On the Equivalence of Weighted Finite-state Transducers

Julien Quint
National Institute of Informatics
Hitotsubashi 2-1-2
Chiyoda-ku
Tokyo 101-8430
Japan
quint@nii.ac.jp

Abstract

Although they can be topologically different, two distinct transducers may actually recognize the same rational relation. Being able to test the equivalence of transducers allows to implement such operations as incremental minimization and iterative composition. This paper presents an algorithm for testing the equivalence of deterministic weighted finite-state transducers, and outlines an implementation of its applications in a prototype weighted finite-state calculus tool.

Introduction

The addition of weights in finite-state devices (where transitions, initial states and final states are weighted) introduced the need to reevaluate many of the techniques and algorithms used in classical finite-state calculus. Interesting consequences are, for instance, that not all non-deterministic weighted automata can be made deterministic (Buchsbaum et al., 2000); or that epsilon transitions may offset the weights in the result of the composition of two transducers (Pereira and Riley, 1997).

A fundamental operation on finite-state transducers in equivalence testing, which leads to applications such as incremental minimization and iterative composition. Here, we present an algorithm for equivalence testing in the weighted case, and describe its application to these applications. We also describe a prototype implementation, which is demonstrated.

1 Definitions

We define a *weighted finite-state automata* (WFST) T over a set of weights $\mathbb{K}$ by an 8-tuple $(\Sigma, \Omega, Q, I, F, E, \lambda, \rho)$ where Σ and Ω are two finite sets of symbols (alphabets), Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, $E \subseteq Q \times \Sigma \cup \{\varepsilon\} \times \Omega \cup \{\varepsilon\} \times \mathbb{K} \times Q$ is the set of transitions, and $\lambda : I \to \mathbb{K}$ and $\rho : F \to \mathbb{K}$ are the initial and final weight functions.

A transition $e \in E$ has a label $l(e) \in \Sigma \cup \{\epsilon\} \times \Omega \cup \{\epsilon\}$, a weight $w(e) \in \mathbb{K}$ and a destination $\delta(e) \in Q$.

The set of weights is a *semi-ring*, that is a system $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ where $\bar{0}$ is the identity element for $\oplus$, $\bar{1}$ is the identity element for $\otimes$, and $\oplus$ is commutative (Berstel and Reteunauer, 1988). The cost of a path in a WFST is the product ($\otimes$) of the initial weight of the initial state, the weight of all the transitions, and the final weight of the final state. When several paths in the WFST match the same relation, the total cost is the sum ($\oplus$) of the costs of all the paths.

In NLP, the *tropical semi-ring* $(\mathbb{R}_+ \cup \{\infty\}, min, +, \infty, 0)$ is very often used: weights are added along a path, and if several paths match the same relation, the total cost is the cost of the path with minimal cost. The following discussion will apply to any semi-ring, with examples using the tropical semi-ring.

2 The Equivalence Testing Algorithm

Several algorithms testing the equivalence of two states are presented in (Watson and Daciuk, 2003), from which we will derive ours. Two states are equivalent if and only if their respective *right language* are equivalent. The right language of a state is the set of words originating from this state. Two deterministic finite-state automata are equivalent if and only if they recognize the same language, that is, if their initial states have the same right language. Hence, it is possible to test the equivalence of two automata by applying the equivalence algorithm on their initial states.

In order to test the equivalence of two WFSTs, we need to extend the state equivalence test algorithm in two ways: first, it must apply to transducers, and second, it must take weights into account. Handling transducers is easily achieved as the labels of transitions defined above are equivalent to symbols in an alphabet (*i.e.* we consider the underlying automaton of the transducer).

Taking weights into account means that for two WFSTs to be equivalent, they must recog-

nize the same relation (or their underlying automata must recognize the same language), *with the same weights*. However, as illustrated by figure 1, two WFSTs can be equivalent but have a different weight distribution. States 1 and 5 have the same right language, but words have different costs (for example, *abad* has a cost of 6 in the top automaton, and 5 in the bottom one). We notice however that the difference of weights between words is constant, so states 1 and 5 are really equivalent modulo a cost of 1.

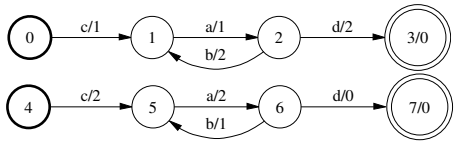

Figure 1: Two equivalent weighted finite-state transducers (using the tropical semi-ring).

Figure 2 shows the weighted equivalence algorithm. Given two states p and q, it returns a true value if they are equivalent, and a false value otherwise. Remainder weights are also passed as parameters w_p and w_q. The last parameter is an associative array S that we use to keep track of states that were already visited.

The algorithm works as follows: given two states, compare their *signature*. The signature of a state is a string encoding its class (final or not) and the list of labels on outgoing transition. In the case of deterministic transducers, if the signature for the two states do not match, then they cannot have the same right language and therefore cannot be equivalent.

Otherwise, if the two states are final, then their weights (taking into account the remainder weights) must be the same (lines 6–7). Then, all their outgoing transitions have to be checked: the states will be equivalent if matching transitions lead to equivalent states (lines 8–12). The destination states are recursively checked. The REMAINDER function computes the remainder weights for the destination states. Given two weights x and y, it returns $\{\bar{1}, x \otimes y^{-1}\}$ if $x < y$, and $\{x^{-1} \otimes y, \bar{1}\}$ otherwise.

If there is a cycle, then we will see the same pair of states twice. The weight of the cycle must be the same in both transducers, so the remainder weights must be unchanged. This is tested in lines 2–4.

The algorithm applies to deterministic WFSTs, which can have only one initial state. To test the equivalence of two WFSTs, we call EQUIV on the respective initial states of the the WFSTs with their initial weights as the remainder weights, and S is initially empty.

3 Incremental minimization

An application of this equivalence algorithm is the incremental minimization algorithm of (Watson and Daciuk, 2003). For every deterministic WFST T there exists at least one equivalent WFST M such that no other equivalent WFST has fewer states (*i.e.* $|Q_M|$ is minimal). In the unweighted case, this means that there cannot be two distinct states that are equivalent in the minimized transducer.

It follows that a way to build this transducer M is to compare every pair of distinct states in Q_A and merge pairs of equivalent states until there are no two equivalent states in the transducer. An advantage of this method is that at any time of the application of the algorithm, the transducer is in a consistent state; if the process has to finish under a certain time limit, it can simply be stopped (the number of states will have decreased, even though the minimality of the result cannot be guaranteed then).

In the weighted case, merging two equivalent states is not as easy because edges with the same label may have a different weight. In figure 3, we see that states 1 and 2 are equivalent and can be merged, but outgoing transitions have different weights. The remainder weights have to be pushed to the following states, which can then be merged if they are equivalent modulo the remainder weights. This applies to states 3 and 4 here.

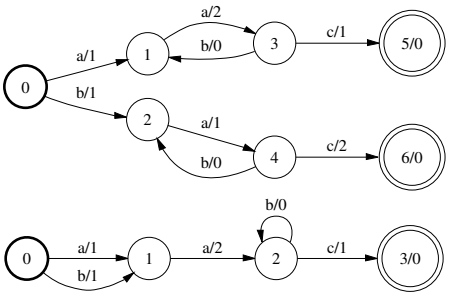

Figure 3: Non-minimal transducer and its minimized equivalent.

4 Generic Composition with Filter

As shown previously (Pereira and Riley, 1997), a special algorithm is needed for the composition of WFSTs. A filter is introduced, whose role is to handle epsilon transitions on the lower side of the top transducer and the upper side of the lower transducer (it is also useful in the unweighted case). In our implementation described in section 5 we have generalized the use of this epsilon-free composition operation to handle two operations that are defined

```
EQUIV(p, w_p, q, w_q, S)
 1  equiv ← FALSE
 2  if S[{p, q}] ≠ NIL
 3     then {w'_p, w'_q} ← S[{p, q}]
 4          equiv ← w'_p = w_p ∧ w'q = w_q
 5     else if SIGNATURE(p) = SIGNATURE(q)
 6             then if FINAL(p)
 7                    then equiv ← w_p ⊗ ρ(p) = w_q ⊗ ρ(q)
 8                  S[{p, q}] ← {w_p, w_q}
 9                  for e_p ∈ E(p), e_q ∈ E(q), l(e_p) = l(e_q)
10                      do {w'_p, w'_q} ← REMAINDER(w_p ⊗ w(e_p), w_q ⊗ w(e_q))
11                         equiv ← equiv ∧EQUIV(δ(e_p), w'_p, δ(e_q), w'_q, S)
12                  DELETE(S[{p, q}])
13  return equiv
```

Figure 2: The equivalence algorithm

on automata only, that is intersection and cross-product. Intersection is a simple variant of the composition of the identity transducers corresponding to the operand automata.

Cross-product uses the exact same algorithm but a different filter, shown in figure 4. The preprocessing stage for both operand automata consists of adding a transition with a special symbol x at every final state, going to itself, and with a weight of $\bar{1}$. This will allow to match words of different lengths, as when one of the automata is "exhausted," the x symbol will be added as long as the other automaton is not. After the composition, the x symbol is replaced everywhere by ϵ.

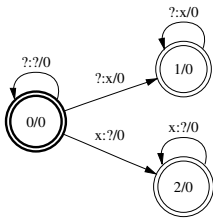

Figure 4: Cross-product filter. The symbol "?" matches any symbol; "x" is a special espilon-symbol introduced in the final states of the operand automata at preprocessing.

The equivalence algorithm that is the subject of this paper is used in conjunction with composition of WFSTs in order to provide an iterative composition operator. Given two transducers A and B, it composes A with B, then composes the result with B again, and again, until a fixed-point is reached. This can be determined by testing the equivalence of the last two iterations. Roche and Schabes (1994) have shown that in the unweighted case this allows to parse context-free grammars with finite-state transducers; in our case, a cost can be added to the parse.

5 A Prototype Implementation

The algorithms described above have all been implemented in a prototype weighted finite-state tool, called `wfst`, inspired from the Xerox tool `xfst` (Beesley and Karttunen, 2003) and the FSM library from AT&T (Mohri et al., 1997). From the former, it borrows a similar command-line interface and regular expression syntax, and from the latter, the addition of weights. The system will be demonstrated and should be available for download soon.

The operations described above are all available in `wfst`, in addition to classical operations like union, intersection (only defined on automata), concatenation, etc. The regular expression syntax is inspired from `xfst` and Perl (the implementation language). For instance, the automaton of figure 3 was compiled from the regular expression `(a/1 a/2 b/0* c/1) | (b/2 a/1 b/0* c/2)` and the iterative composition of two previously defined WFSTs A and B is written `$A %+ $B` (we chose `%` as the composition operator, and `+` refers to the Kleene plus operator).

Conclusion

We demonstrate a simple and powerful experimental weighted finite state calculus tool and have described an algorithm at the core of its operation for

the equivalence of weighted transducers. There are two major limitations to the weighted equivalence algorithm. The first one is that it works only on deterministic WFSTs; however, not all WFSTs can be determinized. An algorithm with backtracking may be a solution to this problem, but its running time would increase, and it remains to be seen if such an algorithm could apply to undeterminizable transducers.

The other limitation is that two transducers recognizing the same rational relation may have non-equivalent underlying automata, and some labels will not match (*e.g.* $\{a, \epsilon\}\{b, c\}$ vs. $\{a, c\}\{b, \epsilon\}$). A possible solution to this problem is to consider the shortest string on both sides and have "remainder strings" like we have remainder weights in the weighted case. If successful, this technique could yield interesting results in determinization as well.

References

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Publications, Stanford, California.

Jean Berstel and Christophe Reteunauer. 1988. *Rational Series and their Languages*. Springer Verlag, Berlin, Germany.

Adam L. Buchsbaum, Raffaele Giancarlo, and Jeffery R. Westbrook. 2000. On the determinization of weighted finite automata. *SIAM Journal on Computing*, 30(5):1502–1531.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 1997. A rational design for a weighted finite-state transducer library. In *Workshop on Implementing Automata*, pages 144–158, London, Ontario.

Fernando C. N. Pereira and Michael Riley. 1997. Speech recognition by composition of weighted finite state automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, pages 431–453. MIT Press, Cambridge, Massachusetts.

Emmanuel Roche and Yves Schabes. 1994. Two parsing algorithms by means of finite state transducers. In *Proceedings of COLING'94*, pages 431–435, Ky$\bar{\text{o}}$t$\bar{\text{o}}$, Japan.

Bruce W. Watson and Jan Daciuk. 2003. An efficient incremental DFA minimization algorithm. *Natural Language Engineering*, 9(1):49–64.