Conversational Syntax Requirements

Kurt Fuqua, Cambridge Mobile
kurt@cambridgemobile.com

Abstract Conversational applications involve the cooperation of multiple concurrent apps. The applications must share a single syntactic grammar which is the superset of each individual application's syntactic grammar since there must be centralized processing. Similarly, the recognized vocabulary must be the superset of the individual vocabularies. These requirements force the development of a single comprehensive grammar rather than the traditional application-specific grammars. The vocabulary must be decoupled from the syntax, moreover the grammar must be used bi-directionally for both analysis and synthesis. Complexities of comprehensive natural language grammar drive the need for extended formalisms. Thus, conversational applications require a radically different architecture both in processing and formalism.

The architectural requirements for processing syntax are radically different than those used for IVRs. A conversational application allows a user to interact with it and multiple other conversational applications in a single, interactive conversation. One of the great processing challenges is that no application knows in advance which one of the apps the user will address in his next sentence. The sentence must be processed *before* a determination is made about the semantic destination of the sentence's content. This is a significant dilemma for structuring the grammar. Tradition application specific grammars presume that all verbal interaction is directed toward a single application exclusively. On a mobile phone application, that assumption never applies. A single mobile phone may have a half dozen voice-enabled mobile apps running concurrently. Each app must be listening for a command relevant to them. This application concurrency forces a radical change in the architecture.

Stateful vs. Stateless Grammar

IVRs use stateful grammars; conversational systems must use stateless grammars. By 'stateful grammar', we mean that after one sentence is processed by the grammar, the state of the system is changed so that the next sentence is processed with a different subgrammar. The first and second sentences are parsed by different subgrammars. This technique is assumed in nearly all IVRs. It has the benefit of limiting the system perplexity thereby improving recognition accuracy. The technique is acceptable if a single app processes every sentence – because it can track the state.

But, what if there are multiple apps involved? Each app can no longer track the state because most of the sentences are not relevant to it. Since there is no state tracking, the system cannot know which subgrammar to use to parse the next sentence. The only practical system is to have every sentence parsed by a single grammar. That single grammar must represent the aggregate of all the sentences in every application. In other words, **a conversational system requires a comprehensive grammar**. This is a fundamental architectural change.

IVRs don't use comprehensive grammars. Instead, there are tiny fragmentary grammars for just a few sentences at a time. Creating a comprehensive grammar for a natural-language such as English requires more powerful formalisms and parsing engines.

Scalable Grammars

In SRGS formalism, the defined syntactic structure and the defined vocabulary are combined into a grammar. The vocabulary cannot be expanded without expanding the grammar. The syntax cannot be defined without defining the vocabulary. Vocabulary

and syntax are inextricably linked. This linkage must be broken in order to allow a comprehensive grammar to be scaled.

Terminal POS

Rather than use literal words as terminal symbols in the grammar, imagine using parts of speech. These parts-of-speech serve as placeholders for words contained in the lexicon. When the grammar calls for an adjective, the engine considers the adjectives in the lexicon. Only the enabled vocabulary needs be considered. If there are six loaded conversational applications, then only the aggregate vocabulary for those applications is enabled in the lexicon. This way the vocabulary can dynamically scale. At any point, the lexicon has enabled only the required vocabulary for the loaded conversational apps.

By using POS as terminal symbols, the syntactic grammar contains very few literals. The vocabulary is dynamic and is essentially populated at run time.

Decoupling Vocabulary

Decoupling the vocabulary from the syntax allows the creation of a comprehensive grammar which is reusable for each application. This can be a superset against which each application is developed. This requires standardization of the part-of-speech symbols to be used as terminals. The terminal POSs can be further qualified by feature tags.

verb< valency=trans >

This qualifies the verb as being transitive in valency. Standardization of the POS, features and morphology processing are assumed. This is addressed my paper *Conversational Lexical Standards*.

The formalism for these comprehensive grammars must also be reversible. Traditional syntactic grammars are used to drive a syntactic parser during analysis. In conversational applications, the identical grammar must drive the 'syntactic composer' during synthesis. The remainder of the paper discusses the requirements of the formalism with this in mind. This is not an attempt to refine SRGS.

Agreement, Features & Inheritance

Comprehensive grammars *cannot* be constructed from context-free grammars (CFGs). Chomsky proved this about 50 years ago. CFGs continue to be used due to their simplicity but the attempt is futile. Realism requires a non-CFG formalism which captures the generality of natural language elegantly.

The most obvious need for non-CFG's is concord. The agreement of the verb in person and number with the subject is the classic example. The solution is to associate grammatical features with terminal symbols (POSs) and phrases. The same system of features used for the POS in the lexicon can be inherited to the phrase level. For example, the NP inherits the feature 'plurality' from the head noun. The features of the concord elements (whether POS or Phrase) can be compared at parse time for agreement. Thus the formalism can retain the universal phrase structure with the addition of associating features and the provision of an inheritance mechanism.

During analysis (parsing), we think of inheritance as moving from the terminals up the parse tree to a phrase. In composition, the process is reversed. Feature attributes of a phrase are transferred downward to the terminal symbols (reverse-inheritance).

Non-increment

It is common in NL for an element of a terminal to be specified in one phase and another element of the same terminal to be specified in an adjacent phrase. The adjacent phrases share a common terminal symbol. This is true of the English auxiliary system. In essence, we don't want the parser to increment the word that it is processing. Thus, a non-increment functionality must be included in the formalism. Here is an example of the English auxiliary with perfect and progressive phases as defined in Lingua.

```
AuxNF ::= (Perf) (Prog) (Pass)
Perf ::= v<"have"> v<mood=pastpt> -
Prog ::= v<"be">   v<mood=prespt> -
Pass ::= v<"be">   v<mood=pastpt> -
```

The '-' symbol following the qualified verb instructs a non-increment operation. Here is an analysis of a perfect progressive statement such as:

> …having been hit …
> Perf Prog verb
> v<'have'>
> v<mood=pastpt>+v<'be'>
> v<mood=prespt>+v<hit>

Start & Resume

Inversion is common in European language such as English especially in interrogatives. The process is fairly simple to describe.

"The first word of the predicate phrase (PP) is inverted with the subject."

> She can come.
> Can she come?

This is **impossible** to perform in a general way with a CFG! The PP is actually split into 2 pieces in the parse tree. The split can occur at many potential places. Another way to describe this is that the parse of the PP is *started* but after the first word, the NP is parsed, then we *resume* the parse of the PP where we left off.

This capability is called start & resume. It is an elegant solution and essential requirement of the syntactic formalism.

Practical Implementations

The parser/composer must be a distinct bidirectional engine which is driven by the syntactic grammar. To be practical, the entire comprehensive grammar must be pre-optimized at compile time. With terminal POS, this is possible. The parser should implement look ahead. Most importantly the parser must be fault-tolerant. Fault-tolerance allows a parser to interpolate a missing word, or possibly two. This is very complex to implement but dramatically improves reliability in high perplexity systems such as needed for conversation.

Conclusion

Conversational concurrent applications require sharing a common syntactic grammar. Each sentence must be processed by a single comprehensive grammar. This is in contrast to the traditional application-specific grammars which are small. The comprehensive grammar must be reusable and general without embedded vocabulary. To allow scalability, the terminal symbols must be part-of-speech. The vocabulary is dynamic as applications are loaded and vocabulary is enabled from a central shared lexicon.

The syntactic formalism requires radical changes from a CFG. There should be features associated with terminals and phrases. The features must be inheritable. Concord rules will operate on and restrict parsing operations. The complexities of real world natural language grammars require non-increment and start-resume operations. These operations are contained in Lingua. Finally, the identical grammar must be completely reversible so that it is equally suited for both analysis and synthesis.

Bibliography

The Scalable Language IPA, Technical Reference Manual, January 2009 (http://slapi.sourceforge.net/)

Lingua Manual, April 2010