Coalgebraic Logic and Synthesis of Mealy Machines

M.M. Bonsangue[1,2], Jan Rutten[2,3], and Alexandra Silva[2]⋆

[1] LIACS - Leiden University
[2] Centrum voor Wiskunde en Informatica (CWI)
[3] Vrije Universiteit Amsterdam (VUA)

Abstract. We present a novel coalgebraic logic for deterministic Mealy machines that is sound, complete and expressive w.r.t. bisimulation. Every finite Mealy machine corresponds to a finite formula in the language. For the converse, we give a compositional synthesis algorithm which transforms every formula into a finite Mealy machine whose behaviour is exactly the set of causal functions satisfying the formula.

1 Introduction

A Mealy machine (S, f) consists of a set S of states and a transition function $f{:}S \to (B \times S)^A$ assigning to each state $s \in S$ and input symbol $a \in A$ a pair $\langle b, s' \rangle$, consisting of an output symbol $b \in B$ and a next state $s' \in S$. Typically one writes

$$f(s)(a) = \langle b, s' \rangle \iff s \xrightarrow{a|b} s'$$

One of the most important applications of Mealy machines is their use in the specification of sequential digital circuits. Taking binary inputs and outputs, there is the well-known correspondence between such binary Mealy machines, on the one hand, and sequential digital circuits built out of logical gates and some kind of memory elements, on the other. In present day text books on logic design [9] — on the construction of sequential digital circuits — Mealy machines are still the most important mathematically exact means for the specification of the intended behaviour of circuits.
There does not seem to exist, however, a generally accepted way of formally specifying Mealy machines themselves. Typically they are "defined" in a natural language such as English. This obviously leads to ambiguities, inconsistencies and plain errors [3].
In this paper, we propose a simple but adequate and expressive logical language for the specification of Mealy machines. Here adequate means that the logical equivalence corresponds to a natural behavioral equivalence on Mealy machines, whereas expressive means that every finite Mealy machine can be represented by a (finite) formula. Finally, simple means that the logic contains precisely what is needed to obtain this goal, and nothing more. The latter point is an important distinguishing factor

⋆ Partially supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BD/27482/2006.

in comparison with some already existing formalisms in the literature, discussed below.
Our approach is, in short, the coalgebraic methodology. Mealy machines are a basic and well-understood family of coalgebras, of the functor $M(S) = (B \times S)^A$. The crucial coalgebraic insight is that the properties of Mealy machines (coalgebras) are fully dictated by (the shape of) their defining functor M. This has led, for instance, to the identification of a *final* Mealy machine, in [12], as the set of all causal stream functions from A^ω to B^ω.
Following the coalgebraic methodology, we apply general insights from so-called coalgebraic modal logic (see e.g. [10, 2]) and define a logic whose basic operations derive directly from the functor M. The equivalence induced by the logic coincides with that induced by the functor M. Further, the logic comes equipped with a proof system for reasoning about universal validity that we prove sound and complete.
From the automata-theoretic point of view, all finite Mealy machines can be specified as a formula in the logic. The main technical contribution of the paper is that we construct (synthesize), conversely, for every formula in the logic a finite Mealy Machine whose behaviour is exactly characterised by the formula.

1.1 Related work

Automata synthesis is a popular and very active research area [11, 13, 6, 3, 4]. Most of the work done on synthesis has as main goal to find a proper and sufficiently expressive type of automata to encode a specific type of logic (such as LTL [13] or μ-calculus [6]).
Technically, the synthesis of a μ-calculus formula consists in translating it into an alternating automaton, reducing the latter into a non-deterministic automaton that is then checked for non-emptiness [6]. The same process has been recently generalized to F-coalgebras in [8]. In this paper, we use a different approach. We construct a deterministic Mealy machine for a formula directly, by considering the formula as a state of the automata containing enough information about its successors.
Although Mealy machines are in one-to-one correspondence with sequential digital circuits, not much work has been done for their specification and synthesis. In [13], a compositional algorithm for synthesizing Generalized Mealy machines (GMMs) from LTL formulae is presented. GMMs are a special class of non-deterministic Mealy machines that have the acceptance condition of generalized Büchi automata. In this paper, we will remain in the world of deterministic Mealy machines, the one corresponding to sequential digital circuits. Moreover, our work exploits the structure of the Mealy machine and, therefore, the resulting logic is simpler than LTL (but expressive enough for deterministic Mealy machines).
The logic most similar to ours is the one presented in [3]. There a logic for formal specification of hardware protocols is presented, and an algorithm for the synthesis of a Mealy machine is given. Their logic corresponds to the conjunctive fragment of LTL. Their synthesis process is standard: first a non-deterministic Büchi automaton is synthesized, secondly a powerset construction is used to make the automaton deterministic and, finally, the propositions on the states are used to determine the inputs and outputs

for each state of the Mealy machine. Because of our coalgebraic approach, the equivalence induced by our logic is canonical, and the logic comes with a proof system that is sound and complete. Further, our synthesis process remains within standard Mealy machines and the behavior of the synthesized automata is exactly characterized by the original formula.
Apart from [12, 4], where synthesis for a special case of 2-adic arithmetic is treated, we did not find any other work on the direct synthesis for deterministic Mealy machines. From these papers we inherit the basic coalgebraic approach, that we use here to derive our expressive logical specification language for Mealy machines.
In summary, the work presented in this paper distinguishes itself from all existing work in the sense that our specification logic is derived directly from the functor, which results in a very simple and consistent logic that has exactly the operators needed to fully specify Mealy machines. Note that being simple does not mean this logic has less expressive power than others. In the context of applications (such as circuits logic design), this logic has all the relevant operators.

Acknowledgements. We would like to thank Clemens Kupke, Helle Hvid Hansen and Yde Venema for valuable suggestions and discussions.

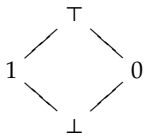
2 Mealy machines

We give the basic definitions on Mealy machines and introduce the notions of simulation and bisimulation.
First we recall the following definition. A (bounded) meet-semilattice is a set B equipped with a binary operation $\wedge_B$ and a constant $\top_B \in B$, such that $\wedge_B$ is commutative, associative and idempotent. The element $\top_B$ is neutral w.r.t. $\wedge_B$. As usual, $\wedge_B$ gives rise to a partial ordering $\leq_B$ on the elements of B:

$$b_1 \leq_B b_2 \Leftrightarrow b_1 \wedge_B b_2 = b_1$$

Every set S can be transformed into a meet-semilattice by taking the collection $\mathcal{P}S$ of all subsets of S with intersection as meet. We use semilattices to represent data structures equipped with an information order: $b_1 \leq_B b_2$ means that b_1 is more *concrete* than b_2.

Our running examples will all use the four element meet-semilattice:

$\mathcal{B} =$ 

Here, the $\top$ element is used for *abstracting* (under-specification) from any concrete data; the $\bot$ element denotes *inconsistency* (over-specification) of information; and the elements 0 and 1 are concrete output values.

Now let A be a finite set and let B be a (possibly infinite) meet-semilattice. A Mealy machine (automaton) (S,f) with inputs in A and outputs in B consists of a set of states S together with a function

$$f{:}S \to (B \times S)^A$$

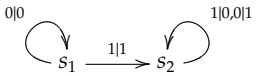
For a given state $s \in S$ and an input $a \in A$, the function f returns a pair $f(s) = \langle b, s' \rangle$, consisting of an output value $b \in B$ and a state $s' \in S$. Typically we will write

$$f(s) = \langle s[a],\, s_a \rangle$$

and call $s[a]$ the *(initial) output on input a* and s_a the *next state on input a*. We shall also use the following convention for the representation of Mealy machines:

$$f(s)(a) = \langle b, s' \rangle \iff s \xrightarrow{a|b} s'$$

Machines where A is the two element set $\{0,1\}$ and B is the meet-semilattice $\mathcal{B}$ are called *binary*, and they are *fully-specified* if only 0 or 1 are used as output elements (and never $\bot$ or $\top$).
For an example, consider the following binary Mealy machine with $S = \{s_1, s_2\}$ and the transition function defined by the following picture.



This machine computes the two's complement of a given binary number.
A Mealy machine (S,f) is a coalgebra of the functor $M{:}Set \to Set$ defined, for any set X, as $M(X) = (B \times X)^A$.
A *homomorphism* from a Mealy machine (S,f) to a Mealy machine (T,g) is a function $h{:}\, S \to T$ preserving initial outputs and next states:

$$h(s)[a] = s[a] \quad \text{and} \quad h(s_a) = h(s)_a$$

(which is equivalent to the condition that $g \circ h = M(h) \circ f$, where the functor M is defined on functions as usual). Next we define the notion of simulation, which can be used to obtain abstraction, and bisimulation, which plays an important role in the minimization of Mealy machines.

Definition 1 ((Bi)simulation for Mealy). *Let (S,f) and (T,g) be two Mealy machines. We call a relation $R \subseteq S \times T$ a* simulation *if for all $(s,t) \in S \times T$ and $a \in A$*

$$s\,R\,t \;\Rightarrow\; (\, s[a] \leq_B t[a] \;\; \textit{and} \;\; s_a\,R\,t_a \,)$$

We call R a bisimulation *relation if both R and its (relational) inverse R^{-1} are simulations.*

We write $s \lesssim t$ (resp. $s \sim t$) whenever there exists a simulation relation (bisimulation relation) containing (s,t); and we call $\lesssim$ and $\sim$ the similarity and bisimilarity relations. By definition, we have $\lesssim \cap \lesssim^{-1} = \sim$.

As an example, consider the following two binary Mealy machines:

Observe that q_3 and q_2 are bisimilar, since $R = \{(q_2, q_3), (q_3, q_3)\}$ is a bisimulation. A minimal machine is obtained by identifying all bisimilar states, yielding our two's complement machine above.

Now, note that the rightmost machine can be simulated by any fully-specified binary machine substituting either 0 or 1 as output for the abstract $\top$ value in the transition from r_1 to r_2. For example, considering the above two's complement machine, we have $s_1 \lesssim r_1$ because $S = \{(s_1, r_1), (s_2, r_2)\}$ is a simulation relation.

Next we recall the construction of a *final* Mealy machine with inputs in A and outputs in B. Finality plays an important role in minimization as well as in the proof system (in Section 3).

Let $A^\omega = \{\, \sigma \mid \sigma{:}\{0, 1, 2, \ldots\} \to A \,\}$, the set of all infinite *streams* over A. For $a \in A$ and $\sigma \in A^\omega$, we define:

$$a{:}\sigma = (a, \sigma(0), \sigma(1), \sigma(2), \ldots) \qquad \sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$$

We call a function $f{:}A^\omega \to B^\omega$ *causal* if for all $\sigma \in A^\omega$ and $n \geq 0$, the nth output value $f(\sigma)(n)$ only depends on the first n input values $(\sigma(0), \ldots, \sigma(n-1))$. Let

$$\Gamma = \{f{:}A^\omega \to B^\omega \mid f \text{ is causal } \}$$

The set Γ can be turned into a Mealy machine (Γ, γ) by defining $\gamma(f)(a) = \langle f[a], f_a \rangle$ as follows:

$$f[a] = f(a{:}\sigma)(0) \text{ (where } \sigma \text{ is arbitrary)} \qquad f_a(\sigma) = f(a{:}\sigma)'$$

(Note that by causality the value of $f(a{:}\sigma)(0)$ only depends on a.) The following theorem is a minor variation on [12, Prop.2.3 and Corr.2.3].

Theorem 2 (Finality of (Γ, γ)). *For every Mealy machine (S, f) there exists a unique homomorphism $h{:}S \to \Gamma$. It satisfies, for all $s, s' \in S$:*

$$s \lesssim s' \iff h(s) \lesssim h(s')$$

where on Γ, similarity coincides with the elementwise ordering induced by B:

$$f \lesssim g \iff \forall \sigma \in A^\omega \, \forall n \geq 0 \,.\, f(\sigma)(n) \leq_B g(\sigma)(n)$$

Since the identity function is always a homomorphism, bisimilarity is equality on Γ. As a consequence, the image $h(S)$ of a Mealy machine S is in fact its minimisation with respect to bisimilarity.

3 Mealy logic

We present a logic for Mealy machines and define its semantics and a satisfaction relation.

Definition 3 (Mealy formulae). *Let A be a set of input actions and let B be a meet-semilattice of output actions. Furthermore, let X be a set of (recursion or) fixed point variables. The set L of* Mealy formulae *is given by the following BNF syntax. For $a \in A$, $b \in B$, and $x \in X$:*

$$\phi ::= tt \mid x \mid a(\phi) \mid a{\downarrow}b \mid \phi \wedge \phi \mid \nu x.\psi$$

where $\psi \in L_g$, the set of guarded *formulae, which is given by:*

$$\psi ::= tt \mid a(\phi) \mid a{\downarrow}b \mid \psi \wedge \psi \mid \nu x.\psi$$

We call $a(\phi)$ a *transition formula* and $a{\downarrow}b$ an *output formula*. Note that our language does not include disjunction or negation. As we will discuss in 3.2, this is a natural restriction and does not decrease the expressiveness of our logic. Moreover, in the same section we will also point out the reasons for only having one type of fixed point operator. Also note that for every unguarded Mealy formula there exists an equivalent guarded formula, as a consequence of [7, Theorem 2.1].
The modal fragment of our logic (*i.e*, the set of closed formulae without the ν operator) is a special case of the coalgebraic logic obtained by a Stone-type duality [1, 2].
In what follows, we shall concentrate on the set L_g^c of formulae that are both guarded and *closed*, that is, without free occurrences of fixed point variables x. We turn the set L_g^c into a Mealy machine (coalgebra)

$$\lambda : L_g^c \to (B \times L_g^c)^A$$

by defining λ as follows. For $a \in A$ and $\phi \in L_g^c$, we write $\lambda(\phi) = \langle \phi[a], \phi_a \rangle$ and we define $\phi[a]$ and ϕ_a by

$$\begin{array}{lll}
tt[a] & = \top_B & tt_a = tt \\
a(\phi)[a'] & = \top_B \ \text{(for any } a' \in A) & (a(\phi))_{a'} = \begin{cases} \phi & \text{if } a = a' \\ tt & \text{otherwise} \end{cases} \\
(a{\downarrow}b)[a'] & = \begin{cases} b & \text{if } a = a' \\ \top_B & \text{otherwise} \end{cases} & (a{\downarrow}b)_{a'} = tt \ \text{(for any } a' \in A) \\
(\phi_1 \wedge \phi_2)[a] & = \phi_1[a] \wedge_B \phi_2[a] & (\phi_1 \wedge \phi_2)_a = (\phi_1)_a \wedge (\phi_2)_a \\
(\nu x.\psi)[a] & = (\psi[\nu x.\psi / x])[a] & (\nu x.\psi)_a = (\psi[\nu x.\psi / x])_a
\end{array}$$

Here, $\psi[\nu x.\psi / x]$ denotes syntactic substitution, replacing in ψ every free occurrence of x by $\nu x.\psi$.
The above definition uses induction on the following complexity measure, which is based on the number of nested unguarded occurrences of ν-formulae:

$$\begin{array}{ll}
N(tt) & = N(a{\downarrow}b) = N(a(\phi)) = 0 \\
N(\phi_1 \wedge \phi_2) & = max\{N(\phi_1), N(\phi_2)\} + 1 \\
N(\nu x.\psi) & = 1 + N(\psi)
\end{array}$$

In order to see that the definition of $\phi[a]$ and ϕ_a is well-formed, note that in the case of $\nu x.\psi$, we have:

$$N(\psi) = N(\psi[\nu x.\psi/x])$$

This can easily be proved by (standard) induction on the syntactic structure of ψ, since ψ is guarded (in x).
Note that the (sub)machine generated by a formula $\phi \in L_g^c$ by repeatedly applying λ will in general be infinite. In Section 4, an algorithm to produce a finite Mealy machine from a formula $\phi \in L_g^c$ will be presented.
Having a Mealy coalgebra structure on L_g^c has two advantages. First, it provides us, by finality of Γ, directly with a natural semantics because of the existence of a (unique) homomorphism:

$$\begin{array}{ccc} L_g^c & \xrightarrow{[\![\cdot]\!]} & \Gamma \\ {\scriptstyle\lambda}\downarrow & & \downarrow{\scriptstyle\gamma} \\ (B \times L_g^c)^A & \xrightarrow[(id\times[\![\cdot]\!])^A]{} & (B \times \Gamma)^A \end{array} \qquad [\![\phi]\!][a] = \phi[a] \quad \text{and} \quad [\![\phi]\!]_a = [\![\phi_a]\!]$$

It assigns to every formula ϕ a causal stream function $[\![\phi]\!]: A^\omega \to B^\omega$.
The second advantage of the Mealy coalgebra structure on L_g^c is that it lets us use the notion of Mealy simulation to define when a Mealy machine (S, f) satisfies a formula $\phi \in L_g^c$, by defining:

$$s \models \phi \quad \Leftrightarrow \quad s \lesssim \phi$$

Proving satisfaction then amounts to the construction of a simulation relation $R \subseteq S \times L_g^c$ between (S, f) and (L, λ) such that $sR\phi$.
The above definition is equivalent to the following, more classical definition of satisfaction. For every valuation $\eta: Var \to \mathcal{P}(S)$, we define a satisfaction relation $\models_\eta$, by induction, as follows:

$$\begin{array}{lll} s \models_\eta tt & & \text{for all } s \\ s \models_\eta a(\phi) & \textit{iff} & s_a \models_\eta \phi \\ s \models_\eta a{\downarrow}b & \textit{iff} & s[a] \leq_B b \\ s \models_\eta \phi_1 \wedge \phi_2 & \textit{iff} & s \models_\eta \phi_1 \textit{ and } s \models_\eta \phi_2 \\ s \models_\eta x & \textit{iff} & s \in \eta(x) \\ s \models_\eta \nu v.\psi & \textit{iff} & \exists T \subseteq S. s \in T \textit{ and } \forall t \in T. t \models_{\eta[T/v]} \psi \end{array}$$

Note that in this definition single occurrences of $x \in X$ are allowed. It can be shown, by a fairly straightforward and not very instructive proof, that the two definitions of satisfaction are equivalent. More precisely, if $\emptyset$ denotes the everywhere empty valuation, we have:

$$s \lesssim \phi \quad \Leftrightarrow \quad s \models_\emptyset \phi$$

for every $\phi \in L_g^c$. We omit the proof and will work in what follows with the definition of satisfaction as simulation.
The following theorem shows that our logic is sufficiently expressive to characterise bisimilarity.

Theorem 4.
(1) For all states s, s' of a Mealy machine (S, f),

$$s \sim s' \quad \textit{iff} \quad \forall \phi \in L^c_g \,.\, s \models \phi \Leftrightarrow s' \models \phi$$

(2) If S is finite *then there exists for any $s \in S$ a formula $\phi_s \in L^c_g$ such that*

$$\forall s' \in S \,.\, s \sim s' \quad \textit{iff} \quad s' \models \phi_s$$

Proof. (1) Because $s \sim s'$ implies $s \lesssim s'$ and $s' \lesssim s$ we have, for any $\phi \in L^c_g$,

$$s \models \phi \iff s \lesssim \phi \iff s' \lesssim \phi \iff s' \models \phi$$

For the converse, note, for any $s \in S$, $a \in A$, and $\phi \in L^c_g$, that $s \models a{\downarrow}s[a]$ and

$$s_a \models \phi \iff s_a \lesssim \phi \iff s \lesssim a(\phi) \iff s \models a(\phi)$$

As a consequence, the following relation

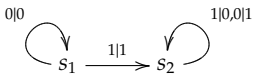
$$R = \left\{ \langle s, s' \rangle \in S \times S \mid \forall \phi \in L^c_g \,.\, s \models \phi \Leftrightarrow s' \models \phi \right\}$$

and its inverse R^{-1} are simulation relations on S. Thus R is a bisimulation.
(2) It is sufficient to construct for a given $s \in S$ a formula ϕ_s with $s \sim \phi_s$. To this end, we associate with every state $s \in S$ a variable $x_s \in X$ and a formula $\phi_s = \nu x_s . \psi_s$ defined by

$$\psi_s = \bigwedge_{a \in A} a(x_{s_a}) \wedge a{\downarrow}s[a]$$

Syntactically replacing free occurrences of $x_{s'}$ by $\phi_{s'}$ will ensure that all ϕ_s will be in L^c_g. By construction, $s \sim \phi_s$. □

Let us illustrate the last construction above. Recall the two's complement Mealy machine presented before:



We define $\phi_1 = \nu x_1 . \psi_1$ and $\phi_2 = \nu x_2 . \psi_2$ by

$$\psi_1 = 0(x_1) \wedge 0{\downarrow}0 \wedge 1(x_2) \wedge 1{\downarrow}1 \qquad \psi_2 = 0(x_2) \wedge 0{\downarrow}1 \wedge 1(x_2) \wedge 1{\downarrow}0$$

Substituting ϕ_2 for x_2 in ψ_1 then yields

$$\phi_1 = \nu x_1 . 0(x_1) \wedge 0{\downarrow}0 \wedge 1(\phi_2) \wedge 1{\downarrow}1 \qquad \phi_2 = \nu x_2 . 0(x_2) \wedge 0{\downarrow}1 \wedge 1(x_2) \wedge 1{\downarrow}0$$

By construction we have $s_1 \sim \phi_1$ and $s_2 \sim \phi_2$.

3.1 Proof system

We now introduce a proof system for assertions of the form $\phi_1 \leq \phi_2$, where $\leq$ is the relation of logical entailment between the closed formulae ϕ_1 and ϕ_2.

$$(\mathit{refl}) \quad \phi \leq \phi \qquad (\mathit{top}) \quad \phi \leq tt$$

$$(\wedge - e1) \quad \phi_1 \wedge \phi_2 \leq \phi_1 \qquad (\wedge - e2) \quad \phi_1 \wedge \phi_2 \leq \phi_2$$

$$(\mathit{trans}) \quad \frac{\phi_1 \leq \phi_2 \quad \phi_2 \leq \phi_3}{\phi_1 \leq \phi_3} \qquad (\wedge - i) \quad \frac{\phi \leq \phi_1 \quad \phi \leq \phi_2}{\phi \leq \phi_1 \wedge \phi_2}$$

$$(a{\downarrow} - \top) \quad tt \leq a{\downarrow}\top_B \qquad (a() - \top) \quad tt \leq a(tt)$$

$$(a{\downarrow} - \wedge) \quad a{\downarrow}b_1 \wedge a{\downarrow}b_2 \leq a{\downarrow}(b_1 \wedge_B b_2) \qquad (a() - \wedge) \quad a(\phi_1) \wedge a(\phi_2) \leq a(\phi_1 \wedge \phi_2)$$

$$(a{\downarrow} - \leq) \quad \frac{b_1 \leq_B b_2}{a{\downarrow}b_1 \leq a{\downarrow}b_2} \qquad (a() - \leq) \quad \frac{\phi_1 \leq \phi_2}{a(\phi_1) \leq a(\phi_2)}$$

$$(\nu - i) \quad \frac{\phi \leq \psi[\phi/x]}{\phi \leq \nu x.\psi} \qquad (\nu - e) \quad \frac{\psi[\nu x.\psi/x] \leq \phi}{\nu x.\psi \leq \phi}$$

The first group of axioms and rules gives to the set of formulae the structure of a meet-semilattice. Further, there are axioms and rules for the two modal operators, showing the interactions between the transition and output formulae with the meet-semilattice structure. Finally, the last two rules $(\nu - i)$ and $(\nu - e)$ can be explained as stating that the term $\nu x.\psi$ is the greatest postfixed point, when viewing the formula ψ as a (monotone) map on formulae.

We write $\vdash \phi_1 \leq \phi_2$ to indicate that the assertion $\phi_1 \leq \phi_2$ is derivable from the above axioms and rules. Note that the converse of $(a{\downarrow} - \wedge)$ is derivable from $(a{\downarrow} - \leq)$ and $(\wedge - i)$. Similarly, also the converses of $(a{\downarrow} - \top)$, $(a() - \top)$ and $(a() - \wedge)$ are derivable.

Theorem 5 (Soundness). *The above proof system is sound, that is, for closed formulae ϕ_1 and ϕ_2, $\vdash \phi_1 \leq \phi_2$ implies that for all Mealy machine (S, f) if $s \models \phi_1$ then $s \models \phi_2$.*

Proof. By induction on the length of proofs. □

Next we turn to the completeness for the *modal* fragment L_m of our Mealy logic L, where a modal formula is a formula with neither fixed point operators nor variables. Note that the (Lindenbaum algebra of) L_m is a meet-semilattice.

Let Θ be the set of all filters of (the Lindenbaum algebra of) L_m, where a *filter* of a meet-semilattice is a non-empty upper closed subset $\mathcal{F}$ such that if $a, b \in \mathcal{F}$ then also $a \wedge b \in \mathcal{F}$. The set Θ can be turned into a Mealy machine (Θ, θ) by defining, for $F \in \Theta$ and $a \in A$, $\theta(F)(a) = \langle F[a], F_a \rangle$, where

$$F[a] = \bigwedge \{b | a{\downarrow}b \in F\} \qquad F_a = \{\phi | a(\phi) \in F\}\,.$$

Note that in order $F[a]$ to be well defined we assume B to be a *finite* meet-semilattice. In case B is infinite, we would need B to be a complete meet-semilattice.

Theorem 6. *For every Mealy machine (S,f) there exists a unique homomorphism $k_S{:}S \to \Theta$. In particular, the homomorphism $k_\Gamma{:}\Gamma \to \Theta$ is an isomorphism.*

As a consequence of Theorem 4, the isomorphism $k_\Gamma{:}\Gamma \to \Theta$ is also an order isomorphism, where the order on Θ is subset inclusion. The logical significance of the above result is that a finitary logic with only finite conjunctions suffices to completely describe all Mealy machines up to bisimilarity. In fact the modal fragment of our logic is a special case of coalgebraic logic obtained by a Stone-type duality [1, 2].
Theorem 6 together with the next lemma gives a logical interpretation of the final coalgebra: its elements correspond to canonical models (in the logical sense) of the Mealy logic.

Lemma 7. *For every modal formula ϕ and filter $F \in \Theta$, $F \models \phi$ if and only if $\phi \in F$.*

Proof. By induction on the structure of ϕ, using the fact that F is a filter and the above definition of $\theta{:}\Theta \to (B \times \Theta)^A$. □

We can finally prove the completeness of the modal fragment of our Mealy logic.

Theorem 8 (Completeness). *For modal formulae ϕ_1 and ϕ_2, if $s \models \phi_1$ implies $s \models \phi_2$ for all Mealy machines (S,f) and $s \in S$, then $\vdash \phi_1 \leq \phi_2$.*

Proof. Assume $\nvdash \phi_1 \leq \phi_2$. It is enough to find a state s in a Mealy machine (S,f) such that $s \models \phi_1$ but $s \not\models \phi_2$. Define $F_{\phi_1} = \{\psi \mid \phi_1 \leq \psi\}$. It is not very difficult to verify that F_{ϕ_1} is a filter, hence it is an element of Θ. Clearly, $\phi_1 \in F_{\phi_1}$ but, by our assumption $\phi_2 \notin F_{\phi_1}$. We can now conclude by applying Lemma 7. □

3.2 Adding negation

The logic we have considered so far contains no negation. Extending the logic with negated formulae is not problematic as long as we consider Mealy machines with outputs in a Boolean algebra B (like the two element set). In this case, we can still turn the set of (possibly negated) formulae into a Mealy coalgebra by extending our definition of λ at the beginning of section 3 with

$$(\neg\phi)[a] = \neg_B(\phi[a]) \qquad (\neg\phi)_a = \neg(\phi)_a\,.$$

It is easy to see that according to this definition negation distributes up to bisimulation over conjunction (de Morgan law), and over the modal operators (a sign that the machine is indeed deterministic). Further, negation is classical, meaning that $\neg(\neg\phi) \sim \phi$. More interestingly, we have that $\neg\nu x.\psi \sim \nu x.\neg\psi$ for all guarded formulae ψ, not necessarily positive.

Clearly, disjunctions and μ-recursive formulae can be defined as derived operators.
From the logical point of view, this means that the Lindenbaum algebra of the resulting logic with negation is the free Boolean algebra over the meet-semilattice of the Mealy logic we considered here. In this case one can apply the isomorphism $\mathcal{UF}ilt(B(L)) \cong Filt(L)$ to obtain analogous soundness and completeness results as above, where L is a meet-semilattice, $B(L)$ is the free Boolean algebra over L and $\mathcal{UF}ilt(B(L))$ is the set of ultrafilters of $B(L)$.

4 Synthesis

We will now describe the synthesis process that will allow to produce a Mealy machine from an arbitrary (closed and guarded) Mealy formula [4]. Each state of the resulting Mealy machine will be a formula constructed in such a way that if s is the state corresponding to a formula ϕ, then $s \sim \phi$. This implies that the semantics of s is exactly the set of causal functions satisfying ϕ.

4.1 Formulae normalization

We have seen that the first group of six axioms and rules of our proof system gives to the set of formulae the structure of a meet-semilattice. In order to guarantee the termination of the synthesis process we will need to identify formulae that are provably equivalent using only these axioms and rules. For instance, the formulae

$$a(tt) \wedge a{\downarrow}b \wedge tt \wedge a{\downarrow}b \text{ and } a(tt) \wedge a{\downarrow}b$$

are equivalent.
To *normalize* a formula ϕ, we need eliminate any redundancy present in the formula: in a conjunction, tt can be eliminated and, by idempotency, the conjunction of two syntactically equivalent formulae can be simplified.
The function *norm* encodes this procedure. We define it by induction on the formula structure as follows:

$$\begin{aligned}
norm(tt) &= tt \\
norm(a(\phi)) &= a(norm(\phi)) \\
norm(a{\downarrow}b) &= a{\downarrow}b \\
norm(\phi_1 \wedge \phi_2) &= conj(rem(flatten(norm(\phi_1) \wedge norm(\phi_2)))) \\
norm(\nu x.\phi) &= \nu x.(norm(\phi))\,.
\end{aligned}$$

Here, *conj* takes a list of formulae $[\phi_1, \ldots, \phi_n]$ and returns the formula $\phi_1 \wedge \ldots \wedge \phi_n$ (*conj* applied to the empty list yields tt), *rem* removes duplicates in a list and *flatten* takes a formula ϕ and produces a list of formulae by omitting brackets and replacing $\wedge$-symbols by commas:

$$\begin{aligned}
flatten(\phi_1 \wedge \phi_2) &= flatten(\phi_1) \cdot flatten(\phi_2) \\
flatten(tt) &= [] \\
flatten(\phi) &= [\phi],\ \phi \in \{a{\downarrow}b, a(\phi_1), \nu x.\phi_1\}
\end{aligned}$$

[4] The source code in Haskell can be downloaded from `www.cwi.nl/~ams/mealy`.

In this definition, $\cdot$ denotes list concatenation and $[\phi]$ the singleton list containing ϕ. Note that an occurrence of tt in a conjunction is eliminated because $\mathit{flatten}(tt) = []$.
For example, the normalization of the two formulae above will result in the same formula – $a(tt) \wedge a{\downarrow}b$.
Note that *norm* still distinguishes the formulae $\phi_1 \wedge \phi_2$ and $\phi_2 \wedge \phi_1$. For simplifying the presentation of the normalization algorithm, we decided not to identify these formulae, since this will not influence termination. However, in the implementation, in order to reduce the number of states, those formulae are identified. In the examples below this situation will never occur.
The *norm* function satisfies the equalities $\mathit{norm}(\mathit{norm}(\phi)) = \mathit{norm}(\phi)$ and $\mathit{norm}(\mathit{norm}(\phi) \wedge \mathit{norm}(\psi)) = \mathit{norm}(\phi \wedge \psi)$, which we will use later for proving termination.

4.2 Synthesis

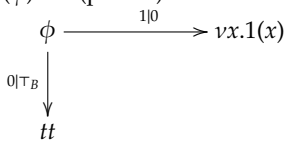
We first describe what happens in a single step of the synthesis process.
The function δ, which does *one-step synthesis* for a single formula, takes a formula $\phi \in L_g^c$ and produces a partial Mealy machine. Below, δ will be used in the function Δ, which synthesises the total Mealy machine.
The function δ is defined, by induction on the complexity measure N defined in Section 3, as follows:

$$
\begin{array}{lll}
\delta(tt)(a) & = \langle \top_B, tt \rangle & \\
\delta(a'(\phi))(a) & = \begin{cases} \langle \top_B, \mathit{norm}(\phi) \rangle & a = a' \\ \langle \top_B, tt \rangle & \mathit{otherwise} \end{cases} & \\
\delta(a'{\downarrow}b)(a) & = \begin{cases} \langle b, tt \rangle & a = a' \\ \langle \top_B, tt \rangle & \mathit{otherwise} \end{cases} & \\
\delta(\phi_1 \wedge \phi_2)(a) & = \delta(\phi_1)(a) \sqcap \delta(\phi_2)(a) & \\
\delta(\nu x.\phi)(a) & = \langle b, \mathit{norm}(\phi') \rangle \ \textbf{where} \ \langle b, \phi' \rangle = \delta(\phi[\nu x.\phi / x])(a) &
\end{array}
$$

where $\sqcap$ is defined as: $\langle b_1, \phi_1 \rangle \sqcap \langle b_2, \phi_2 \rangle = \langle b_1 \wedge_B b_2, \mathit{norm}(\phi_1 \wedge \phi_2) \rangle$.
Note that this function is very similar to the function λ presented in Section 3. In fact, the difference is the normalization that is now being applied to the formulae so that a finite machine will be produced.
As an example, consider the formula $\phi = 1{\downarrow}0 \wedge (\nu x.1(x))$, specifying a binary Mealy machine. We can easily compute that $\delta(\phi)(0) = \langle \top_B, tt \rangle$ and

$$
\begin{array}{lll}
\delta(\phi)(1) & = \delta(1{\downarrow}0)(1) \sqcap \delta(\nu x.1(x))(1) & \\
& = \langle 0, tt \rangle \sqcap \langle \top_B, \nu x.1(x) \rangle & \\
& = \langle 0, \nu x.1(x) \rangle &
\end{array}
$$

So, $\delta(\phi)$ is a (partial) finite function represented by the following diagram.



To compute the entire Mealy machine that satisfies ϕ, we need to apply δ to the new states generated at each step repeatedly until all states in the automata have their transitions/outputs fully defined.

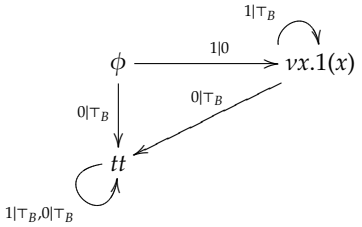
We implement this procedure with the auxiliary function D. The arguments of this function are two sets of states: $sts \subseteq L_g^c$, the states that still need to be processed and $vis \subseteq L_g^c$, the states that already have been visited (synthesized). For each $\phi \in sts$, D computes $\delta(\phi)$ and produces an intermediate transition function (possibly partial) by taking the union of all those $\delta(\phi)$. Then, it collects all new states appearing in this step and recursively computes the transition function for those.

$$D(sts, vis) = \begin{cases} \emptyset & sts = \emptyset \\ trans \cup D(newsts, vis') & otherwise \end{cases}$$
$$\textbf{where } trans = \{\langle \phi, \delta(\phi)\rangle \mid \phi \in sts\}$$
$$sts' = collectStates(trans)$$
$$vis' = sts \cup vis$$
$$newsts = sts' \setminus vis'$$

The function Δ takes a Mealy formula $\phi \in L_g^c$ and returns a Mealy machine that satisfies ϕ:

$$\Delta(\phi) = (dom(f), f) \quad \textbf{where } f = D(\{norm(\phi)\}, \emptyset)$$

The function *dom* returns the domain of a finite function. The finiteness and termination of the synthesis algorithm will be proved in appendix B. Let us look at an example. For the formula ϕ presented above $\Delta(\phi) = (S, f)$, where $S = \{tt, \phi, \nu x.1(x)\}$ and f is represented by the following diagram.



Note that the Mealy machine produced by Δ is not minimal. In this example, the states tt and $\nu x.1(x)$ are bisimilar and could be identified.

The (special) output value $\top_B$ allows us to define *underspecified* machines: if a given formula does not contain information about the output value for a given input a, then we do not return as output a concrete value but instead $\top_B$. If $\top_B$ is replaced by any other element $b \in B$ the resulting machine will still satisfy ϕ.

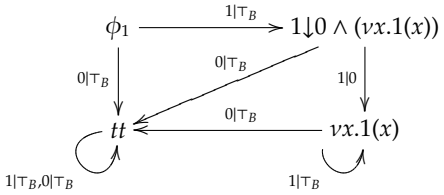
Let us see a few other examples of the synthesis process. To simplify the presentation, we consider again binary machines and, moreover, the formulae presented below will only have information for the input 1. Therefore, for the 0 input δ will always return $\langle \top_B, tt\rangle$.

Let us start with $\phi_1 = 1(1{\downarrow}0) \wedge (\nu x.1(x))$. We have:

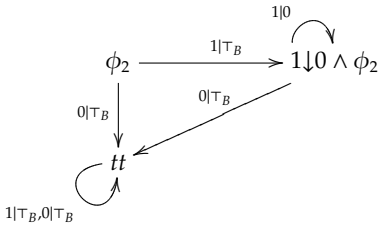
$$\begin{aligned} \delta(\phi_1)(1) &= \delta(1(1{\downarrow}0))(1) \sqcap \delta(\nu x.1(x))(1) \\ &= \langle \top_B, 1{\downarrow}0\rangle \sqcap \langle \top_B, \nu x.1(x)\rangle \\ &= \langle \top_B, 1{\downarrow}0 \wedge (\nu x.1(x))\rangle \end{aligned}$$

We now repeat the process for $1{\downarrow}0 \wedge (\nu x.1(x))$, which will yield $\delta(1{\downarrow}0 \wedge (\nu x.1(x)))(1) = \langle 0, \nu x.1(x)\rangle$. Finally, we calculate $\delta(\nu x.1(x))(1) = \langle \top_B, \nu x.1(x)\rangle$.

The complete Mealy machine is represented in the following diagram:



Now, take $\phi_2 = \nu x.1(1\downarrow 0) \wedge 1(x)$. Because $1(1\downarrow 0)$ has no x's one could be tempted to assume that the automaton for ϕ_2 would be the same as the one for ϕ_1. However, that is not the case. The synthesis algorithm will produce the following automaton for ϕ_2.



As a last example, let $\phi_3 = \nu x.1(x \wedge (\nu y.1(y) \wedge 1\downarrow 0))$. We have:

$$\begin{aligned}\delta(\phi_3)(1) &= \delta(1(\phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0)))(1)\\ &= \langle \top_B, \phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0)\rangle\end{aligned}$$

and

$$\begin{aligned}&\delta(\phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0))(1)\\ &= \delta(\phi_3)(1) \sqcap \delta(\nu y.1(y) \wedge 1\downarrow 0)(1)\\ &= \langle \top_B, \phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0)\rangle \sqcap \langle 0, \nu y.1(y) \wedge 1\downarrow 0\rangle\\ &= \langle 0, \mathit{norm}(\phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0) \wedge (\nu y.1(y) \wedge 1\downarrow 0))\rangle\\ &= \langle 0, \phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0)\rangle\end{aligned}$$

Note that if *norm* would not have been applied, the resulting state $\phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0) \wedge (\nu y.1(y) \wedge 1\downarrow 0)$ would be regarded as a new state, even though it is equivalent to $\phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0)$. Moreover, applying δ to this state (for input 1) would yield again an equivalent but (syntactically) different state, namely $\phi_3 \wedge (\nu y.1(y) \wedge 1\downarrow 0) \wedge (\nu y.1(y) \wedge 1\downarrow 0) \wedge (\nu y.1(y) \wedge 1\downarrow 0)$. This illustrates that the function λ, defined in Section 3, generally produces an infinite machine. However, the identifications made by *norm* ensure the termination of the synthesis process, which we will formally prove in appendix B.

5 Conclusions and future work

We have given a coalgebraic account of Mealy machines, focussing on a logical specification language for them. Despite its simplicity, the logic is expressive in the sense that all Mealy machines can be characterized by finite formulae, but also in the sense that logical equivalence corresponds to bisimulation. Further, the logic is sound and the modal fragment complete for all Mealy machines.

The specification language is finitary and includes a fixed point operator. Other temporal operators can be defined as derived operators. Interestingly, the language is already expressive enough to characterize all Mealy machines even without negation and disjunction. Even stronger, for binary Mealy machines the addition of negation does not increase the expressive power of the logic. This situation is typical also of deterministic finite automata: the addition of negation in regular expressions does not increase the class of languages that they characterize, even though regular languages are closed under complement.

Our main result is an algorithm for the synthesis of a Mealy machine from a formula. Our synthesis algorithm is compositional, in the sense that the semantics of the Mealy machine synthesized from a formula can be obtained by suitably composing the semantics of the Mealy machines synthesized from sub-formulae.

In this paper we have explored the synthesis of one particular type of automata, the Mealy machines. With a small variation of the logic one can easily obtain a similar result for Moore automata. More generally, different type of automata can be obtained by varying the functor under consideration on the category of sets. It would be interesting to generalize the present result in order to synthesize coalgebras for different functors.

References

1. M. M. Bonsangue and A. Kurz. Duality for logics of transition systems. In Sassone, ed., *FoSSaCS*, vol. 3441 of *LNCS*, p. 455–469. Springer, 2005.
2. M. M. Bonsangue and A. Kurz. Presenting functors by operations and equations. In Aceto and Ingólfsdóttir, eds., *FoSSaCS*, vol. 3921 of *LNCS*, p. 172–186. Springer, 2006.
3. E. M. Clarke, S. M. German, Y. Lu, H. Veith, and D. Wang. Executable protocol specification in esl. In W. A. H. Jr. and S. D. Johnson, eds., *FMCAD*, vol. 1954 of *LNCS*, p. 197–216. Springer, 2000.
4. H. H. Hansen, D. Costa, and J. J. M. M. Rutten. Synthesis of mealy machines using derivatives. *ENTCS*, 164(1):27–45, 2006.
5. D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.
6. O. Kupferman and M. Vardi. μ-calculus synthesis. In M. Nielsen and B. Rovan, eds., *MFCS'00*, vol. 1893 of *LNCS*, p. 497–507. Springer, 2000.
7. O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.
8. C. Kupke and Y. Venema. Coalgebraic automata theory: basic results. Technical Report SEN-E0701, CWI, The Netherlands, 2007.
9. A. B. Marcovitz. *Introduction to Logic Design*. McGraw-Hill, 2005.
10. L. Moss. Coalgebraic logic. *Annals of Pure and Applied Logic*, 96, 1999.
11. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL'89*, p. 179–190, 1989.
12. J. J. M. M. Rutten. Algebraic specification and coalgebraic synthesis of mealy automata. *ENTCS*, 160:305–319, 2006.
13. S. Tini and A. Maggiolo-Schettini. Compositional synthesis of generalized mealy machines. *Fundam. Inform.*, 60(1-4):367–382, 2004.

A Proof of Theorem 6

We will present the proof of Theorem 6, presented in section 3.1.

Proof (Theorem 6). For a Mealy machine (S, f), we define the function $k_S{:}S \to \Theta$ by $k(s) = \{\phi \mid s \models \phi\}$, for $s \in S$. It is easy to verify that $k(s)$ is a filter. Next we show that k is an homomorphism. In fact, for each $a \in A$, we have

$$\begin{array}{lll} k(s)[a] = \bigwedge\{b | a{\downarrow}b \in k(s)\} & & \text{definition of } (\Theta, \theta) \\ \quad = \bigwedge\{b | s \models a{\downarrow}b\} & & \text{definition of } k(s) \\ \quad = \bigwedge\{b | s[a] \leq b\} & & \\ \quad = s[a] & & \end{array}$$

and also

$$\begin{array}{lll} k(s)_a = \{\phi | a(\phi) \in k(s)\} & & \text{definition of } (\Theta, \theta) \\ \quad = \{\phi | s \models a(\phi)\} & & \text{definition of } k(s) \\ \quad = \{\phi | s_a \models \phi\} & & \\ \quad = k(s_a) & & \end{array}$$

It remains to prove uniqueness. Let $g{:}S \to \Theta$ be another homomorphism, and let $s \in S$. We prove that $\phi \in g(s)$ if and only if $\phi \in k(s)$ by induction on the structure of ϕ. We consider only two cases, as the others follow easily because both $g(s)$ and $k(s)$ are filters. We start with the case $\phi \equiv a{\downarrow}b$.

$$\begin{array}{lll} a{\downarrow}b \in g(s) \Leftrightarrow g(s)[a] \leq b & & \text{def. of } (\Theta, \theta) \\ \quad \Leftrightarrow s[a] \leq b & & g \text{ is an homomorphism} \\ \quad \Leftrightarrow s \models a{\downarrow}b & & \\ \quad \Leftrightarrow a{\downarrow}b \in k(s) & & \end{array}$$

In case $\phi \equiv a(\psi)$ we have:

$$\begin{array}{lll} a(\psi) \in g(s) \Leftrightarrow \psi \in g(s)_a & & \text{def. of } (\Theta, \theta) \\ \quad \Leftrightarrow \psi \in g(s_a) & & g \text{ is an homomorphism} \\ \quad \Leftrightarrow \psi \in k(s_a) & & \text{induction hypothesis} \\ \quad \Leftrightarrow a(\psi) \in k(s) & & \end{array}$$

The fact that $k_\Gamma{:}\Gamma \to \Theta$ is an isomorphism follows from the uniqueness (up to isomorphism) of final coalgebras. □

B Proof of termination

We prove that our synthesis algorithm will always deliver a finite automaton for a given closed and guarded Mealy formula.

Definition 9. *For any $\phi \in L^c_g$ we define the closure of ϕ as the smallest set $cl(\phi)$ satisfying the following conditions:*

$$\begin{array}{l} \phi \in cl(\phi) \\ \phi_1 \wedge \phi_2 \in cl(\phi) \Rightarrow \phi_1, \phi_2 \in cl(\phi) \\ a(\phi_1) \in cl(\phi) \Rightarrow \phi_1 \in cl(\phi) \\ \nu x.\phi_1 \in cl(\phi) \Rightarrow \phi_1[\nu x.\phi_1/x] \in cl(\phi) \end{array}$$

One can easily prove that the set $cl(\phi)$ is finite for any formula ϕ (as is well-known from [5]). We can also prove that $\phi' \in cl(\phi) \Rightarrow cl(\phi') \subseteq cl(\phi)$.

Definition 10. *For any $\phi \in L_g^c$ we define the super-closure of ϕ as*

$$supercl(\phi) = \left\{ norm(\bigwedge_{\psi \in S} \psi) \mid S \in \mathcal{P}(cl(\phi)) \right\}$$

where $\bigwedge\{\psi_1, \ldots, \psi_n\} = \psi_1 \wedge \ldots \wedge \psi_n$. Because $cl(\phi)$ is finite, then $supercl(\phi)$ is also finite.
Note that if $\phi_1, \phi_2 \in supercl(\phi)$, then $norm(\phi_1 \wedge \phi_2) \in supercl(\phi)$. This follows directly from the fact that if S_1 and S_2 are subsets of $cl(\phi)$ then $S_1 \cup S_2$ also is and that $norm(norm(\phi_1) \wedge norm(\phi_2)) = norm(\phi_1 \wedge \phi_2)$.
Also note that the formula tt will always be an element of the super-closure, corresponding to the empty conjunction.

Theorem 11. *Let $\phi \in L_g^c$, $a \in A$ and $\delta(\phi)(a) = \langle b, \psi \rangle$. Then $\psi \in supercl(\phi)$.*

Proof. We will use induction on the complexity measure N (defined in Section 3).
If $N(\phi) = 0$, then $\phi \in \{tt, a{\downarrow}b, a(\phi')\}$. The result follows directly from the definition of δ and $supercl(\phi)$.
Now, suppose that $N(\phi) = k + 1$. Then $\phi \in \{\phi_1 \wedge \phi_2, \nu x.\phi'\}$.
For $\phi = \phi_1 \wedge \phi_2$, let $\delta(\phi)(a) = \langle b, \psi \rangle$ and $\delta(\phi_i)(a) = \langle b_i, \psi_i \rangle$, $i = 1, 2$. Then:

$$\psi = norm(\psi_1 \wedge \psi_2) \qquad \text{definition of } \delta$$

By the induction hypothesis, $\psi_i \in supercl(\phi_i) \subseteq supercl(\phi)$. This implies $\psi_1 \wedge \psi_2 \in supercl(\phi)$ and thus $norm(\psi_1 \wedge \psi_2) \in supercl(\phi)$.
For $\phi = \nu x.\phi'$, let $\delta(\phi)(a) = \langle b, \psi \rangle$ and $\delta(\phi'[\nu x.\phi'/x])(a) = \langle b', \psi' \rangle$. Then:

$$\psi = norm(\psi') \qquad \text{definition of } \delta$$

By the induction hypothesis, $\psi' \in supercl(\phi'[\nu x.\phi'/x]) \subseteq supercl(\phi)$. This implies that $\psi' \in supercl(\phi)$ and thus $norm(\psi') \in supercl(\phi)$.

□

Theorem 12. *For a given formula $\phi \in L_g^c$, $D(\{\phi\}, \emptyset)$ terminates.*

Proof. We know that $\delta(\phi)(a) \in supercl(\phi)$ and for all $\psi \in supercl(\phi) \Rightarrow supercl(\psi) \subseteq supercl(\phi)$. Therefore, we have an upper bound for the number of states that will need to be processed. Since D guarantees that no state is processed twice, the set *newsts* will eventually be empty and, therefore, $D(\{\phi\}, \emptyset)$ terminates. □

This theorem (together with the fact that the function *dom* terminates) concludes the proof that Δ terminates and that our synthesis algorithm will always return a finite machine for a given closed and guarded formula.