string into tokens. It then passes the tokens through the FST morphology. This results in a set of stems and tags that the LFG grammar builds into phrases and assigns a well-formed functional structure. There may be multiple sets which are all passed to the grammar for parsing. To make this more concrete, consider the sentence in (5a). Once (5a) is run through the FST morphology the result will be as in (5b).

(5) a. Boys appeared.
b. boy +Noun +Pl appear +Verb +PastBoth +123SP .

XLE looks up the lexical entry for each of these items. These are shown in (6). Note that tags and punctuation have lexical entries, just as stems do.

(6)

| | | | |
|---|---|---|---|
| boy | N | XLE | @(NOUN boy). |
| +Noun | N_SFX | XLE | @(PERS 3) @(EXISTS NTYPE). |
| +Pl | NNUM_SFX | XLE | @(NUM pl). |
| appear | V | XLE | @(V-SUBJ appear). |
| +Verb | V_SFX | XLE | @(EXISTS VTYPE). |
| +PastBoth | VTNS_SFX | XLE | @(TENSE past). |
| +123SP | VPERS_SFX | XLE. | |
| . | PERIOD | *. | |

A lexical entry begins with a head (e.g, *boy*) that matches the stem or tag provided by the morphology. This is followed by a part of speech, e.g, *N*. Next comes a code that XLE uses to determine whether this entry is allowed to match items coming from the transducer morphology, indicated by XLE, or whether it can match against an unanalyzed token, indicated by *. The vast majority of words go through the morphology. A token reading with * is used only when the grammar writer wishes to allow an alternative to the information provided by the morphology or wishes to provide an entry for a token not found in the morphology: both of these cases are rare because the FST morphologies have excellent lexical coverage. Finally, the remainder of the lexical entry comprises calls to templates which provide the functional information used by the grammar. For example, the *+Noun* tag calls templates that provide the feature PERS with the value 3 and that require the f-structure to have a value for NTYPE. As seen by the entries for *+123SG* and the period, some lexical items do not call any templates.

The grammar combines the stems and tags into low-level phrase-structure nodes using sublexical rules. Sublexical rules are similar to regular LFG rules, although they tend to be much simpler in that they do not have complex functional annotations. For example, the sublexical rule for common nouns like *boys* would look like (7).

(7) N $\longrightarrow$ N_BASE N_SFX_BASE NNUM_SFX_BASE.

The only difference between regular rules and sublexical rules is in the display: there is a display toggle to hide the sublexical information so that the leaves of the tree are the inflected tokens instead of the

stem and the tags. The display with sublexical information is shown in Figure 1 for our example sentence *Boys appeared*.[3] The corresponding f-structure is shown in Figure 2; here the person and tense information provided by the tags is seen.

XLE provides an additional feature which makes the use of FST morphologies ideal for simplifying lexicon development. For lexical items with unpredictable subcategorization frames, it is necessary to provide a lexical entry for each lexical item. This is primarily the case for verbs, but also for adjectives that take oblique prepositional phrases and nouns that take clausal arguments, for example. However, the vast majority of lexical items have entirely predictable subcategorization frames. That is, knowing the part of speech and some inflectional information is sufficient for determining the lexical entry. This is the case for most common and proper nouns, adjectives, adverbs, and numbers. XLE defines a special lexical entry *-unknown* which matches any stem that comes from the morphology. This lexical entry contains these predictable parts of speech with the associated template calls. Consider the simplified entry for *-unknown* in (8).

(8) -unknown N XLE @(NOUN %stem);
A XLE @(ADJ %stem);
ADV XLE @(ADVERB %stem).

This entry states that any item can be a noun N, an adjective A, or an adverb ADV. The *%stem* is a variable that denotes the unknown stem and makes it available to the template. For example, if the stem is *boy*, the form *boy* would be passed to the NOUN template. Obviously, most words are not nouns, adjectives, *and* adverbs and so the grammar should not build analyses for all of these possibilities for each stem. The appropriate restrictions are provided by the sublexical rules: the noun entry will only go through if the morphology assigns the stem in question the appropriate noun tags, etc.

A stem is matched against the *-unknown* entry only as a last resort, if it matches no other entry in the lexicon. Thus, given the lexical entries shown in (6), *boy* would not match *-unknown*. But the explicit entry for *boy* can be removed to simplify the lexicon, and *boy* would then match against the N option in the *-unknown* lexical entry in (8).

By integrating FST morphologies into XLE grammars, it is possible to significantly reduce the size of the lexicons and the amount of time spent in their development. Not only do the FST morphologies make it so that only stem forms need lexical entries, but the existence of the *-unknown* entry means that the majority of lexical items need no explicit lexical entry at all. Instead, they are built on the fly based on information in the morphology FST and the sublexical rules.

### 2.2 Multiple FSTs

The previous section discussed how FST morphologies are integrated into XLE. In this section we briefly review the methods described by (Kaplan and Newman, 1997) for obtaining finer control over the morphology output provided to the deep grammar.

Although the FST morphologies used by the XLE ParGram grammars are extremely large, they can never be complete. New words are constantly being added to the language, by being borrowed, morphed from one part of speech to another, or newly coined. Fortunately, novel words tend to follow regular morphological patterns. This can be exploited by the grammars by creating a guesser FST morphology which will guess the part of speech and other inflectional tags based on the surface form of a word. For example, English words ending in *-ed* can be past tense verbs or deverbal adjectives, words

[3]The detail oriented reader will notice that the part of speech categories in the tree and the sublexical rule in (7) end in _BASE while the lexical entries in (6) do not. XLE automatically adds these endings and uses them in determining how to display these lexical items.

ending in *-s* can be plural nouns or third singular verbs, and words beginning with a capital letter can be proper names.

However, in general it is not desirable to have the grammar guess at a surface word form unless this form cannot be found in the regular morphology. The order in which FSTs (both tokenizers and morphologies) apply is specified in the MORPHCONFIG.

(9) STANDARD ENGLISH MORPHOLOGY (1.0)
TOKENIZE: ../common/english.tok.parse.fst
ANALYZE: ../common/english.infl.fst
../common/english.guesser.fst
MULTIWORD: english.standard.mw.fst

In the sample MORPHCONFIG in (9), the FST morphologies are ordered. XLE will apply the words to the FST morphologies *in order* until an analysis is found. So, for a word like *boys* which is known by the standard English morphology english.infl.fst, XLE will apply that FST to *boys*; this results in the output *boy +Noun +Pl*; XLE then stops applying FSTs to *boys* and moves on to the next word. However, for a word like *Saakashvili*, which is a name not known to the English morphology, XLE will first apply the standard English morphology, but this will result in no output; so, it then applies the next FST english.guesser.fst. Due to the initial capital, the guesser will produce a stem *Saakashvili* and the appropriate tags for a proper noun, as well as a tag *+Guessed*.[4]

Generally, the XLE LFG grammars use two morphological FSTs: a large standard morphology and a guesser. However, grammars adapted for specific corpora (Kaplan et al., 2002), may use additional FSTs. These FSTs can cover novel terminology, multiword expressions, or words that are unknown to the standard morphology but which the grammar writer does not wish to have go through the guesser (e.g., verbs specific to the corpus). For example, the English EUREKA grammar was specialized to parse copier repair tips, and a morphological FST was added to recognize the regular patterns that comprise part and fault numbers, such as those in (10a). In addition, this grammar uses a special FST to recognize multiword copier part names, as in (10b).

(10) a. 600T40506
D-034
PL3-E11

b. cam follower
CRT controller PWB
airknife solenoid assembly

Thus, being able to use multiple FST morphologies for a grammar allows for finer control of the types of words recognized by the grammar and the types of analyzes that they receive.

## 3 Tokenizers and Shallow Markup

The use of FST morphologies described above presupposes that the string of characters of a sentence has been split up into individual tokens. Tokenizing is relatively simple, although not trivial, when processing written text with standard punctuation marks. It becomes more complicated when the input string contains shallow markup, such as part of speech tagging. In this section, we first discuss standard tokenization using FSTs and how this interfaces with the deep grammars. We then examine the modifications that are needed to allow for shallow markup of the grammar input.

### 3.1 Tokenizers

The process of tokenization breaks up a string (e.g. a sentence) into tokens (e.g. words). As with the output of the FST morphologies, the output of the tokenizer need not be deterministic and all possible

[4] All guessed words are provided with a distinctive tag. This makes it possible to provide them with a feature in the f-structure marking their source, which can be used in debugging and by applications to determine the reliability of the analysis. In addition, it allows the grammar writer to defi ne OT marks (Frank et al., 2001) to (dis)prefer analyses with guessed words.

tokenizations are passed to the next component, namely the morphologies. Most deep grammars work on tokenized input, in the simplest case treating all punctuation as spaces. Here we describe a more complex system that allows the grammar writer to use the information provided by the punctuation to constrain the grammar. In English,[5] tokenization involves: breaking off punctuation, breaking off clitics, and lower casing certain letters. We consider these in turn.

Breaking off punction is often a simple task. Consider the examples in (11) where the string is on the left and the tokenized form on the right. TB indicates a token boundary.[6]

(11) a. I see them. $\Longrightarrow$ I TB see TB them TB . TB

b. the dog, a poodle, $\Longrightarrow$ the TB dog TB , TB a TB poodle TB , TB

In each example, the punctuation has been split off from the preceding word. This is crucial because the morphology and lexicon will recognize forms like *them* and *dog* but not forms like *them.* and *dog.*, which are what occur in the string.

It turns out that in many languages, it is not enough simply to insert token boundaries around punctuation. This is because it may be much easier for a punctuation-sensitive syntactic grammar to require punctuation marks for certain constructions even though the rules of normal English typography do not allow those marks to appear in the surface string. For example, the rules for English appositives and parentheticals are simpler if they specify that those constructions are set off by commas, but according to rules of English haplology, those marks are not always present in the string. In English, sentence periods consume commas, as in (12a), and abbreviatory periods, as in (12b).

(12) a. Find the dog, a poodle. $\Longrightarrow$ Find TB the TB dog TB , TB a TB poodle TB , TB . TB

b. Go to Palm Dr. $\Longrightarrow$ Go TB to TB Palm TB Dr. TB . TB

In (12a), a comma must be inserted by the tokenizer between the *poodle* token and the . token. In fact, an arbitrary number of commas can potentially be inserted to account for nested parentheticals; this is straightforward to do with FSTs.[7] In (12b), the ending period token must allow for an abbreviatory period that is part of the *Dr.* token. All of these possibilities are non-deterministically passed to the FST morphology to avoid potentially incorrect pruning of analyses.

Another role of the tokenizer in English is to split off clitics. An example is seen in (13).

(13) I'll go. $\Longrightarrow$ I TB 'll TB go TB . TB

By splitting off the clitic auxiliary *'ll*, the grammar can treat this auxiliary similarly to its full counterpart *will*.

The above examples are relatively straightforward and are intended to provide a feel for the tokenization process. This process can become more complex because it is not always clear when to break off punctuation. For example, hyphens may be word internal or join separate words; in the former case they should not be broken off by the tokenizer while in the latter they should. The syntactic processing algorithms in XLE operate on arbitrary regular languages and thus allow for the tokenizing transducer to be nondeterministic as well as not linear-bounded. For example, any sentence final period will allow for zero or more haplology commas.

[5]The Japanese ParGram grammar uses the well-known ChaSen tokenizer (Asahara and Matsumoto, 2000). This tokenizer combines some of the features of a tokenizer with those of a morphology. It breaks the string of characters into words and then provides some basic part of speech tagging for those words. For details on how the ChaSen tokenizer was integrated into the XLE Japanese grammar, see (Masuichi et al., 2003).

[6]By convention XLE requires a final token boundary but not an initial one. Nothing crucial rests on this.

[7]This behavior is provided by transducers that formally have the property of not being linear bounded. A technical adjustment to LFG's prohibition against nonbranching dominance chains is required to avoid computational difficulties that could otherwise arise from the composition of an LFG grammar and a non-linear-bounded transducer.

Another source of non-determinism for English tokenizers is to lowercase capitalized words in certain positions. This may occur in sentence initial position, as in (14a), although it must be optional, as in (14b). In addition, it may occur in other positions, such as after colons, as in (14). (In the examples below, only the lowercasing is shown, not the insertion of TB token boundaries.)

(14) a. The boy left. $\Longrightarrow$ the boy left.
b. Mary left. $\Longrightarrow$ Mary left.

(15) The boy left: He was unhappy. $\Longrightarrow$ the boy left: he was unhappy.

Thus, a tokenizer needs to know which environments (e.g., sentence initially and after colons) trigger optional lower casing. Otherwise, the input tokens will not match the morphology and lexicon. For example, *Bush* and *bush* are different words with different morphological and syntactic properties.

To provide an integrated example, consider (16). The initial word can be lower cased or not and haplology commas are provided for, shown here by curly braces for the disjunction and the Kleene star notation for the haplology.

(16) Bush saw them. $\Longrightarrow$ { Bush | bush } TB saw TB them TB [, TB]* . TB

As the rules of tokenization vary from language to language, it is necessary to write tokenizers for each language. This can be done relatively easily with off-the-shelf finite-state technology (e.g., that which comes with (Beesley and Karttunen, 2003)). XLE provides a default parsing and generation tokenizer which simply splits off punctuation. The French and German grammars have used the English tokenizer with moderate success, but these are being specialized for those languages to improve accuracy.

### 3.2 Shallow Markup

As discussed in (Kaplan and King, 2003), it is possible to use FSTs in conjunction with minor changes to the XLE LFG grammars to allow for input strings which contain shallow markup. The idea behind using shallow markup is that such markup may decrease ambiguity and increase accuracy while decreasing the amount of processing time. Shallow markup can include part of speech (POS) tagging (17) and named entities (18).

(17) a. I/PRP_ saw/VBD_ her/PRP_ duck/VB_.
b. I/PRP_ saw/VBD_ her/PRP$_ duck/NN_.

(18) a. <person>General Mills</person> bought it.
b. <company>General Mills</company> bought it.

An evaluation of the usefulness of this markup in conjunction with deep grammars is discussed in (Kaplan and King, 2003). Basically, the POS tagging was found to improve speed but to decrease accuracy because of compatibility failures between the tagging and the correct analysis; further investigations indicate that using partial POS tagging with improved back-off techniques when matching fails can result in an increase in speed and accuracy. The named entity markup improved both speed and accuracy, in part because the internal structure of proper names was not built by the parser. Here we summarize how FSTs can be used to process this marked up input in such a way that the deep grammar can operate on it correctly.

The first issue is that the normal rules of tokenization discussed above, such as lowercasing and comma haplology, need to skip the markup. The second is that this markup should not be broken up or tokenized as if it were standard punctuation. Finally, the markup must be interpreted correctly by the grammar: with POS tags this is done at the level of the FST morphology; with named entities this is done by a combination of FST tokenization and the grammar.

**Part of Speech Tagging** POS tags are not relevant to tokenization, but the tokenizing transducer must be modified so that the POS tags do not interfere with the other patterns that the tokenizer is concerned with. The proper effect of a POS tag is to reduce the number of outputs from the morphological analyzer. Consider the words *walks* in (19).[8]

(19) She walks/VBZ_.

The morphological analyzer normally would produce two outputs for *walks* of which only one is appropriate in this context, the verb. However, with unmarked strings, XLE passes both analyses to the grammar. The POS tags are used to trim the analyses early by only passing the morphological analyses compatible witht the POS tagger to the grammar.

Obtaining the desired behavior requires a specification of the morphological output sequences that are consistent with a given POS tag. This is done by a hand-written mapping table that states legal correspondences between POS tags and the morphological analyzer tags.

Given this mapping table, we produce a FST that filters the normal output of the morphology FST. The filtering FST, for example, allows pairs of lemmas and morphological indicators to be followed by VBZ if and only if the indicators are compatible with the allowable sequences drawn from the table. Thus the indicator +Noun is not allowed in front of VBZ. The filtering FST also maps the POS tags to epsilon, so that they do not appear in its output. This FST is put in a cascade with the tokenizer and morphology FSTs. The overall effect is that the tags in the input are preserved by the tokenizer, pass through the morphological analyzer, and then cause incompatible morphological-indicator sequences to be discarded. Thus only the contextually appropriate interpretations are passed on to the grammar. Another version of the filtering FST allows through both the compatible sequences and the incompatible ones, but marks the incompatible ones with an additional tag so that the grammar can disprefer parses with these tags, only using them on a second pass if parsing with the compatible tags fails; this two-stage parsing is controlled by OT marks (Frank et al., 2001).

**Named Entities** The named entities appear in the text as XML mark-up, as in (20).

(20) <company>General Mills</company> bought it.

The tokenizer was modified to include an additional tokenization of the strings whereby the material between the XML mark-up was treated as a single token with a special morphological tag on it, as in (21a). The underscore represents the literal space occurring in the middle of the named-entity. As a fall back mechanism, the tokenization that the string would have received without that mark-up is also produced, as in (21b).

(21) a. General_Mills TB +NamedEntity TB bought TB it TB . TB

b. General TB Mills TB bought TB it TB . TB

The morphological analyzer was modified to allow the additional morphological indicator +NamedEntity to pass through. A lexical entry was added for that indicator and the grammar was changed to allow it to follow nouns. The grammar first tries to parse only with the named enity version. If this parse fails, then a second pass is tried without any markup; as with the POS tagging fall-back mechanism, this second pass grammar is controlled by OT marks (Frank et al., 2001). Since the named entity finder recognition is quite good for proper names, this second pass is rarely needed, thereby increasing both speed and accuracy of the grammar.

[8] In their initial experiments, (Kaplan and King, 2003) used deterministic, gold standard POS tagging derived from the Penn Treebank. However, the POS tag filter can also allows for alternative POS tags for a given lexical item, to prevent early pruning when the tagger is not certain.

## 4 Conclusions

We have described a language-processing architecture that carefully integrates finite-state techniques for tokenization and morphological analysis with XLE's algorithms for parsing and generation with LFG grammars. This architecture has important theoretical consequences, as it allows many generalizations about the properties of words and strings to be factored out of the specification of more complicated syntactic patterns. It also offers significant practical advantages, since it allows the information included in independently developed, large-scale lexical resources to be used directly as a component of grammar. This avoids the cost of redundant specification and helps to boot-strap broad-coverage capabilities in a cost-effective way. The ParGram grammars have particularly benefited from XLE's facilities for integrating finite-state technology with deep LFG grammars, allowing for the development of truly broad-coverage deep grammars (for coverage and accuracy evaluation statistics, see (Riezler et al., 2002) on the English grammar, (Masuichi et al., 2003) on the Japanese grammar, and (Kaplan et al., 2004) for a comparison of the English grammar to shallow parsers).

## References


M. Asahara and Y. Matsumoto. 2000. Extended models and tools for high-performance part-of-speech tagger. In *Proceedings of COLING*, pages 21–27.

K. Beesley and L. Karttunen. 2003. *Finite State Morphology*. CSLI Publications.

M. Butt, T.H. King, M.-E. Niño, and F. Segond. 1999. *A Grammar Writer's Cookbook*. CSLI Publications.

M. Butt, H. Dyvik, T.H. King, H. Masuichi, and C. Rohrer. 2002. The Parallel Grammar project. In *COLING 2002: Workshop on Grammar Engineering and Evaluation*, pages 1–7.

M. Dalrymple. 2001. *Lexical Functional Grammar*. Academic Press.

A. Frank, T.H. King, J. Kuhn, and J.T. Maxwell. 2001. Optimality theory style constraint ranking in large-scale LFG grammars. In P. Sells, editor, *Formal and Empirical Issues in Optimality Theoretic Syntax*, pages 367–397. CSLI Publications.

R.M. Kaplan and J. Bresnan. 1982. Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. The MIT Press.

R.M. Kaplan and M. Kay. 1981. Phonological rules and fi nite state transducers. Presented at the annual meeting of the LSA.

R.M. Kaplan and M. Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

R.M. Kaplan and T.H. King. 2003. Low-level mark-up and large-scale LFG grammar processing. In *Proceedings of the LFG03 Conference*. CSLI On-line Publications.

R.M. Kaplan and J.T. Maxwell. 1996. LFG grammar writers workbench. ftp://ftp.parc.xerox.com/pub/lfg.

R.M. Kaplan and P. Newman. 1997. Lexical resource reconciliation in the Xerox Linguistic Environment. In *Computational environments for grammar development and linguistic engineering*, pages 54–61.

R.M. Kaplan, T.H. King, and J.T. Maxwell. 2002. Adapting existing grammars: The XLE approach. In *COLING 2002: Workshop on Grammar Engineering and Evaluation*, pages 29–35.

R.M. Kaplan, S. Riezler, T.H. King, J.T. Maxwell, A. Vasserman, and R. Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of HLT-NAACL*.

L. Karttunen, R.M. Kaplan, and A. Zaenen. 1992. Two level morphology with composition. In *Proceedings of the 14th International Conference on Computational Linguistics*, pages 141–148.

H. Masuichi, T. Ohkuma, H. Yoshimura, and Y. Harada. 2003. Japanese parser on the basis of the Lexical-Functional Grammar formalism and its evaluation. In *Proceedings of The 17th Pacific Asia Conference on Language, Information and Computation (PACLIC17)*, pages 298–309.

J.T. Maxwell and R.M. Kaplan. 1989. An overview of disjunctive constraint satisfaction. In *Proceedings of the International Workshop on Parsing Technologies*, pages 18–27.

S. Riezler, T.H. King, R.M. Kaplan, R. Crouch, J.T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of ACL*.