WIKIPEDIA

Finite-state transducer

A **finite-state transducer** (**FST**) is a finite-state machine with two memory *tapes*, following the terminology for Turing machines: an input tape and an output tape. This contrasts with an ordinary finite-state automaton, which has a single tape. An FST is a type of finite-state automaton that maps between two sets of symbols.[1] An FST is more general than a finite-state automaton (FSA). An FSA defines a formal language by defining a set of accepted strings while an FST defines relations between sets of strings.

An FST will read a set of strings on the input tape and generates a set of relations on the output tape. An FST can be thought of as a translator or relater between strings in a set.

In morphological parsing, an example would be inputting a string of letters into the FST, the FST would then output a string of morphemes.

Contents

Overview

An automaton can be said to *recognize* a string if we view the content of its tape as input. In other words, the automaton computes a function that maps strings into the set {0,1}. Alternatively, we can say that an automaton *generates* strings, which means viewing its tape as an output tape. On this view, the automaton generates a formal language, which is a set of strings. The two views of automata are equivalent: the function that the automaton computes is precisely the indicator function of the set of strings it generates. The class of languages generated by finite automata is known as the class of regular languages.

The two tapes of a transducer are typically viewed as an input tape and an output tape. On this view, a transducer is said to *transduce* (i.e., translate) the contents of its input tape to its output tape, by accepting a string on its input tape and generating another string on its output tape. It may do so nondeterministically and it may produce more than one output for each input string. A transducer may also produce no output for a given input string, in which case it is said to *reject* the input. In general, a transducer computes a relation between two formal languages.

Each string-to-string finite-state transducer relates the input alphabet Σ to the output alphabet Γ. Relations R on $\Sigma^*\times\Gamma^*$ that can be implemented as finite-state transducers are called **rational relations**. Rational relations that are partial functions, i.e. that relate every input string from Σ^* to at most one Γ^*, are called **rational functions**.

Finite-state transducers are often used for phonological and morphological analysis in natural language processing research and applications. Pioneers in this field include Ronald Kaplan, Lauri Karttunen, Martin Kay and Kimmo Koskenniemi.[2] A common way of using transducers is in a so-called "cascade", where transducers for various operations are combined into a single transducer by repeated application of the composition operator (defined below).

Formal construction

Formally, a finite transducer T is a 6-tuple (Q, Σ, Γ, I, F, δ) such that:

- Q is a finite set, the set of *states*;
- Σ is a finite set, called the *input alphabet*;
- Γ is a finite set, called the *output alphabet*;
- I is a subset of Q, the set of *initial states*;
- F is a subset of Q, the set of *final states*; and
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q$ (where ε is the empty string) is the *transition relation*.

We can view (Q, δ) as a labeled directed graph, known as the *transition graph* of T: the set of vertices is Q, and $(q, a, b, r) \in \delta$ means that there is a labeled edge going from vertex q to vertex r. We also say that a is the *input label* and b the *output label* of that edge.

NOTE: This definition of finite transducer is also called *letter transducer* (Roche and Schabes 1997); alternative definitions are possible, but can all be converted into transducers following this one.

Define the *extended transition relation* δ^* as the smallest set such that:

- $\delta \subseteq \delta^*$;

- $(q, \epsilon, \epsilon, q) \in \delta^*$ for all $q \in Q$; and
- whenever $(q, x, y, r) \in \delta^*$ and $(r, a, b, s) \in \delta$ then $(q, xa, yb, s) \in \delta^*$.

The extended transition relation is essentially the reflexive transitive closure of the transition graph that has been augmented to take edge labels into account. The elements of δ^* are known as *paths*. The edge labels of a path are obtained by concatenating the edge labels of its constituent transitions in order.

The *behavior* of the transducer T is the rational relation $[T]$ defined as follows: $x[T]y$ if and only if there exists $i \in I$ and $f \in F$ such that $(i, x, y, f) \in \delta^*$. This is to say that T transduces a string $x \in \Sigma^*$ into a string $y \in \Gamma^*$ if there exists a path from an initial state to a final state whose input label is x and whose output label is y.

Weighted automata

Finite State Transducers can be weighted, where each transition is labelled with a weight in addition to the input and output labels. A Weighted Finite State Transducer (WFST) over a set K of weights can be defined similarly to an unweighted one as an 8-tuple T=(Q, Σ, Γ, I, F, E, λ, ρ), where:

- Q, Σ, Γ, I, F are defined as above;
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q \times K$ (where ε is the empty string) is the finite set of transitions;
- $\lambda : I \to K$ maps initial states to weights;
- $\rho : F \to K$ maps final states to weights.

In order to make certain operations on WFSTs well-defined, it is convenient to require the set of weights to form a semiring.[3] Two typical semirings used in practice are the log semiring and tropical semiring: unweighted automata may be regarded as having weights in the Boolean semiring.[4]

Stochastic FST

Stochastic FSTs (also known as probabilistic FSTs or statistical FSTs) are presumably a form of weighted FST.

Operations on finite-state transducers

The following operations defined on finite automata also apply to finite transducers:

- Union. Given transducers T and S, there exists a transducer $T \cup S$ such that $x[T \cup S]y$ if and only if $x[T]y$ or $x[S]y$.
- Concatenation. Given transducers T and S, there exists a transducer $T \cdot S$ such that $x[T \cdot S]y$ if and only if there exist x_1, x_2, y_1, y_2 with $x = x_1x_2, y = y_1y_2, x_1[T]y_1$ and $x_2[S]y_2$.
- Kleene closure. Given a transducer T, there exists a transducer T^* with the following properties:

$$\epsilon[T^*]\epsilon; \quad \textbf{(k1)}$$

$$w[T^*]y \text{ and } x[T]z \text{ then } wx[T^*]yz; \quad \textbf{(k2)}$$

  and $x[T^*]y$ does not hold unless mandated by (**k1**) or (**k2**).

- Composition. Given a transducer T on alphabets Σ and Γ and a transducer S on alphabets Γ and Δ, there exists a transducer $T \circ S$ on Σ and Δ such that $x[T \circ S]z$ if and only if there exists a string $y \in \Gamma^*$ such that $x[T]y$ and $y[S]z$. This operation extends to the weighted case.[5]

  This definition uses the same notation used in mathematics for relation composition. However, the conventional reading for relation composition is the other way around: given two relations T and S, $(x, z) \in T \circ S$ when there exist some y such that $(x, y) \in S$ and $(y, z) \in T$.

- Projection to an automaton. There are two projection functions: π_1 preserves the input tape, and π_2 preserves the output tape. The first projection, π_1 is defined as follows:

  Given a transducer T, there exists a finite automaton $\pi_1 T$ such that $\pi_1 T$ accepts x if and only if there exists a string y for which $x[T]y$.

  The second projection, π_2 is defined similarly.

- Determinization. Given a transducer T, we want to build an equivalent transducer that has a unique initial state and such that no two transitions leaving any state share the same input label. The powerset construction can be extended to transducers, or even weighted transducers, but sometimes fails to halt; indeed, some non-deterministic transducers do not admit equivalent deterministic transducers.[6] Characterizations of determinizable transducers have been proposed[7] along with efficient algorithms to test them:[8] they rely on the semiring used in the weighted case as well as a general property on the structure of the transducer (the twins property).
- Weight pushing for the weighted case.[9]
- Minimization for the weighted case.[10]
- Removal of epsilon-transitions.

Additional properties of finite-state transducers

- It is decidable whether the relation [T] of a transducer T is empty.
- It is decidable whether there exists a string y such that x[T]y for a given string x.
- It is undecidable whether two transducers are equivalent.[11] Equivalence is however decidable in the special case where the relation [T] of a transducer T is a (partial) function.
- If one defines the alphabet of labels $L = (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\})$, finite-state transducers are isomorphic to NDFA over the alphabet L, and may therefore be determinized (turned into deterministic finite automata over the alphabet $L = [(\Sigma \cup \{\epsilon\}) \times \Gamma] \cup [\Sigma \times (\Gamma \cup \{\epsilon\})]$) and subsequently minimized so that they

have the minimum number of states.

Applications

Context-sensitive rewriting rules of the form $a \to b / c \_ d$, used in linguistics to model phonological rules and sound change, are computationally equivalent to finite-state transducers, provided that application is nonrecursive, i.e. the rule is not allowed to rewrite the same substring twice.[12]

Weighted FSTs found applications in natural language processing, including machine translation, and in machine learning.[13][14] An implementation for part-of-speech tagging can be found as one component of the [15] library.

See also

- Mealy machine
- Moore machine
- Morphological dictionary
- foma (software)
- Tree transducer

Notes

1. Jurafsky, Daniel (2009). *Speech and Language Processing*. Pearson. ISBN 9789332518414.
2. Koskenniemi 1983
3. Berstel, Jean; Reutenauer, Christophe (2011). *Noncommutative rational series with applications*. Encyclopedia of Mathematics and Its Applications. **137**. Cambridge: Cambridge University Press. p. 16. ISBN 978-0-521-19022-0. Zbl 1250.68007 (https://zbmath.org/?format=complete&q=an:1250.68007).
4. Lothaire, M. (2005). *Applied combinatorics on words*. Encyclopedia of Mathematics and Its Applications. **105**. A collective work by Jean Berstel, Dominique Perrin, Maxime Crochemore, Eric Laporte, Mehryar Mohri, Nadia Pisanti, Marie-France Sagot, Gesine Reinert, Sophie Schbath, Michael Waterman, Philippe Jacquet, Wojciech Szpankowski, Dominique Poulalhon, Gilles Schaeffer, Roman Kolpakov, Gregory Koucherov, Jean-Paul Allouche and Valérie Berthé. Cambridge: Cambridge University Press. p. 211. ISBN 0-521-84802-4. Zbl 1133.68067 (https://zbmath.org/?format=complete&q=an:1133.68067).
5. Mohri 2004, pp. 3–5
6. [1] (http://www.let.rug.nl/~vannoord/papers/preds/node22.html)
7. Mohri 2004, pp. 5–6
8. Allauzen 2003
9. Mohri 2004, pp. 7–9
10. Mohri 2004, pp. 9–11
11. Griffiths 1968
12. "Regular Models of Phonological Rule Systems" (http://acl.ldc.upenn.edu/J/J94/J94-3001.pdf) (PDF). Retrieved August 25, 2012.
13. Kevin Knight and Jonathan May (2009). "Applications of Weighted Automata in Natural Language Processing". In Manfred Droste; Werner Kuich; Heiko Vogler. *Handbook of Weighted Automata*. Springer Science & Business Media. ISBN 978-3-642-01492-5.
14. "Learning with Weighted Transducers" (http://www.cs.nyu.edu/~mohri/pub/fsmnlp08.pdf) (PDF). Retrieved April 29, 2017.
15. OpenGrm (http://opengrm.org/)

References

- Allauzen, Cyril; Mohri, Mehryar (2003). "Efficient Algorithms for Testing the Twins Property" (http://www.cs.nyu.edu/~mohri/pub/twins.pdf) (PDF). *Journal of Automata, Languages and Combinatorics*. **8** (2): 117–144.
- Koskenniemi, Kimmo (1983), *Two-level morphology: A general computational model of word-form recognition and production* (http://www.ling.helsinki.fi/~koskenni/doc/Two-LevelMorphology.pdf) (PDF), Department of General Linguistics, University of Helsinki
- Mohri, Mehryar (2004). "Weighted Finite-State Transducer Algorithms: An Overview" (http://www.cs.nyu.edu/~mohri/pub/fla.pdf) (PDF). *Formal Languages and Applications*. **148** (620): 551–564. doi:10.1007/978-3-540-39886-8_29 (https://doi.org/10.1007%2F978-3-540-39886-8_29).
- Griffiths, T. V. (1968). "The unsolvability of the Equivalence Problem for Λ-Free nondeterministic generalized machines". **15** (3). ACM: 409–413.

External links

- OpenFst (http://openfst.org/), an open-source library for FST operations.
- Stuttgart Finite State Transducer Tools (http://www.cis.uni-muenchen.de/~schmid/tools/SFST/), another open-source FST toolkit
- java FST Framework (http://jsalatas.ictpro.gr/java-fst-framework-api-review/), an open-source java FST Framework capable of handling OpenFst text format.
- Vcsn (http://vcsn.lrde.epita.fr/), an open-source platform (C++ & IPython) platform for weighted automata and rational expressions.
- Finite State Morphology--The Book (http://web.stanford.edu/~laurik/fsmbook/home.html) XFST/ LEXC, a description of Xerox's implementation of finite-state transducers intended for linguistic applications.
- FOMA (https://code.google.com/archive/p/foma/), an open-source implementation of most of the capabilities of the Xerox XFST/ LEXC implementation.

Further reading

- Jurafsky, Daniel; James H. Martin (2000). *Speech and Language Processing*. Prentice Hall. pp. 71–83. ISBN 0-13-095069-6.
- Kornai, Andras (1999). *Extended Finite State Models of Language*. Cambridge University Press. ISBN 0-521-63198-X.
- Roche, Emmanuel; Yves Schabes (1997). *Finite-state language processing*. MIT Press. pp. 1–65. ISBN 0-262-18182-7.
- Beesley, Kenneth R.; Lauri Karttunen (2003). *Finite State Morphology*. Center for the Study of Language and Information. ISBN 1-57586-434-7.
- Roark, Brian; Richard Sproat (2007). *Computational Approaches to Morphology and Syntax*. Oxford University Press. ISBN 0-19-927478-9.
- Berstel, Jean (1979). *Transductions and Context-free Languages*. Teubner Verlag.. Free PDF version (http://www-igm.univ-mlv.fr/~berstel/LivreTransductions/LivreTransductions.html)