A

Algebra-Coalgebra Duality in Brzozowski's Minimization Algorithm

F. BONCHI, ENS Lyon, Université de Lyon LIP (UMR 5668)
M.M. BONSANGUE, LIACS - Leiden University[1]
H.H. HANSEN, Radboud University Nijmegen[1]
P. PANANGADEN, McGill University
J.J.M.M. RUTTEN, Centrum Wiskunde & Informatica[2]
A. SILVA, Radboud University Nijmegen[3]

We give a new presentation of Brzozowski's algorithm to minimize finite automata, using elementary facts from universal algebra and coalgebra, and building on earlier work by Arbib and Manes on a categorical presentation of Kalman duality between reachability and observability. This leads to a simple proof of its correctness and opens the door to further generalizations. Notably, we derive algorithms to obtain minimal, language equivalent automata from Moore, non-deterministic and weighted automata.

Categories and Subject Descriptors: F.1.1 [**Theory of Computation**]: Models of Computation; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms

General Terms: Algorithms, Theory

Additional Key Words and Phrases: algebra, automata, coalgebra, duality

1. INTRODUCTION

Duality plays a fundamental role in many areas of mathematics, computer science, systems theory and even physics. For example, the familiar concept of Fourier transform is essentially a duality result: an instance of Pontryagin duality, see, for example the standard textbook [Rudin 1962]. Another basic instance, known to undergraduates, is the duality of a finite-dimensional vector spaces V over some field k, and the space of linear maps from V to k, which is itself a finite-dimensional vector space. Building on this self-duality, a fundamental principle in systems theory due to [Kalman 1959] captures the duality between the concepts of *observability and controllability* (to be explained below). The latter was further extended to automata theory (where controllability amounts to *reachability*) in [Arbib and Zeiger 1969], and in various papers [Arbib and Manes 1974; 1975a; 1975c; 1975b; 1980a; 1980b] where Arbib and Manes explored algebraic automata theory in a categorical framework; see also the excellent collection of papers [Kalman et al. 1969] where both automata theory and systems theory is presented.

Authors' addresses: F. Bonchi, ENS Lyon, Université de Lyon, France; M. M. Bonsangue, Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands; P. Panangaden, McGill University, Montreal, Quebec, Canada; J. Rutten, CWI, Amsterdam, The Netherlands; H. Hansen and A. Silva, Intelligent systems section, Radboud University Nijmegen, The Netherlands.
[1]Also affiliated to CWI, Amsterdam, The Netherlands.
[2]Also affiliated to Radboud University Nijmegen, The Netherlands.
[3]Also affiliated to CWI, Amsterdam, The Netherlands & HASLab / INESC TEC, Universidade do Minho, Braga, Portugal.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

In a system with states and transitions triggered by actions, it may be the case that an observer cannot discern the state of the system. Instead, he can see an *observable* of some quantity or property that does not uniquely fix the state. However, if one can use a sequence of observations following a prescribed course of actions to determine the state then one says that the system is observable. Similarly, a system is said to be *controllable* if a sequence of actions can drive it into a desired state irrespective of the initial state. These concepts are natural for control theory but seem less related to automata. It was a significant synthesis of [Arbib and Manes 1975a] to see that this makes sense in automata theory as well.

The main contribution of the present paper is to exploit this duality to explain a rather unexpected and surprising algorithm for minimizing automata due to [Brzozowski 1962]: Starting with a (possibly non-deterministic) automaton that accepts a language L, one reverses its transitions, makes it deterministic, takes the part that is reachable, and then repeats all of this once more. The surprise is that the result will be a minimal deterministic automaton that accepts L. Although we are presently not concerned with complexity and performance, we briefly mention that although the worst case complexity of the algorithm is exponential, it often performs well in practice, see e.g. [Champarnaud et al. 2002; Tabakov and Vardi 2005].

Though an elementary description and correctness proof of the algorithm is not very difficult (see for instance [Sakarovitch 2009, Cor. 3.14]), this proof does not really "explain" why it works. Here, we aim at supplying the conceptual reason and provide a proof that the algorithm works because of the simple duality between *reachability* and *observability* mentioned above. We first present a reformulation of Arbib and Manes' duality result in terms of elementary algebra and coalgebra [Rutten 2000], from which we will derive (the correctness of) Brzozowski's algorithm as a corollary. We mention that one of the first papers to study minimal realizations using category theory is [Goguen 1972]. Although duality does not play a role there, our definitions of reachability and observability are essentially the same as in [Goguen 1972].

Our reasons for giving this new formulation of Brzozowski's algorithm are the following.

First, the duality between reachability and observability is in itself a very beautiful result that, unfortunately, it is not very well-known in the computer science community. The work of Kalman [Kalman 1959; Kalman et al. 1969] is, of course, well known in the systems community but not, for example, in the programming languages and logics community.

Secondly, basic notions of algebra and coalgebra turn out to be the natural mathematical settings for the modelling of reachability and observability, respectively, and provide an elementary proof and understanding of the algorithm, paving also the way to several extensions. The elementary proof (in sections 2–4) uses only the notions of sets and functions. As a result, this part of the paper should be accessible to anyone with a very basic understanding of automata. We then present a more formal, categorical proof (in Section 9), which identifies the relevant categories and functors. While it is not essential to understand the rest of the paper, it captures the essence of duality most clearly.

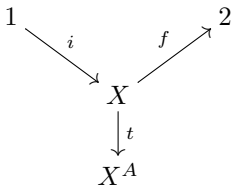
Thirdly, our proof of Brzozowski's algorithm is easy to generalise. The present paper contains the straightforward generalisation of the algorithm to Moore automata in Section 5. As an application of this generalisation we show how to use it to minimize automata corresponding to expressions of Kleene algebra with tests (KAT) in Section 6 . The construction for Moore automata is then used to obtain Brzozowski-like algorithms for non-deterministic automata (NDA) in Section 7, and weighted automata (WA) in Section 8. More precisely, these algorithms take as input an NDA or a WA, and produces as output a *minimal Moore automaton* that accepts the same (weighted)

language as the original one. As for the deterministic case, the Brzozowski algorithms for NDA and WA are based on performing twice the operations of reversing and determinization but such operations now get a different interpretation. We emphasize that the output automaton is *not* state-minimal as an NDA or a WA (as for example studied in [Arbib and Manes 1975c]). In fact, as shown in Example 8.3 the state space of our resulting *deterministic and minimal* Moore automaton may even be infinite.

This paper is an extended version of the paper [Bonchi et al. 2012], which only contained the proof of correctness of Brzozowski's algorithm and the straightforward extension to Moore automata; thus sections 6– 9 are new.

2. REACHABILITY AND OBSERVABILITY

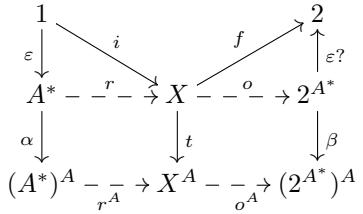
Let $1 = \{0\}$, $2 = \{0, 1\}$ and let A be any set. A deterministic automaton with inputs from A is given by the following data:



(1)

That is, a set X of states, a transition function $t\colon X \to X^A$ mapping each state $x \in X$ to a function $t(x)\colon A \to X$ that sends an input symbol $a \in A$ to a state $t(x)(a)$, an initial state $i \in X$ (formally denoted by a function $i\colon 1 \to X$), and a set of final (or accepting) states given by a function $f\colon X \to 2$, sending a state to 1 if it is final and to 0 if it is not.

We introduce *reachability* and *observability* of deterministic automata by means of the following diagram:



(2)

in the middle of which we have our automaton (X, t, i, f).

On the left, we have the set A^* of all words over A. We view this as an automaton with the empty word ε as initial state and with transition function

$$\alpha\colon A^* \to (A^*)^A \qquad \alpha(w)(a) = w \cdot a$$

On the right, we have the set 2^{A^*} of all languages over A, also viewed as an automaton. The transition function of this automaton is

$$\beta\colon 2^{A^*} \to (2^{A^*})^A \qquad \beta(L)(a) = \{w \in A^* \mid a \cdot w \in L\} \tag{3}$$

where $\beta(L)(a)$ is the so-called *(left) a-derivative of the language L*; and a final state function

$$\varepsilon?\colon 2^{A^*} \to 2$$

that maps a language to 1 if it contains the empty word, and to 0 if it does not. Notice that in the automaton on the left we do not care about a final state, only the initial

state matters; this makes sense from the point of view of reachability, we only care where we can get to from the initial state. Similarly for the automaton on the right, we do not care about the initial state, only about the final state.

Horizontally, we have functions r and o that we will introduce next. First we define x_w, for $x \in X$ and $w \in A^*$, inductively by

$$x_\varepsilon = x \qquad x_{w \cdot a} = t(x_w)(a)$$

i.e., x_w is the state reached from x by inputting (all the letters of) the word w. With this notation, we now define

$$\begin{array}{ll} r\colon A^* \to X & o\colon X \to 2^{A^*} \\ r(w) = i_w & o(x)(w) = f(x_w) \end{array}$$

Thus r sends a word w to the state i_w that is reached from the initial state $i \in X$ by inputting the word w; and o sends a state x to the language it accepts. That is, switching freely between languages as maps and languages as subsets,

$$o(x) = \{ w \in A^* \mid f(x_w) = 1 \} \tag{4}$$

We think of $o(x)$ as the semantics or the *behavior* of the state x.

The functions r and o are homomorphisms in the precise sense that they make the triangles and squares of diagram (2) commute. In order to understand the latter, we note that at the bottom of the diagram, we use, for $f\colon V \to W$, the notation

$$f^A\colon V^A \to W^A$$

to denote the function defined by $f^A(\phi)(a) = f(\phi(a))$, for $\phi\colon A \to V$ and $a \in A$.

One can readily see that the function r is uniquely determined by the functions i and t; similarly, the function o is uniquely determined by the functions t and f. In categorical terms, the unique existence of r is a consequence of A^* being an initial algebra of the functor $1 + (A \times -)$; similarly, the unique existence of o rests on the fact that 2^{A^*} is a final coalgebra of the functor $2 \times (-)^A$.

Having explained diagram (2), we can now give the following definition.

Definition 2.1 (*reachability, observability, minimality*). A deterministic automaton (X, t, i, f) is *reachable* if r is surjective, it is *observable* if o is injective, and it is *minimal* if it is both reachable and observable.

Thus (X, t, i, f) is reachable if all states are reachable from the initial state, that is, for every $x \in X$ there exists a word $w \in A^*$ such that $i_w = x$; and (X, t, i, f) is observable if different states recognize different languages, in other words, if they have different observable behavior. This explains the use of the word "observable", namely, an automaton is observable if its states can be unambiguously identified with their observable behaviour. Note that our definition of a minimal automaton coincides with the standard one, and for a fixed language L, the minimal automaton accepting L is unique up to isomorphism.

3. CONSTRUCTING THE REVERSE OF AN AUTOMATON

Next we show that by reversing the transitions, and by swapping the initial and final states of a deterministic automaton, one obtains a new automaton accepting the reversed language. By construction, this automaton will again be deterministic. Moreover, if the original automaton is reachable, the resulting one is observable.

Our construction will make use of the following operation:

$$2^{(-)}: \quad \begin{array}{c} V \\ {\scriptstyle f}\downarrow \\ W \end{array} \quad \mapsto \quad \begin{array}{c} 2^V \\ \uparrow{\scriptstyle 2^f} \\ 2^W \end{array}$$

which is defined, for a set V, by $2^V = \{S \mid S \subseteq V\}$ and, for $f\colon V \to W$ and $S \subseteq W$, by

$$2^f : 2^W \to 2^V \qquad 2^f(S) = \{v \in V \mid f(v) \in S\}$$

(In categorical terms, this is the contravariant powerset functor.)

The reverse construction. Given the transition function $t\colon X \to X^A$ of a deterministic automaton, we apply, from left to right, the following three transformations:

$$\begin{array}{c} X \\ {\scriptstyle t}\downarrow \\ X^A \end{array} \;\Big\|\; \begin{array}{c} X \times A \\ \downarrow \\ X \end{array} \;\Big|\; \begin{array}{c} 2^{X\times A} \\ \uparrow \\ 2^X \end{array} \;\Big\|\; \begin{array}{c} (2^X)^A \\ \uparrow{\scriptstyle 2^t} \\ 2^X \end{array}$$

The single, vertical line in the middle corresponds to an application of the operation $2^{(-)}$ introduced above. The double lines, on the left and on the right, indicate isomorphisms that are based on the operations of *currying* and *uncurrying*. The end result consists of a new set of states 2^X together with a new transition function (which by abuse of notation we denote by 2^t)

$$2^t :\ 2^X \to (2^X)^A \qquad 2^t(S)(a) = \{x \in X \mid t(x)(a) \in S\}$$

which maps any subset $S \subseteq X$, for any $a \in A$, to the set of all its a-predecessors. Note that the reversed transition function 2^t is again deterministic.

Initial becomes final. Applying the operation $2^{(-)}$ to the initial state (function) of our automaton X gives

$$\begin{array}{c} 1 \\ {\scriptstyle i}\downarrow \\ X \end{array} \;\Big|\; \begin{array}{c} 2 \\ \uparrow{\scriptstyle 2^i} \\ 2^X \end{array}$$

(where we write 2 for 2^1), by which we have transformed the initial state i into a final state function 2^i for the new automaton 2^X. We note that according to this new function 2^i, a subset $S \subseteq X$ is final (that is, is mapped to 1) precisely when $i \in S$.

Reachable becomes observable. Next we apply $2^{(-)}$ to the entire left hand-side of diagram (2), that is, to both t and i and to α and ε, as well as to the functions r and r^A. This yields the following commuting diagram:

$$\begin{array}{ccc} & & 2 \\ & {\scriptstyle 2^i}\nearrow & \uparrow{\scriptstyle 2^\varepsilon} \\ 2^X & \xrightarrow{2^r} & 2^{A^*} \\ {\scriptstyle 2^t}\downarrow & & \downarrow{\scriptstyle 2^\alpha} \\ (2^X)^A & \xrightarrow[2^{r^A}]{} & (2^{A^*})^A \end{array} \tag{5}$$

We note that for any language $L \in 2^{A^*}$, we have $2^{\varepsilon}(L) = \varepsilon?(L)$ and, for any $a \in A$,

$$2^{\alpha}(L)(a) = \{w \in A^* \mid w \cdot a \in L\}$$

The latter resembles the definition of $\beta(L)(a)$ but it is different in that it uses $w \cdot a$ instead of $a \cdot w$. By the universal property (of finality) of the triple $(2^{A^*}, \beta, \varepsilon?)$, there exists a unique homomorphism $rev\colon 2^{A^*} \to 2^{A^*}$ as shown here

$$\begin{array}{ccc} & & 2 \\ & {\scriptstyle 2^{\varepsilon}}\nearrow & \uparrow{\scriptstyle \varepsilon?} \\ 2^{A^*} & \xrightarrow[rev]{} & 2^{A^*} \\ \downarrow{\scriptstyle 2^{\alpha}} & & \downarrow{\scriptstyle \beta} \\ (2^{A^*})^A & \xrightarrow[rev^A]{} & (2^{A^*})^A \end{array} \tag{6}$$

which sends a language L to its reverse

$$rev(L) = \{w \in A^* \mid w^R \in L\}$$

where w^R is the reverse of w.

Combining diagrams (5) and (6) yields the following commuting diagram:

$$\begin{array}{ccccc} & & & & 2 \\ & {\scriptstyle 2^i}\nearrow & & {\scriptstyle 2^{\varepsilon}}\nearrow & \uparrow{\scriptstyle \varepsilon?} \\ 2^X & \xrightarrow[2^r]{} & 2^{A^*} & \xrightarrow[rev]{} & 2^{A^*} \\ \downarrow{\scriptstyle 2^t} & & \downarrow{\scriptstyle 2^{\alpha}} & & \downarrow{\scriptstyle \beta} \\ (2^X)^A & \xrightarrow[2^{r^A}]{} & (2^{A^*})^A & \xrightarrow[rev^A]{} & (2^{A^*})^A \end{array}$$

Thus we see that the composition of rev and 2^r (is the unique function that) makes the following diagram commute:

$$\begin{array}{ccc} & & 2 \\ & {\scriptstyle 2^i}\nearrow & \uparrow{\scriptstyle \varepsilon?} \\ 2^X & \xrightarrow[O]{} & 2^{A^*} \\ \downarrow{\scriptstyle 2^t} & & \downarrow{\scriptstyle \beta} \\ (2^X)^A & \xrightarrow[O^A]{} & (2^{A^*})^A \end{array} \qquad O = rev \circ 2^r \tag{7}$$

One can easily show that it satisfies, for any $S \subseteq X$,

$$O(S) = \{w^R \in A^* \mid i_w \in S\} \tag{8}$$

Final becomes initial. The following bijective correspondence

$$\begin{array}{ccc} 2 & \Vert & 1 \\ {\scriptstyle f}\uparrow & \Vert & \downarrow{\scriptstyle f} \\ X & \Vert & 2^X \end{array}$$

(again an instance of currying) transforms the final state function f of the original automaton X into an initial state function of our new automaton 2^X, which we denote again by f. It will induce, by the universal property of $(A^*, \varepsilon, \alpha)$, a unique homomorphism R as follows:

$$\begin{array}{ccc} 1 & & \\ {\scriptstyle\varepsilon}\downarrow & \searrow^{f} & \\ A^* & \xrightarrow{R} & 2^X \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle 2^t} \\ (A^*)^A & \xrightarrow[R^A]{} & (2^X)^A \end{array} \tag{9}$$

Putting everything together. By now, we have obtained the following, new deterministic automaton:

$$\begin{array}{ccccc} 1 & & & & 2 \\ {\scriptstyle\varepsilon}\downarrow & \searrow^{f} & & \nearrow^{2^i} & \uparrow{\scriptstyle\varepsilon?} \\ A^* & \xrightarrow[R]{} & 2^X & \xrightarrow[O]{} & 2^{A^*} \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle 2^t} & & \downarrow{\scriptstyle\beta} \\ (A^*)^A & \xrightarrow[R^A]{} & (2^X)^A & \xrightarrow[O^A]{} & (2^{A^*})^A \end{array} \tag{10}$$

where the above diagram is simply the combination of diagrams (9) and (7) above.

THEOREM 3.1. *Let (X, t, i, f) be a deterministic automaton and let $(2^X, 2^t, f, 2^i)$ be the reversed deterministic automaton constructed like above.*

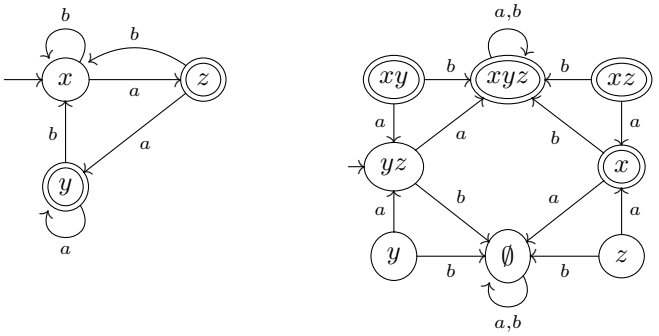
(1) If (X, t, i, f) is reachable, then $(2^X, 2^t, f, 2^i)$ is observable.
(2) If (X, t, i, f) accepts the language L, then $(2^X, 2^t, f, 2^i)$ accepts $rev(L)$.

PROOF. As the operation $2^{(-)}$ transforms surjections into injections (and since rev is a bijection), reachability of (X, t, i, f) implies observability of $(2^X, 2^t, f, 2^i)$. The second statement follows from the fact that we have

$$\begin{aligned} O(f) &= \{w \in A^* \mid 2^i(f_w) = 1\} \\ &= \{w^R \in A^* \mid i_w \in f\} \qquad \text{[by identity (8)]} \\ &= rev(\{w \in A^* \mid i_w \in f\}) \\ &= rev(o(i)) \end{aligned}$$

□

We consider the following two automata. In the picture below, an arrow points to the initial state and a double circle indicates that a state is final:



(11)

The automaton on the left is reachable (but not observable, since y and z accept the same language $\{a,b\}^*a+1$). Applying the reverse construction yields the automaton on the right, which is observable (all the states accept different languages) but not reachable (e.g., the state $\{x,y\}$, denoted by xy, is not reachable from the initial state $\{y,z\}$). Furthermore, the language accepted by the automaton on the right, $a\{a,b\}^*$, is the reverse of the language accepted by the automaton on the left, which is $\{a,b\}^*a$.

4. BRZOZOWSKI'S ALGORITHM

As an immediate consequence, we obtain the following version of Brzozowski's algorithm.

COROLLARY 4.1. *Given a deterministic automaton accepting a language L,*

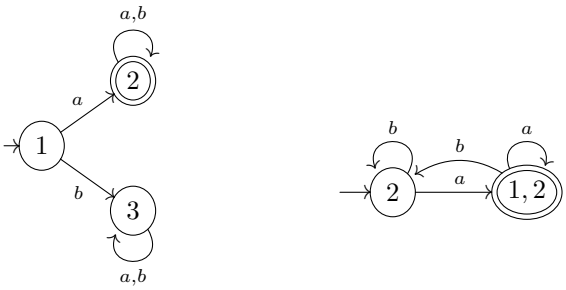
(*1*) *apply the reverse construction,*
(*2*) *take the reachable part,*
(*3*) *apply the reverse construction,*
(*4*) *take the reachable part.*

The resulting automaton is the minimal automaton accepting L.

PROOF. The automaton obtained after steps (1) and (2) is reachable and accepts $rev(L)$. After step (3), the automaton is observable and accepts $rev(rev(L))=L$. Finally, taking reachability in step (4) yields a minimal automaton accepting L. □

Note that in Corollary 4.1, if the original automaton is already reachable, then the automaton obtained after step (2) is a minimal automaton accepting $rev(L)$.

We saw that applying the reverse construction (step (1)) to the left automaton in (11) resulted in the automaton on the right in (11). By taking the reachable part of the latter (step (2)), we obtain the automaton depicted below on the left (where $1=\{y,z\}$, $2=\{x,y,z\}$ and $3=\emptyset$):



(12)

The automaton on the right in (12) is obtained by, once more, reversing-determinizing (step (3)) and taking the reachable part (step (4)). It is the minimization of the automaton we started with.

Note that the original algorithm in [Brzozowski 1962] works with non-deterministic automata, while Corollary 4.1 is restricted to deterministic automata. In Section 7, we will show how to treat non-deterministic automata and in Section 8 we will further generalize to weighted automata. First, in the next section, we extend our result to deterministic Moore automata.

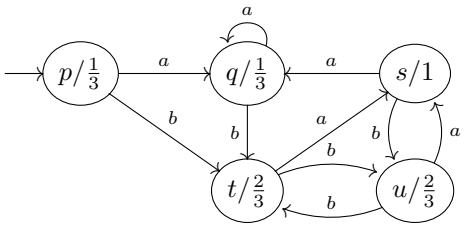
5. MOORE AUTOMATA

Moore automata generalise deterministic automata by allowing outputs in an arbitrary set B, rather than just 2. Formally, a Moore automaton with inputs in A and

outputs in B consists of a set of states X, an initial state $i\colon 1 \to X$, a transition function $t\colon X \to X^A$ and an output function $f\colon X \to B$. Moore automata accept functions in B^{A^*} (that is functions $\phi\colon A^* \to B$) instead of languages in 2^{A^*}.

Here is in a nutshell how our story above can be generalised to Moore automata. We can redraw diagram (2) by simply replacing 2 with B. We then define reachability, observability, and minimality as before. Next, we adopt our procedure of reversing transitions by using (the contravariant functor) $B^{(-)}$ instead of $2^{(-)}$: for all sets V, $B^V = \{\phi \mid \phi\colon V \to B\}$ and, for all functions $g\colon V \to W$, the function $B^g\colon B^W \to B^V$ maps each $\phi \in B^W$ to $B^g(\phi) = \phi \circ g$. Finally, all the results discussed above will also hold for Moore automata. We note that although the output set B may be infinite, if the state space X is finite, then the range of the output function $B_0 = f[X] \subseteq B$ is also finite, and we can view the Moore machine as having output in B_0. Consequently, the state space B_0^X of the reversed Moore machine is also finite. The next example illustrates the minimization of a Moore automaton.

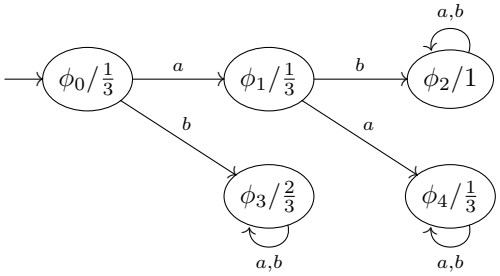
We consider the following Moore automaton with inputs in $A = \{a, b\}$ and output in the subset $B = \{\frac{1}{3}, \frac{2}{3}, 1\}$ of the rational numbers $\mathbb{Q}$. In the picture below, the output value r of a state x is indicated inside the circle by x/r:



(13)

The automaton accepts a function in B^{A^*} mapping every word w ending with ba to 1, every word ending with b to $\frac{2}{3}$ and every other word to $\frac{1}{3}$. Clearly, the automaton is reachable from p. However, it is not observable, since, for example, the states p and q accept the same function.

Applying the reverse construction (step (1)) yields a Moore automaton with B^S as set of states, where $S = \{p, q, s, t, u\}$ is the set of states of the original automaton. The output value of a state $\phi\colon S \to B$ is given by $\phi(p)$, where p is the initial state of the original automaton. Further, the output function of the original automaton becomes the new initial state, i.e., the function $\phi_0\colon S \to B$ mapping p and q to $\frac{1}{3}$, t and u to $\frac{2}{3}$, and s to 1. The reachable part of the (finite) state space B^S can be computed using a standard least fixpoint algorithm that starts from the initial state ϕ_0, and iteratively adds successor states until no new states are found. We obtain the following automaton that has only five states.

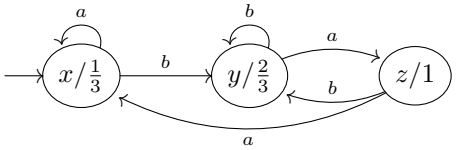


(14)

We do not spell out the full definition of the above states. As an example, the state ϕ_1 consists of the map assigning p, q and s to $\frac{1}{3}$ (that is $\phi_0(q)$), and t, u to 1 (that is $\phi_0(s)$).

Note that the function in B^{A^*} accepted by the state ϕ_0 maps each word $w \in \{a, b\}^*$ to the same value where the reverse word w^R is mapped by the function accepted by the original automaton in (13). More precisely, it maps words which begin with ab to 1, words which begin with b to $\frac{2}{3}$, and all other words to $\frac{1}{3}$.

If we repeat the reverse construction one more time (step (3)), and take the reachable automaton from the new initial state (step (4)), we obtain the minimal Moore automaton equivalent to the one in (13):



(15)

6. KLEENE ALGEBRA WITH TESTS

Kleene algebra with tests (KAT) are a simple, but powerful, extension of regular expressions and Kozen's coinductive calculus of KAT [Kozen 1997; 2008] provides a method for deriving a Moore automaton from a KAT expression. In this section, we show how the algorithm above for Moore automata can be applied in order to obtain a minimal automaton recognizing a KAT expression.

We will first recall the coalgebraic theory of Kleene algebra with tests from [Kozen 2008]. The proof of the existence of finite automata for KAT expressions in Section 6.4 is essentially the same as for the case of (classical) deterministic automata and regular expressions. It can thereby be seen as a simplification of the proof given in [Kozen 2008, Section 5.1].

6.1. Expressions

Let Σ be a set of *primitive action* symbols $p, q \in \Sigma$ and let T be a finite set of *primitive test* symbols $t \in T$. We define the set $BExp$ of (Boolean) *tests* as the Boolean terms over T:

$$b \in BExp ::= \quad t \in T \mid b_1 b_2 \mid b_1 + b_2 \mid \bar{b} \mid 0 \mid 1$$

The set of *atoms* α, β is $At = 2^T$. Let $\equiv$ denote Boolean equivalence on the set $BExp$. The quotient of $BExp$ with respect to $\equiv$ is then a Boolean algebra B satisfying

$$B \;=\; (BExp/\equiv) = \quad 2^{2^T} \;= 2^{At}$$

(the second equality is actually an isomorphism). Note that the atoms of the Boolean algebra B are (indeed) given by the set At. We denote the $\equiv$-equivalence class of $b \in BExp$ by $[b]$ and define, for $\alpha \in At$, $\alpha \leq b \iff \alpha \in [b] \in 2^{At}$ or, equivalently, $\alpha \leq b \iff \alpha + b \equiv b$. Next we define the set Exp of ***KAT expressions*** e, f by

$$e \in Exp ::= \quad p \in \Sigma \mid b \in BExp \mid ef \mid e + f \mid e^*$$

6.2. Automata on guarded strings

We define the set GS of *guarded strings* x, y by

$$GS = \; (At \times \Sigma)^* At$$

We shall denote the elements of $At \times \Sigma$ by roman letters a, b and the elements of GS by strings $x = a_1 a_2 \cdots a_n \alpha \in GS$ for $n \geq 0$, $a_i \in At \times \Sigma$ and $\alpha \in At$.

An *automaton* on guarded strings consists of a set of states S together with an output function

$$f \colon S \to 2^{At}$$

and a transition function

$$t\colon S \to S^{At\times\Sigma}$$

mapping every $s \in S$, for every $a \in At \times \Sigma$, to a next state $t(s)(a) \in S$, also denoted by $s_a = t(s)(a)$ Such automata are, in other words, coalgebras $(S, \langle f, t\rangle)$ of the set functor $F(S) = 2^{At} \times S^{At\times\Sigma}$.

The carrier of the *final* coalgebra of this functor consists, as usual, of the set

$$(2^{At})^{(At\times\Sigma)^*} \;\cong\; 2^{At(At\times\Sigma)^*} \;\cong\; 2^{(At\times\Sigma)^*At} \;=\; 2^{GS}$$

that is, the set of languages over guarded strings. It carries an automaton structure

$$f\colon 2^{GS} \to 2^{At} \quad \text{and} \quad t\colon 2^{GS} \to (2^{GS})^{At\times\Sigma}$$

given, for a language $K \in 2^{GS}$ and $a \in At \times \Sigma$ by

$$\begin{aligned} f(K) \;&=\; \{\alpha \in At \mid\; \alpha \in K\} = \; K \cap At \\ t(K)(a) \;&=\; K_a \;= \{x \in GS \mid\; ax \in K\} \end{aligned}$$

The latter is the familiar derivative (or left quotient) of languages (over the alphabet $At \times \Sigma$).

6.3. The automaton of KAT expressions

By using (a slight variation of) the syntactic Brzozowski derivatives, also the set Exp of KAT expressions can be supplied with an automaton structure

$$f\colon Exp \to 2^{At} \quad \text{and} \quad t\colon Exp \to Exp^{At\times\Sigma}$$

First we define the output $f(e)$ by induction on the structure of $d, e \in Exp$:

$$f(p) = \emptyset \quad f(b) = \{\alpha \in At \mid \alpha \leq b\,\} \quad f(de) = f(d) \cap f(e) \quad f(d{+}e) = f(d) \cup f(e) \quad f(e^*) = At$$

Next we define the a-derivative $e_a = t(e)(a) \in Exp$, for $e \in Exp$ and $a = \langle \alpha, q\rangle \in At \times \Sigma$, again by induction on the structure of e:

$$p_a = \; p_{\langle\alpha,q\rangle} = \begin{cases} 1 \text{ if } p = q \\ 0 \text{ if } p \neq q \end{cases} \qquad (de)_a = \; (de)_{\langle\alpha,q\rangle} = \begin{cases} d_a f + e_a & \text{if } \alpha \in o(d) \\ d_a f & \text{if } \alpha \notin o(d) \end{cases}$$

$$b_a = 0 \qquad (e + f)_a = e_a + f_a \qquad (e^*)_a = e_a e^*$$

By finality, there now exists a unique homomorphism

$$\begin{array}{ccc} Exp & \overset{o}{- - - - -} & 2^{GS} \\ {\scriptstyle\langle t,f\rangle}\downarrow & & \downarrow{\scriptstyle\langle t,f\rangle} \\ 2^{At} \times Exp^{At\times\Sigma} & - - - - - \to & 2^{At} \times (2^{GS})^{At\times\Sigma} \end{array}$$

which assigns to each KAT expression the language (of guarded strings) that it denotes.

6.4. Finite automata for KAT expressions

For every $e \in Exp$ and $w \in (At \times \Sigma)^*$, we define the repeated derivative e_w by $e_\varepsilon = e$ and $e_{wa} = \; (e_w)_a$.

PROPOSITION 6.1. For $e, f \in Exp$ and $w \in (At \times \Sigma)^*$, the repeated derivatives $(ef)_w$, $(e+f)_w$ and $(e^*)_w$ are of the form

$$\begin{aligned}(ef)_w &= e_{t_1} f + \cdots + e_{t_k} f + f_{u_1} + \cdots + f_{u_l} \\ (e+f)_w &= e_w + f_w \\ (e^*)_w &= e_{v_1} e^* + \cdots + e_{v_m} e^*\end{aligned}$$

for $k, l, m \geq 0$, $t_i, u_i, v_i \in (At \times \Sigma)^*$.

PROOF. The proof is by straightforward induction on the syntactic structure of KAT expressions. □

By the fact that the repeated derivatives of basic KAT expressions $p \in \Sigma$ and $b \in BExp$ are contained in $\{0, 1\}$ and by Proposition 6.1, we can construct for any KAT expression $e \in Exp$ a finite automaton with e as designated (initial) state. This automaton is essentially the subautomaton generated by e quotiented with idempotence. This means that in the repeated derivatives of e we remove double occurrences of expressions g in sums of the form $\cdots + g + \cdots + g + \cdots$. Modulo this reduction, there are only finitely many repeated derivatives. For instance, suppose by induction that we have proved that the number of repeated derivatives of expressions e and f is bounded by $\sharp_w e \leq N$ and $\sharp_w f \leq M$. Then it follows from Proposition 6.1 that the total number $\sharp_w ef$ of repeated derivatives of ef (with double occurrences of expressions removed) is bounded by 2^{N+M}.

Note that it is very easy to come up with much sharper bounds for $\sharp_w e$, but all we wanted to show here is the *existence* of finite automata for KAT expressions. Also note that the procedure described in the paragraph above does *not* amount to taking the quotient of Exp with respect to the equivalence induced by the axioms ACI (associativity - commutativity - idempotency) of $+$. For instance, for primitive test symbols $p \neq q$, the KAT expressions $p + q$ and $q + p$ will *not* be identified; our procedure yields two different (but bisimilar) automata for these expressions.

6.5. Brzozowski meets Kozen: a minimization algorithm for KAT

In this section, we show how to obtain a minimal automaton on guarded strings corresponding to a KAT expression e.

We will use the following two KAT expressions, over the one letter alphabets $\Sigma = \{p\}$ and $T = \{b\}$, to illustrate the algorithm

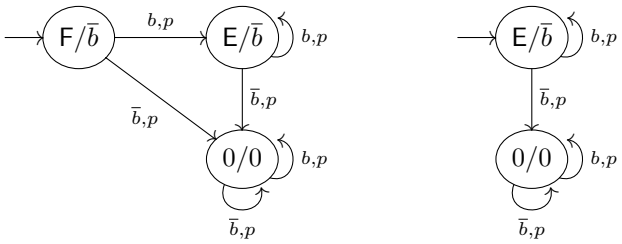
$$\mathsf{E} = (bp)^* \overline{b} \qquad \mathsf{F} = \overline{b} + bp(bp)^* \overline{b}$$

The expressions above denote, respectively, the following two simple imperative programs

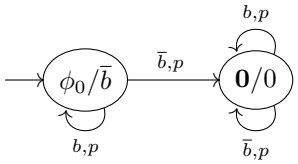
```
while b                 if b
    p;                   then {
                             p ;
                             while b
                                 p;
                             }
                          else skip;
```

Intuitively, it is easy to see that the two programs are equivalent. We will show that the automaton corresponding to the expression F can be minimized to the automaton

corresponding to E. The subautomata generated by E and F in the automaton of KAT expressions are the following:



By applying reverse construction (step (1)) to the automaton on the left and then taking its reachable part (step (2)), we obtain the following automaton.



where $\phi_0 \colon \{\mathsf{F}, \mathsf{E}, 0\} \to B$ is the function defined by

$$\phi_0(\mathsf{F}) = \overline{b} \qquad \phi_0(\mathsf{E}) = \overline{b} \qquad \phi_0(0) = 0$$

and $\mathbf{0} \colon \{\mathsf{F}, \mathsf{E}, 0\} \to B$ is the function mapping every state to 0.

In this particular example, when we execute steps (3) and (4) for the above automaton, we recover an isomorphic automaton. This is as expected, since the automaton above is precisely the automaton for the expression E where the `while` loop is completely folded.

7. NON-DETERMINISTIC AUTOMATA

A non-deterministic automaton with input from a finite alphabet A is given by a finite set X of states, a transition function $t \colon X \to \mathcal{P}_\omega(X)^A$ that sends a state $x \in X$ and an input symbol $a \in A$ to a finite set of states $t(x)(a)$, a set of initial states given by a function $i \colon 1 \to \mathcal{P}_\omega(X)$, and a set of final (or accepting) states given by a function $f \colon X \to 2$ which sends a state to 1 if it is final and to 0 if it is not.

In order to find a minimal deterministic automaton recognizing the same language of a non-deterministic automaton, we could first determinize the original automaton and then apply Brzozowski's algorithm in Corollary 4.1. Next we show how one can directly construct, from a non-deterministic automaton, a deterministic one recognizing the reverse language. Once we have this automaton, we can apply Theorem 3.1 and obtain a minimal deterministic automaton recognizing the language of the automaton we start with. This saves one determininization step since determinization of the original automaton and step (1) of Corollary 4.1 are essentially combined into one determinization step.

7.1. The reverse of a non-deterministic automaton

Our construction will make use of the following *lower inverse* operation:

$$\begin{array}{ccc} V & & \mathcal{P}_\omega(V) \\ {\scriptstyle f}\downarrow & \mapsto & \uparrow{\scriptstyle f^\diamond} \\ \mathcal{P}_\omega(W) & & \mathcal{P}_\omega(W) \end{array}$$

which is defined, for $f\colon V \to \mathcal{P}_\omega(W)$ and $S \subseteq W$, by

$$f^\diamond\colon \mathcal{P}_\omega(W) \to \mathcal{P}_\omega(V) \qquad f^\diamond(S) = \{v \in V \mid f(v) \cap S \neq \emptyset\}$$

(In topological terms, $f^\diamond$ is the lower inverse of a multifunction f.) Note that $f^\diamond(\emptyset) = \emptyset$ and that $f^\diamond(U_1 \cup U_2) = f^\diamond(U_1) \cup f^\diamond(U_2)$.

Now, given the transition function $t\colon X \to \mathcal{P}_\omega(X)^A$ of our non-deterministic automaton, we apply, from left to right, the following three transformations:

$$\begin{array}{c|c|c|c} X & X \times A & \mathcal{P}_\omega(X \times A) & \mathcal{P}_\omega(X)^A \\ \downarrow{\scriptstyle t} & \downarrow & \uparrow & \uparrow{\scriptstyle t^\diamond} \\ \mathcal{P}_\omega(X)^A & \mathcal{P}_\omega(X) & \mathcal{P}_\omega(X) & \mathcal{P}_\omega(X) \end{array}$$

The single, vertical line in the middle corresponds to an application of the lower inverse operation introduced above. The double lines on the left indicate the isomorphism based on the operations of *currying* and *uncurrying*, whereas the double lines on the right indicate the isomorphism between $\mathcal{P}_\omega(X \times A)$ and $\mathcal{P}_\omega(X)^A$ which holds since we assume A to be finite. The end result consists of a *deterministic transition function* on the set of states $\mathcal{P}_\omega(X)$, which by abuse of notation we simply denote by $t^\diamond$ (in analogy with the notation 2^t used in Section 3). Concretely,

$$t^\diamond\colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(X)^A \quad \text{where} \quad t^\diamond(S)(a) = \{x \in X \mid t(x)(a) \cap S \neq \emptyset\}$$

which maps any subset $S \subseteq X$, for any $a \in A$, to the set of all states that have an a-transition to a state in S. Note that this lower inverse construction yields a deterministic transition function.

If we apply this lower inverse construction to the initial states $i\colon 1 \to \mathcal{P}_\omega(X)$ of our original non-deterministic automaton, we obtain the function $i^\diamond\colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(1) \cong 2$ with $i^\diamond(S) = 1$ if and only if $i(*) \cap S \neq \emptyset$, where $1 = \{*\}$. Thus we have transformed the set of initial states i into a *final state map* $i^\diamond$ for the new deterministic automaton on $\mathcal{P}_\omega(X)$, according to which a subset $S \subseteq X$ is final (that is, is mapped to 1) precisely when it contains some initial state of the original automaton.

As a last step, we transform the final state function $f\colon X \to 2$ of the original non-deterministic automaton X into an *initial state* of our new deterministic automaton $\mathcal{P}_\omega(X)$, as before, by using the bijective correspondence between functions from X to 2 and elements of 2^X.

Putting everything together we now have constructed a deterministic automaton which recognizes the reverse of the language accepted by the original automaton.

THEOREM 7.1. *Let (X, t, i, f) be a non-deterministic automaton accepting the language L. Then the language accepted by the deterministic automaton $(\mathcal{P}_\omega(X), t^\diamond, f, i^\diamond)$ is $rev(L)$.*

PROOF. We show that, for all $x \in X$ and for all $w \in A^*$

$$t(x)(w) \cap f \neq \emptyset \iff x \in t^\diamond(f)(w^R) \tag{16}$$

Note that here we are using the inductive extensions of t and $t^\diamond$ to words, with $t(q)(\varepsilon) = \{q\}$ and $t^\diamond(S)(\varepsilon) = S$. Also note that equation (16) is a slightly more general statement than the theorem, since we are not requiring x to be an initial state.

The proof is by induction on the length of words $w \in A^*$. For the empty word ε, note that

$$t(x)(\varepsilon) \cap f \neq \emptyset \iff \{x\} \cap f \neq \emptyset \iff x \in f \iff x \in t^\diamond(f)(\varepsilon).$$

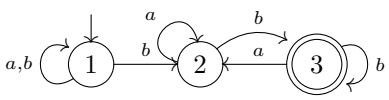
Take $a \in A$ and $w \in A^*$.

$$\begin{aligned}
t(x)(aw) \cap f \neq \emptyset &\iff \left(\bigcup_{y \in t(x)(a)} t(y)(w) \right) \cap f \neq \emptyset && \text{(definition of } t \text{ on words)} \\
&\iff \exists_{y \in t(x)(a)}\ t(y)(w) \cap f \neq \emptyset \\
&\iff \exists_{y \in t(x)(a)}\ y \in t^\diamond(f)(w^R) && \text{(induction hypothesis)} \\
&\iff t(x)(a) \cap t^\diamond(f)(w^R) \neq \emptyset \\
&\iff x \in t^\diamond(t^\diamond(f)(w^R))(a) && \text{(definition of } t^\diamond\text{)} \\
&\iff x \in t^\diamond(f)(w^R a) && \text{(definition of } t^\diamond \text{ on words)} \\
&\iff x \in t^\diamond(f)((aw)^R)
\end{aligned}$$

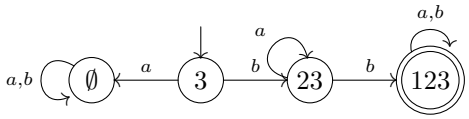
□

Now, by replacing step (1) in Corollary 4.1, by the lower inverse construction, we obtain the original Brzozowski algorithm [Brzozowski 1962] for non-deterministic automata. Indeed, since $(\mathcal{P}_\omega(X), t^\diamond, f, i^\diamond)$ is deterministic and accepts $rev(L)$, by taking its reachable part (step (2)), and by reversing it (step (3)), we obtain a deterministic automaton that is observable and accepts $rev(rev(L)) = L$. By taking its reachable part again (step (4)), we obtain a deterministic automaton that is minimal and accepts L.

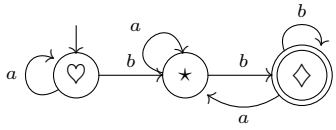
Example 7.2. We apply this algorithm to the following automaton (taken from [Adámek et al. 2012]):



which recognizes the language $L = \{wb \mid |w|_b \geq 1\}$ consisting all words ending in a b and containing at least two b's. Applying the lower inverse construction and taking reachability we obtain the automaton



which recognizes the reverse language $rev(L) = \{bw \mid |w|_b \geq 1\}$. We can now reverse (step (3)) and take the reachable part (step (4)), obtaining the following deterministic automaton



which is the minimal *deterministic* automaton recognizing the language $L = \{wb \mid |w|_b \geq 1\}$.

8. WEIGHTED AUTOMATA

Next we will generalize the above construction for non-deterministic automata to weighted automata over certain semirings.

8.1. Semirings and semimodules

Recall that a *semiring* is a tuple $(\mathbb{S}, +, \cdot, 0, 1)$ where $(\mathbb{S}, +, 0)$ and $(\mathbb{S}, \cdot, 1)$ are monoids, the former of which is commutative, and multiplication distributes over finite sums:

$$r \cdot 0 = 0 = 0 \cdot r \qquad r \cdot (s + t) = r \cdot s + r \cdot t \qquad (r + s) \cdot t = r \cdot t + s \cdot t$$

We just write $\mathbb{S}$ to denote a semiring. In this section we require a semiring to be *commutative*, which means that the monoid $(\mathbb{S}, \cdot, 1)$ is also commutative.

Examples of commutative semirings are: every field, the Boolean semiring 2, the semiring $(\mathbb{N}, +, \cdot, 0, 1)$ of natural numbers, and the tropical semiring $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$. The semiring of languages $(\mathcal{P}_\omega(A^*), \cup, \cdot, \emptyset, \varepsilon)$ with concatenation as multiplication is an example of a non-commutative semiring.

For a semiring $\mathbb{S}$, an $\mathbb{S}$-*semimodule* is a commutative monoid $(M, +, 0)$ with a left-action $\mathbb{S} \times M \to M$ denoted by juxtaposition rm for $r \in \mathbb{S}$ and $m \in M$, such that for every $r, s \in \mathbb{S}$ and every $m, n \in M$ the following laws hold:

$$\begin{array}{rclcrcl} (r+s)m & = & rm + sm & \qquad & r(m+n) & = & rm + rn \\ 0m & = & 0 & & r0 & = & 0 \\ 1m & = & m & & r(sm) & = & (r \cdot s)m \end{array}$$

Every semiring $\mathbb{S}$ is an $\mathbb{S}$-semimodule, where the action is taken to be just the semiring multiplication. Semilattices are another example of semimodules (for the Boolean semiring $\mathbb{S}$).

An $\mathbb{S}$-semimodule homomorphism is a monoid homomorphism $h \colon M_1 \to M_2$ such that $h(rm) = rh(m)$ for each $r \in \mathbb{S}$ and $m \in M_1$. $\mathbb{S}$-semimodule homomorphisms are also called *linear maps*. The set of all linear maps from a $\mathbb{S}$-semimodule M_1 to M_2 is denoted by $Lin(M_1, M_2)$.

Free $\mathbb{S}$-*semimodules* over a set X exist and can be constructed using the functor $V \colon \mathsf{Set} \to \mathsf{Set}$ defined on sets X and maps $h \colon X \to Y$ as follows:

$$\begin{aligned} V(X) &= \{\, \varphi \colon X \to \mathbb{S} \mid \varphi \text{ has finite support} \,\}, \\ V(h(\varphi)) &= \big(y \mapsto \textstyle\sum_{x \in h^{-1}(y)} \varphi(x)\big), \end{aligned}$$

where a function $\varphi \colon X \to \mathbb{S}$ is said to have finite support if $\varphi(x) \neq 0$ holds only for finitely many elements $x \in X$. One can think of $V(X)$ as consisting of all formal linear combinations of elements of X. In fact, $V(X)$ is the free $\mathbb{S}$-semimodule on X when equipped with the following pointwise $\mathbb{S}$-semimodule structure:

$$(\varphi_1 + \varphi_1)(x) = \varphi_1(x) + \varphi_2(x) \quad (s\varphi_1)(x) = s \cdot \varphi_1(x)\,.$$

Free semimodules enjoy the following universal property: for every function $h \colon X \to M$ from a set X to a semimodule M, there exists a unique linear map $h^\sharp \colon V(X) \to M$ that is called the *linear extension* of h. In the following, we will often identify a function with its linear extension (and thus we will often use h in place of $h^\sharp$). A *basis* of an $\mathbb{S}$-semimodule M is a subset X of M such that the linear extension of the function $X \hookrightarrow M$ is an isomorphism (that is, $V(X)$ and M are isomorphic as $\mathbb{S}$-semimodules).

Similarly to vector spaces, we define for an $\mathbb{S}$-semimodule M over a commutative semiring $\mathbb{S}$ its *dual space* $M^\star$ to be the set $Lin(M, \mathbb{S})$ of all linear maps between M and $\mathbb{S}$, endowed with the $\mathbb{S}$-semimodule structure obtained by taking pointwise addition and monoidal action: $(g + h)(m) = g(m) + h(m)$, and $(sh)(m) = s \cdot h(m)$. Note that $\mathbb{S} \cong V(1)$ and that $\mathbb{S}^\star = Lin(\mathbb{S}, \mathbb{S}) \cong \mathbb{S}$.

Unlike vector spaces, not all $\mathbb{S}$-semimodules are free semimodules (just as not all modules over a ring are free modules). An important observation in [Worthington 2009] is that for a commutative semiring $\mathbb{S}$, if $M = V(X)$ is a free $\mathbb{S}$-semimodule with finite

non-empty set X as basis, then the dual space $M^\star$ is a free $\mathbb{S}$-semimodule with as basis the following set (of the same cardinality as X):

$$\{x^\star \in Lin(M, \mathbb{S}) \mid x \in X \text{ and } x^\star(y) = 1 \text{ if } x = y, \text{ and } 0 \text{ otherwise}\}.$$

That is, $x^\star \colon M \to \mathbb{S}$ is the projection on the x-component. By considering elements of M as column vectors, the elements of $M^\star$ are row vectors. For a linear map $h \colon M_1 \to M_2$ between $\mathbb{S}$-semimodules M_1 and M_2 the *transpose* $h^T \colon M_2^\star \to M_1^\star$ is the map defined by

$$h^T(\varphi) = \varphi \circ h$$

for every $\varphi \in M_2^\star = Lin(M_2, \mathbb{S})$. It is easy to see that $h^T(\varphi) \in M_1^\star$. Note that in matrix-notation h^T is indeed the linear map given by the transposed matrix of h. From the commutativity of $\mathbb{S}$ it follows that $(g \circ h)^T = h^T \circ g^T$. Finally, we note that $V(X)$ and $V(X)^\star$ are isomorphic, since they are freely generated from bases of the same cardinality.

8.2. The reverse of a weighted automaton

A *weighted automaton* with finite input alphabet A and weights over a semiring $\mathbb{S}$ is given by a set of states X, a function $t \colon X \to V(X)^A$ (encoding the transition relation in the following way: the state $x \in X$ can make a transition to $y \in X$ with input $a \in A$ and weight $s \in \mathbb{S}$ if and only if $t(x)(a)(y) = s$), a final state function $f \colon X \to \mathbb{S}$ associating an output weight with every state, and an initial state function $i \colon 1 \to V(X)$.

It will be convenient to describe weighted automata using matrix-vector notation. The initial state function i is then a column vector, and the final state function f is a row vector. The transition function t can be seen as an A-indexed collection of maps $t_a \colon X \to V(X)$ which by linearity uniquely determines a linear map $t_a^\sharp \colon V(X) \to V(X)$. Hence t corresponds to an A-indexed collection of $X \times X$-matrices t_a where $t_a(y, x) = t(x)(a)(y)$ for all $x, y \in X$. We denote matrix multiplication by $\circ$. Given a state vector v in $V(X)$, the next state vector after reading letter a is given by the product $t_a \circ v$, and the output of a state vector v is the product $f \circ v$. The inductive extension of t from letters to words amounts to matrix-multiplication with $t_{a_0 \dots a_n} = t_{a_n} \circ \dots \circ t_{a_0}$ for $a_0 \dots a_n \in A^+$ and t_ε is equal to the identity matrix. Like Moore automata, weighted automata recognize functions in $\mathbb{S}^{A^*}$ which are usually referred to as *formal power series (over $\mathbb{S}$)*, and hereafter denoted by σ and ρ. More precisely, the formal power series recognized by a weighted automaton (X, t, i, f) is the function that maps $w \in A^*$ to $f(t(i)(w)) \in \mathbb{S}$, or in matrix notation $w \mapsto f \circ t_w \circ i$.

Notice that if we take $\mathbb{S}$ to be the Boolean semiring then weighted automata are precisely non-deterministic automata (because V and $\mathcal{P}_\omega$ are naturally isomorphic).

Next we recall from [Worthington 2009] how to construct from a weighted automaton (X, t, i, f) a deterministic Moore automaton recognizing the reverse language. The state space of this *reverse Moore automaton* will be $V(X)^\star$. Given the transition function $t \colon X \to V(X)^A$ of a weighted automaton, we apply, from left to right, the following three transformations:

$$\begin{array}{c|c|c|c} X & X \times A & V(X \times A)^\star & (V(X)^\star)^A \\ \downarrow{\scriptstyle t} & \downarrow & \uparrow & \uparrow{\scriptstyle t^T} \\ V(X)^A & V(X) & V(X)^\star & V(X)^\star \end{array}$$

Again, we abuse notation and simply write t^T for the end result, ignoring the isomorphisms. As before, the double lines on the left indicate the isomorphism based on the operations of *currying* and *uncurrying*, whereas the double lines on the right indicate

the isomorphism between $V(X \times A)^\star$ and $(V(X)^\star)^A$ obtained from the isomorphism $V(X \times A) \cong V(X)^A$ (for A and X finite sets) and the fact that $M^\star \cong M$ whenever M is free. The single, vertical line in the middle corresponds to an application of the transpose operation to the linear extension of the function $X \times A \to V(X)$. The end result consists of a *deterministic transition function* on the set of states $V(X)^\star$:

$$t^T \colon V(X)^\star \to (V(X)^\star)^A \qquad\qquad t^T(\varphi)(a)\left(\sum s_i x_i\right) = \sum s_i \cdot \varphi(t(x_i)(a))$$

where $\varphi \in V(X)^\star$, $a \in A$ and $\sum s_i x_i$ is an element in $V(X)$ expressed as a formal sum. In the special case of non-deterministic automata, transposing is easily seen to correspond to reversal of transitions.

If we transpose the (linear extension of the) initial state function $i \colon 1 \to V(X)$ of our original weighted automaton, we obtain the (linear) function $i^T \colon V(X)^\star \to V(1)^\star \cong \mathbb{S}$ with $i^T(\varphi) = \varphi(i(*))$, where $1 = \{*\}$. This function will give the output weight associated to each state in the reverse Moore automaton.

As a last step, we transpose the (linear extension of the) final state function $f \colon X \to \mathbb{S}$ of the original weighted automaton we obtain the map $f^T \colon \mathbb{S} \to V(X)^\star$ (recall that $\mathbb{S}^\star \cong \mathbb{S}$) defined by $f^T(s)\left(\sum s_i x_i\right) = \sum s \cdot s_i f(x_i)$. Because of linearity we can restrict its domain to the multiplicative unit of $\mathbb{S}$ and obtain the initial state of the reverse Moore automaton as the linear map $f^T(1) \colon \sum s_i x_i \mapsto \sum s_i f(x_i) = f^\sharp\left(\sum s_i x_i\right)$. Note that f^T is indeed the column vector obtained by transposing the row vector f.

THEOREM 8.1. *Let (X, t, i, f) be a weighted automaton over a commutative semiring $\mathbb{S}$ and a finite input alphabet A recognizing a formal power series $\sigma \colon A^* \to \mathbb{S}$. The* reverse Moore automaton *constructed above $(V(X)^\star, t^T, f^T, i^T)$ recognizes the power series σ^R which is defined for all $w \in A^*$ by*

$$\sigma^R(w) = \sigma(w^R)$$

where w^R is the reversed string of w.

PROOF. We need to show for all $w \in A^*$ that (using matrix notation)

$$f \circ t_w \circ i = i^T \circ t^T_{w^R} \circ f^T. \tag{17}$$

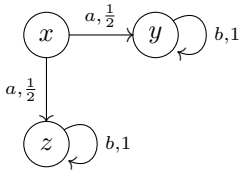
Since $f \circ t_w \circ i$ is in $\mathbb{S}$, i.e., it is a 1×1-matrix, it is equal to its own transpose, and hence (17) follows if we can show that $t^T_w = t^T_{w^R}$ for all $w \in A^*$. We prove this by induction on the length of w. For $w = \varepsilon$, t^T_ε and t_{ε^R} are both equal to the identity matrix. For the induction step, we have for $a \in A$ and $u \in A^*$:

$$t^T_{au} \;=\; (t_u \circ t_a)^T \;=\; t^T_a \circ t^T_u \;\overset{\text{(IH)}}{=}\; t^T_a \circ t^T_{u^R} \;=\; t^T_{u^R a} \;=\; t^T_{(au)^R}. \qquad \square$$

Now, by replacing step (1) in Corollary 4.1, by the reverse Moore automaton construction, we obtain a Brzozowski algorithm for weighted automata. Indeed, starting with a weighted automaton (X, t, i, f) which recognizes σ, the reverse Moore automaton $(V(X)^\star, t^T, f^T, i^T)$ recognizes σ^R, by taking its reachable part (step (2)), and reversing it (step (3)), we obtain a Moore automaton that is observable and recognizes σ. By taking its reachable part again (step (4)), we obtain a Moore automaton that is minimal and recognizes σ.

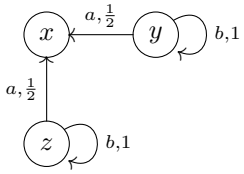
Example 8.2. We illustrate this algorithm with an example over the semiring of real numbers and the alphabet $A = \{a, b\}$. Take $X = \{x, y, z\}$, $i = (1\ 0\ 0)^T$ and $f =$

$(1\ \ 2\ \ 2)$, with the transition function represented below



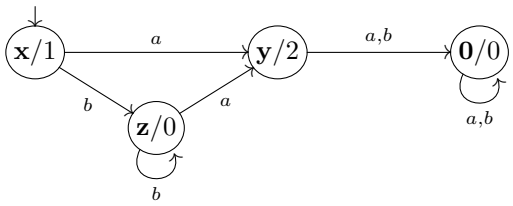
$$t_a = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & 0 \end{pmatrix} \qquad t_b = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This weighted automaton recognizes the power series σ that assigns 1 to ε, 2 to words in ab^* and 0 to any other word. The reverse Moore automaton has initial state $f^T = (1\ \ 2\ \ 2)^T$, output map (represented as a vector) $i^T = (1\ \ 0\ \ 0)$, and transition function $t^T \colon V(X)^\star \to (V(X)^\star)^A$ as represented below. Recall that $V(X)^\star \cong V(X)$ and t^T is determined by its action on the basis vectors $x^\star, y^\star, z^\star$ which we simply denote x, y, z.



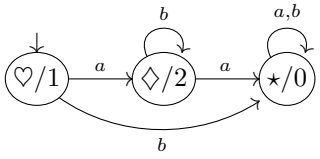
$$t_a^T = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad t_b^T = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The reachable part of the reverse Moore automaton is depicted here:



where $\mathbf{x} = f^T = (1\ \ 2\ \ 2)^T$, $\mathbf{y} = (2\ \ 0\ \ 0)^T$, $\mathbf{z} = (0\ \ 2\ \ 2)^T$ and $\mathbf{0} = (0\ \ 0\ \ 0)^T$.

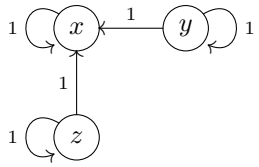
We can now easily see that this new automaton recognizes the power series ρ that assigns 1 to ε, 2 to words in b^*a and 0 to any other word. It is easy to see that indeed $\rho(w) = \sigma(w^R)$, for any $w \in A^*$. Now, we have a deterministic Moore automaton and we can execute steps (3) and (4) in order to obtain the minimal *Moore automaton* recognizing the power series σ.



where $\heartsuit$, $\diamondsuit$ and $\star$ stand for the functions $\{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{0}\} \to \mathbb{R}$ represented as vectors as, respectively, $(1\ \ 2\ \ 0\ \ 0)^T$, $(2\ \ 0\ \ 2\ \ 0)^T$ and $(0\ \ 0\ \ 0\ \ 0)^T$.

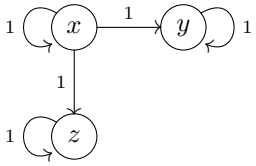
Example 8.3. We give another example of the construction with an example over the semiring of real numbers and a singleton alphabet $A = \{a\}$. Take $X = \{x, y, z\}$, $i = (1\ \ 0\ \ 1)^T$ and $f = (1\ \ 1\ \ 2)$, with transition function represented below (we omit the

label a)



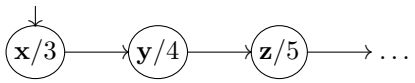
$$t = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This weighted automaton recognizes the power series σ that maps a^n into $3 + n$. Since $A = \{a\}$, each word w is equal to its reversed w^R and thus $\sigma^R = \sigma$. The reverse Moore automaton has initial state $f^T = (1\ 1\ 2)^T$, final state function $i^T = (1\ 0\ 1)$ and transition function:

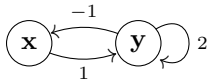


$$t^T = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Differently from the example above, the reachable part of the reverse Moore automaton is now infinite (below, $\mathbf{x} = f^T = (1\ 1\ 2)^T$, $\mathbf{y} = (1\ 2\ 3)^T$ and $\mathbf{z} = (1\ 3\ 4)^T$).



However, since $V(X)^\star$ is generated from a finite basis (for the properties discussed at the end of Section 8.1), then also the reachable states might be finitely generated. This is indeed the case of the above example where $\mathbf{z}$ can be expressed as linear combination of $\mathbf{x}$ and $\mathbf{y}$ ($\mathbf{z} = 2\mathbf{y} - \mathbf{x}$). Intuitively, the above infinite Moore automaton can be finitely represented by the following weighted automaton



having as initial state function $i = (1\ 0)^T$ and final state function $f = (3\ 4)$. Now, by applying steps (3) and (4) to the obtained Moore automaton (or to its finite representation), we get the exactly the same automaton (since, in this special case, $\sigma = \rho$).

We conclude this section by remarking that in general, and not unexpectedly, we cannot apply the algorithm above to all semirings. In particular, in the example above, if we would have taken the automaton to be over the semiring of natural numbers, then the reachable part of the reverse Moore automaton would not be finitely based (like the two states weighted automaton above). Note that as we remarked above $\mathbf{z} = 2\mathbf{y} - \mathbf{x}$ which requires negative coefficients. In general, we will be able to guarantee that the reachable part of the reverse Moore automaton is finitely based only for Noetherian semirings [Ézik and Maletti 2011; Bonsangue et al. 2012].

9. ADJUNCTIONS OF AUTOMATA: A CATEGORICAL PERSPECTIVE

In this section we will make explicit the categorical picture that underlies Theorem 3.1. The main observation here is that the dual self-adjunction on the category Set (induced by the contravariant powerset functor) extends to one on the category of deterministic automata. We will also see that Brzozowski's minimisation algorithm can be described

succinctly in terms of functors between categories of automata. From this categorical picture the Brzozowski algorithm for Moore automata is an immediate and easy generalisation.

9.1. Self-dual adjunction of Set

The category of sets and functions is denoted by Set. There is an adjunction between Set and $\mathsf{Set}^{\mathrm{op}}$ induced by the contravariant powerset functor $2^{(-)}\colon \mathsf{Set} \to \mathsf{Set}^{\mathrm{op}}$ (as illustrated on the left in (18)) which means that there is a natural bijection between morphisms as shown on the right in (18). Recall that a morphism $X \to Y$ in $\mathsf{Set}^{\mathrm{op}}$ is a morphism $Y \to X$ in Set, i.e. a function from Y to X. The $^{\mathrm{op}}$-notation indicates the contravariance. In particular, the functor 2^{op} does exactly the same as 2; the $^{\mathrm{op}}$-superscript just keeps track of the direction.

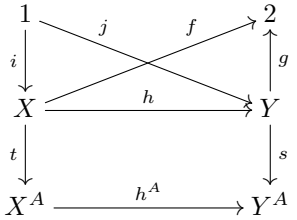
$$\mathsf{Set} \underset{2^{\mathrm{op}}}{\overset{2}{\rightleftarrows}} \mathsf{Set}^{\mathrm{op}} \qquad \frac{X \to 2^Y \quad \text{in Set}}{2^X \to Y \quad \text{in } \mathsf{Set}^{\mathrm{op}}} \tag{18}$$

In terms of set functions, the bijection of morphisms is given by taking exponential transpose which we denote by $f \mapsto \hat{f}$ (in both directions):

$$\frac{f\colon X \to 2^Y \quad \text{in Set}}{\hat{f}\colon Y \to 2^X \quad \text{in Set}} \qquad \text{given by:} \quad y \in f(x) \iff x \in \hat{f}(y) \tag{19}$$

9.2. Categories of automata

We denote by DA the category of deterministic automata and automaton morphisms. A deterministic automaton is both a $1 + (A \times -)$-algebra (initial state plus transition structure) and a $2 \times (-)^A$-coalgebra (output function plus transition structure), and an *automaton morphism* is a function that respects both structures. That is, an automaton morphism between automata (X, t, i, f) and (Y, s, j, g) is a map $h\colon X \to Y$ such that the following diagram commutes:

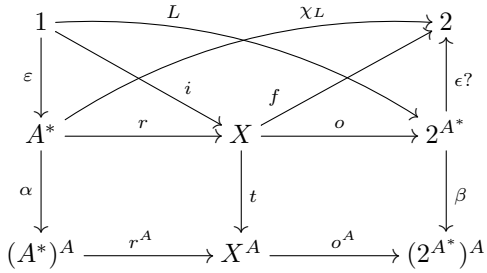


(20)

For example, in diagram (2) (on page 3) the reachability map r is a morphism of $1 + (A \times -)$-algebras, and the observability map o is a morphism of $2 \times (-)^A$-coalgebras, but not vice versa.

The category DA has neither initial nor final objects, since automaton morphisms must preserve the accepted language. For a given language $L \subseteq A^*$ we will therefore restrict our attention to the full subcategory $\mathsf{DA}(L)$ of DA which has as its objects all the deterministic automata that accept L. The category $\mathsf{DA}(L)$ has both an initial object and a final object that are obtained as follows. Consider again the diagram (2) which has on the left the initial $1+(A\times-)$-algebra $(A^*, \alpha, \varepsilon)$ and on the right the final $2\times(-)^A$-coalgebra $(2^{A^*}, \beta, \varepsilon?)$. The initial object of $\mathsf{DA}(L)$ is obtained by adding to $(A^*, \alpha, \varepsilon)$ as output function χ_L, the characteristic function of L. The final object of $\mathsf{DA}(L)$ is obtained by adding the language L as initial state to $(2^{A^*}, \beta, \varepsilon?)$. The reachability and

observability maps are now both $\mathsf{DA}(L)$ morphisms, and hence they are the initial, respectively final, morphism in $\mathsf{DA}(L)$:



(21)

The notions of reachability and observability can now be formulated in terms of initiality and finality:

— An automaton in $\mathsf{DA}(L)$ is reachable if the initial morphism r is surjective.
— An automaton in $\mathsf{DA}(L)$ is observable if the final morphism o is injective.

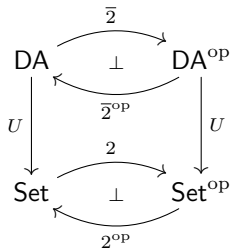
Note that the final morphism from the initial object $(A^*, \alpha, \varepsilon, \chi_L)$ assigns to a word $w \in A^*$ the set of all words $u \in A^*$ such that $wu \in L$. So the kernel of the final observability map is nothing but the Myhill-Nerode equivalence of L, and the image is a minimal automaton that accepts L which has as its states the set $\{\beta(L)(w) \mid w \in A^*\}$ of all derivatives of L. The picture in (21) thus nicely captures the well known equivalence:

(1) L is regular iff
(2) the set of derivatives of L is finite iff
(3) the Myhill-Nerode equivalence has finite index.

9.3. Self-dual adjunction of DA

The definition of the reverse automaton in Section 3 defines a map $\overline{2}$ on the objects of DA using the contravariant powerset functor 2 on Set. It can easily be verified that by taking $\overline{2}(f) = 2^f = f^{-1}$ for an automaton morphism f, that $\overline{2}$ is a contravariant functor on DA. We will now show that the functor $\overline{2}$ lifts the self-dual adjunction on Set to DA.

PROPOSITION 9.1. *The functor $\overline{2}$ is a lifting of the contravariant powerset functor 2 to the category* DA*, which means that the following diagram commutes (U denotes the forgetful functor which sends an automaton to its state set):*



In particular, $\overline{2}$ induces a self-dual adjunction on DA.

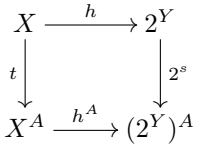
PROOF. The commutativity of the diagram is clear from the definition of the reverse automaton. To see that we indeed have an adjunction on DA we have to show that for

all $\mathcal{X} = \langle X, t, i, f\rangle$ and $\mathcal{Y} = \langle Y, s, j, g\rangle$ in DA there is a natural bijection of morphisms:

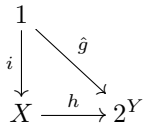
$$\frac{\mathcal{X} \to \overline{2}^{\mathrm{op}}(\mathcal{Y}) \quad \text{in DA}}{\overline{2}(\mathcal{X}) \to \mathcal{Y} \quad \text{in DA}^{\mathrm{op}}} \tag{22}$$

The above follows by showing that the bijection via exponential transpose given in (19) restricts to one between DA-morphisms, that is, we have to prove that a function $h\colon X \to 2^Y$ is an DA-morphism exactly when its exponential transpose $\hat{h}\colon Y \to 2^X$ is.

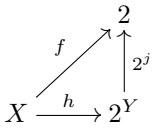
First we note that in the construction of the reverse automaton, the final state function is turned into the initial state function by taking exponential transpose. (This was left implicit in Section 3.) Thus we want to prove that for all functions $h\colon X \to 2^Y$, the following diagrams



(i)

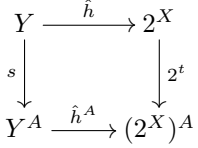


(ii)

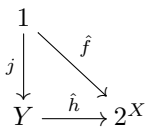


(iii)

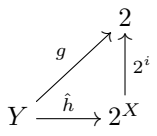
commute if and only if



(i')



(ii')



(iii')

commute. This will be easy to prove using the following fact: For any function $e\colon Z \to Y$, the exponential transpose of $\hat{h} \circ e$ is equal to $2^e \circ h$, i.e.,

$$\frac{Z \xrightarrow{e} Y \xrightarrow{\hat{h}} 2^X}{X \xrightarrow{h} 2^Y \xrightarrow{2^e} 2^Z} \tag{23}$$

since for all $x \in X, z \in Z$:

$$x \in \hat{h}(e(z)) \iff e(z) \in h(x) \iff z \in e^{-1}(h(x)) = 2^e(h(x)).$$

Now [(iii) $\Leftrightarrow$ (ii')] and [(ii) $\Leftrightarrow$ (iii')] are easily obtained from (23):

$$\begin{aligned} f = 2^j \circ h &\iff \hat{f} = \hat{h} \circ j \quad \text{and} \\ g = 2^i \circ \hat{h} &\iff \hat{g} = h \circ i. \end{aligned}$$

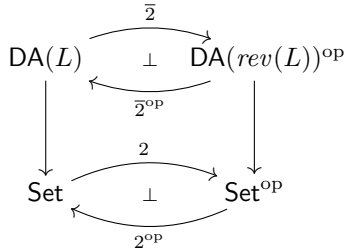
To see that [(i) $\Leftrightarrow$ (i')] holds, note that the commutativity of these diagrams can be formulated parametric in $a \in A$. We will use the notation t_a for the transition function for label a, i.e., $t_a\colon X \to X$ where $t_a(x) = t(x)(a)$. We have:

$$\begin{aligned} &\text{(i) commutes} \iff \text{for all } a \in A\colon 2^{s_a} \circ h = h \circ t_a \\ &\text{(i') commutes} \iff \text{for all } a \in A\colon 2^{t_a} \circ \hat{h} = \hat{h} \circ s_a. \end{aligned}$$

By taking transposes on both sides (and using $\hat{\hat{h}} = h$) it follows by (23) that (i) commutes iff (i') does. $\square$

Since the reverse automaton accepts the reverse language, the above adjunction restricts to one between $\mathsf{DA}(L)$ and $\mathsf{DA}(rev(L))^{\mathrm{op}}$.

COROLLARY 9.2. *There is a lifting of adjunctions:*



The functor $\overline{2}$ maps the initial object of $\mathsf{DA}(L)$ to the final object of $\mathsf{DA}(rev(L))$ and surjections to injections, hence using the definitions of reachability and observability given below (21), Theorem 3.1(1) is immediate. We restate the duality result in the present setting.

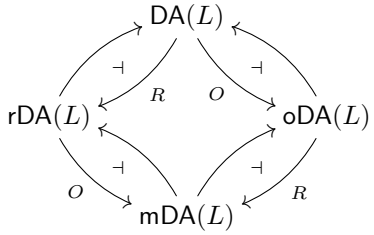
COROLLARY 9.3. *Let $\mathcal{A}$ be the initial object in $\mathsf{DA}(L)$, $\mathcal{Z}$ the final object in $\mathsf{DA}(rev(L))$, and let $\mathcal{X}$ be an automaton in $\mathsf{DA}(L)$. The initial reachability morphism $r\colon \mathcal{A} \to \mathcal{X}$ in $\mathsf{DA}(L)$ is mapped by $\overline{2}$ to the final morphism $\overline{2}(r)\colon \overline{2}(\mathcal{X}) \to \overline{2}(\mathcal{A}) = \mathcal{Z}$ in $\mathsf{DA}(rev(L))$. Consequently, if $\mathcal{X}$ is reachable, then $\overline{2}(\mathcal{X})$ is observable:*

$$r\colon \mathcal{A} \twoheadrightarrow \mathcal{X} \qquad \overset{\overline{2}}{\mapsto} \qquad o\colon \overline{2}(\mathcal{X}) \rightarrowtail \mathcal{Z}$$

9.4. Functorial reachability and observability

The subcategories of reachable and observable automata and their relationship play a central role in Brzozowski's algorithm. It is well known that we can make an automaton reachable by restricting to its reachable states, and we can make it observable by dividing out by the kernel $\sim$ of the final observability morphism o, that is, $x \sim y$ iff $o(x) = o(y)$. (The relation $\sim$ is also known as observational equivalence, bisimilarity or language equivalence.) These operations are functors, in fact, they are a so-called coreflector and reflector, respectively (cf. [Adámek et al. 2009]). More precisely, the inclusion functor $\mathsf{rDA} \hookrightarrow \mathsf{DA}$ of reachable DAs into DAs has a right adjoint R which sends a DA to its reachable part; and the inclusion functor $\mathsf{oDA} \hookrightarrow \mathsf{DA}$ of observable DAs into DAs has a left adjoint O which sends a DA to its observable quotient.

The coreflector R and reflector O restrict in the expected way to observable and reachable automata that accept a given language L. Clearly, applying O to a reachable automaton, or applying R to an observable automaton, yields a minimal automaton accepting the same language. This is illustrated in the following diagram where mDA denotes the full subcategory of minimal automata. (The restriction of R to $\mathsf{oDA}(L)$ we denote also by R, similarly for O):



(24)

9.5. Brzozowski's algorithm, categorically

The composition $R \circ O$ (and also $O \circ R$) describes the direct way of minimising an automaton, namely by taking the quotient with observational equivalence and mak-

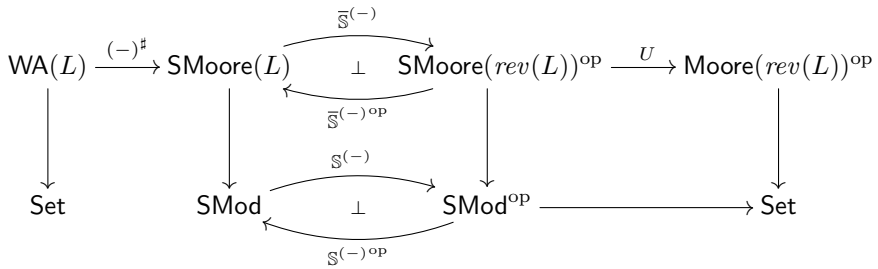
ing it reachable. Brzozowski's algorithm avoids the quotient construction by replacing it with taking reachability in the dual category of $\mathsf{DA}(rev(L))$. As we already know, this works thanks to the fact that $\overline{2}$ transforms reachability into observability (cf. Theorem 3.1(i) and Corollary 9.3). In categorical terms, Corollary 9.3 says that the functor $\overline{2}$ restricts to $\overline{2}\colon \mathsf{rDA}(L) \to \mathsf{oDA}(rev(L))^{\mathrm{op}}$. Dually, the functor $\overline{2}^{\mathrm{op}}$ restricts to $\overline{2}^{\mathrm{op}}\colon \mathsf{rDA}(rev(L))^{\mathrm{op}} \to \mathsf{oDA}(L)$. Brzozowski's algorithm is thus the traversal of the following diagram from $\mathsf{DA}(L)$ to $\mathsf{mDA}(L)$ following the solid edges: $R \circ \overline{2}^{\mathrm{op}} \circ R^{\mathrm{op}} \circ \overline{2}$. (Note that R^{op} can simply be thought of as doing the same as R.)

$$\begin{array}{ccc} \mathsf{DA}(L) & \xrightarrow{\overline{2}} & \mathsf{DA}(rev(L))^{\mathrm{op}} \\ {\scriptstyle O}\,\vdots & & \downarrow{\scriptstyle R^{\mathrm{op}}} \\ \mathsf{oDA}(L) & \xleftarrow{\overline{2}^{\mathrm{op}}} & \mathsf{rDA}(rev(L))^{\mathrm{op}} \\ \downarrow{\scriptstyle R} & & \\ \mathsf{mDA}(L) & & \end{array} \tag{25}$$

9.6. Generalisation to Moore automata and weighted automata

The generalisation of Brzozowski's algorithm to Moore automata described in Section 5 is immediate in the categorical picture, since for every set B the contravariant Hom-functor $B^{(-)} = \mathsf{Set}(-, B)$ induces a self-dual adjunction on Set via exponential transpose; $B = 2$ is just a special case. All technical results such as (23) hold for an arbitrary set B.

The Brzozowski-style algorithm for weighted automata (of which NDAs are a special case) described in Section 8 can be explained categorically as follows. The key observation is that the language semantics of weighted automata is obtained by viewing a weighted automaton $\mathcal{X} = (X, t, i, f)$ as a Moore automaton $\mathcal{X}^\sharp = (V(X), t^\sharp, i, f^\sharp)$ in the category SMod of $\mathbb{S}$-semimodules and $\mathbb{S}$-semimodule homomorphisms. We denote the corresponding category as SMoore. We refer the reader to [**?**] for a detailed explanation of this construction on linear weighted automata (which are Moore automata over vector spaces), and to [Silva et al. 2010] for a novel perspective of this construction as a generalized powerset construction. Moore automata over SMod are reversed using the contravariant Hom-functor $\mathbb{S}^{(-)} = \mathsf{SMod}(-, \mathbb{S})$ which sends an $\mathbb{S}$-semimodule to its dual space, and which lifts to a dual adjunction of Moore automata over SMod. The construction of the reverse Moore machine of a weighted automaton is formally the composition $\overline{\mathbb{S}} \circ (-)^\sharp$. The situation is summarized here:



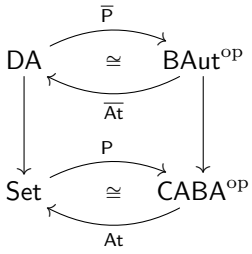
In the Brzozowski-style algorithm for weighted automata, we take the reachable part of the reverse Moore automaton. Note, however, that taking reachability is not

a legal operation in SMoore, since the reachable part of the state space may not be a (sub)semimodule. Therefore, we first forget the semimodule structure on the state space, and simply view the reverse Moore automaton as an "unstructured" Moore automaton over Set (the corresponding category is denoted by Moore). When the semiring $\mathbb{S}$ is infinite, this step may result in an infinite state space. Summarizing, the Brzozowski-style algorithm for weighted automata is obtained as the functor composition $R \circ \overline{B}^{\mathrm{op}} \circ R^{\mathrm{op}} \circ U \circ \overline{\mathbb{S}} \circ (-)^\sharp$ where U denotes the functor forgetting semimodule structure.

9.7. Duality via Boolean algebras

We briefly describe the duality between automata and Boolean automata that gives rise to a different approach to finding the minimal automaton that accepts a language L, cf. [Gehrke 2009; Gehrke et al. 2008; Roumen 2011].

A *Boolean automaton* is a deterministic automaton (X, t, i, f) where X is a complete, atomic Boolean algebra and t and f are Boolean algebra homomorphisms. A morphism of Boolean automata is an automaton morphism which is also a Boolean homomorphism. In [Roumen 2011], it is shown that the discrete duality between Set and complete atomic Boolean algebras (CABA) can be lifted to one between DA and Boolean automata (BAut), as illustrated in the diagram below. Here we write P for the contravariant functor that sends a set to its powerset algebra, and At for the operation that sends a Boolean algebra to its set of atoms. The lifting $\overline{\mathsf{P}}$ is essentially the same as $\overline{2}$, only now we take the Boolean algebra structure on the new state space 2^X into account. It can easily be verified that the reverse automaton is, in fact, a Boolean automaton. In the other direction, given a Boolean automaton (X, t, i, f) a (set-based) automaton is obtained by taking the atoms of X and restricting the reversed transition function 2^t to atoms.



Note that the above is a dual equivalence, so for any automaton $\mathcal{X} \in \mathsf{DA}$, $(\overline{\mathsf{At}} \circ \overline{\mathsf{P}})(\mathcal{X}) \cong \mathcal{X}$. A minimal deterministic automaton accepting a given language $L \subseteq A^*$ can now be obtained as the dual of the Boolean automaton $\mathcal{X}_r$ (accepting $rev(L)$) generated by the subset of right-derivatives of L in the Boolean algebra of all languages. In particular, minimisation via BAut is not the same as Brzozowski's algorithm, since taking the reachable part of $\overline{\mathsf{P}}(\mathcal{X})$ is not a valid operation in BAut, as in general, the reachable part of the state space is not a Boolean subalgebra of 2^X. For example, the reachable part of the automaton on the right of (11) consist of the states $\{x, y, z\}, \{y, z\}, \emptyset$ which do not form a Boolean algebra.

Another approach to minimisation via duality is given in [Bezhanishvili et al. 2012] which uses finite Boolean algebras with operators (FBAO) as the dual category. In fact, FBAO is just the subcategory of BAut consisting of finite Boolean automata, and their duality between partially observable deterministic automata and FBAO is a restriction of the above one to finite sets. Instead of taking the reachable part in the dual category (which is not valid in FBAO, as remarked above), their construction takes the subalgebra of modally definable subsets. This subalgebra is the image of the initial morphism in the category of all Boolean algebras with operators, and so it is a categorical analogue of taking reachability which is taking the image of the initial morphism in $\mathsf{DA}(L)$.

10. DISCUSSION

We have given a new description and a new correctness proof of Brzozowski's algorithm [Brzozowski 1962] for the minimization of deterministic automata. The novel proof relies on the duality between reachability and observability, a principle which has been originally introduced in systems theory [Kalman 1959] and then extended to automata theory [Arbib and Zeiger 1969; Arbib and Manes 1975a; 1980b]. Our formulation uses, albeit implicitly, standard elements of universal algebra and coalgebra [Rutten 2000].

In addition, we have shown how to generalise the algorithm to deterministic Moore automata, and applied it in the context of KAT expressions, non-deterministic and weighted automata. It is important to remark here that our minimimal realizations are Moore (deterministic) automata: our work does *not* concern notions of minimal (or canonical) non-deterministic automata or weighted automata. In this context, we mention that our Definition 2.1 is essentially the same as that of [Goguen 1972] which contains one of the first categorical treatments of minimal realizations, although duality does not play a role there. Finally, we have provided a categorical perspective where the dual adjunction underlying our construction is made precise, and Brzozowski's algorithm is shown to be functorial.

The duality between reachability and observability has been studied also in other contexts. In [Bidoit et al. 2001], this duality is used to establish several analogies between concepts from observational (coalgebraic) and constructor-based (algebraic) specifications. In the present paper, we have highlighted how Brzozowski's algorithm integrates both concepts.

Yet another approach, similar in spirit to ours, can be found in [Bezhanishvili et al. 2012], where a Stone-type duality between automata and their logical characterization is taken as a basis for minimization. We briefly discussed this work in Section 9.7, but the precise connection between that work and the present paper, remains to be better understood.

Stone-type dualities between automata and formal languages have previously been studied using the Boolean algebra structure of formal languages, see e.g. [Gehrke 2009; Gehrke et al. 2008; Roumen 2011]. Our picture does not involve Boolean algebras, and is in our view closer to the original formulation of Brzozowski than the Boolean algebra-based approaches.

Crucial for our work was the combined use of both algebra and coalgebra. Notably, it was important to include in the definition of automaton an initial state, which is in essence an algebraic concept. In coalgebra, one typically models automata without initial states. In that respect, so-called well-pointed coalgebras [Adámek et al. 2012], which are coalgebras with a designated initial state, may have some relevance for the further generalization of Brzozowski's algorithm.

A somewhat different notion of non-deterministic Moore automata has recently been introduced in [Castiglione et al. 2011]. It basically consists of a non-deterministic automaton with output in a set that comes equipped with a commutative and associative operator. Interesting for our context is their variant of Brzozowski's algorithm for the construction of a minimal deterministic Moore automaton that is equivalent to a given non-deterministic one. It would be interesting to study if our algorithm extends also to coherent non-deterministic Moore automata [Castiglione et al. 2011].

Brzozowski's minimization algorithm has also been combined with Brzozowski's method for deriving deterministic automata from regular expressions in [Watson 2000]. It would be useful to investigate how such a combination can be generalized to, for example, Kozen's calculus of Kleene algebras with tests [Kozen 1997], which would simplify the procedure we present in the current paper, eliminating one half of the Brzozowski's algorithm. All that is required for our approach is to construct from

a KAT expression an automaton accepting the guarded language denoted by the reverse of the input expression (represented as a function). Defining such an algorithm remains as future work.

Recently, in [Silva et al. 2010], we have presented a generalisation of the powerset construction for the determinisation of many different types of automata. Examples include Rabin's probabilistic automata [Rabin 1963] and weighted automata [Schützenberger 1961], to which our method applies in spite of the fact that the resulting deterministic (Moore) automaton is infinite. We would like to combine the results of the present paper with those of [Silva et al. 2010] to generalise Brzozowski's algorithm to probabilistic automata.

Acknowledgements. This work has been carried out under the Dutch NWO projects *CoRE: Coinductive Calculi for Regular Expressions* and *Behavioural Differential Equations*. The work of Helle Hvid Hansen was partially funded by the NWO under the Veni grant `639.021.231`. The work of Alexandra Silva was partially supported by Fundação para a Ciência e a Tecnologia, Portugal, under grant number `SFRH/BPD/71956/2010` and `FCOMP-01-0124-FEDER-020537` (program COMPETE). The work of Filippo Bonchi was partially supported by the PEPS-CNRS project CoGIP and by the project ANR 12IS02001 PACE. The work of Prakash Panangaden was funded by a grant from Natural Sciences and Engineering Research Council of Canada.

REFERENCES

ADÁMEK, J., BONCHI, F., HÜLSBUSCH, M., KÖNIG, B., MILIUS, S., AND SILVA, A. 2012. A coalgebraic perspective on minimization and determinization. See Birkedal [2012], 58–73.

ADÁMEK, J., HERRLICH, H., AND STRECKER, G. E. 2009. *Abstract and Concrete Categories - The Joy of Cats*. Dover Publications.

ADÁMEK, J., MILIUS, S., MOSS, L. S., AND SOUSA, L. 2012. Well-pointed coalgebras (extended abstract). See Birkedal [2012], 89–103.

ARBIB, M. AND ZEIGER, H. 1969. On the relevance of abstract algebra to control theory. *Automatica 5*, 589–606.

ARBIB, M. A. AND MANES, E. G. 1974. Machines in a category: An expository introduction. *SIAM Review 16*, 163–192.

ARBIB, M. A. AND MANES, E. G. 1975a. Adjoint machines, state-behavior machines, and duality. *Journal of Pure and Applied Algebra 6,* 3, 313 – 344.

ARBIB, M. A. AND MANES, E. G. 1975b. Extensions of semilattices. *The American Mathematical Monthly 82(7)*, 744–746.

ARBIB, M. A. AND MANES, E. G. 1975c. Fuzzy machines in a category. *Bulletin of the Australian Mathematical Society 13(2)*, 169–210.

ARBIB, M. A. AND MANES, E. G. 1980a. Foundations of system theory: The Hankel matrix. *Journal of Computer and System Sciences 20*, 330–378.

ARBIB, M. A. AND MANES, E. G. 1980b. Machines in a category. *Journal of Pure and Applied Algebra 19*, 9–20.

BEZHANISHVILI, N., PANANGADEN, P., AND KUPKE, C. 2012. Minimization via duality. In *Proceedings of WoLLIC' 12*, L. Ong and R. de Queiroz, Eds. Lecture Notes in Computer Science, vol. 7456. Springer, 191–205.

BIDOIT, M., HENNICKER, R., AND KURZ, A. 2001. On the duality between observability and reachability. In *FoSSaCS*, F. Honsell and M. Miculan, Eds. Lecture Notes in Computer Science, vol. 2030. Springer, 72–87.

BIRKEDAL, L., Ed. 2012. *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*. Lecture Notes in Computer Science, vol. 7213. Springer.

BONCHI, F., BONSANGUE, M. M., RUTTEN, J. J. M. M., AND SILVA, A. 2012. Brzozowski's algorithm (co)algebraically. In *Logic and Program Semantics*, R. L. Constable and A. Silva, Eds. Lecture Notes in Computer Science, vol. 7230. Springer, 12–23.

BONSANGUE, M. M., MILIUS, S., AND SILVA, A. 2012. Sound and complete axiomatizations of coalgebraic language equivalence. *ACM TOCL 13*.

BRZOZOWSKI, J. A. 1962. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical Theory of Automata*. MRI Symposia Series, vol. 12. Polytechnic Press, Polytechnic Institute of Brooklyn, 529–561.

CASTIGLIONE, G., RESTIVO, A., AND SCIORTINO, M. 2011. Nondeterministic Moore automata and Brzozowski's algorithm. In *CIAA*, B. Bouchou-Markhoff, P. Caron, J.-M. Champarnaud, and D. Maurel, Eds. Lecture Notes in Computer Science, vol. 6807. Springer, 88–99.

CHAMPARNAUD, J.-M., KHORSI, A., AND PARANTHOËN, T. 2002. Split and join for minimizing: Brzozowskis algorithm. In *PSC'02 Proceedings, Prague Stringology Conference*, M. Balík and M. Simánek, Eds. Number DC-2002-03 in Report. 96–104.

ÉZIK, Z. AND MALETTI, A. 2011. The category of simulations for weighted tree automata. *International Journal of Foundations of Computer Science (IJFCS) 22(8)*, 1845–1859.

GEHRKE, M. 2009. Stone duality and the recognisable languages over an algebra. In *CALCO*, A. Kurz, M. Lenisa, and A. Tarlecki, Eds. Lecture Notes in Computer Science, vol. 5728. Springer, 236–250.

GEHRKE, M., GRIGORIEFF, S., AND PIN, J.-E. 2008. Duality and equational theory of regular languages. In *ICALP (2)*, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds. Lecture Notes in Computer Science, vol. 5126. Springer, 246–257.

GOGUEN, J. 1972. Minimal realization of machines in closed categories. *Bulletin of the American Mathematical Society 78(5)*, 777–783.

KALMAN, R. 1959. On the general theory of control systems. *IRE Transactions on Automatic Control 4,* 3, 110–110.

KALMAN, R. E., FALB, P. L., AND ARBIB, M. A. 1969. *Topics in Mathematical Systems Theory*. McGraw Hill.

KOZEN, D. 1997. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst. 19,* 3, 427–443.

KOZEN, D. 2008. On the coalgebraic theory of Kleene algebra with tests. Tech. Rep. `http://hdl.handle.net/1813/10173`, Computing and Information Science, Cornell University. March.

RABIN, M. O. 1963. Probabilistic automata. *Information and Control 6,* 3, 230–245.

ROUMEN, F. 2011. Canonical automata via duality. Unpublished note.

RUDIN, W. 1962. *Fourier Analysis on Groups*. John Wiley and Sons.

RUTTEN, J. J. M. M. 2000. Universal coalgebra: a theory of systems. *Theor. Comput. Sci. 249,* 1, 3–80.

SAKAROVITCH, J. 2009. *Elements of Automata Theory*. Cambridge University Press.

SCHÜTZENBERGER, M. P. 1961. On the definition of a family of automata. *Information and Control 4,* 2-3, 245–270.

SILVA, A., BONCHI, F., BONSANGUE, M. M., AND RUTTEN, J. J. M. M. 2010. Generalizing the powerset construction, coalgebraically. In *FSTTCS*, K. Lodaya and M. Mahajan, Eds. LIPIcs, vol. 8. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 272–283.

TABAKOV, D. AND VARDI, M. 2005. Experimental evaluation of classical automata constructions. In *Proceedings of LPAR*, G. Sutcliffe and A. Voronkov, Eds. Lecture Notes in Artificial Intelligence, vol. 3835. Springer, 396–411.

WATSON, B. W. 2000. Directly constructing minimal DFAs: Combining two algorithms by Brzozowski. In *CIAA*, S. Yu and A. Paun, Eds. Lecture Notes in Computer Science, vol. 2088. Springer, 311–317.

WORTHINGTON, J. M. 2009. Automata, representations, and proofs. Ph.D. thesis, Cornell University, Ithaca, NY, USA. AAI3376696.