

1.A Appendix: An Overview of Finite-State Automata and Transducers

In this appendix I give an overview of regular languages and relations, and their associated computational devices, finite-state acceptors (FSA's) and finite-state transducers (FST's). The coverage here is necessarily brief, and for further discussion other sources are recommended. Finite-state acceptors and regular languages are discussed in any good introduction to the theory of computation: see, for example (Harrison, 1978; Hopcroft and Ullman, 1979; Lewis and Papadimitriou, 1981). There are fewer introductory works on transducers. One reasonably accessible discussion (dealing with transducers) can be found in (Kaplan and Kay, 1994). One might also consult the third chapter of (Sproat, 1992) for an in-depth introduction to the use of finite-state transducers in computational phonology and morphology. For transducers (as well as weighted acceptors), there is a recent paper by Mohri (1997) that discusses various formal properties and algorithms, and various other relevant works are cited therein.

1.A.1 Regular languages and finite-state automata

Basic to the theory of automata is the notion of an *alphabet* of symbols; the entire alphabet is conventionally denoted Σ . The *empty string* is denoted by ϵ , which is *not* an element of Σ ; also, the empty string is distinct from the *empty set* \emptyset . Σ^* denotes the set of all strings — including ϵ — over the alphabet Σ .

It is usual to define a *regular language* with a recursive definition such as the following (modeled on that of (Kaplan and Kay, 1994, page 338)):

1. \emptyset is a regular language
2. For all symbols $a \in \Sigma \cup \epsilon$, $\{a\}$ is a regular language
3. If L_1, L_2 and L are regular languages, then so are
 - (a) $L_1 \cdot L_2$, the *concatenation* of L_1 and L_2 : for every $w_1 \in L_1$ and $w_2 \in L_2$, $w_1 w_2 \in L_1 \cdot L_2$
 - (b) $L_1 \cup L_2$, the *union* of L_1 and L_2
 - (c) L^* , the *Kleene closure* of L . Using L^i to denote L concatenated with itself i times, $L^* = \cup_{i=0}^{\infty} L^i$.

While the above definition is complete, regular languages observe additional *closure* properties:

- *Intersection*: if L_1 and L_2 are regular languages then so is $L_1 \cap L_2$.
- *Difference*: if L_1 and L_2 are regular languages then so is $L_1 - L_2$, the set of strings in L_1 that are not in L_2 .
- *Complementation*: if L is a regular language, then so is $\Sigma^* - L$, the set of all strings over Σ that are *not* in L . (Of course, complementation is merely a special case of difference.)

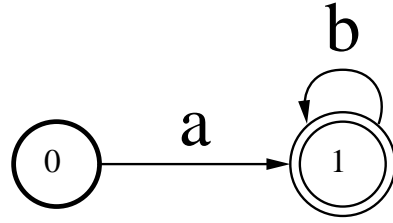


Figure 1.3: An acceptor for ab^* . The heavy-circled state (0) is (conventionally) the initial state, and the double-circled state is the final state.

- *Reversal*: if L is a regular language, then so is $Rev(L)$, the set of reversals of all strings in L .

Regular languages are sets of strings, and they are usually notated using *regular expressions*. A fundamental result of automata theory are the so-called Kleene's theorems, which demonstrate that regular languages are exactly the languages that can be recognized using *finite-state automata*, where this computational device can be defined as follows (Harrison, 1978; Hopcroft and Ullman, 1979; Lewis and Papadimitriou, 1981):

A finite-state automaton is a quintuple $M = (K, s, F, \Sigma, \delta)$ where:

1. K is a finite set of states
2. s is a designated initial state
3. F is a designated set of final states
4. Σ is an alphabet of symbols, and
5. δ is a transition relation from $K \times \Sigma$ to K

As a simple example, consider the (infinite) set of strings: $\{a, ab, abb, abbb \dots\}$ — i.e. the set consisting of a followed by zero or more bs . The most compact regular expression denoting this set is ab^* . Furthermore, the language can be recognized by the finite-state machine given in Figure 1.3.

1.A.2 Regular relations and finite-state transducers

Regular n-relations can be defined in a way entirely parallel to regular languages. Again, the definition given here is modeled on that of Kaplan and Kay (1994):

1. \emptyset is a regular n-relation
2. For all symbols $a \in [(\Sigma \cup \epsilon) \times \dots \times (\Sigma \cup \epsilon)]$, $\{a\}$ is a regular n-relation
3. If R_1, R_2 and R are regular n-relations, then so are

- (a) $R_1 \cdot R_2$, the (*n*-way) concatenation of R_1 and R_2 : for every $r_1 \in R_1$ and $r_2 \in R_2$, $r_1 r_2 \in R_1 \cdot R_2$
- (b) $R_1 \cup R_2$
- (c) R^* , the *n*-way Kleene closure of R .

One can think of regular *n*-relations as *accepting* strings of a relation stated over an *m*-tuple of symbols, and mapping them to strings of a relation stated over a *k*-tuple of symbols, where $m + k = n$. We can therefore speak more specifically of $m \times k$ -relations. As in the case of regular languages, there are further closure properties that regular *n*-relations obey:¹¹

- *Composition*: if R_1 is a regular $k \times m$ -relation and R_2 is a regular $m \times p$ -relation, then $R_1 \circ R_2$ is a regular $k \times p$ -relation. Composition will be explained below.
- *Reversal*: if R is a regular *n*-relation, then so is $Rev(R)$.
- *Inversion*: if R is a regular $m \times n$ -relation, then R^{-1} , the *inverse* of R , is a regular $n \times m$ -relation.

One computes the inverse of a transducer by simply switching the input and output labels. The fact that regular relations are closed under inversion has an important practical consequence for systems based on finite-state transducers, namely that they are fully bidirectional. Thus, as we noted in Section 1.2.2, a model of spelling (mapping from the ORL to Γ) can be turned into a model of reading (mapping from Γ to the ORL) by simply inverting the FST implementing $M_{ORL \rightarrow \Gamma}$.

For most practical applications of *n*-relations $n = 2$ (so that *k* and *m* are obviously both 1).¹² In this case we can speak of a relation as mapping from strings of one regular language into strings of another. In this work we will be concerned exclusively with 2-relations, and we will use the term *regular relations* with that meaning throughout.

The computational device corresponding to a regular relation is a *finite-state transducer*. The definition of FST can be modeled on the definition of FSA's given above, so we will merely illustrate by example, rather than essentially repeat the definition. Say we have an alphabet $\Sigma = \{a, b, c, d\}$ and a regular relation over that alphabet expressed by the set: $\{(a, c), (ab, cd), (abb, cdd), (abbb, cddd) \dots\}$. This relation thus consists of a mapping to *c* followed by zero or more *b*'s mapping to *d*. This relation can be represented compactly by the *two-way regular expression* $a:c(b:d)^*$. Figure 1.4, depicts an FST that computes this relation. We refer to the expressions on the lefthand side of the ':' as the input side, and the expressions on the righthand side as the output side. Thus, in Figure 1.4, the input side is characterizable by the regular expression ab^* , and the output side by the expression cd^* .

Composition of regular relations has the same interpretation as composition of functions: if R_1 and R_2 are regular relations, then applying $R_1 \circ R_2$ to an input

¹¹The omission of difference, complementation and intersection are intentional. In general, regular relations are *not* closed under these operations, though some important subclasses of regular relations are. See (Kaplan and Kay, 1994) for further discussion.

¹²One exception is the work of Kiraz (1999).

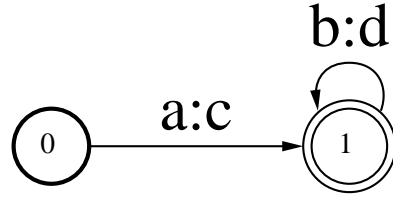
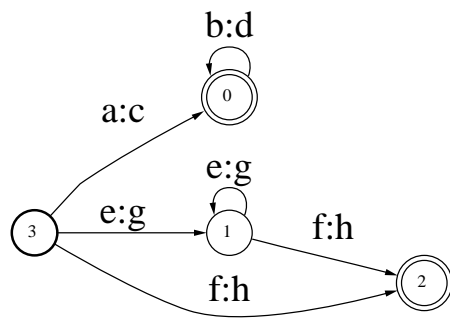


Figure 1.4: An FST that accepts $a:c (b:d)^*$.

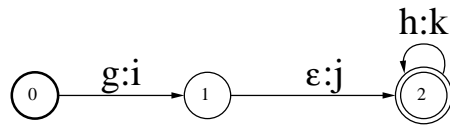
expression I is the same as applying R_1 to I first and then applying R_2 to the output. Figure 1.5 depicts two transducers, labeled T_1 and T_2 . T_1 computes the relation expressible as $(a:c (b:d)^*) \mid ((e:g)^* f:h)$ (where \mid denotes disjunction), whereas T_2 computes $g:i \epsilon:j h:k^*$ (with the $\epsilon:j$ term inserting a j). The result of composing the two transducers together — $T_1 \circ T_2$ — is a transducer that computes the trivial relation, $e:i \epsilon:j f:k$. In this particular case, though both T_1 and T_2 express relations with infinite domains and ranges, the result of composition merely maps the string ef to ijk .

One other notion that is worth mentioning is the notion of *projection* onto one dimension of a relation. For example, for a 2-way relation R , $\pi_1(R)$ projects R onto the first dimension and $\pi_2(R)$ projects onto the second dimension. Projection applied to an FST produces an FSA corresponding to one side of the transducer. Thus the first projection (π_1) of the transducer in Figure 1.4 is the acceptor in Figure 1.3.

T_1



T_2



T_3

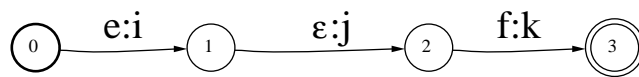


Figure 1.5: Three transducers, where $T_3 = T_1 \circ T_2$