A Unified Language Model Architecture for Web-based Speech Recognition Grammars

Wesley Holland, Daniel May, Julie Baca, Georgios Lazarou and Joseph Picone

Center for Advanced Vehicular Systems
Mississippi State University
{wholland, may, baca, glaz, picone}@cavs.msstate.edu

Abstract *— Supporting popular language model grammar formats, such as JSGF and XML-SRGS, has been an important step forward for the speech recognition community, particularly with respect to integration of human language technology with Internet-based technologies. Industry standard formats, though conceptually straightforward implementations of context free grammars, contain restrictions that pose serious challenges when applied to aspects of the speech recognition problem. This paper compares and contrasts these formats, discusses the implications for speech recognition systems, and presents a unified language model architecture to support transparent conversion between various language model formats. This architecture requires the conversion of higher-level grammar specifications such as the JSGF and SRGS into lower-level theoretical structures such as Augmented Backus-Naur form and Standard Backus-Naur form. The public domain implementation of this architecture provides a framework for future advancements in language model conversion and web-based speech recognition applications.*

Keywords *— context free grammars, grammar transformations, speech recognition*

1. INTRODUCTION

Several industry standard grammar specifications such as the Java Speech Grammar Format (JSGF) [1] and the W3C XML Speech Recognition Grammar Specification (XML-SRGS) [1] were created to support development of voice-enabled Internet applications. While these standards allow for the specification of context free grammars (CFGs), most language models for automatic speech recognition have a regular grammar equivalent and can therefore be modeled as finite state machines (FSMs). To support language model creation using these standards, we developed a suite of software tools in our public domain speech recognition toolkit [3] to convert between these grammar formats.

Issues of theoretical equivalence and restrictions on conversions between regular and context free grammars have been studied and described extensively. No algorithm has been proven to perform conversions from arbitrary CFGs generating regular languages to FSMs without assuming certain restrictions on the grammar, such as the absence of center-embedded non-terminals [4]. However, software tools have been developed for conversions between FSMs and CFGs which assume such restrictions on the grammars handled [5]. A similar restriction is assumed in the dynamic grammar compilation described in [6]. This work examined the use of finite-state transducers for dynamic grammar compilation, and thus focused on this aspect rather than the complexities of specific CFG to FSM conversions.

Some commercial toolkits also provide conversions between specific CFGs, e.g., XML-SRGS to JSGF [7] or to a specific FSM. The difficulty with commercial products, however, is that conversion to FSMs for other recognition engines may not be easily supported. In addition, our experience has shown that the specifics of each of the individual CFG-based Internet grammar formats present unique and significant challenges. No publication to date has adequately articulated in detail the challenges of converting these CFGs to FSMs in a manner that is modular and extensible, and therefore more easily used and widely applicable. For example, the treatment of weights and probabilities in XML-SRGS differs fundamentally from that common to most speech recognition systems by treating weights on loops differently than weights on non-repeating arcs. XML-SRGS thus requires a level of interpretation that other CFG formats may not. Such features heighten the need for modular software architecture built upon the appropriate theoretical abstractions. The remainder of this paper describes the technical issues encountered in the design and implementation of our conversion process along with our solutions to these issues and offers insight into future development of robust, general purpose, and verifiable language model conversion tools.

2. GRAMMAR FORMATS

This section provides an overview of existing language model grammar specifications for the Internet and examines how these are used in speech recognition systems. First, however, for reference in the following sections, we mention our internal language model format, known as ISIP Hierarchical Digraph (IHD).

Similar to the numerous recognizer-specific language model representations in common use, IHD is implemented as a set of hierarchically layered FSMs.

As shown in Fig. 1, each FSM layer is a generic directed graph class or DiGraph. IHD binds these layers of FSMs together into levels. The top-most level contains a single DiGraph, with the nodes of this DiGraph mapping to more complete DiGraphs at the level below it. For example, the top-most layer might represent the sentence level, the level below that the word level, the next the state level.

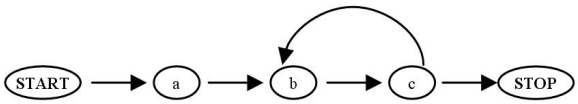

Fig. 2. FSM for regexp a(bc)+

Our goal was to provide a bidirectional conversion tool that could systematically convert to and from JSGF and XML-SRGS, i.e., down to the phone and state level, so that recognition could be performed in these popular formats by our speech engine. While not all recognition systems supporting alternate grammar formats provide this capability, we believe that it is an important feature that reduces development efforts and experimental setup time. We also required the tool to provide the same level of conversion in reverse in order to allow our internal language models to be used on other recognition systems.

The subtle but important distinctions between JSGF and XML-SRGS proved challenging to the task of developing general purpose conversion tools. However, our deepened understanding of these nuances led to the design of a modular software architecture providing conversion tools for these and future grammar formats. The remainder of this section presents key theoretical and syntactic features of each format, including similarities and differences. Section 0 presents the developed unified language model architecture.

2.1 BNF and ABNF

JSGF and XML-SRGS are theoretically equivalent in expressive and computational power, with both adhering in principle to Backus-Naur Form (BNF) [8], a formal notation for CFGs, and more specifically to two equivalent variants: Extended BNF (EBNF) [9] and Augmented BNF (ABNF). Many detailed descriptions of BNF and its variants exist. We briefly introduce key features relevant to our discussion.

Stated simply, BNF defines a method for describing production rules in a CFG, including terminal and non-terminal symbols for rules, and a selection of alternatives among rules. Though numerous variants of the syntax exist, an example rule in a BNF grammar might be:

<A> ::= <B>|c

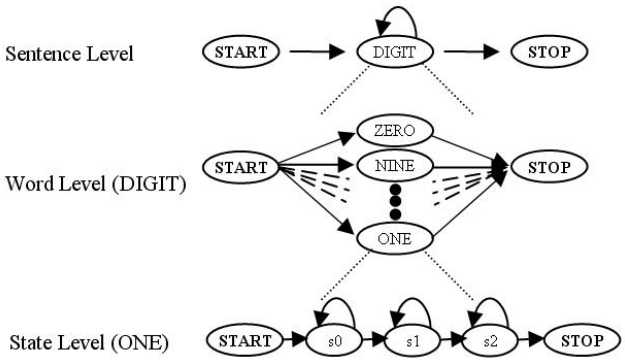

Fig. 1. IHD Format

where non-terminals are represented in capital letters, A, B, surrounded by brackets <> and can appear on the lefthand side (LHS) or righthand side (RHS) of the rule demarcated by the := symbol. Terminals are often expressed in lower case (though not required), but more importantly can appear only on the rule RHS. While concatenation is often implied, it can, for emphasis, be explicit. In such cases, concatenation is typically expressed with the comma. Finally, selection or branching among alternative rule definitions is expressed by the | symbol.

BNF also allows the use of recursive rules in a grammar. Such rules directly or indirectly reference themselves. An example of direct recursion might be: <A> ::= a<A>. The use of directly or indirectly recursive rules is useful to represent repetitive actions in an FSM. Consider the simple FSM in Fig. 2. This FSM recognizes the regular expression, a(bc)+ that could be represented in BNF with the production rules:

<S> ::= <A>
<A>::= aB
<B> ::= (bc)|(bcB)

The use of the non-terminal B on the RHS in rule 3 is recursive and indicates that subgraph bc is a cycle that can be repeated one or more times. However, for simple rules such as this, the cycle in this FSM could be represented using the Kleene + operator, a standard notation for regular expressions which denotes 1 or more repetitions. EBNF extends BNF to support the use of structures such as the * and + for repetition, as well as others. (EBNF also has many variants, but its origins date to [9].) This allows for the creation of a more intuitive set of production rules for regular expressions, so that rules A and B above can be reduced to:

<A> ::= a (bc)+

2.2 JSGF and XML-SRGS

The lack of standardized specification for many concepts essential for speech recognition made BNF and ABNF less directly useful for this purpose. This led to the creation of the first industrial grammar specification, JSGF. Although JSGF and XML-SRGS both provide expressive equivalence to EBNF, they differ wildly in structure. For instance, the rule in Fig. 2 could be represented in JSGF as:

<A> = a(bc)+;

Note that the + operator is supported directly as well as the use of parentheses. Consider the same rule in XML- SRGS:

```
<rule>
  <item> a </item>
  <item repeat='1-'>
    <item> b</item>
    <item> c </item>
  </item>
</rule>
```

The <rule> tag marks this as a production rule. The terminal symbols are marked with <item> tags, and the "repeat= '1-'"

attribute on the <item> grouping surrounding the non-terminals b and c denotes the Kleene + operation (1 or more repetitions) applied to these tokens.

Clearly, the JSGF syntax is more similar to EBNF than XML-SRGS. The differences are due in large part to their origins: XML was designed initially as a markup language for general Internet usage and later modified to provide support for spoken language; JSGF was designed from the outset to support spoken language applications. The W3C SRGS attempted to address these issues first, by using JSGF as a theoretical model in defining the XML-SRGS, and second, by developing a standard specification for ABNF [2]. ABNF is an EBNF variant, with origins dating back to Arpanet. Any ABNF-SRGS can be mapped to XML-SRGS. The previous example could be written as shown below, using *1 for + before the item (bc) to be repeated:

```
<A> = a *1 (bc)
```

The subtle distinctions between syntaxes complicated the task of identifying underlying theoretical structures. Beyond the syntax issues, however, JSGF and XML-SRGS include programmatic constructs such as scope resolution, as well as methods for specifying weights, an essential feature for practical speech recognition.

Another interesting set of issues arose with respect to recursion. While a conformant JSGF grammar processor must provide support for recursive rules (right recursive), this support is optional for conforming XML-SRGS processors. Though a detailed discussion is beyond the scope of this paper, several arguments can be made in favor of this support. First, any right recursive rule can be rewritten using the Kleene * and + operators where more appropriate. Second, speech recognizers typically use regular grammars, which must be either left or right linear, and thus can contain only left or right recursion. since including this capability would not affect grammars which did not support recursion, but the reverse was not true, we chose to include it in all our grammar format conversion tools.

Other specific features of XML with respect to weights and probabilities complicated our design decisions. These are discussed below.

2.3 XML-SRGS Weights and Probabilities

In XML-SRGS, the structures available for specification of probabilistic alternation do not correspond in a straight-forward manner to weighted transitions of a FSM. These structures take the form of the *weight* attribute and the *repeat-prob* attribute. It is important to note that in XML-SRGS, these are two different logical entities. During translation to an accepting FSM, these attributes contribute in a complex manner to the determination of arc transition weights; specifically, *weight* attributes correspond to weights on the nodes of a Moore-machine style FSM and *repeat-prob* attributes correspond to weights on backward-facing arcs of a Mealy-machine style FSM. This characteristic voids the possibility of utilizing standard Mealy-Moore conversions on an XML-SRGS grammar and complicates the XML-SRGS to accepting FSM translation. An XML-SRGS grammar for the FSM in Fig. 3 using both the *weight* attribute and the *repeat-prob* attribute is shown below:

```
<grammar>
 <rule>
  <item> a </item>
  <one-of>
   <item weight='4'> b </item>
  </one-of>
  <item repeat='1-' repeat_prob='.43'> c </item>
 </rule>
</grammar>
```

The first item in this grammar, a, has no weight since its incoming arc has no weight. The second item, b, has a weight on its node which may be interpreted as a weight on the transition from a to b. The SRGS dictates that an item may not have a weight unless its immediate enclosing tag is a <one-of>. SRGS does, however, allow for a single item to be enclosed in a <one-of> in order to specify a weight where one would not otherwise be allowed. Since speech recognition systems typically have weights on all arcs, this limitation on the SRGS *weight* attribute is significant.

Also important, the *repeat-prob* attribute can only be used with the SRGS *repeat* looping attribute. Recall this attribute implements the EBNF loop extensions for Kleene operations. In this example, "repeat = '1-'" is equivalent to +. An additional limitation is that repeat probability values must lie between 0.0 and 1.0. This complicates matters for recognizers that utilize logarithmic probabilities.

If recursion is allowed, grammars may be authored in XML-SRGS without this dual Mealy/Moore nature in favor of a strict Moore-style FSM. The grammar in Fig. 3 would be represented as shown below without repeat probabilities, but with recursive rules:

```
<grammar>
  <rule>
    <item> a </item>
     <one-of>
        <item weight='4'> <ruleref uri="#B"/> </item>
     </one-of>
    </rule>
    <rule id="B">
    <item> b </item>
          <item> <ruleref uri="#C"/> </item>
    </rule>
```

```
<rule id="C">
 <item> c </item>
 <one-of>
    <item weight='.43'> <ruleref uri="#C"/> </item>
    <item> <ruleref special="NULL"> </item>
</one-of>
</rule>
</grammar>
```

This avoids the use of a repeat probability on node 'c' by putting a weight on the recursive rule reference at the end of this grammar.

3. SOFTWARE ARCHITECTURE

Most industry standard grammar specifications, such as JSGF and XML-SRGS, cannot be directly converted into a finite state machine representation to be used for speech recognition purposes. In order to convert to a finite state machine, we must convert the high-level grammar to the lowest level of the Chomsky hierarchy. We represent this form using normalized BNF which consists of the following rule types:

$A \rightarrow a,B$
$A \rightarrow B$
$A \rightarrow \varepsilon$

Where 'A' and 'B' are non-terminal symbols, 'a' is a terminal symbol, and 'ε' is the epsilon symbol. From this lowest-level representation, the conversion to a finite state machine representation is relatively straightforward. It is also easy to directly convert to a high-level grammar representation.

Rather than implement conversions directly from high-level representations to normalized BNF, we chose to use an intermediate ABNF format. This approach has two main advantages. The first is that JSGF and XML-SRGS were both designed for easy mapping to an ABNF. The second is that classic algorithms exist for conversion from ABNF to normalized BNF.

Once the underlying theoretical structures of each format were understood in detail, it was clear that producing a verifiably robust conversion tool required a software process with specific modules, corresponding to the theoretical stages of the conversion. The first stage would create a common ABNF grammar format to which any other format could be converted; the ABNF-SRGS was an obvious choice. The next stage would entail processing the ABNF to remove extensions and thus standardize representation of weights and probabilities. Thus, we designed our conversion process to include the following steps and corresponding software modules: 1) convert the XML-SRGS, or JSGF to an equivalent ABNF, 2) convert the ABNF to remove the EBNF extensions and produce a clean BNF with or without recursion, 3) convert the BNF to XML-SRGS, JSGF, or IHD. The redesigned conversion process is shown in Fig. 4.

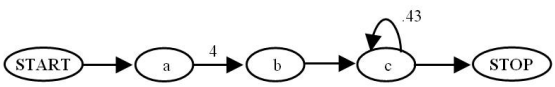

Fig. 3. FSM for XML Weights and Probabilities

3.1 JSGF/XML-SRGS to ABNF Conversion

Clearly, a primary challenge in converting from JSGF or XML-SRGS to ABNF is the transformation of syntax. As previously noted, the JSGF syntax is very similar to ABNF and this transformation is trivial. Transformation of the XML-SRGS syntax, however, requires some explanation. The basic XML-SRGS tokens and rule references map easily to ABNF terminals and non-terminals. XML-SRGS <item/> tags are used to denote grouping; as such, they map well to ABNF parentheses.

The XML-SRGS markup attributes, however, do not map so clearly to ABNF. While the repeat='0-' and repeat='1-' item attributes can be converted into the Kleene star and plus, respectively, XML-SRGS also allows for repeat attributes such as repeat='m-n'. Such detail cannot be represented in ABNF. Thus, conversion from XML-SRGS includes the duplication of such repeats into enumerated alternatives, as illustrated in Fig. 5.

As mentioned in Section 0, if recursion is allowed, XML-SRGS may be viewed as describing FSMs with weights placed on items, rather than transitions. Likewise, JSGF places weights on items. This corresponds to a Moore finite state machine [10]. However, the finite state machines used in IHD and common to many speech recognition systems are of the Mealy variety, with weights on arcs. Hence, a conversion is necessary.

We chose to implement this Moore to Mealy machine transformation during the conversion to ABNF due to the differences in weight specification techniques between JSGF and XML-SRGS. While JSGF is an exclusively Moore specification, XML-SRGS, as previously discussed, additionally allows for the attachment of probabilities to the conceptual equivalent of Kleene closures, repeat attributes. This added complexity requires that the JSGF and XML-SRGS Mealy to Moore conversions be executed in separate modules.

3.2 ABNF to BNF conversion

ABNF uses regular expression operators to avoid the

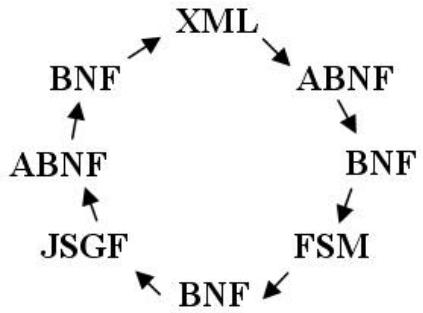

Fig. 4. Conversion Design

XML-SRGS

```
<item repeat='1-2'>
    a
    b
</item>
```

ABNF

```
<S>::=(ab)|(abab)
```

FSM

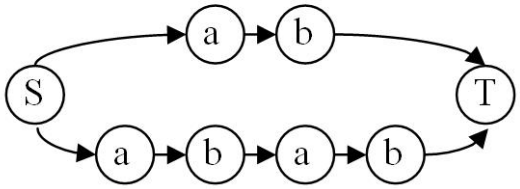

Fig. 5. Enumerated alternatives

recursion inherent in most BNF grammars. Thus, the conversion from ABNF to BNF will, in general, increase the number of productions. This conversion is primarily concerned with the simplification of ABNF rules containing regular expression operators into combinations of BNF productions. Table 1 illustrates a few examples of ABNF grammars and their BNF equivalents.

Our developed algorithm for this conversion is centered around the expression of a grammar through description of its transitions. The first step in each iteration of this algorithm is to find a transition corresponding to the concatenation operator, the Kleene star, or the Kleene plus. Once a transition is located, conversion proceeds through determination of the set of symbols to the left and the set of symbols to the right of the transition (i.e., the "to"s and "from"s). Treatment of each transition in this manner results in a complete description of the associated grammar in a simple BNF form for easy conversion to an accepting FSM.

In our system, this conversion operates by examining each concatenation token, Kleene star token, and Kleene plus token in all nesting levels of a given production and determining what, if any, transition it implies. Determination of involved symbols is aided by the recursive functions, "findLeftSymbols" and "findRightSymbols", which return sets containing the left and right symbols for a given transition token. For example, the conversion of **a,b,((c,d)|(e,f))+** begins with the examination of the first concatenation symbol. In this case, findLeftSymbols and findRightSymbols return "a" and "b", respectively. The next treated symbol is the second concatenation symbol. In this case, findLeftSymbols returns "b" and findRightSymbols recursively discovers and returns "c" and "e". The third and fourth concatenation symbols are straightforward in the manner of the first. The final treated symbol is the Kleene plus. In this case, findLeftSymbols and findRightSymbols are both called on the preceding expression. Then, rules are created to indicate transitions from each right symbol to each left symbol. This example is illustrated in Fig 6.

Table 11. ABNF-BNF equivalence

ABNF	**BNF**
S::= A | B	S::=A
	S::=B
S::=(A)+	S::=A1
	A1::=A, A1
	A1::=A
S::=(A)*	S::=A1
	A1::=A, A1
	A1::= ε

Although this example does not include multiple rules or non-terminal symbols, the inclusion of these items is a simple matter. For each occurrence of a non-terminal symbol, the findLeftSymbol or findRightSymbol method, as the case may be, should be called upon the expression corresponding to the expansion of the given non-terminal.

3.3 BNF to FSM conversion

Once a grammar is in a BNF representation, conversion to a finite-state machine is straightforward. This conversion involves scanning the set of BNF rules, and identifying unique non-terminal/terminal symbol pairs. These pairs correspond to nodes in the resulting FSM. The arcs of the FSM are derived from the concatenation operators between terminal and non-terminal symbols. For instance, the FSM of Fig. 7 can be represented by the following normalized BNF:

S::=A
A::=**a**,B
B::=**b**,C
B::=b,A
C::=**c**,T
T::=ε

The non-terminal/terminal symbol pairs are in bold to show that three unique nodes have been identified in this graph. The concatenation operators are represented by the ',' character, and translate into arcs.

4. SUMMARY AND CONCLUSIONS

Supporting popular CFG-based language model formats has been an important priority in our research. The important nuances of each CFG format implementation have presented equal challenges to producing robust conversion tools. Further, assignment of weights and probabilities for these styles must be carefully considered in converting to other formats, including CFG or regular grammars. We have addressed these issues by incorporating additional stages and corresponding software modules in our process which perform generic conversions to and from common EBNF and BNF grammar formats. These enhancements have advanced an important goal for the speech research community — producing verifiably robust conversion tools to support popular CFG-based language model standards.

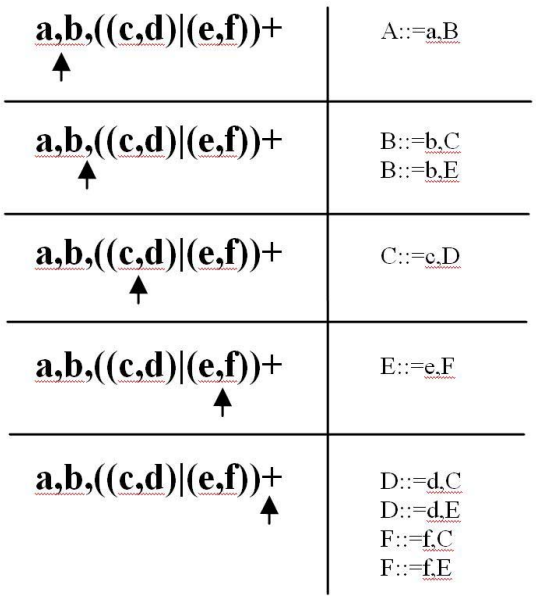

Fig. 6. ABNF to BNF example

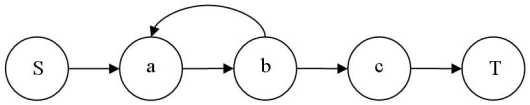

Fig. 7. FSM for (a,b)+c

The software modules developed for this research are in the public domain, and numerous tools in our speech recognition toolkit currently utilize these conversions. Related software in our toolkit includes a graphical language network design tool, which allows construction of a language model in all of the formats of Section 0, and a language model tester, which tests for equality of language models in varying formats.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. IIS-0414450. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Java Speech Grammar Format Specification, Version 1, Sun Microsystems Developer Network, October 26, 1998 (see http://java.sun.com/products/java-media/speech/forDevelopers/ JSGF/JSGF).

[2] A. Hunt and S. McGlashan, Eds., W3C Speech Recognition Grammar Specification Version 1.0, March 16, 2004 (see http://www.w3.org/TR/speech-grammar/).

[3] J. Picone, et al., "A Public Domain C++ Speech Recognition Toolkit,", ISIP, Mississippi State University, Mississippi State, MS, USA, March 2003.

[4] N. Chomsky, "On Certain Formal Properties of Grammars," *Info. and Control*, Vol. 2, 1959, pp. 137-167.

[5] M. Mohri, "Weighted Grammar Tools: The GRM Library," in J.C. Junqua and G. van Noord (eds), *Robustness in Language and Speech Technology*, .Kluwer Academic Publishers, 2000.

[6] J. Schalkwyk, L. Hetherington, and E. Story, "Speech Recognition with Dynamic Grammars Using Finite-State Transducers," *Eurospeech 2003*, pp. 1969-1972, 2003.

[7] "Converting JSGF to SRGS," http://publib.boulder.ibm.com/ infocenter/pvcvoice/51x/index.jsp?topic=/com.ibm.voicetools.grammar.doc/tgrj2srgs.html, IBM Corporation, White Plains, New York, USA, April 2006.

[8] P. Naur, "Revised Report on the Algorithmic Language Algol 60," *Com. of the ACM*, Vol. 7, No. 12, pp. 735–736, 1963.

[9] N. Wirth, "What Can We Do About the Unnecessary Diversity of Notation for Syntactic Definitions," *Com. of the ACM*, Vol. 20, No. 11, pp. 822–823, 1977.

[10] J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 2001.