Optimizing an engine network that allows dynamic masking

Frédéric Tendeau
frederic.tendeau@acm.org

Abstract

We present a method to compile and fully determinize & minimize speech recognition networks where any relevant sub-network can be switched on/off efficiently at runtime. We show how the general algorithm for determinizing general weighted transducers can be applied for this task.¹

1. Introduction

Dialogue applications allow the machine to ask questions to the speaker, e.g. when an ambiguity arises or in order to refine a particular request progressively. In this way, the machine gathers information during earlier stages of the dialogue and can exploit this, among other things, to improve the performances of the speech recognition. One way to improve the accuracy and the speed is to lower the perplexity and that is the purpose of dynamic masking: some paths are momentarily disconnected, which simplifies the searching for the best recognition path. For example, in an automatic switchboard application: once you know in which department the sought person works, you can mask the names belonging to all other departments. The application can use the grammar to point out the parts to disconnect.

In many dialogue applications, spoken utterances are modelled by a Context-Free Grammar (CFG). The VoiceXML Speech Recognition Grammar Format is a CFG format: ABNF and its XML twin. Most of the time, the expressive power of these CFGs is limited to the definition of regular languages by allowing only one kind of recursion (left or right) so that the dialogue grammar is easily compiled into a finite-state machine² (FSM) which is the network of the speech recognition engine, the device on which the search algorithm runs to perform the recognition.

In this context, dynamic masking consists in switching on/off some pieces of the FSM by removing/re-inserting some transitions directly on the network. A masking instruction refers to some non-terminal of the CFG: when A is masked, its contribution momentarily disappears from the network, as if it was not present in the CFG at compilation.

Dynamic masking should not be confused with dynamic activation that is used in VoiceXML [7] and for which a neat definition is given in [5], together with appropriate algorithms for compilation and optimization. Dynamic activation consists in adding new start symbols to the CFG at runtime, extending its language momentarily; another important difference with masking is that when de-activated, the contribution of a non-terminal remains in the grammar everywhere it appears in the right-hand side of a rule —it disappears only from the top-level, the root level in VXML terminology. Optimization techniques borrowed from finite-state technology have proven to be extremely effective in speech recognition because they improve dramatically the three most important characteristics of an engine: speed, footprint and accuracy [1], [2], [4]. The speed improvement is obtained thanks to determinization: in each state and for every word, there is at most one possible state to go to; the reduction of footprint results from minimization (and practically also from determinization); and the accuracy improvement is the benefit of both determinization and minimization: determination guarantees that every sentence prefix is considered in at most one search hypothesis and minimization maximizes the number of recombinations, therefore, the set of hypotheses (which is bounded in size) is used optimally.

Finite-state optimization and dynamic masking are very often considered as antagonist: a piece of engine network that is enabled for dynamic masking is generally not seen as optimizable in the speech recognition community, or only for pieces that always have the same masking status which heavily limits the practical effectiveness.

The purpose of this article is to show how Mohri’s algorithm [2] can be applied in order to fully determinize an engine network where dynamic masking is possible. First we address the compilation of grammars into maskable networks then we detail an algebra of masks and we show that it meets the requirements for using the general determinization algorithm over finite-state transducers. A discussion and concluding remarks follow.

2. Compilation of masks

We are going to adapt the algorithm proposed by Nederhof [6], which compiles a CFG into a finite-state automaton (FSA). We want to compile a CFG into a finite-state transducer (FST) with boolean labels on the transitions: those labels carry the on/off status of the mask. We call Ω the domain for boolean labels and we use the operator ⊗ to compose (accumulate) masks; \( \overline{\mathbb{T}} \in \Omega \) is neutral for \( \otimes \).

¹This work has been carried out in the speech recognition lab. of Dialoca in Brussels, recently bankrupted.
²In fact, the CFG formalism remains useful for practical reasons. It allows for an easy specification of the semantic extraction from the spoken utterances. It is also very convenient to isolate sub-grammars thus to design modular grammar components.
Let’s consider a CFG \((S, N, \Sigma, P)\) where \(S\) is the start symbol, \(N\), the set of non-terminals, \(\Sigma\) the alphabet and \(P\) the set of context-free rules. Let \(V\) be the vocabulary \(\Sigma \cup N\).

Each non-terminal \(A\) is associated with a predicate \(p_A\) that reflects its masking status at runtime. When \(A\) is not maskable then \(p_A\) equals the constant value \texttt{true}. We are going to tag each transition of the network with a boolean label that holds its masking status: an expression consisting of \(p_A\) predicates connected with \texttt{or} and \texttt{and} operations. Formally, if we call \(\Pi\) the set \(\{p_A | A \in N\}\), then the domain of masking tags is \(\Omega = 2^\Pi\), i.e. a transition is tagged by a set of subsets of \(\Pi\), which represents a predicate expression in disjunctive (normal) form. For example the tag \(\{\{p_A, p_B\}, \{p_A, p_C\}\} \in \Omega\) means that the associated transition is \texttt{on} if non-terminals \(A\) and \(B\) are \texttt{on}, or \(A\) and \(C\) are \texttt{on}: the innermost commas stand for \texttt{and}, the outermost, for \texttt{or}. We will show later that the mask \(\{(\texttt{true})\}\) is actually \(\top\).

We compile a grammar into a finite-state transducer \((Q, q_i, F, \Sigma, \epsilon, \Delta)\), where \(Q\) is the set of states, \(q_i \in Q\) is the initial state, \(F \subseteq Q\) is the subset of final states, \(\epsilon\) is the empty word and \(\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Omega \times Q\) the set of transitions, carrying both a word and a mask. In some cases (a rule \(A \rightarrow \epsilon\) or in case of recursion), \(\epsilon\) can arise on transitions but removing it is a classical operation over FSMs, that involves \(\otimes\) in the case of FSTs.

Figure 1 shows the compilation algorithm. It assumes that we have a non-recursive CFG. Recursions are an issue for compiling a CFG into a FSA but, once this is solved, see [6], adding masks is easy. So the only reason for this limitation here is the sake of briefness and clarity: the FSA algorithm we rely on is much simpler. Figure 2 shows the example of a grammar that is compiled in Figure 3 into the non-deterministic transducer \(T_G\), considering that only \(A\) and \(B\) are maskable. This compilation runs as follows: line 1 creates states 1 (\(= q_i\)) and 2; line 2 makes state 2 final; line 4 calls compile (1, \(S, \top, 2\)). The if statement triggers line 12, which considers the only rule with \(S\), hence line 13 calls compile (1, \(\text{Open Names Close}, \top \otimes \top = \top, 2\)). Because \(\alpha\) consists of at least 2 symbols, the condition line 7 holds with \(X = \text{Open}, Y = \text{Names}, \beta = \text{Close}\) then line 8 adds state 3, line 9 calls compile (1, \(\text{Open}, \top, 3\); we skip this call. Line 10 calls compile (3, \(\text{Names Close}, \top, 2\)). Again line 7 triggers, but with \(X = \text{Names}, Y = \text{Close}, \beta = \epsilon\) hence state 5 is added and line 10: compile (3, \(\text{Names}, \top, 5\)); line 11 wins and line 12 enumerates the three alternatives for \(\text{Names}\); the first is \(A\) thus compile (3, \(A, \top, 5\) ) triggers line 11 again, with Jim Jackson as first alternative therefore, line 13 calls compile (3, \(\text{Jim Jackson}, \top \otimes \{(p_A)\} = \{(p_A)\}, 5\)), then line 7 fires, line 8 adds state 6 and line 9: compile (3, \(\text{Jim}, \{(p_A)\}, 6\) which adds, line 6, the transition between states 3 and 6 with Jim and mask \(\{(p_A)\}\). Etc.

```
procedure main
  1  Q ← \{q_i, q_f\}
  2  F ← \{q_f\}
  3  Δ ← ∅
  4  compile (q_i, S, T, q_f)

procedure compile (q ∈ Q, α ∈ V*, m ∈ Ω, q’ ∈ Q) is
  5  if α = ε or α = a ∈ Σ then
  6      add transition (q, α, m, q’) to Δ
  7  elseif α = XYβ with X, Y ∈ V and β ∈ V* then
  8      add new state q” to Q
  9      compile (q, X, m, q”)
 10     compile (q”, Yβ, T, q’)
 11  else a = A ∈ N
 12     for each A → γ ∈ P do
 13        compile (q, γ, m ⊔ \{(p_A)\}, q’)
```

Figure 1: Grammar compilation

```
S    =  Open Names Close
Open =  pass me | ε
Close =  please | ε
Names =  A | B | C
A     =  Jim Jackson | Sue Smith
B     =  Sue Smith
C     =  Jim Moore

Figure 2: Grammar G

Figure 3: Non-deterministic transducer T_G
```
3. Determinization with masks

3.1. Semiring

We want to define ⊕ and ⊗ operations in Ω in such a way that our example \{\{p_A, p_B\}, \{p_A, p_C\}\} can also be written \(p_A \odot p_B \odot p_A \odot p_C\)—assuming \(\odot\) has priority over \(\odot\).

Let’s start with the innermost level of masks: the structure \(2^{|\Pi|}, \cup, 0\) is the commutative and-monoid that we call \(\mathcal{M}\). For full masks, we introduce the or by taking the power-set generalization: \((2^{|\Pi|}, \odot, \emptyset, \{\emptyset\}, \{\{\}\})\), where

- \(\odot\) can be seen as or; it is the usual union operation over \(2^{|\Pi|}\) and \(\emptyset\) is its neutral element: obviously, \((2^{|\Pi|}, \odot, \emptyset)\) is a commutative monoid \((2^{|\Pi|})\);

- \(\odot\) represents and; it is defined in the following way: \((\pi\text{ symbols stand for elements of }\mathcal{M}; n, m \geq 0)\):

\[
\begin{align*}
\{\pi_1, ..., \pi_n\} \odot \emptyset &= \emptyset \\
\emptyset \odot \{\pi_1, ..., \pi_n\} &= \emptyset \\
\{\pi_1, ..., \pi_n\} \odot \{\pi_1', ..., \pi_m\} &= \bigvee_{1 \leq i \leq n, j \leq m} \{\pi_i \cup \pi_j'\}
\end{align*}
\]

For example: \(\{(p_A), (p_B)\} \odot \{(p_C), (p_D)\} = \{(p_A, p_C), (p_A, p_D), (p_B, p_C), (p_B, p_D)\}\).

Trivially, \(\{\emptyset\}\) is neutral for \(\odot\) and it’s easy to see that this operation is associative and distributive with respect to \(\odot\). The structure \((2^{|\Pi|}, \odot, \emptyset, \{\emptyset\}, \{\{\}\})\) is then the semiring of masks \((\Omega, \odot, \emptyset, \emptyset)\).

Notice that our semiring is commutative because \(\odot\) is commutative—trivial from \(\odot\) definition. Notice also that \(\odot\) and \(\odot\) are idempotent: \(\forall x \in \Omega : x \odot x = x = x \odot x\)—also trivial from definitions. Remark that in a boolean domain, operators are naturally commutative and idempotent.

The factorisation of some \(x = \{\pi_1, ..., \pi_n\}\) consists in looking for a product \(\{\delta_1, ..., \delta_m\} \odot \{\pi_1, ..., \pi_p\} = x\). Such a decomposition is always possible because we have at least \(x = \{\pi_1, ..., \pi_n\} \odot \emptyset\). Sometimes, it’s the only possibility. The set of all possibilities provides the range of all factors of \(x\). We can then define a division in our semiring in the following way: \(x \odot y = \{z : \exists z \text{ such that } x = y \odot z\}
\begin{align*}
&z \text{ if } \exists z \text{ such that } x = y \odot z \\
v \text{ otherwise, where } x = u \odot v, y = u \odot w \\
&\text{s. t. } u \text{ and } w \text{ have only } \mathcal{T} \text{ as common factor}
\end{align*}

In addition to the semiring axioms, [3] introduces a new condition that is only implicit in [2]: the semiring must be weakly (left) divisible: every \(x\) must be divisible by \(x \odot y\) (provided \(x \odot y \neq \emptyset\)) and the quotient must be identical to that of \(u \odot x\) divided by \((u \odot x) \odot (u \odot y)\) (provided \(u \neq \emptyset\)), which our division clearly satisfies.

Therefore the general algorithm for FST determinization can be used.

3.2. Improved representation

Let’s consider the mask \(\{\{p_A, p_B\}, \{p_B\}\}\). It represents an on value when the following condition holds: \(p_A\) and \(p_B\) are on or \(p_B\) alone is, which is equivalent to the condition: \(p_B\) is on. Hence masks \(M_1 = \{\{p_A, p_B\}, \{p_B\}\}\) and \(M_2 = \{\{p_B\}\}\) represent the same masking status: \(M_1 \equiv M_2\). We can keep the simplest. Formally: for \(\{\pi_1, ..., \pi_n\} \in 2^{|\Pi|}\), the improved representation keeps only the smaller \(\pi_i (1 \leq i \leq n)\) subsets, where small refers to the natural partial order in \(2^{|\Pi|};\). Because of \(\subset\) being a partial order, the improved representation of a \(\{\pi_1, ..., \pi_n\}\) can range from a single \(\{\pi_k\}\) to \(\{\pi_1, ..., \pi_n\}\).

In this representation, \(\mathcal{T}\) is an annihilator for \(\oplus\). Indeed, \(\{\\}\) being smaller than any other subsets of \(\Pi\), we have, for any \(\pi \in 2^{|\Pi|} : \{\\}\) \(\oplus\) \(\{\pi\} \equiv \{\\}\). Remark that true be an annihilator for \(\oplus\) is no surprise.

Figure 4 shows \(T_G^{det}\), the determinization of \(T_G\) in the improved representation; it turns out to be also minimal. You can see that the two non-deterministic transitions on Jim in \(T_G\) become a single transition in \(T_G^{det}\) with a \(\oplus\)-summed mask: \(\{\{p_C\}\} \oplus \{\{p_A\}\} = \mathcal{T} \oplus \{\{p_A\}\} = \mathcal{T}\); the (power) state reached contains states 6 and 9 that are associated with the residual masks (resulting from \(\odot\)-division) that turn out to be the masks of the original non-deterministic transitions; these residuals reappear in the outgoing transitions on Jackson and Moore. The same holds for the non-determinism on Sue.

Figure 4 illustrates that the resulting FSM is deterministic, whatever the masking status of all maskable non-terminals.

3.3. Halt

Mohri’s algorithm is a semiring generalization of the power-set construction used in the determinization algorithm for finite-state automata: instead of building sets in \(2^{|\Omega|}\), he builds sets in \(2^{|\Omega|} \times \Omega\) (where \(\Omega\) is an arbitrary semiring), which explains why the algorithm may not terminate: for some \((q_1, ..., q_k) \in 2^{|\Omega|}\) there can be infinitely many \((q_1, m_1), ..., (q_k, m_k) \in 2^{|\Omega|} \times \Omega\).

Our basic set of predicates is finite: II, hence \(2^{|\Omega|}\) is finite as well, therefore there is only a finite set of possible tuples.
\((q_1, m_1), \ldots, (q_k, m_k)\). The determinization algorithm for transducers over the masking semiring always terminates.\(^3\)

4. Discussion

The improved representation significantly reduces the size and the number of masking tags that need to be kept at runtime. Remember that \(\mathcal{T}\) being constant, every transition tagged by this value is not influenced by runtime changes of the masking status.

As for the size of the determinized FST, there is a theoretical danger of combinatorial explosion due to the number of \((\{q_1, m_1\}, \ldots, \{q_k, m_k\})\) for each \((q_1, \ldots, q_k)\) of the FSA. Indeed, during determinization, a lot of \(\oplus\) and \(\otimes\) operations are performed, but in our case, both these operations are idempotent, which avoids a factorial explosion. Still, the mask domain counts \(2^{2^{(1)}}\) elements; even if we remember that only some non-terminals are maskable, the double exponential makes it quickly huge. The improved representation shrinks this number but it remains a theoretical risk. Now practically, this problem vanishes completely.

Experiments on some CFGs using grammars of 4-8,000 public ways of Paris area (streets, avenues...) show the same trend reported in [4]: not only determinization does not trigger a combinatorial explosion but we always observed that it reduces the size of networks. Yet, in [4], the \(\Omega\) domain is not finite and \(\otimes\) is not idempotent.

Moreover, we always experienced that, with or without masks, the determinized network has the same topology: the presence of masks never implied additional states. Despite this trend is not surprising given Mohri’s observations and the properties of this semiring, it still has to be confirmed: either practically by further testing with other grammars, or theoretically by proving our semiring guarantees this property.

5. Conclusion

While compiling a FSM from a CFG, we introduced a parameter on every transition, that is a boolean expression over the set of predicates \(\{p_A\}\) 

\[\text{masking status of } A \in N,\]

and that reflects the masking status of the transition. We described an algebra for such expressions that matches semiring axioms and that lets the determinization terminate. This shows that one can get advantages of both dynamic masking and finite-state determinization. Besides, keep in mind that a backward determinization, followed by a forward determinization yields a minimal FST (Brzozowski algorithm).

Starting from CFGs is not mandatory for the technique as such: they offer only a convenient way to refer to some sub-graphs. One could as well have another way to define the set of sentences and to refer to sub-graphs, and still follow the steps described here: the key feature is the masking semiring, commutative, twice idempotent, with its improved representation.

The semiring we described can be combined with whatever semiring: the cartesian product of two semirings is still a semiring. Indeed, we developed a string-to-string transducer (dealing with the semantic interpretation) and a string-to-weight transducer (for the search algorithm), both compiled from the same ABNF grammar and both featuring masks — that are updated in a synchronized way at runtime.

6. Acknowledgements

This communication owes a lot to Lionel Delphin-Poulat. Many thanks for fruitful discussions and useful references.

7. References


