Speech Recognition with Dynamic Grammars Using Finite-State Transducers

Johan Schalkwyk, Lee Hetherington[†] *and Ezra Story*

SpeechWorks International
Boston, MA USA
{johans,ezra}@speechworks.com

[†]Spoken Language Systems Group
MIT Laboratory for Computer Science
Cambridge, MA USA
ilh@mit.edu

Abstract

Spoken language systems, ranging from interactive voice response (IVR) to mixed-initiative conversational systems, make use of a wide range of recognition grammars and vocabularies. The recognition grammars are either static (created at design time) or dynamic (dependent on database lookup at run time). This paper examines the compilation of recognition grammars with an emphasis on the dynamic (changing) properties of the grammar and how these relate to context-dependent speech recognizers. By casting the problem in the algebra of finite-state transducers (FSTs) we can use the composition operator for fast-and-efficient compilation and splicing of dynamic recognition grammars within the context of a larger precompiled static grammar.

1. Introduction

Spoken language systems, ranging from interactive voice response (IVR) to mixed-initiative conversational systems, make use of a wide range of recognition grammars and vocabularies. The recognition grammars are either static (created at design time) or dynamic (dependent on database lookup at run time). This paper examines the compilation of recognition grammars with an emphasis on the dynamic (changing) properties of the grammar and how these relate to context-dependent speech recognizers.

We present an efficient technique that addresses dynamic changes of a grammar, while preserving cross-word context-dependent constraints (section 3). By casting the problem in the algebra of finite-state transducers (FSTs) we can use the composition operator for fast-and-efficient compilation and splicing of dynamic recognition grammars while still preserving the context (e.g., phonology, triphonic models etc…) of a larger precompiled static grammar.

Through a mechanism referred to as late-binding (section 4.2) we show how this algorithm gives flexibility to the design of complex grammars resulting in large memory savings across multiple channels of an application.

2. Finite-State Representation

Many of the components of a speech recognizer can be represented by a weighted finite-state transducer [1, 2]. This includes the grammar, lexicon, phonological rules, context-dependent expansions, and associated acoustic models. Each component introduces a finer-level description of the language moving from words (G), through phonemes (L), phones (P), and finally the context-dependent model labels (C).

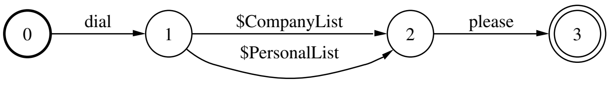

Figure 1: *Name Dialing grammar G with dynamic splice points.*

Formally this cascade can be described as a sequence of compositions of finite-state transducers:

$$R = C \circ P \circ L \circ G\ . \tag{1}$$

Within this framework the grammar (G) represents the finite-state expansion of the context-free language. Except for a particular subset of context-free rules (those that cannot be factored into left-linear and right-linear partitions), most grammars written using context-free rules can be converted to a finite-state acceptor. Grammars that cannot be converted directly to a finite-state representation can be approximated using finite-depth recursion.

3. Dynamic Grammars

For illustration purposes we concentrate on a name dialing application consisting of a large static component (company wide address book) and a smaller dynamic component (personal address book). This particular recognition grammar can be expressed as a set of context-free rules using the W3C XML grammar specification language [3]:

```
<?xml version="1.0"?>
 <rule id="ROOT" scope="public">
   dial
    <one-of>
      <item> <ruleref uri="#CompanyList"> </item>
      <item> <ruleref uri="#PersonalList"> </item>
    </one-of>
   please
 </rule>
 <rule id="CompanyList">
   <one-of>
    <item> Steve </item>
    <item> Jim   </item>
   </one-of>
 <rule>
</grammar>
```

Figure 1 depicts a finite-state expansion of the example name dialing grammar. Using either reverse lookup (ANI) of the end-users telephone number or a user specific access ID, the

[†]Lee Hetherington was sponsored in part by industrial consortia supporting the MIT Oxygen Alliance and the MIT Spoken Language Systems Group Affiliates Program.

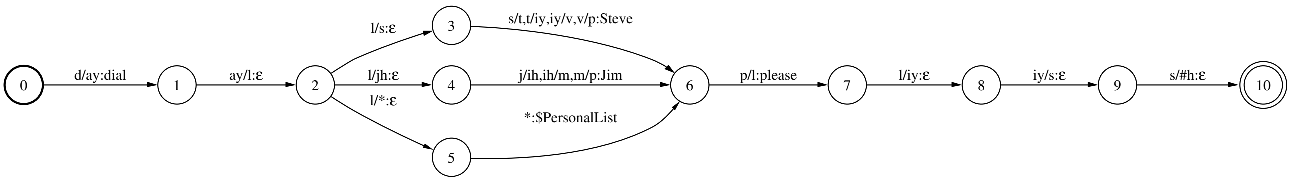

Figure 2: *Expanded right context: $C_r LG$. The arcs for "Steve" and "Jim" have been condensed to save space in the figure.*

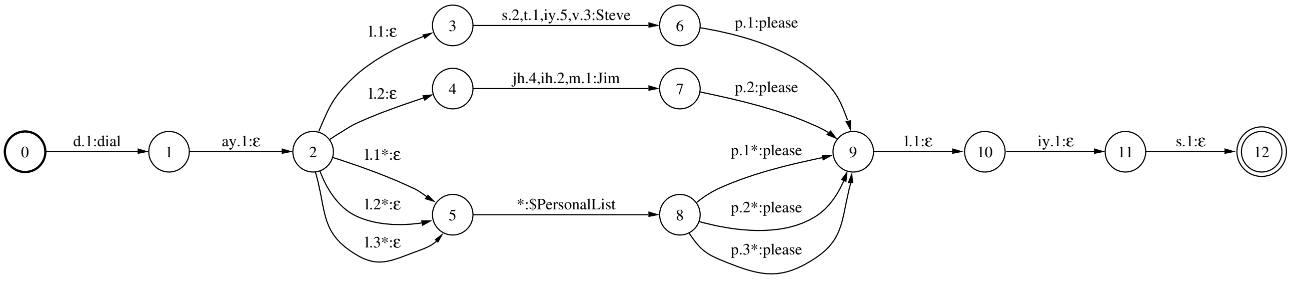

Figure 3: *Left context resolved and mapped to model labels using decision tree: $C'LG = C_l \circ rev(C_r LG)$.*

system will load and compile the personal address book component of the grammar. The dynamic component is then bound at run time with the static preloaded component.

Since this grammar is dependent on a dynamic component (personal address book), it cannot be fully expanded into a flat weighted finite-state transducer. Due to the potentially large size of the static component (e.g., company wide address book that could have tens of thousands of entries) we would like to partially compile the grammar before recognition time in order to minimize grammar loading latency. Therefore the compilation of the static component has to take into account the run-time splicing of an arbitrary dynamic component, and furthermore the splicing needs to preserve the context-dependent cross-word effects (e.g., phonology and modeling).

Since both the acoustic modeling C and the phonology P use context dependency, all context dependency within the recognition cascade is contained within $C \circ P$. Without loss of generality, we can consider all context dependency to be contained within a single FST $C' = C \circ P$. In this case, C' maps from phonemes to context-dependent acoustic model labels.

In general, C' will contain both left and right context dependency, and we have found that factoring

$$C' = C_l \circ C_r \tag{2}$$

to aid in the efficiency of applying C' to $L \circ G$, most importantly for reducing latency in constructing dynamic subgrammars at run time. Here, C_r encodes the right contextual effects and C_l encodes the left contextual effects (similar to the phonological rule transducer decomposition described in [4]).

Referring back to Figure 1, we need to expand the right context of the word "dial" to incorporate the known expansions of the words in the company wide address book, as well as the unknown right contextual expansions of the words in the personal address book. C_r scans the transducer $L \circ G$ from right to left, labeling each arc in $L \circ G$ with the associated right context, including unspecified right contexts. We can accomplish this *right-to-left* composition as follows:

$$C_r LG = \mathrm{rev}(\mathrm{rev}(C_r) \circ \mathrm{rev}(L \circ G)) \ , \tag{3}$$

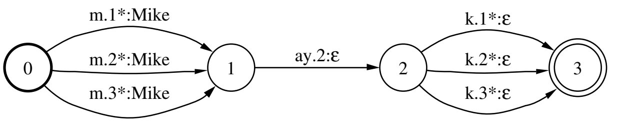

Figure 4: *Context-Dependent expansion of dynamic component* **$PersonalList**.

where rev$(\cdot)$ reverses a transducer. Figure 2 depicts the first phase of the context-dependent labeling process, resolution of the right context by composing with C_r. Note the presence of the unknown context indicated using the ***** (star) phoneme in **l.***.

Similarly, we need to expand the left context, and for the word "please" this will have to account for the known and dynamic words to the left. This is accomplished by composing transducer C_l, which scans from left to right, labeling each arc with the associated left context:

$$C'LG = C_l \circ C_r LG \ . \tag{4}$$

Figure 3 depicts the result of composing with C_l, yielding $C'LG$, and at this point both left and right context are resolved where possible. In the figure, the labels *phone.i* refer to the numbered leaves of an associated phonemic decision tree. Since the right context of the phone **l** is unknown at state 2, all possible realizations of **l** with left context **ay** are present. Likewise, at state 9, the left context of the phone **p** is unknown, and all possible realizations of **p** with right context **l** are present.

Equation 4 is repeated for each component in the grammar, including the dynamic run-time components (e.g., **$PersonalList**). Figure 4 depicts this expansion for an example personal entry "Mike." Both the left and right contextual effects of the dynamic component are unknown (indicated using the *****) at the start and end, and the proper connections will be made at run time.

At run time we need to resolve the cross-word transitions at the boundaries of dynamic grammar components. For example, in Figure 3 we need to select the appropriate context-dependent models to use leaving state 2 and arriving at state 9, and in Figure 4 the appropriate models leaving the initial state 0 and arriv-

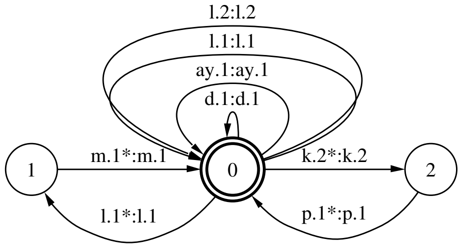

Figure 5: *Cross-word constraint connection machine.*

ing at the final state 3. Simply splicing in a subgrammar component (on-the-fly) allows for inappropriate connections, which can be filtered by composition with an enforcement transducer E:

$$R = E \circ \text{splice}(C'LG, D) \ . \tag{5}$$

Here, the on-the-fly operation splice$(C'LG, D)$ resolves dynamic subgrammar references in $C'LG$ using a dictionary of subgrammars D and splices them into place when needed in a context-*independent* manner, similar to how a recursive transition network (RTN) operates. On-the-fly composition with the special-purpose transducer E enforces that appropriate cross-word connections are created. E is designed as a pass-through of all context-dependent models that are fully specified, but at the same time allows only compatible sequences of partially specified word-boundary arcs. For example, Figure 5 shows a portion of such a transducer E. It passes through fully specified (non-*) labels but places constraints on partially specified (*) labels. For example, **k.2* p.1*** is a legal sequence that produces the fully resolved sequence **k.2 p.1**, but **k.2* p.2*** is not legal and produces no label sequence. The net result of $E \circ$ splice$(C'LG, D)$ is the context-*dependent* splicing of dynamic subgrammar components, and this is performed on the fly during recognition.

4. Implementation Considerations

4.1. Recursive Transition Networks (RTNs)

In the preceding discussion the grammar G represents a finite-state expansion of the language. An alternative to full expansion is to do this process on the fly. In one such approach [5], the process of construction of a recognition graph is deferred until a particular non-terminal is encountered during recognition. Here a recursive transition network (RTN) is formed. The RTN includes a separate graph for each non-terminal and the paths through the graph represent the possible sequences of elements (terminals and non-terminals). At recognition time when a non-terminal is encountered on an arc, a recursive "call" is made for that non-terminal. Through these recursive calls allowable word sequences are formed without having to expand the grammar into a single overall network.

Full expansion of a word-level grammar into a finite-state transducer that accepts context-dependent phonemes prior to recognition can significantly reduce the amount of computation required during the recognition phase [6]. Also when the grammar is fully expanded we can use standard finite-state optimization techniques (e.g., determinization, minimization, and weight pushing) to further improve the computation/memory trade-off. However in many cases the fully expanded grammar may be too large.

On the other hand, recognition based on run-time expansion of the word-level graph requires additional CPU time to do the expansion since the run-time graph is not fully minimized and not necessarily deterministic. The advantage of delayed expansion is a compact representation of the language constraints.

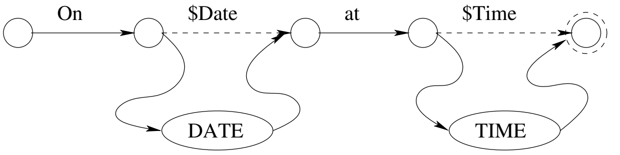

Figure 6: *Recursive call within a FSM grammar to a precompiled Date grammar.*

The proposed algorithm for compiling dynamic grammars where the context-dependent constraints are encoded into the arcs of the edge parts of the machine provides a way of combining aspects of pre-computation of context-dependent phoneme graphs as well as dynamic processing of grammar constraints to provide a configurable trade-off between data size and recognition-time computation.

In principal this implementation corresponds to a standard RTN network, but instead of a stack recursion into a particular rule, this approach allows a stack recursion into a fully expanded graph (for example a precompiled date or time grammar, see Figure 6). Each component of the grammar, the top-level carrier phrase grammar as well as low level components like date, time, and personal address book phrase structured grammars, can be compiled and optimized separately, with the inclusion of the cross-word context-dependent phoneme constraints. Thus, an RTN representation can be used in place of the splice operator of Equation 5.

4.2. Late Binding

In our name dialing example, we may be running a system which runs this application on many channels at the same time. Each channel will be simultaneously processing a different caller. Using the RTN and the delayed compilation techniques described above, we can bind an external personal addressbook grammar to the appropriate rule in the top level name dialer grammar for each caller. For a large system, we want to do this in such a way that we share the potentially large top-level grammar amongst all the callers to the system without having to pay the computational cost of recompiling it and the cost of having duplicates of the FST in memory.

The primary difficulty in doing so is that the top-level and the external component to be bound in have no knowledge of one another when compiled. This lack of knowledge is reflected in the mapping of the output labels of the grammar transducers to vocabulary and rule transitions. Each component is built with a different alphabet of arc labels, which corresponds to different namespaces mapping strings to integer indices within the implementation.

A potential solution is to share a master namespace, or vocabulary list, amongst all grammars on a system. This is not practical for systems where generic components (e.g., currency subgrammar) may be bound into many different top-level grammars which are not known apriori (e.g., account transfer and stock purchase). In addition, when dealing with systems with a dynamically generated lexicon, the list of possible words and combinations of words (tokens) is limitless.

Our solution is to create a thin "shim" layer which when placed on top of an FST, maps from one namespace to another.

In practice, this shim layer can be an order of magnitude smaller than the FST itself because it is populated on the fly and only contains those states and arcs encountered during a particular recognition run. This layer allows an FST to work within the context of a different namespace than it was compiled with, without paying the time and memory cost of either recompiling the FST or duplicating the entire FST and relabeling it.

The late binding process then consists of merging the namespaces assosiated with each component grammar together into a merged namespace, and then placing a shim over each component to map labels from its namespace to the merged namespace. In this way, a large top level grammar can be shared among all the simultaneous callers to a system without having to duplicate the grammar FST. The same holds true for large component grammars, such as a currency grammar being bound into a top-level checking account grammar.

4.3. Context-Dependent Splicing without Composition

As discussed in previously in section 3, we perform a context-independent splice of a dynamic subgrammar into its carrier grammar and then enforce context-dependent constraints by on-the-fly composition with the enforcement transducer E. One consequence of this approach is that every state and arc traversed during recognition must be dynamically created on the fly.

In general, dynamically created arcs can be more costly than simply returning the static arcs of a static FST. There might be significant efficiency gains if the portions of the top-level grammar component that do not immediately border dynamic subcomponents could be served to the Viterbi decoder simply by returning static arcs as opposed to dynamically creating arcs. Obviously, with the use of any dynamic grammar subcomponents not all arcs can be static, but perhaps a large fraction of the states could yield static arcs. How much benefit can be realized from this approach depends on the grammars involved. If the top-level grammar is relatively small compared to the dynamic subcomponents (e.g., small carrier grammar and large dynamic word list subgrammar), most arcs would need to be dynamic. However, if the top-level grammar is relatively large (e.g., an n-gram where there are relatively few dynamic word-classes) and late binding is not in use, then a large fraction of the resulting arcs could remain static. (If the technique of late binding discussed in section 4.2 is utilized, all arcs would need to remain dynamic due to the mapping of label indices.)

One of us (Hetherington) has implemented direct context-dependent FST splicing that enforces context-dependent marker label constraints internally (i.e., not using FST composition per se) and produces dynamic arcs only where necessary. We have performed some comparisons of constraint enforcement by composition vs. by such code within the context of the MIT SUMMIT speech recognition system running within the Jupiter weather information access domain [7]. With a top-level bigram containing 1183 static words and 3 dynamic word classes ($City, $City-State, and $State) containing a total of 1495 words, we found that during recognition, only 40% of the arcs accessed need to be created dynamically. The other 60% can be passed directly from the static top-level grammar. Within the SUMMIT system, this yields a recognition speed 18% faster than if on-the-fly composition with E is used to enforce constraints after performing context-independent splicing. In this case, we find that using the three dynamic word classes is only 1% slower than if all three were statically compiled into the top-level bigram grammar. Thus, the technique of context-dependent FST splicing appears to work well even for relatively large top-level grammars such as n-grams.

5. Conclusions

Through-out this paper we have concentrated on finding generic solutions that fit with-in the finite-state transducer framework. This methodology has the further advantage that it hides the complexity of dynamic cross-word connections, reusable grammar components and namespace resolution from the Viterbi decoder. Since the output is always a finite-state transducer the algorithms presented above can be run without any modifications to the decoder. Due to the pruning properties of the Viterbi decoder only a small fraction of the dynamically expanded graph is traversed.

Dynamic grammars are an important aspect of spoken language systems. As shown in our example dynamic grammars may require a mixture of precompiled components and user-specific components, where the latter is only known at run time. More complex dialogue design such as "one-step correction"[†] requires the ability to quickly merge a confirmation grammar with a previously activated collection grammar (such as city names). The demands of these scenarios requires efficient methods to compile, combine and re-use grammar components. By casting the problem in the algebra of finite-state transducers we can address these problems in an efficient and flexible manner.

6. Acknowledgements

The author's would like to thank Michael Riley and Mehryar Mohri for fruitful discussions about finite-state techniques. An international patent application for the methods related to dynamic grammars (section 3) was published on January 9, 2003 (PCT WO 03/005345(A1)).

7. References

[1] M. Mohri and M. Riley, "Integrated context-dependent networks in very large vocabulary speech recognition," in *Proc. European Conf. Speech Communication and Technology*, Budapest, Sept. 1999, pp. 811–814.

[2] M. Mohri and M. Riley, "Network optimizations for large vocabulary speech recognition," *Speech Communication*, vol. 28, pp. 1–12, May 1999.

[3] A. Hunt and S. McGlashan, "Speech recognition grammar specification version 1.0," http://www.w3.org/TR/speech-grammar, June 2002.

[4] L. Hetherington, "An efficient implementation of phonological rules using finite-state transducers," in *Proc. European Conf. Speech Communication and Technology*, Aalborg, Sept. 2001, pp. 1599–1602.

[5] M. K. Brown and S. C. Glinski, "Context-free large vocabulary connected speech recognition," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, Adelaide, Apr. 1994, vol. 2, pp. 145–148.

[6] S. Kanthak, H. Ney, M. Riley, and M. Mohri, "A comparison of two LVR search optimization techniques," in *Proc. Int. Conf. Spoken Language Processing*, Denver, Sept. 2002, pp. 1309–1312.

[7] V. Zue, S. Seneff, J. Glass, J. Polifroni, C. Pao, T. J. Hazen, and L. Hetherington, "Jupiter: a telephone-based conversational interface for weather information," *IEEE Trans. Speech and Audio Processing*, vol. 8, no. 1, pp. 85–96, Jan. 2000.

[†] The term one-step correction refers to the scenario where the spoken language system allows correction during the confirmation phase of the dialogue, e.g. No, "I want to fly to Boston", as apposed to "Austin".