Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Coalgebra, concurrency, and control

J.J.M.M. Rutten

Software Engineering (SEN)

SEN-R9921 November 30, 1999

Report SEN-R9921
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Coalgebra, Concurrency, and Control

J.J.M.M. Rutten
Email: janr@cwi.nl
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract
Coalgebra is used to generalize notions and techniques from concurrency theory, in order to apply them to problems concerning the supervisory control of discrete event systems. The main ingredients of this approach are the characterization of controllability in terms of (a variant of) the notion of bisimulation, and the observation that the family of (partial) languages carries a final coalgebra structure. This allows for a pervasive use of coinductive definition and proof principles, leading to a conceptual unification and simplification and, in a number of cases, to more general and more efficient algorithms.

1991 Mathematics Subject Classification: 68Q10, 68Q55
1991 ACM Computing Classification System: D.3, F.1, F.3
Keywords & Phrases: Coalgebra, concurrency, control theory, Moore automaton, bisimulation, coinduction, discrete event system, controllability

Contents

1 Introduction **3**

2 A few mathematical preliminaries **4**

3 Moore automata **4**

4 Coinduction and finality **6**

5 The automaton of partial languages **7**

6 Operators on partial languages **9**

7 Coinduction and regularity **11**

8 Controllability **15**

9 Supremal controllable sublanguages **18**

10 Non-blocking languages **22**

11 Output-control consistency **24**

12 Notes and discussion **26**

13 Appendix: coinductive definitions **27**

14 Appendix: Nerode equivalence **28**

15 Appendix: cat and mouse **29**

1 Introduction

Coalgebra can be understood as a theory that deals with behavioural aspects of dynamic systems in a rather wide sense. Behaviour is often appropriately viewed as consisting of both dynamics and observations, which have to do with change of states and partial access to states, respectively. Therefore many types of automata are typical examples of coalgebras: their dynamics is given by state transitions, and what can be observed of a given state is, for instance, whether it is accepting or not. But many other types of examples exist as well, such as the data type of streams (infinite sequences), with the convention that of a stream, only the first element is actually observable, and where change of state consists of removing the first element (thus creating the possibility of observing the second element of the stream).

Precisely as the notion of congruence is central to the theory of (universal) algebra, much of the theory of (universal) coalgebra [Rut96, JR97] is centered around the notion of *bisimulation*. This notion was originally invented by Park [Par81] and Milner [Mil80], in order to formalize the behavioural equivalence of concurrent processes. Bisimulation was later introduced into the world of coalgebra by Aczel and Mendler [AM89], who gave a categorical definition of bisimulation that applies to arbitrary coalgebras. The scope of this new definition of bisimulation is far more general than the original one, and applies to many types of systems, including not only Park and Milner's concurrent processes, but also various kinds of data types and automata. Using the (generalized) notion of bisimulation, Aczel [Acz88] formulated a principle of *coinduction*, in very much the same way as Milner had introduced his 'bisimulation proof method': In order to prove that two processes are behaviourally equivalent (bisimilar), it is sufficient to establish the existence of a bisimulation relation between them. This principle of coinduction is particularly useful for coalgebras that are *final*, a notion dual to that of initiality in algebra. On a final coalgebra, bisimilarity coincides with equality, whereby proving equality amounts to constructing bisimulation relations.

The aim of the present paper is to apply coalgebraic reasoning to some problems regarding the control of dynamical systems. More specifically, we shall be dealing with the supervisory control of discrete event systems, introduced by Ramadge and Wonham [RW89, WR87]. In their formulation, the problem of controllability amounts to restraining the behaviour of a system by forbidding (but only when necessary) certain events (inputs) at certain moments, in such a way that the resulting behaviour satisfies a given specification, and moreover such that a fixed family of so-called uncontrollable events are never forbidden.

Discrete event systems are just deterministic (Moore) automata with partial transition functions. They will be defined here as coalgebras, along with the notions of homomorphism and bisimulation. The main relevance of these notions is their role in the characterization of the automaton $\mathcal{L}$ of *partial languages*. These partial languages are actually pairs of languages (subsets of words), and represent what in control theory are called the marked behaviour and the closed behaviour of automata. Using the notion of *input derivative* [Brz64, Con71], the set $\mathcal{L}$ can be supplied with an automaton structure, which is *final* among all automata. The finality of $\mathcal{L}$ gives rise to both definitions and proofs by coinduction. Since this will be our main instrument for tackling the problems of controllability, relatively much time is spent on the introduction and illustration of both definitions and proofs by coinduction. Again using the notion of derivative, coinductive definitions of operators on languages take the shape of what could be called *behavioural differential* equations (in the same style as Conway's *differential calculus of events* [Con71]). In Section 6, coinductive definitions of a number of operators on languages are giving, including a new operator called supervised product. Section 7 contains a rather long list of examples of proofs by coinduction, which we hope will allow the reader to become familiar with this somewhat unusual way of reasoning.

It is not until Section 8, then, that we actually turn to the problem of controllability. Varying the notion of bisimulation, we introduce *control relations* (and *partial bisimulations*), which generalize Ramadge and Wonham's notion of controllable sublanguage, and which are used in a new coinduction principle for proving controllability: In order to prove that one language is controllable with respect to another, it is sufficient to establish the existence of a control (or a partial bisimulation) relation between them, just as showing the equality of languages by coinduction amounts

to finding (ordinary) bisimulation relations. Here we can profit from the coinductive definitions of the operators on partial languages, since they are formulated in terms of derivatives, which is exactly what is needed for the construction of control and partial bisimulation relations.

Sections 9 and 10 deal with the problem of constructing supremal controllable sublanguages, which was originally solved by Ramadge and Wonham. Here a solution is presented which is based on the finality of the automaton of partial languages, and which uses the coinductive characterizations of controllability mentioned above. Showing the existence of a supremal controllable sublanguage (of a given language and satisfying a certain specification) then amounts to the observation that control and partial bisimulation relations are closed under arbitrary unions, as is the case for ordinary bisimulations. Varying the well-known characterization of bisimilarity as the greatest fixed-point of a monotone operator on a set of relations [Mil89], repeated here for partial Moore automata in Section 3, next leads to algorithms for computing automaton representations for such supremal languages. These algorithms are mildly more general than the ones in [WR87], and are easily shown correct by coinduction. In Section 11, yet another illustration of the coalgebraic approach is presented, by giving a coinductive definition of the transformation of a (Moore) automaton into one that is *output-control-consistent* [Won99]. The procedural reading of this coinductive definition gives rise to a new and rather efficient algorithm. Section 12 makes some comparisons with existing work, including the one paper on the relation between bisimulation and supervisory control we are aware of [BL98], mentions some further references to the literature, and discusses some topics for future research. Finally, some proofs and examples have been included as appendices, together with a brief note on the relation between the final automaton $\mathcal{L}$ and the algebraic notion of Nerode equivalence.

2 A few mathematical preliminaries

Partial functions: Partial functions between sets X and Y will be modelled as ordinary (total) functions $\phi : X \to (1 + Y)$. Here 1 is any one-element set, suggestively written as $1 = \{\Uparrow\}$, and $+$ is disjoint union. For x in X, $\phi(x) = \Uparrow$ means that $\phi(x)$ is undefined, also simply denoted by $\phi(x)\Uparrow$. Dually, $\phi(x)\Downarrow$ denotes that $\phi(x)$ *is* defined, that is, $\phi(x) \in Y$. The domain of ϕ is defined as the set $dom(\phi) = \{x \in X \mid \phi(x)\Downarrow\}$.
Words: Let $A^* = \{\langle a_1, \cdots, a_n\rangle \mid n \geq 0 \text{ and } a_i \in A\}$ be the set of all words over A, including the empty word ($n = 0$), denoted by ε. For a in A, a^* will be a shorthand for $\{a\}^*$. Let vw denote the concatenation of two words v and w in A^*; $v \leq w$ denotes that v is a prefix of w. The prefix-closure of a set of words $V \subseteq A^*$ is the set $\overline{V} = \{w \in A^* \mid \exists\, v \in V :\ w \leq v\}$. The set V is prefix-closed if $V = \overline{V}$. Furthermore, we define the a-derivative of V by $V_a = \{w \in A^* \mid aw \in V\}$. More generally, for a word w in A^*, $V_w = \{v \in A^* \mid wv \in V\}$.
Power set: The power set of a set X is the collection of all its subsets: $\mathcal{P}(X) = \{V \mid V \subseteq X\}$.

3 Moore automata

Let A and B be two arbitrary sets. A *partial Moore automaton* (or *Moore machine*) with inputs in A and outputs in B is a pair $S = (S, \langle o, t\rangle)$ consisting of a set S of *states*, and a pair of functions

$$\langle o, t\rangle : S \to B \times (1 + S)^A,$$

consisting of an output function $o : S \to B$ and a transition function $t : S \to (1 + S)^A$. The function t assigns to each state s in S a function $t(s) : A \to (1 + S)$, which for any input symbol (also called *event*) a in A is either undefined: $t(s)(a)\Uparrow$, meaning that from s no a-transition can take place; or specifies the state that is reached after the input symbol a has been consumed: $t(s)(a) \in S$. We shall usually say Moore automaton, omitting the word partial.

The following notation will be helpful. We write $s \xrightarrow{a}$ iff $t(s)(a)\Downarrow$, in which case we define $s_a = t(s)(a)$ and write $s \xrightarrow{a} s_a$. More generally, define $s \xrightarrow{\varepsilon}$ to hold always, put $s_\varepsilon = s$ and write $s \xrightarrow{\varepsilon} s$. Assuming, by induction, that $s \xrightarrow{w} s_w$ has been defined, define $s \xrightarrow{wa}$ iff $t(s_w)(a)\Downarrow$, put

$s_{wa} = t(s_w)(a)$, and write $s \xrightarrow{wa} s_{wa}$. Some further notation: $s\longrightarrow$ iff there exists a in A with $s \xrightarrow{a}$; and $s \xrightarrow{a}\!\!\!\!\!/\;$ iff $t(s)(a)\Uparrow$.

A *homomorphism* between Moore automata $S = (S, \langle o, t\rangle)$ and $S' = (S', \langle o', t'\rangle)$ is a function $f : S \to S'$ with, for all $s \in S$ and $a \in A$:

$$o'(f(s)) = o(s) \text{ and: } s \xrightarrow{a} \text{ iff } f(s) \xrightarrow{a}, \text{ in which case: } f(s)_a = f(s_a).$$

It is straightforward to see that the family of all Moore automata and homomorphisms between them, is a category: identity mappings are homomorphisms, and the functional composition of two homomorphisms is again a homomorphism. In fact, a Moore automaton is a *coalgebra* of the functor $F = B \times (1 + -)^A : Set \to Set$, which are pairs (S, α) consisting of a set S and a function $\alpha : S \to F(S)$ (denoted above by $\langle o, t\rangle$). Correspondingly, homomorphisms of Moore automata are precisely F-coalgebra homomorphisms. Although much of what follows is based on general notions and insights from the abstract theory of coalgebra—such as the notion of bisimulation to be introduced below—we have tried to keep the paper self-contained and do not assume any prior knowledge on coalgebra or categories. The interested reader is referred to [Rut96] and [JR97].

A Moore automaton $S' = (S', \langle o', t'\rangle)$ is a *subautomaton* of $S = (S, \langle o, t\rangle)$ if $S' \subseteq S$ and the inclusion function $i : S' \to S$ is a homomorphism. Given $S = (S, \langle o, t\rangle)$ and S', the functions o' and t' in that case are uniquely determined. For a state s in S, $\langle s\rangle$ denotes the subautomaton *generated* by s: it is the smallest subautomaton of S containing s, and can be obtained by including all states from S that are reachable via a finite number of transitions from s. Homomorphisms map subautomata to subautomata: for a homomorphism $f : S \to T$ and subautomaton $S' \subseteq S$, $f(S')$ is a subautomaton of T. For s in S, moreover, $f(\langle s\rangle) = \langle f(s)\rangle$.

A *bisimulation* between two Moore automata $S = (S, \langle o, t\rangle)$ and $S' = (S', \langle o', t'\rangle)$ is a relation $R \subseteq S \times S'$ with, for all s in S, s' in S':

$$\text{if } s\,R\,s' \text{ then } \begin{cases} (i) & o(s) = o'(s'), \\ (ii) & \forall\, a \in A :\ s \xrightarrow{a} \Rightarrow\ (s' \xrightarrow{a} \text{ and } s_a\ R\ s'_a\,), \text{ and} \\ (iii) & \forall\, a \in A :\ s' \xrightarrow{a} \Rightarrow\ (s \xrightarrow{a} \text{ and } s_a\ R\ s'_a\,) \end{cases}$$

(Sometimes we shall write $\langle s, s'\rangle \in R$ rather than $s\,R\,s'$.) A bisimulation between S and itself is called a bisimulation *on* S. Unions and (relational) compositions of bisimulations are bisimulations again. We write $s \sim s'$ whenever there exists a bisimulation R with $s\,R\,s'$. This relation $\sim$, called the *bisimilarity* relation, is the union of all bisimulations and, therewith, the greatest bisimulation. The greatest bisimulation on one and the same Moore automaton, again denoted by $\sim$, is an equivalence relation. The notions of homomorphism and bisimulation are closely related: a function $f : S \to S'$ is a homomorphism if and only if its graph relation $\{\langle s, f(s)\rangle \mid s \in S\}$ is a bisimulation.

Two states are bisimilar iff they give rise to the same outputs, now and in any possible future:

Proposition 3.1 For states s and s' in Moore automata S and S',

$$s \sim s' \text{ iff } \forall\, w \in A^* :\ s \xrightarrow{w} \Leftrightarrow s' \xrightarrow{w}, \text{ in which case } o(s_w) = o'(s'_w).$$

Bisimilarity is thus characterized as observational equivalence (with respect to the observation of outputs). The implication from left to right follows by induction on the length of words, using the fact that $\sim$ itself is a bisimulation relation. The reverse implication follows from the observation that the set of all pairs $\langle s, s'\rangle$ in $S \times S'$ satisfying the formula on the right of the 'iff' above, is a bisimulation relation.

Bisimulation relations and bisimilarity can be characterized as (post-)fixed points of a monotone operator $\Phi : \mathcal{P}(S \times S') \to \mathcal{P}(S \times S')$, which is defined, for a relation $R \subseteq S \times S'$ between Moore automata S and S', by

$$\Phi(R) = \{\langle s, s'\rangle \in S \times S' \mid s \text{ and } s' \text{ satisfy conditions (i), (ii), and (iii) above }\}$$

Proposition 3.2 Let S and S' be two Moore automata.

1. A relation $R \subseteq S \times S'$ is a bisimulation iff R is a post-fixed point of Φ: that is, $R \subseteq \Phi(R)$.
2. Consequently, bisimilarity is the greatest fixed point of Φ: $\sim = \mathrm{gfp}\,(\Phi)$.
3. It can be obtained as a countable intersection of finite iterations of Φ: $\sim = \bigcap_{n\geq 0} \Phi^n(S \times S')$, where $\Phi^0 = id$ and $\Phi^{n+1} = \Phi \circ \Phi^n$.

Proof: The first statement is by the definition of bisimulation relation. Since the greatest fixed point of a monotone operator on a complete lattice equals the union of all its post-fixed points, and bisimilarity is defined as the union of all bisimulation relations, (2) follows. The inclusion from left to right, in (3), follows by an easy induction from $\sim \subseteq S \times S'$, $\sim \subseteq \Phi(\sim)$, and the monotonicity of Φ. For the inclusion from right to left, it is sufficient to prove that the infinite intersection is a post-fixed point of Φ, since $\sim$ is the union of all post-fixed points of Φ: Putting $I = \bigcap_{n\geq 0} \Phi^n(S \times S')$, we have to show that $I \subseteq \Phi(I)$. Consider $\langle s, s' \rangle \in I$. Because $I \subseteq \Phi(S \times S')$, it follows that $o(s) = o(s')$. Supposing $s \xrightarrow{a}$, it follows from $I \subseteq \Phi^{n+1}(S \times S')$ that $s' \xrightarrow{a}$ and $\langle s_a, s'_a \rangle \in \Phi^n(S \times S')$, for all $n \geq 0$, which implies $\langle s_a, s'_a \rangle \in I$. The other case, starting with the assumption that $s' \xrightarrow{a}$, is proved similarly. Thus $\langle s, s' \rangle \in \Phi(I)$. □

The operator Φ can be used to compute for a given relation $R \subseteq S \times S'$, the greatest bisimulation $\tilde{R}$ contained in R: by a minor variation on Proposition 3.2, we have that $\tilde{R} = \bigcap_{n\geq 0} \Phi^n(R)$.

Note that though (1) and (2) in Proposition 3.2 hold by general considerations for many other types of coalgebras as well, (3) depends on the fact that our Moore automata are *deterministic*: for a state s and input symbol a, there is at most one successor state s_a. Generally, (3) does not hold for non-deterministic systems (for which there is not a unique successor s_a for a given state s and input symbol a).

4 Coinduction and finality

Using the notion of *input derivative*, a Moore automaton $\mathcal{L}_B$ (over A and B) is defined that satisfies a proof principle called coinduction and that is final among all Moore automata. It is universal in the sense that its states represent minimal realisations of all possible behaviours of all possible Moore automata.

Consider the following set of partial functions from A^* to B:

$$\mathcal{L}_B = \{\phi : A^* \to (1 + B) \mid dom(\phi) \neq \emptyset \text{ and } dom(\phi) \text{ is prefix-closed } \}$$

A reasonable name for the an element ϕ of this set $\mathcal{L}_B$ might be *Moore language*, notably in view of Section 5, where the special case of $B = 2$ will be treated, for which such elements will represent (partial) languages indeed.

The set $\mathcal{L}_B$ can be turned into a Moore automaton using the following notion. Consider a Moore language ϕ in $\mathcal{L}_B$ and an input symbol a in A. If $a \in dom(\phi)$ then we define the *a-derivative* or input derivative ϕ_a of ϕ as the partial function

$$\phi_a : A^* \to (1 + B), \quad \phi_a(w) = \begin{cases} \phi(aw) & \text{if } aw \in dom(\phi) \\ \Uparrow & \text{otherwise} \end{cases}$$

If $a \notin dom(\phi)$ then the function ϕ_a itself is undefined. Note that if ϕ_a is defined then $dom(\phi_a)$ is prefix-closed because $dom(\phi)$ is; moreover, it is non-empty since $a \in dom(\phi)$ implies $\varepsilon \in dom(\phi_a)$. Thus ϕ_a is an element of $\mathcal{L}_B$. Next an output function $o_B : \mathcal{L}_B \to B$ and a transition function $t_B : \mathcal{L}_B \to (1 + \mathcal{L}_B)^A$ are defined, for $\phi \in \mathcal{L}_B$, by

$$o_B(\phi) = \phi(\varepsilon), \quad t_B(\phi)(a) = \begin{cases} \phi_a & \text{if } \phi_a \text{ is defined} \\ \Uparrow & \text{otherwise} \end{cases}$$

(Note that $\phi(\varepsilon) \in B$, since the fact that $dom(\phi)$ is non-empty and prefix-closed implies $\varepsilon \in dom(\phi)$.) Thus we have obtained a Moore automaton $(\mathcal{L}_B, \langle o_B, t_B \rangle)$. Note that our notation

of $\phi_a = t_B(\phi)(a)$, whenever ϕ_a is defined, is consistent with the notational convention from the beginning of Section 3, of writing $s_a = t(s)(a)$ (whenever this is defined), for a state s in a Moore automaton $(S, \langle o, t\rangle)$.

Theorem 4.1 The Moore automaton $(\mathcal{L}_B, \langle o_B, t_B\rangle)$ satisfies the principle of *coinduction*: for all ϕ and ψ in $\mathcal{L}_B$, if $\phi \sim \psi$ then $\phi = \psi$.

The theorem can be used as a proof principle: in order to prove $\phi = \psi$, it is sufficient to establish the existence of a bisimulation relation on $\mathcal{L}_B$ containing the pair $\langle \phi, \psi\rangle$.
Proof: One can easily prove by induction on the length of words w in A^* that $\phi \xrightarrow{w}$ (in the automaton $\mathcal{L}_B$) iff $w \in dom(\phi)$, in which case $o_B(\phi_w) = \phi(w)$. If $\phi \sim \psi$, for Moore languages ϕ and ψ then, by Proposition 3.1, $\phi \xrightarrow{w}$ iff $\psi \xrightarrow{w}$, in which case $o_B(\phi_w) = o_B(\psi_w)$. It follows that $dom(\phi) = dom(\psi)$ and for w in $dom(\psi)$, $\phi(w) = \psi(w)$, that is, $\phi = \psi$. □

The above coinduction principle, and a number of variations still to follow, will be used time and again in the remainder of the paper. A first example follows right away.

Theorem 4.2 The Moore automaton $(\mathcal{L}_B, \langle o_B, t_B\rangle)$ is *final*: For any Moore automaton $(S, \langle o, t\rangle)$ there exists a unique homomorphism $l : S \to \mathcal{L}_B$. This homomorphism is characterized by the following equivalence: for $s, s' \in S$, $s \sim s'$ iff $l(s) = l(s')$.

Proof: For the existence part of the theorem, define $l : S \to \mathcal{L}_B$, for s in S, by $w \in dom(l(s))$ iff $s \xrightarrow{w}$, in which case $l(s)(w) = o(s_w)$. Uniqueness of l follows from the observation that for any two homomorphisms $f, g : S \to \mathcal{L}_B$, the set $\{\langle f(s), g(s)\rangle \in \mathcal{L}_B \times \mathcal{L}_B \mid s \in S\}$ is a bisimulation relation on $\mathcal{L}_B$. Then $f = g$ follows by coinduction (Theorem 4.1). The latter part of the theorem is immediate from the definition of l and Proposition 3.1. □

The Moore language $l(s)$ is called the *behaviour* of the state s of the automaton S. We also say that s *represents* the language $l(s)$, and that $l(s)$ is *accepted* by the state s. Yet another formulation is to say that the language $l(s)$ is *realized* by the state s (in the Moore automaton S).

The uniqueness part of Theorem 4.2 can in fact be easily seen to be equivalent to the coinduction *proof* principle of Theorem 4.1. As we shall see in Section 6, the existence part gives rise to coinductive *definitions*.

An immediate consequence of Theorem 4.2 is the fact that the subautomaton $\langle \phi\rangle \subseteq \mathcal{L}_B$ generated by ϕ, which is given by $\langle \phi\rangle = \{\phi_w \mid w \in dom(\phi)\}$ is a minimal Moore automaton accepting ϕ. In other words, $\langle \phi\rangle$ is a minimal realisation of ϕ. (See also Section 14, where this subautomaton, for the special case of $B = 2$, is characterized as a quotient of A^* with respect to Nerode equivalence.)

Taking $S = \mathcal{L}_B$ in Theorem 4.2, one obtains the corollary that the behaviour $l(\phi)$ of a Moore language ϕ, viewed as a state of the Moore automaton $\mathcal{L}_B$, is equal to the language ϕ itself (since the identity mapping is a homomorphism and hence equal to l). Moore languages could therefore be said to "do (behave) as they are", a slogan which applies more generally to any final coalgebra.

5 The automaton of partial languages

Of special interest are Moore automata for which (the set of inputs A is arbitrary and) the set of outputs is $B = 2 = \{0, 1\}$, the set of Booleans. We shall refer to them as *partial automata* or simply *automata*, and reserve the term *Moore* automaton for those situations where B is arbitrary. In this section, the structure of the final automaton $\mathcal{L}_2$ is analyzed in some detail, which consists of so-called partial languages. Yet another coinduction principle is introduced, formulated in terms of simulation relations.

The output function of an automaton $(S, \langle o, t\rangle)$ is now a function $o : S \to 2$, indicating whether a state s in S is *accepting*: $o(s) = 1$, denoted by $s{\downarrow}$, or not: $o(s) = 0$, denoted by $s{\uparrow}$.

Moore languages, the elements of the final Moore automaton $\mathcal{L}_2$, have now a particularly simple description:

$$\mathcal{L}_2 \cong \mathcal{L} = \{(V, W) \mid V \subseteq W \subseteq A^*\,,\ W \neq \emptyset\,,\ \text{and } W \text{ is prefix-closed } \}$$

To a Moore language $\phi : A^* \to 1 + 2$ in $\mathcal{L}_2$, a pair $\langle V, W\rangle \in \mathcal{L}$ is assigned, defined by $W = dom(\phi)$ and $V = \{w \in W \mid \phi(w) = 1(\in 2)\}$. Conversely, to a pair $\langle V, W\rangle \in \mathcal{L}$ a function $\phi \in \mathcal{L}_2$ is assigned, given for $w \in A^*$ by $\phi(w) = 1$ if $w \in V$, $= 0$ if $w \notin V$ and $w \in W$, and is undefined if $w \notin W$. This defines a bijection between $\mathcal{L}_2$ and $\mathcal{L}$.

The elements of $\mathcal{L}$ are called *partial languages* or simply *languages*, and are denoted by $L = (L^1, L^2)$. Classically, a language V is defined as a set of words $V \subseteq A^*$ or, equivalently, a total function $V : A^* \to 2$. Such languages can be embedded into $\mathcal{L}$ by mapping $V : A^* \to 2$ to the pair (V, A^*). The image of this embedding clearly consists of all languages $L = (L^1, L^2)$ in $\mathcal{L}$ with $L^2 = A^*$. Such languages will be called *classical* (another name could be *total*).

Furthermore the following constants are introduced, one for each of the elements in 2 and one for each input symbol a in A:

$$0 = (\emptyset, \{\varepsilon\}),\ \ 1 = (\{\varepsilon\}, \{\varepsilon\}),\ \ a = (\{a\}, \{\varepsilon, a\})$$

The automaton structure on $\mathcal{L}$ is obtained using the definition of $\mathcal{L}_2$ (Section 4), in combination with the isomorphism $\mathcal{L} \cong \mathcal{L}_2$ above. An explicit description is as follows. Let a in A be an input symbol. The *a-derivative* or *input derivative* L_a of a language $L = (L^1, L^2)$ in $\mathcal{L}$ is defined whenever $a \in L^2$, and is given by

$$L_a = (L^1_a, L^2_a) \quad \text{iff } a \in L^2$$

(recalling that for a set $V \subseteq A^*$, $V_a = \{w \in A^* \mid aw \in V\}$). If $a \notin L^2$ then L_a is undefined. More generally, we define

$$L_w = (L^1_w, L^2_w) \quad \text{iff } w \in L^2$$

As before, a-derivatives are the basis for an automaton structure $(\mathcal{L}, \langle o_{\mathcal{L}}, t_{\mathcal{L}}\rangle)$ on $\mathcal{L}$, which is now given, for $L \in \mathcal{L}$, by

$$o_{\mathcal{L}}(L) = \begin{cases} 1 & \text{if } \varepsilon \in L^1 \\ 0 & \text{if } \varepsilon \notin L^1 \end{cases} \quad , \quad t_{\mathcal{L}}(L)(a) = \begin{cases} L_a & \text{if } L_a \text{ is defined} \\ \Uparrow & \text{otherwise} \end{cases}$$

Note that if L_a is defined then $L^1_a \subseteq L^2_a$, $L^2_a \neq \emptyset$, and L^2_a is prefix-closed. Thus L_a is an element of $\mathcal{L}$, indeed. Using the notational conventions introduced before, one has

$$L\!\downarrow \text{ iff } \varepsilon \in L^1\,, \quad L \xrightarrow{w} L_w \text{ iff } L_w \text{ is defined } \text{ iff } w \in L^2$$

For the constant languages 0, 1 and a defined above, there is thus the following behaviour:

$$0\!\uparrow,\quad 1\!\downarrow,\quad a\!\uparrow,\quad 0 \not\to,\quad 1 \not\to,\quad a \xrightarrow{a} 1,\quad a \overset{b}{\not\to} \text{ for } b \neq a$$

The following theorem repeats, for future reference, Theorems 4.1 and 4.2 for the present case.

Theorem 5.1 (1) The automaton $(\mathcal{L}, \langle o_{\mathcal{L}}, t_{\mathcal{L}}\rangle)$ satisfies the principle of coinduction: for all K and L in $\mathcal{L}$, if $K \sim L$ then $K = L$. (2) The automaton $(\mathcal{L}, \langle o_{\mathcal{L}}, t_{\mathcal{L}}\rangle)$ is *final*: For any automaton $(S, \langle o, t\rangle)$ there exists a unique homomorphism $l : S \to \mathcal{L}$. This homomorphism is characterized by the following equivalence: for $s, s' \in S$, $s \sim s'$ iff $l(s) = l(s')$. □

We shall see many examples of definitions by coinduction, based on part (2) of the theorem, in Section 6, and of proofs by coinduction, based on part (1), in Section 7. For an automaton S, the unique homomorphism $l : S \to \mathcal{L}$ now maps a state s in S to

$$l(s) = (L^1_s, L^2_s) = (\{w \mid s \xrightarrow{w} \text{ and } s_w\!\downarrow\},\ \{w \mid s \xrightarrow{w}\})$$

In control theory, these sets are sometimes called the marked and the closed behaviour of s, a terminology which will not be used here.

The presence of an ordering on the set 2 of observations: $0 \leq 1$, allows for the following generalisation of the notion of bisimulation, together with a corresponding coinduction proof principle. A *simulation* between two automata $S = (S, \langle o, t\rangle)$ and $S' = (S', \langle o', t'\rangle)$ is a relation $R \subseteq S \times S'$ with, for all s in S, s' in S':

$$\text{if } s\,R\,s' \text{ then } \begin{cases} (i) & o(s) \leq o'(s'), \text{ and} \\ (ii) & \forall\, a \in A:\ s \xrightarrow{a} \Rightarrow\ (s' \xrightarrow{a} \text{ and } s_a\ R\ s'_a\,) \end{cases}$$

Note that (i) is equivalent to: if $s\!\downarrow$ then $s'\!\downarrow$. Unions and (relational) compositions of simulations are simulations again. We write $s \leq s'$ whenever there exists a simulation R with $s\,R\,s'$. This relation $\leq$, called the *similarity* relation, is the union of all simulations and, therewith, the greatest simulation. Writing $K \subseteq L$ iff $K^1 \subseteq L^1$ and $K^2 \subseteq L^2$, for languages $K = (K^1, K^2)$ and $L = (L^1, L^2)$ in $\mathcal{L}$, there is the following theorem.

Theorem 5.2 For all K and L in $\mathcal{L}$, if $K \leq L$ then $K \subseteq L$. □

(The converse implication also holds.) The proof of this theorem is an easy variation on the proof of Theorem 5.1(1). Note that for automata, a relation R is a bisimulation iff both R and R^{-1}, the inverse of R, are simulations. Therefore, Theorem 5.2 implies Theorem 5.1(1). (This relies on the fact that our automata are deterministic. Generally, this does not hold for deterministic automata.)

There is yet another characterization of the final automaton $\mathcal{L}$, stemming from [Rut98] (varying on a result from [Bre98]), which has featured in certain metric models of concurrent programming languages (cf. [dB91] and [BV96]). It will play no role in the rest of the paper. Let $A^\infty_\delta = A^* \cup A^\omega \cup A^* \cdot \delta$, where A^* is as before, A^ω is the set of all infinite words over A, and $A^* \cdot \delta = \{w \cdot \delta \mid w \in A^*\}$, (assuming that $\delta \notin A$). For an infinite word $w = a_1 a_2 a_3 \cdots$ in A^ω and a natural number $n \geq 1$, the n-th truncation of w is given by $w[n] = a_1 \cdots a_n$. Let for a word w in A^* and a subset $V \subseteq A^\infty_\delta$, the w-derivative of V be defined by $V_w = \{v \in A^\infty_\delta \mid wv \in V\}$, where concatenation of words is extended to A^∞_δ in the obvious way. Call V (metrically) *closed*[1] if, for an infinite word w in A^ω, $V_{w[n]} \neq \emptyset$ for all $n \geq 1$ implies $w \in V$. Typically, a^∞ is closed, whereas a^* is not. The set V is *consistent* if for all words w in A^*, $\delta \in V_w$ iff $V_w = \{\delta\}$. For instance, $\{ab, ac, b\delta\}$ is consistent whereas $\{ab, a\delta\}$ is not.

Theorem 5.3 $\qquad \mathcal{L}(\cong\ \mathcal{L}_2\,) \cong \{V \subseteq A^\infty_\delta \mid V \text{ is non-empty, closed, and consistent}\}$

Proof: The following two mappings constitute a bijection. From left to right, a partial language $L = (L^1, L^2)$ is sent to

$$L^1 \cup \{w \cdot \delta \mid w \notin V,\ w \in W,\ \forall\, a \in A:\ w \cdot a \notin W\,\} \cup \{w \in A^\omega \mid \forall\, n \geq 0:\ w[n] \in W\}$$

Conversely, a non-empty, closed, and consistent set $V \subseteq A^\infty_\delta$ is mapped to the partial language $(V \cap A^*, \overline{V})$, where $\overline{V}$ is the set of all prefixes of words in V (with the convention that $v \leq w \cdot \delta$ iff $v \leq w$). □

6 Operators on partial languages

This section introduces a number of operators on partial languages, including the synchronized (shuffle) product and an operation called supervised product. These definitions will be coinductive and are given by equations that resemble (partial) differential equations from classical analysis, since they are formulated in terms of a-derivatives. In [Rut99], a unified treatment of both operators on automata and classical differential equations is given. Although the present section contains some new operators, all results are variations on similar constructions in [Rut98] and [Rut99], a reason why proofs are given in one of the appendices (Section 13).

For languages K and L in $\mathcal{L}$, the following operators will be introduced:

[1] The terminology is explained by the fact that this definition is equivalent to being closed with respect to the metric topology on A^∞_δ induced by the Baire metric.

$K + L$: sum

KL: concatenation

K^*: Kleene star (or repetition)

$K \parallel_I L$: synchronous product (for $I \subseteq A$)

$K /_I L$: supervised product (for $I \subseteq A$)

These operators are defined for languages $K = (K^1, K^2)$ and $L = (L^1, L^2)$ in $\mathcal{L}$, and thus generalize the corresponding definitions for classical languages (included in $\mathcal{L}$ as $L = (L, A^*)$). The definitions take the shape of specifications that are *behavioural* in the following sense. Recall that the behaviour of a language L, viewed as a state of the automaton $\mathcal{L}$, is determined by the values of $o_{\mathcal{L}}(L)$ and $t_{\mathcal{L}}(L)$, which tell us whether L is accepting or not (recall that $L\!\downarrow$ iff $\varepsilon \in L$), and what the steps are that L can take. The latter are defined, for any input symbol a in A, in terms of a-derivatives: $L \xrightarrow{a} L_a$ iff $a \in L^2$. The definitions below, now, specify the behaviour of (the result of applying) the operators, by explicitly describing their a-derivatives and observations. In Section 13, it is proved that this type of definition is justified, that is, that there exist unique operators satisfying the clauses below. Since the key ingredient in the proof is the fact that $\mathcal{L}$ is a final automaton, we say that these operators are *defined by coinduction*:

$$(K + L)_a = K_a + L_a \qquad (K + L)\!\downarrow \text{ iff } K\!\downarrow \text{ or } L\!\downarrow$$

$$(KL)_a = \begin{cases} K_a\, L & \text{if } K\!\uparrow \\ K_a\, L + L_a & \text{if } K\!\downarrow \end{cases} \qquad (KL)\!\downarrow \text{ iff } K\!\downarrow \text{ and } L\!\downarrow$$

$$(K^*)_a = K_a\, K^* \qquad (K^*)\!\downarrow$$

$$(K \parallel_I L)_a = \begin{cases} (K_a \parallel_I L) + (K \parallel_I L_a) & \text{if } a \notin I \\ K_a \parallel_I L_a & \text{if } a \in I \end{cases} \qquad (K \parallel_I L)\!\downarrow \text{ iff } K\!\downarrow \text{ and } L\!\downarrow$$

$$(K /_I L)_a = \begin{cases} K_a /_I L_a & \text{if } K \xrightarrow{a} \text{ and } L \xrightarrow{a} \\ 0 /_I L_a & \text{if } K \not\xrightarrow{a} \text{ and } L \xrightarrow{a} \text{ and } a \in I \\ \Uparrow & \text{otherwise} \end{cases} \qquad (K /_I L)\!\downarrow \text{ iff } L\!\downarrow$$

We adhere (here and in the remainder of the paper) to the following convention regarding the (un)definedness of plus, concatenation, and the synchronous product:

$$\Uparrow + X = X + \Uparrow = X, \quad \Uparrow X = \Uparrow$$

$$\Uparrow \parallel_I X = X \parallel_I \Uparrow = \Uparrow$$

For instance, if K_a is undefined and $X = L$ then $K_a + L$ should be read as L whereas both $K_a L$ and $K_a \parallel_I L$ are undefined. As a further illustration, note that it follows that $(KL)_a = L_a$ in case K_a is undefined, $K\!\downarrow$, and L_a is defined.)

Sum, concatenation, and star generalize the usual operations on (total) languages to partial languages (although their definitions look different from the classical inductive ones). The synchronized product $K \parallel_I L$, with respect to a set of input symbols $I \subseteq A$, allows K and L to *shuffle* their respective a-steps as long as a is not in I; if a is in I then K and L should take that a-step synchronously. There are two special cases that deserve a notation of their own, called the (free) shuffle product, and the intersection of K and L, respectively:

$$K \parallel L = K \parallel_\emptyset L\,, \quad K \wedge L = K \parallel_A L$$

Note that in the definition of the synchronous product, regardless of what $I \subseteq A$ is, there is no restriction on the 'alphabets' of (that is, the input symbols occurring in) the languages K and L. Notably, these need not be disjoint in the definition of the shuffle product.

In the supervised product $K /_I L$, the language K acts as a *supervisor* that restricts the behaviour of L by forbidding certain steps: Only those a-steps of L are allowed that either K itself can take as well, or that are in I, where the set I is a given set of favoured actions. In Section 8 and further, the typical choice for I will be the set of so-called *uncontrollable* actions, which no supervisor can ever block. After an a-step of both K and L, the resulting language L_a will be supervised by K_a, which has taken the passing of the a-step into account as well. In case $L \xrightarrow{a}$, $K \not\xrightarrow{a}$ and $a \in I$, the supervised product transforms into $0 /_I L_a$, implying that in the future only those steps of L_a are allowed that are in I. For the purpose of the present paper, this type of supervisor will be sufficient but, more generally, one often defines a supervisor as a function $\sigma : A^* \to \mathcal{P}(A)$. The supervision by a (partial) language K corresponds to a supervisor σ_K which is defined, for any word w in A^*, by

$$\sigma_K(w) = \{a \in A \mid K \xrightarrow{wa}\} \cup I$$

In Section 9, we shall use the following generalisation of the sum, defined on families of languages:

$$(\bigcup\{K_i \mid i \in I\})_a = \bigcup\{(K_i)_a \mid i \in I \text{ and } K_i \xrightarrow{a}\},$$

$$(\bigcup\{K_i \mid i \in I\})\downarrow \text{ iff } (\exists\, i \in I :\ (K_i)\downarrow)$$

One can prove by coinduction that some of the above definitions of the operators on $\mathcal{L}$ could also have been given using formulae familiar from classical language theory as in, for instance,

$$\begin{aligned} K + L &= (K^1 \cup L^1, K^2 \cup L^2) \\ KL &= (\{vw \mid v \in K^1,\ w \in L^1\},\ \{vw \mid v \in K^2,\ w \in L^2\}) \\ K^* &= \bigcup_{n \geq 0} K^n \end{aligned}$$

For the synchronized and supervised products, such 'internal' descriptions (in terms of the elements of the languages involved) are possible as well, but they are rather awkward and not very practical to use. More importantly, however, none of these internal descriptions will be of much use to us: all our proofs will be by coinduction, requiring the construction of bisimulation relations. Since bisimulation relations are defined in terms of a-steps, such constructions are most easily based on the behavioural specifications above. This will be illustrated by many examples in Section 7.

7 Coinduction and regularity

A procedural explanation of proofs by coinduction is given, which is shown to be effective for regular partial languages. A series of examples concludes this section, which may help the reader to acquire the skill of *input derivation* and the technique of proofs by coinduction.

The strength of the present coinduction proof principle is that it can be used to prove any valid equality of partial languages, and that it always works in the same way: In order to prove an equality $K = L$ of partial languages K and L in $\mathcal{L}$, one defines a relation R in stages. The first pair to be included is $\langle K, L\rangle$. Next the following step is repeated until it does not yield any new pairs: for any pair $\langle M, N\rangle$ already present in R and for any input symbol a in A, one investigates whether both M_a and N_a are defined, in which case the pair $\langle M_a, N_a\rangle$ is added to R. If neither M_a nor N_a is defined, no pair has to be added to R, and we continue. If only one of them is defined, the procedure aborts, and we conclude that very moment that $K \neq L$. When adding a pair $\langle M, N\rangle$ to R, at any stage of its construction, we should check whether $M\downarrow$ iff $N\downarrow$. If this condition is not fulfilled the procedure aborts and, again, the conclusion is that $K \neq L$. If the procedure never aborts then the relation R is, by construction, a bisimulation and $K = L$ follows, by coinduction (Theorem 5.1(1)).

The coinductive proof principle is effective, that is, terminates in finitely many steps, for partial languages that are regular. First this notion will be formally defined and compared to the classical notion of regularity, and then a series of examples of coinductive proofs will be presented.

A language L in $\mathcal{L}$ is *regular* if it is an element of the set $\mathcal{R} \subseteq \mathcal{L}$, which is inductively defined by

1. $0 \in \mathcal{R}$, $1 \in \mathcal{R}$, and $a \in \mathcal{R}$ (for $a \in A$)
2. if $K \in \mathcal{R}$ and $L \in \mathcal{R}$ then $(K + L) \in \mathcal{R}$, $(KL) \in \mathcal{R}$ and $K^* \in \mathcal{R}$

The class of regular partial languages contains all classical regular languages, but has many more elements. Notably, concatenation of a classical language L with the constant language 0 (classically defined as the empty set) yields 0 again: $0L = L0 = 0$. Recalling the definition of the constant partial language $0 = (\emptyset, \{\varepsilon\})$, one still has $0L = 0$, for partial languages $L \in \mathcal{L}$, but generally, $L0 \neq 0$: for instance, $a0 = (\emptyset, \{\varepsilon, a\})$. (See items 7 and 8 in Examples 7.5 below.) The behaviour of $a0$ consists of the ability of taking an a-step to 0, which then blocks; neither $a0$ nor 0 are accepting.

The following theorem immediately implies the effectivity of the coinductive proof method. In its proof, we shall need the familiar laws for sum, formulated in the lemma below. The proof of the lemma provides a first simple exercise in coinductive reasoning.

Lemma 7.1 For languages K, L, and M in $\mathcal{L}$,

$$K + K = K\,, \quad K + L = L + K\,, \quad (K + L) + M = K + (L + M)$$

Proof: To prove the first law, consider the relation $R = \{\langle K + K, K\rangle \mid K \in \mathcal{L}\}$ on $\mathcal{L}$. For a pair $\langle K + K, K\rangle$ in R, note that $(K + K)\downarrow$ iff $K\downarrow$, and that $\langle (K + K)_a, K_a\rangle = \langle K_a + K_a, K_a\rangle \in R$. This proves that R is a bisimulation relation, from which the first law follows by coinduction. For the commutativity law, a first attempt for the definition of a bisimulation relation might be, similarly, $T = \{\langle K + L, L + K\rangle \mid K, L \in \mathcal{L}\}$. Trying to prove that this is indeed a bisimulation, one investigates a-steps of pairs $\langle K + L, L + K\rangle$ in T, for input symbols a in A. If both K_a and L_a are defined then $\langle (K + L)_a, (L + K)_a\rangle = \langle K_a + L_a, L_a + K_a\rangle \in T$, indeed. However, if K_a is defined and L_a is not, we find that $\langle (K + L)_a, (L + K)_a\rangle = \langle K_a, K_a\rangle$, which is not (at once recognizable as) an element of T. The remedy should be clear: rather than working with T, one shows that $T \cup \{\langle K, K\rangle \mid K \in \mathcal{L}\}$ is a bisimulation, which is trivial. Then the second law follws by coinduction, after all. The third law is proved in the same manner. □

Theorem 7.2 For a language L in $\mathcal{L}$, L is regular iff the subautomaton $\langle L\rangle \subseteq \mathcal{L}$ generated by L is finite.

Proof: ($\Rightarrow$) For any language L in $\mathcal{L}$, $\langle L\rangle = \{L_w \mid w \in A^* \text{ and } L \xrightarrow{w}\}$. This set is clearly finite for the constant languages 0, 1, and a. Next assume that both $\langle K\rangle$ and $\langle L\rangle$ are finite. It is immediate from the definition of $(K + L)_a$ that $\langle K + L\rangle$ is finite. For KL one can show, by induction on the length of words w, that $(KL)_w = K' + L' + \cdots + L''$, for some $K' \in \langle K\rangle$ and $L', \ldots, L'' \in \langle L\rangle$. Using the idempotency, commutativity, and associativity laws for $+$, it follows that there are only finitely many such languages, which proves that $\langle KL\rangle$ is finite. Similarly, one shows that $(K^*)_w = K'K^* + \cdots + K''K^*$, for some languages $K', \ldots, K'' \in \langle K\rangle$, implying that $\langle K^*\rangle$ is finite.
($\Leftarrow$) This half of the theorem is proved in very much the same way as the half of Kleene's Theorem that states that the language recognized by a finite automaton is regular. Only sketching the proof, therefore, we observe that the behaviour of the finite automaton $\langle L\rangle$ is determined by a finite set of finite equations of the shape $K = aK_a + \cdots + bK_b + x$, for any K in $\langle L\rangle$, where the set $\{a, \ldots, b\} = \{a \mid K \xrightarrow{a}\}$ and where $x = 1$, if $K\downarrow$, and $x = 0$, otherwise. Solving this system of equations using some of the familiar laws for languages (notably including: if $L\uparrow$ and $K = LK + M$ then $K = L^*M$, proved in Examples 7.5.14 below), one finds regular expressions for each of the elements K in $\langle L\rangle$, including L itself. □

Corollary 7.3 For regular languages K and L in $\mathcal{R}$, let $k = |\langle K \rangle|$ and $l = |\langle L \rangle|$ be the sizes of the subautomata of $\mathcal{L}$ they generate. The coinductive proof method described above decides in at most $k \times l$ steps whether $K = L$ or not.

Proof: Equality of K and L follows by coinduction from the construction of a bisimulation relation containing the pair $\langle K, L \rangle$, consisting of pairs $\langle K_w, L_w \rangle$ for words w in A^*. Since $K_w \in \langle K \rangle$ and $L_w \in \langle L \rangle$, for any word w, there are at most $k \times l$ many of such pairs. □

The other operators, synchronized product and supervised product, preserve regularity as well.

Proposition 7.4 If K and L are regular languages then so are $K \parallel_B L$ and $K /_B L$.

Proof: Consider regular languages K and L. By Theorem 7.2, it is sufficient to prove that $\langle K \parallel_B L \rangle$ and $\langle K /_B L \rangle$ are finite subautomata of $\mathcal{L}$ or, equivalently, that there are only finitely many derivates $(K \parallel_B L)_w$ and $(K /_B L)_w$. This follows from the definition of these respective products in a similar way to the first half of the proof of Theorem 7.2. □

Examples 7.5 Things are trivial at the outset here, but rapidly become somewhat more interesting.

1. $(1)_a$ and $(0)_a$ are undefined, $1\downarrow$ and $0\uparrow$.
2. $(a)_a = 1$, and $(a)_b$ is undefined, for $b \neq a$, and $a\uparrow$.
3. $(a + b)_a = (a + b)_b = 1$
4. $((a + b)^*)_a = (a + b)_a(a + b)^* = 1(a + b)^*$, and similarly $((a + b)^*)_b = 1(a + b)^*$.
5. Since $(1K)_a = K_a$ and $1K\uparrow$ iff $K\uparrow$, the relation $\{\langle K, 1K \rangle \mid K \in \mathcal{L}\} \cup \{\langle K, K \rangle \mid K \in \mathcal{L}\}$ is a bisimulation on $\mathcal{L}$. Thus $1K = K$, by coinduction.
6. As a consequence: $((a + b)^*)_a = ((a + b)^*)_b = (a + b)^*$.
7. $0K = 0$ and $0 + K = K$, since $\{\langle 0K, 0 \rangle \mid K \in \mathcal{L}\}$ and $\{\langle 0 + K, K \rangle \mid K \in \mathcal{L}\}$ are bisimulations.
8. $K0 = (\emptyset, K^2)$ (and so, generally, $K0 \neq 0$): $R = \{\langle K0, (\emptyset, K^2) \rangle \mid K \in \mathcal{L}\}$ is a bisimulation, since both $(K0)\uparrow$ and $(\emptyset, K^2)\uparrow$, and $\langle (K0)_a, (\emptyset, K^2)_a \rangle = \langle K_a, (\emptyset, K_a^2) \rangle \in R$.
9. For the law $K(L + M) = KL + KM$, we consider the relation

   $$R = \{\langle K(L + M) + N,\ KL + KM + N \rangle \mid K, L, M, N \in \mathcal{L}\}$$

   and reason as follows to show that it is a bisimulation (assuming that $K\downarrow$, the case of $K\uparrow$ being similar):

   $$\begin{array}{ll} & (K(L + M) + N)_a \\ = & K_a\,(L + M) + L_a + M_a + N_a \\ R & K_aL + K_aM + L_a + M_a + N_a \\ = & K_aL + L_a + K_aM + M_a + N_a \\ = & (KL)_a + (KM)_a + N_a \\ = & (KL + KM + N)_a \end{array}$$

10. $(b^*a)_a = 1$ and $(b^*a)_b = b^*a$, hence $((b^*a)^*)_a = (b^*a)^*$ and $((b^*a)^*)_b = (b^*a)(b^*a)^*$.
11. Let $K = (b^*a)^*ab^*$. Then $K\uparrow$ and assuming $a \neq b$, we find $K_a = K + b^*$ and $K_b = (b^*a)K$.

12. With K as above, $(K + b^*)K^* = (a + b)^*$: the following two-element relation

$$R = \{\langle (K + b^*)K^*, (a + b)^* \rangle,\ \langle ((b^*a)K + b^*)K^*, (a + b)^* \rangle\}$$

can be shown to be a bisimulation relation on $\mathcal{L}$, from which the equality follows by coinduction. Treating one interesting case in detail, we reason as follows:

$$\begin{array}{ll} & ((K + b^*)K^*)_b \\ = & (K_b + (b^*)_b)K^* + K_bK^* \quad [\text{note that } (K + b^*)\downarrow] \\ = & K_bK^* + b^*K^* + K_bK^* \\ = & K_bK^* + b^*K^* \\ = & ((b^*a)K)K^* + b^*K^* \\ = & ((b^*a)K + b^*)K^* \\ R & (a + b)^* \\ = & ((a + b)^*)_b \end{array}$$

13. The reader is invited to take courage now, and to try and prove the following equality: $[(b^*a)^*ab^*]^* = 1 + a(a + b)^* + (a + b)^*aa(a + b)^*$. A proof by coinduction can be found in [Rut98]. It uses a 5-element bisimulation relation, which contains the two-element relation R of Example 12 as a subset.

14. If $L\uparrow$ and $K = LK + M$ then $K = L^*M$: Consider languages K, L, and M in $\mathcal{L}$ with $L\uparrow$ and $K = LK + M$. It will be sufficient to prove that the relation

$$R = \{\langle (UK + V), (UL^*M + V) \rangle \mid U, V \in \mathcal{L}\}$$

which includes $\langle K, L^*M \rangle = \langle 1K + 0, 1(L^*M) + 0 \rangle$, is a bisimulation. First observe that, for languages U and V in $\mathcal{L}$, $(UK + V)\downarrow$ iff $(UL^*M + V)\downarrow$, since $K\downarrow$ iff $M\downarrow$. Assuming $U\downarrow$ (the other case being similar), one has

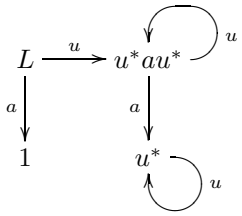
$$\begin{array}{ll} & (UK + V)_a \\ = & (UK)_a + V_a \\ = & U_aK + K_a + V_a \\ = & U_aK + (LK + M)_a + V_a \quad [\text{since } K = LK + M, \text{ by assumption}] \\ = & U_aK + L_aK + M_a + V_a \quad [\text{since } L\uparrow, \text{ by assumption}] \\ = & (U_a + L_a)K + (M_a + V_a) \\ R & (U_a + L_a)(L^*M) + (M_a + V_a) \\ = & U_a(L^*M) + L_aL^*M + M_a + V_a \\ = & (U(L^*M))_a + V_a \\ = & (U(L^*M) + V)_a \end{array}$$

This proves that R is a bisimulation relation.

15. Here are some examples of derivatives of the shuffle and synchronized product:

$$\begin{array}{rcl} ([ab]^* \parallel_{\{a\}} [ac]^*)_a & = & b[ab]^* \parallel_{\{a\}} c[ac]^* \\ ([ab]^* \parallel [ac]^*)_a & = & (b[ab]^* \parallel [ac]^*) + ([ab]^* \parallel c[ac]^*) \\ ([ab]^* \parallel [cd]^*)_c & = & [ab]^* \parallel d[cd]^* \\ (([ab]^* \parallel [cd]^*) \parallel_{\{c\}} [ce]^*)_c & = & ([ab]^* \parallel d[cd]^*) \parallel_{\{c\}} e[ce]^* \end{array}$$

16. Let $A = \{a, u\}$ and $L = a + uu^*au^*$. By computing the derivatives of L: $L_a = 1$, $L_u = u^*au^*$, $(u^*au^*)_u = u^*au^*$, $(u^*au^*)_a = u^*$, $(u^*)_u = u^*$, we obtain the following picture of the automaton $\langle L \rangle \subseteq \mathcal{L}$ generated by L:

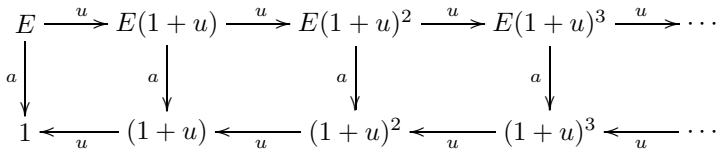


Note that the automaton is finite because L is regular (by Theorem 7.2).

17. This example illustrates Theorem 7.2 again, by showing a language that is not regular and the corresponding infinite automaton it generates. Let $A = \{a, u\}$ and let the language E be given by the following coinductive definition:

$$E_a = 1, \quad E_u = E(1 + u), \quad E{\uparrow}$$

(The unique existence of a language E satisfying these equations can be proved along the lines of the proof of the well-definedness of the operators in Section 13.) Computing derivatives again, we find $(E(1 + u))_a = 1 + u$ and $(E(1 + u))_u = E_u(1 + u) = E(1 + u)^2$. Continuing this way, the following picture of $\langle E \rangle$ is obtained:



One can easily prove that all of the expressions occurring in this picture represent different languages or, equivalently, that none of them are pairwise bisimilar. Thus the automaton $\langle E \rangle$ is infinite, indeed, and consequently, the language E is not regular.

8 Controllability

A notion of controllability is introduced in terms of *control relations* on automata. Using them to characterize the corresponding notion from control theory (of controllable sublanguage), a coinduction proof principle for controllability is obtained. Strengthening the definition of control relation, we then introduce *partial bisimulations*, which have a pleasant characterization: two languages K and L are partially bisimilar iff the result of supervising L with K equals K.

Here and in the remainder of this paper, let the set of input symbols be given by $A = C + U$, the disjoint union of two sets C and U, whose elements are called, respectively, *controllable* input symbols (or events) and *uncontrollable* input symbols. A relation $R \subseteq S \times S'$ between two automata $S = (S, \langle o, t \rangle)$ and $S' = (S', \langle o', t' \rangle)$ is a *control relation* if, for all s in S and s' in S':

$$\text{if } s\,R\,s' \text{ then } \begin{cases} (i) & \forall\, a \in A:\ (s \xrightarrow{a} \text{ and } s' \xrightarrow{a}) \Rightarrow s_a\ R\ s'_a\,, \text{ and} \\ (ii) & \forall\, u \in U:\ s' \xrightarrow{u} \Rightarrow (s \xrightarrow{u} \text{ and } s_u\ R\ s'_u) \end{cases}$$

(Note that the 'and $s_u\ R\ s'_u$' in (ii) already follows from (i).) Unions and (relational) compositions of control relations are control relations again. We write $s \geq^U s'$ whenever there exists a control relation R with $s\,R\,s'$, and say that s *is controllable with respect to* s'. The intuition is that the behaviour of (that is, the language represented by) s can be obtained by suitably constraining the behaviour of s', where constraining means forbidding certain steps that s' (at some stage) can

take, but in such a way that only steps in C and no steps in U are forbidden. The relation $\geq^U$, called the *controllability* relation, is the union of all control relations and, therewith, the greatest control relation.

Clearly, the definition of control relation is a variation on both the definitions of simulation and bisimulation. Note that unlike those definitions, it does not depend on the observation functions o and o', since it does not contain any requirements regarding acceptance of states. We have chosen the symbol $\geq^U$ for the controllability relation, since it suggests that U-steps from the right component should be mimicked by the left component.

The above definition of controllability is consistent with the following notion from control theory: A (classical) language $K \subseteq A^*$ is *controllable with respect to* a (classical) language $L \subseteq A^*$ (which is the closed behaviour of an automaton $\mathcal{G}$) if

$$\overline{K}U \cap L \subseteq \overline{K}$$

where $\overline{K}$ is the prefix-closure of K and where $\overline{K}U = \{vu \in A^* \mid v \in \overline{K} \text{ and } u \in U\}$. In order to make the connection between the two definitions precise, we first generalize the latter to partial languages: A language $K = (K^1, K^2)$ in $\mathcal{L}$ is controllable with respect to another language $L = (L^1, L^2)$ in $\mathcal{L}$ iff $K^2U \cap L^2 \subseteq K^2$. Now there is the following theorem.

Theorem 8.1 For all K and L in $\mathcal{L}$, $K \geq^U L$ iff $K^2U \cap L^2 \subseteq K^2$.

The implication from left to right serves again as a coinductive proof principle, similar to Theorems 4.1 and 5.2. The procedural interpretation of the coinductive proof principle, described in Section 7, again applies: Proving that a language K is controllable with respect to another language L amounts to the step by step construction of a control relation R, starting with the pair $\langle K, L\rangle$. If at some stage of the construction the relation contains a pair $\langle M, N\rangle$ with $N \xrightarrow{u}$ and $M \not\xrightarrow{u}$, for some u in U, then the procedure aborts, and we conclude that K is not controllable with respect to L. Otherwise the procedure is continued until no new pairs are found. Then the result is, by construction, a control relation, and it follows from Theorem 8.1 that K is controllable with respect to L. Also the proof of Corollary 7.3 applies again: if the languages K and L are regular then the procedure is effective, because in that case, the number of pairs $\langle K_w, L_w\rangle$, for words w in A^*, is finite.

Proof of Theorem 8.1: ($\Rightarrow$) Consider languages K and L in $\mathcal{L}$ with $K \geq^U L$, and consider $w \in A^*$ and $u \in U$ with $w \in K^2$ and $wu \in L^2$. Since L^2 is prefix-closed, also $w \in L^2$. Therefore $K \xrightarrow{w}$ and $L \xrightarrow{w}$, whence $K_w \geq^U L_w$, by repeated application of (i) above to the control relation $\geq^U$. Now $wu \in L^2$ implies $L_w \xrightarrow{u}$, and thus $K_w \xrightarrow{u}$, by (ii) above. Equivalently, $wu \in K^2$, which was to be shown.

($\Leftarrow$) Let $R = \{\langle M, N\rangle \mid M, N \in \mathcal{L} \text{ and } M^2U \cap N^2 \subseteq M^2\}$. Consider languages M and N with $M\,R\,N$, and suppose $M \xrightarrow{a}$ and $N \xrightarrow{a}$, for an input symbol $a \in A$. Then $M^2U \cap N^2 \subseteq M^2$ implies $M_a^2U \cap N_a^2 \subseteq M_a^2$, whence $M_a\,R\,N_a$. If $N \xrightarrow{u}$, for $u \in U$, then $u \in N^2$. Since $u = \varepsilon u \in M^2U$, this implies $u \in M^2$, whence $M \xrightarrow{u}$. This proves that R is a control relation. □

Examples 8.2 Here are some simple examples of proofs of controllability.

1. Let $C = \{a, b\}$, $U = \{u, v\}$, $K = avb$, and $L = (a(u + vb))^*$. Then $K \not\geq^U L$, since $L_a = (u + vb)L \xrightarrow{u}$ and $K_a = vb \not\xrightarrow{u}$.
2. With C, U, and L as in the previous example, let $K = au + av$. Then $K \geq^U L$ because $\{\langle K, L\rangle, \langle u + v, (u + vb)L\rangle, \langle 1, L\rangle, \langle 1, bL\rangle\}$ is a control relation.

Next the notion of *partial bisimulation* is introduced. It generalizes that of control relation and is possibly more natural, as is illustrated by the fact that it can be precisely characterized in terms of supervisors (see Theorem 8.3 below): A relation $R \subseteq S \times S'$ between two automata $S = (S, \langle o, t\rangle)$ and $S' = (S', \langle o', t'\rangle)$ is a *partial bisimulation* if, for all s in S and s' in S':

$$\text{if } s\,R\,s' \text{ then } \begin{cases} (i) & o(s) = o'(s'), \\ (ii) & \forall\, a \in A:\ s \xrightarrow{a} \Rightarrow\ (s' \xrightarrow{a} \text{ and } s_a\ R\ s'_a\,), \text{ and} \\ (iii) & \forall\, u \in U:\ s' \xrightarrow{u} \Rightarrow\ (s \xrightarrow{u} \text{ and } s_u\ R\ s'_u\,) \end{cases}$$

Unions and (relational) compositions of partial bisimulations are partial bisimulations again. We write $s \sim_U s'$ whenever there exists a partial bisimulation R with $s R s'$. This relation $\sim_U$, called the *partial bisimilarity* relation, is the union of all partial bisimulations and, therewith, the greatest partial bisimulation. The name *partial* bisimulation is clearly motivated by the fact that its definition can be obtained by weakening the definition of bisimulation, more specifically, by requiring the condition in clause (iii) to hold for uncontrollable input symbols $u \in U$ only rather than for any a in A.

The following theorem gives a precise characterization of the notion of partial bisimilarity.

Theorem 8.3 For languages K and L in $\mathcal{L}$, $K \sim_U L$ iff $K = (K /_U L)$.

The implication from left to right of this theorem has again a procedural reading: In order to show, for two given languages K and L, that K can be obtained from L by means of a supervised product (with K), it is sufficient to establish the existence of a partial bisimulation containing the pair $\langle K, L \rangle$. Again, the procedure is complete, and effective for regular K and L.

Proof of Theorem 8.3: ($\Rightarrow$) Assuming $K \sim_U L$, we shall prove $K \sim (K /_U L)$, from which the equality then follows by coinduction (Theorem 5.1(1)). Define

$$R = \{ \langle M, (M /_U N) \rangle \mid M, N \in \mathcal{L} \text{ and } M \sim_U N \}$$

In order to show that R is a bisimulation relation on $\mathcal{L}$, consider M and N with $M \, R \, N$. First of all note that $M{\downarrow}$ iff $N{\downarrow}$, since $M \sim_U N$, iff $(M /_U N){\downarrow}$, by the definition of the supervised product. If $M \xrightarrow{a}$, for a in A, then $N \xrightarrow{a}$ whence $(M /_U N) \xrightarrow{a}$. In that case, $(M /_U N)_a = (M_a /_U N_a)$ and since $M_a \sim_U N_a$, $M_a \, R \, (M_a /_U N_a)$. If $(M /_U N) \xrightarrow{a}$, for a in A, then either: $M \xrightarrow{a}$ and $N \xrightarrow{a}$, or: $M \not\xrightarrow{a}$, $N \xrightarrow{a}$, and $a \in U$. The latter possibility, however, is contradicted by the fact that $M \sim_U N$, so we are back in the previous case. This proves that R is a bisimulation relation.
($\Leftarrow$) For the reverse implication, we show that

$$T = \{ \langle M, N \rangle \mid M, N \in \mathcal{L} \text{ and } M = (M /_U N) \}$$

is a partial bisimulation. For M and N with $M \, T \, N$, $M{\downarrow}$ iff $(M /_U N){\downarrow}$ iff $N{\downarrow}$. If $M \xrightarrow{a}$, for a in A, then $(M /_U N) = M \xrightarrow{a}$. This implies $N \xrightarrow{a}$, by the definition of $(M /_U N)$, and $M_a \, T \, N_a$, since $M_a = (M /_U N)_a = (M_a /_U N_a)$. Conversely, if $N \xrightarrow{u}$, for u in U this time, then $(M /_U N) \xrightarrow{u}$, and thus $M = (M /_U N) \xrightarrow{u}$ as well. We have $M_u = (M /_U N)_u = (M_u /_U N_u)$, where the latter equality follows from the fact that both $M \xrightarrow{u}$ and $N \xrightarrow{u}$. Thus $M_u \, T \, N_u$. This proves that T is a partial bisimulation. □

Examples 8.4 Here are some examples of partially bisimilar languages:

1. Let $C = \{a\}$, $U = \{u, v\}$, $K = u(a + vv^*0)$, and $L = ua + uvv^*av^*$. Because

   $$\{ \langle K, L \rangle, \langle a + vv^*0, a + vv^*av^* \rangle, \langle 1, 1 \rangle, \langle v^*0, v^*av^* \rangle \}$$

   is a partial bisimulation, $K \sim_U L$ (and hence $K = K /_U L$, by Theorem 8.3).

2. Let $C = \{c_1, c_2\}$ and $U = \{p_1, p_2\}$, let $K = c_1p_1c_2M + 1$ with $M = (c_1p_1p_2c_2 + c_1p_2p_1c_2 + p_2c_1p_1c_2)^*p_2$, and let $L = (c_1p_1)^* \parallel (c_2p_2)^*$. (See Example 9.5 for an operational intuition for these languages.) We claim that $K \sim_U L$, because the following relation is readily shown to be a partial bisimulation: $\{ \langle K, L \rangle, \langle p_1c_2M, p_1(c_1p_1)^* \parallel (c_2p_2)^* \rangle, \langle c_2M, L \rangle, \langle M, (c_1p_1)^* \parallel p_2(c_2p_2)^* \rangle, \langle (p_1p_2c_2 + p_2p_1c_2)M, p_1(c_1p_1)^* \parallel p_2(c_2p_2)^* \rangle, \langle p_2c_2M, (c_1p_1)^* \parallel p_2(c_2p_2)^* \rangle \}$.

It is immediate from the definitions that any partial bisimulation is also a simulation, as well as a control relation. More precisely, there is the following characterization of partial bisimilarity, which can be proved along the lines of Theorems 5.2 and 8.1. It will play no role in the sequel.

Proposition 8.5 For languages $K = (K^1, K^2)$ and $L = (L^1, L^2)$ in $\mathcal{L}$,

$$K \sim_U L \text{ iff } K \leq L,\ K \geq^U L, \text{ and } K^1 = K^2 \cap L^1$$

(A language K for which $K^1 = K^2 \cap L^1$, is sometimes called L^1-closed.) □

9 Supremal controllable sublanguages

The problem to be solved here is the following: Given two languages E and L, we want to find the largest language K in $\mathcal{L}$ such that both $K \leq E$ and $K{\sim_U}L$. (The set of input symbols is again assumed to consist of controllable and uncontrollable actions: $A = C + U$.) Intuitively, L represents some given unconstrained behaviour that should be minimally restricted, as before by forbidding controllable steps only, in such a way that the resulting behaviour satisfies the specification E. In this section, the existence of the language K is proved and a procedure for the construction of an automaton representing K is described. This procedure is effective whenever the languages E and L are regular.

Before we proceed, let us mention that there are quite a few problems related to the one above, which can be tackled by means of the techniques described below. For instance, one might look for the largest language K with $K \leq E$ and $K \geq^U L$. Yet another problem is to find the largest K with $K \leq E$ and $K{\sim_U}L$, such that K is moreover *non-blocking*. This latter problem will be dealt with in all detail in Section 10, using a minor variation of the present results, the first of which is the following theorem.

Theorem 9.1 Let E and L be two languages, and suppose there exists a language F in $\mathcal{L}$ with $F \leq E$ and $F{\sim_U}L$. Then there exists a largest language K in $\mathcal{L}$ satisfying $K \leq E$ and $K{\sim_U}L$. It is given by

$$K = \bigcup \{ F \in \mathcal{L} \mid F \leq E \text{ and } F{\sim_U}L \}$$

Proof: The requirement on the existence of a language F with $F \leq E$ and $F{\sim_U}L$ ensures that the union above is not taken over the empty set. The theorem is a direct consequence of the observation that, for any family of languages $\{F_i\}_{i\in I}$, if $F_i \leq E$ for all $i \in I$ then $(\bigcup_{i\in I} F_i) \leq E$, and if $F_i{\sim_U}L$ for all $i \in I$ then $(\bigcup_{i\in I} F_i){\sim_U}L$. The first of these implications is trivial. For the second, assume $F_i{\sim_U}L$ for all $i \in I$, and let $\{R_i\}_{i\in I}$ be a corresponding family of partial bisimulations. Since the relation $R = \{\langle \bigcup_{j\in J} F_j,\, N\rangle \mid J \subseteq I \text{ and } \forall\, j \in J :\ \langle F_j, N\rangle \in R_j\}$ is a partial bisimulation, which is readily verified, it follows that $(\bigcup_{i\in I} F_i){\sim_U}L$. □

An immediate consequence of this theorem is the existence of an optimal supervisor for L, ensuring that the specification E will be satisfied: By Theorem 8.3, we have that $K = K /_U L$.

Next an automaton will be constructed that represents K. It is obtained by means of a fixed point construction that resembles the description, in Proposition 3.2, of bisimilarity as a greatest fixed point. It is a little different, however, and the difference is somewhat subtle.

The following construction will be carried out: for any two automata S and T with designated (initial) states s in S and t in T, we shall construct an automaton X with designated state x such that: x 'satisfies' s, that is, $x \leq s$; such that $x{\sim_U}t$; and such that x is maximal (with respect to $\leq$) among all states (in any automaton) with those two properties. The solution to the original problem, that is, a representation for K, will then be obtained, in Corollary 9.4 below, by taking $s = E$ and $t = L$ in the automaton $\mathcal{L}$ of partial languages.

As a preliminary, we first show how any relation $H \subseteq S \times T$ between automata S and T can be turned into an automaton by defining acceptance and transitions, for any $\langle p, q\rangle$ in H and a in A, by

$$\langle p, q\rangle \downarrow \text{ iff } (p \downarrow \text{ and } q\downarrow)$$

$$\langle p, q\rangle_a = \begin{cases} \langle p_a, q_a\rangle & \text{if } p \xrightarrow{a} \text{ and } q \xrightarrow{a} \text{ and } \langle p_a, q_a\rangle \in H \\ \Uparrow & \text{otherwise} \end{cases}$$

The various relations that will come up below will be considered as automata in this sense. Now consider any two automata S and T with designated (initial) states s in S and t in T. Let the relation $R \subseteq S \times T$ be defined by

$$R = \{\langle s_w, t_w\rangle \mid w \in A^* \text{ and } s \xrightarrow{w} \text{ and } t \xrightarrow{w}\}$$

The set R thus carries an automaton structure as described above. Consider the following monotone operation Φ, which maps a relation $H \subseteq S \times T$ to

$$\Phi(H) = \{\langle p,q\rangle \in H \mid (q\!\downarrow \Rightarrow\ p\!\downarrow) \text{ and } \forall\, u \in U:\ q\xrightarrow{u} \Rightarrow\ (p\xrightarrow{u} \text{ and } \langle p_u, q_u\rangle \in H\,)\}$$

and define

$$\tilde{R} = \bigcap_{n\geq 0} \Phi^n(R)$$

The set $\tilde{R}$ is again an automaton but note that it is generally not a subautomaton of R: for a given state $\langle p,q\rangle \in \tilde{R}$, some of the states that were reachable in R may have disappeared from $\tilde{R}$. Also note that the computation of the automaton $\tilde{R}$ is effective in case the subautomata $\langle s\rangle \subseteq S$ and $\langle t\rangle \subseteq T$ generated by s and t are finite (in which case the languages $l(s)$ and $l(t)$ they represent are regular, by Theorem 7.2): If $e = |\langle s\rangle|$ and $l = |\langle t\rangle|$ then the automaton R consists of at most $e \times l$ states. This implies that $\tilde{R}$ is obtained from R in at most $e \times l$ applications of Φ.

Theorem 9.2 Let the automaton $\tilde{R}$ be as defined above:

1. For any state v in an automaton V: if $v \leq s$ and $v{\sim_U}t$ then $\langle s,t\rangle \in \tilde{R}$ and $v \leq \langle s,t\rangle$.
2. If $\langle s,t\rangle \in \tilde{R}$ then $\langle s,t\rangle \leq s$ and $\langle s,t\rangle{\sim_U}t$.

For the proof of the theorem, the following lemma will be used. It says that $\tilde{R}$ is the greatest fixed point of Φ that is contained in R.

Lemma 9.3 For Φ and $\tilde{R}$ as defined above,

1. $\tilde{R} = \Phi(\tilde{R})$
2. For $R' \subseteq R$: if $R' \subseteq \Phi(R')$ then $R' \subseteq \tilde{R}$.

Proof of Lemma 9.3: By the definition of Φ, $\Phi(H) \subseteq H$ for any $H \subseteq S \times T$, proving the inclusion from right to left in the first statement. Using the first statement, the second statement is immediate by the monotonicity of Φ. To prove $\tilde{R} \subseteq \Phi(\tilde{R})$, consider $\langle p,q\rangle \in \tilde{R}$. Because $\tilde{R} \subseteq \Phi(R)$, $q\!\downarrow$ implies $p\!\downarrow$. Next suppose $q\xrightarrow{u}$ for some u in U. Since $\langle p,q\rangle \in \Phi^{n+1}(R)$, for $n \geq 0$, we have $p\xrightarrow{u}$ and $\langle p_u, q_u\rangle \in \Phi^n(R)$, for $n \geq 0$, implying $\langle p_u, q_u\rangle \in \tilde{R}$. Thus $\langle p,q\rangle \in \Phi(\tilde{R})$. □

Proof of Theorem 9.2: For 1, consider a state v in an automaton V with $v \leq s$ and $v{\sim_U}t$. Let $H \subseteq V \times S$ be a simulation relation with $v\,H\,s$, and let $Q \subseteq V \times T$ be a partial bisimulation with $v\,Q\,t$. Define

$$P = \{\langle p,q\rangle \in R \mid \exists\, z \in V:\ z\,H\,p \text{ and } z\,Q\,q\}$$

We show that $P \subseteq \Phi(P)$: for $\langle p,q\rangle \in P$, there is $z \in V$ with $z\,H\,p$ and $z\,Q\,q$. If $q\!\downarrow$ then $z\!\downarrow$, since Q is a partial bisimulation, whence $p\!\downarrow$, because H is a simulation relation. If $q\xrightarrow{u}$, for $u \in U$, then $z\xrightarrow{u}$, $p\xrightarrow{u}$, with $z_u\,H\,p_u$ and $z_u\,Q\,q_u$ and thus $\langle p_u, q_u\rangle \in P$, for the same reasons. This proves that $\langle p,q\rangle \in \Phi(P)$, hence $P \subseteq \Phi(P)$. Consequently, by Lemma 9.3, $P \subseteq \tilde{R}$. In other words: if $z\,H\,p$ and $z\,Q\,q$ then $\langle p,q\rangle \in \tilde{R}$. Note that this implies in particular that $\langle s,t\rangle \in \tilde{R}$. Another consequence is that the set $P' = \{\langle z, \langle p,q\rangle\rangle \mid z \in V,\, p \in S,\, q \in T,\, z\,H\,p \text{ and } z\,Q\,q\}$ is a relation $P' \subseteq V \times \tilde{R}$. It is easily seen to be a simulation relation, which proves $v \leq \langle s,t\rangle$.

For 2, we assume that $\langle s,t\rangle \in \tilde{R}$. For the inequality $\langle s,t\rangle \leq s$, it is sufficient to show that $\{\langle\langle p,q\rangle, p\rangle \mid \langle p,q\rangle \in \tilde{R}\,\}$ is a simulation relation between $\tilde{R}$ and S, which is immediate from the definition of the automaton $\tilde{R}$. For $\langle s,t\rangle{\sim_U}t$, consider the relation $H = \{\langle\langle p,q\rangle, q\rangle \mid \langle p,q\rangle \in \tilde{R}\,\}$. We prove that H is a partial bisimulation between $\tilde{R}$ and T. For a pair $\langle p,q\rangle \in \tilde{R}$:

(i) $\langle p,q\rangle\!\downarrow$ iff ($p\!\downarrow$ and $q\!\downarrow$) iff $q\!\downarrow$, since $\langle p,q\rangle \in \tilde{R} = \Phi(\tilde{R})$.

(ii) If $\langle p,q\rangle\xrightarrow{a}$, for a in A, then both $p\xrightarrow{a}$ and $q\xrightarrow{a}$ and $\langle p_a, q_a\rangle$ in $\tilde{R}$.

(iii) If $q \xrightarrow{u}$, for u in U, then $p \xrightarrow{u}$ and $\langle p_u, q_u \rangle \in \tilde{R}$, since $\langle p, q \rangle \in \tilde{R} = \Phi(\tilde{R})$.

Thus H is a partial bisimulation, which concludes the proof of 2. □

An application of Theorem 9.2 to our original problem yields the following corollary.

Corollary 9.4 Let E and L be two languages, and suppose there exists a language F with $F \leq E$ and $F \sim_U L$. The construction above yields an automaton $\tilde{R}$ with $\langle E, L \rangle \in \tilde{R}$ and

$$l(\langle E, F \rangle) = (K =) \bigcup \{ F \in \mathcal{L} \mid F \leq E \text{ and } F \sim_U L \}$$

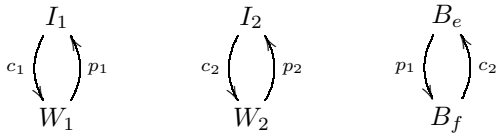
($l : \tilde{R} \to \mathcal{L}$ is the unique homomorphism given by the finality of $\mathcal{L}$).

Proof: In the construction of $\tilde{R}$ above, take $S = T = \mathcal{L}$, $s = E$, and $t = L$. Putting $V = \mathcal{L}$ and $v = F$, the assumptions of the corollary validate the conditions of part 1 of Theorem 9.2. As a consequence, $\langle E, L \rangle \in \tilde{R}$ and $F \leq \langle E, L \rangle$ and $F \sim_U \langle E, L \rangle$. Because $l : \tilde{R} \to \mathcal{L}$ is a homomorphism, $\langle E, L \rangle$ and $l(\langle E, L \rangle)$ are bisimilar, hence $F \leq l(\langle E, L \rangle)$ and $F \sim_U l(\langle E, L \rangle)$. Part 2 of Theorem 9.2 implies $l(\langle E, L \rangle) \leq E$ and $l(\langle E, L \rangle) \sim_U L$. This proves $l(\langle E, L \rangle) = K$. □

Example 9.5 Let $A = \{c_1, c_2, p_1, p_2\}$ and define

$$I_1 = (c_1 p_1)^*, \;\; W_1 = p_1 I_1, \;\; I_2 = (c_2 p_2)^*, \;\; W_2 = p_2 I_2, \;\; B_e = (p_1 c_2)^*, \;\; B_f = c_2 B_e$$

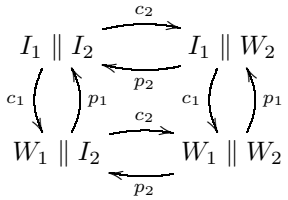
The language I_1 is the behaviour of an abstract machine that repeatedly consumes some input by executing c_1 and then (after some internal computation not modelled here) produces some output by executing p_2. One may view I_1 as the (initial) *idle* state and W_1 as the *working* state of the machine. Similarly, I_2 and W_2 constitute a second such machine. The states B_e and B_f are the *empty* and the *full* state of a buffer, whose role will be explained below. There are the following transitions:



The unconstrained behaviour of this example is the shuffle product of I_1 and I_2:

$$L = I_1 \parallel I_2$$

in which the steps of both machines are interleaved:



The outputs p_1 produced by machine 1 are intended to be consumed as inputs c_2 by machine 2. Moreover, we assume that before machine 1 produces a next output, its previous output must have been consumed by machine 2. None of these assumptions are modelled by the behaviour L, and so our aim is to restrict L in such a way that these requirements are met. An elegant way of specifying them is to assume the presence of a one-place buffer such as the one introduced above. In the empty state B_e, the buffer can store an output from machine 1 by synchronizing with the action p_1, meanwhile moving to the full state B_f. Next it can pass this output on to machine 2 by synchronizing with c_2, returning to the empty state. The specification E can now be defined by

$$E = (L \parallel_{\{p_1, c_2\}} B_e) = L \hat{\parallel} B_e$$

using $\hat{\|}$ as a shorthand for the synchronized product $\|_{\{p_1,c_2\}}$. Note that $E \leq L$. A last assumption is that a machine can be forbidden to consume an input, that is, to execute its c step, but once a machine has entered its working state, the production of its output is autonomous. Formally, this is modelled by assuming

$$C = \{c_1, c_2\}, \quad U = \{p_1, p_2\}$$

The problem now has become to find the largest language K such that $K \leq E$ and $K \sim_U L$. Applying the method of Theorem 9.2, the relation R turns out to consist of the following 8 pairs:

$$\begin{aligned} R &= \{\langle E_w, L_w\rangle \mid w \in A^* \text{ and } E \xrightarrow{w} \text{ and } L \xrightarrow{w}\} \\ &= \{\, \langle\, (I_1 \parallel I_2)\hat{\|}B_e,\, (I_1 \parallel I_2)\,\rangle,\ \langle\, (I_1 \parallel I_2)\hat{\|}B_f,\, (I_1 \parallel I_2)\,\rangle, \\ &\quad \langle\, (I_1 \parallel W_2)\hat{\|}B_e,\, (I_1 \parallel W_2)\,\rangle,\ \langle\, (I_1 \parallel W_2)\hat{\|}B_f,\, (I_1 \parallel W_2)\,\rangle, \\ &\quad \langle\, (W_1 \parallel I_2)\hat{\|}B_e,\, (W_1 \parallel I_2)\,\rangle,\ \langle\, (W_1 \parallel I_2)\hat{\|}B_f,\, (W_1 \parallel I_2)\,\rangle, \\ &\quad \langle\, (W_1 \parallel W_2)\hat{\|}B_e,\, (W_1 \parallel W_2)\,\rangle,\ \langle\, (W_1 \parallel W_2)\hat{\|}B_f,\, (W_1 \parallel W_2)\,\rangle\,\} \end{aligned}$$

The set R carries an automaton structure as defined above: the only accepting state is the first element, and transitions are determined by a-derivatives, as in

$$\langle\, (W_1 \parallel I_2)\hat{\|}B_e,\, (W_1 \parallel I_2)\,\rangle \xrightarrow{p_1} \langle\, (I_1 \parallel I_2)\hat{\|}B_f,\, (I_1 \parallel I_2)\,\rangle$$

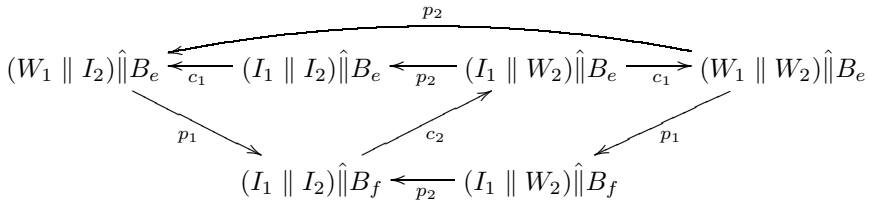
For the computation of $\tilde{R}$, one application of Φ is sufficient, since

$$\tilde{R} = \Phi(R) = R - \{\, \langle\, (W_1 \parallel I_2)\hat{\|}B_f,\, (W_1 \parallel I_2)\,\rangle,\ \langle\, (W_1 \parallel W_2)\hat{\|}B_f,\, (W_1 \parallel W_2)\,\rangle\,\}$$

The removal of, for instance, the first of these pairs is explained by

$$(W_1 \parallel I_2) \xrightarrow{p_1} \text{ and } (W_1 \parallel I_2)\hat{\|}B_f \not\xrightarrow{p_1}$$

Denoting the elements of $\tilde{R}$ by their first components only, the automaton looks like



The language K we were after is now obtained as the language represented by the state $(I_1 \parallel I_2)\hat{\|}B_e$ in the automaton above: $K = l((I_1 \parallel I_2)\hat{\|}B_e)$. Solving the equations corresponding to this 6-state automaton, the following explicit description is obtained:

$$K = c_1p_1c_2\,(c_1p_1p_2c_2 + c_1p_2p_1c_2 + p_2c_1p_1c_2)^*p_2 \,+\, 1$$

Example 9.6 This example is intended to illustrate that if the languages E and L are not regular then the construction of $\tilde{R}$ out of R in general does take infinitely many steps. Consider $A = C + U$ with $C = \{a\}$ and $U = \{u\}$, let $L = a + uu^*au^*$, and let E be given by the following coinductive definition:

$$E_a = 1, \quad E_u = E(1 + u), \quad E{\uparrow}$$

Although L is clearly regular, E is not: see examples 16 and 17 in Examples 7.5 for a description of the subautomata $\langle L\rangle \subseteq \mathcal{L}$ and $\langle E\rangle \subseteq \mathcal{L}$ they generate. Setting out to find the largest language K with $K \leq E$ and $K \sim_U L$, we compute

$$\begin{aligned} R &= R' \cup \{\, \langle (1+u)^k,\, u^*\rangle \mid k \geq 0\,\}, \text{ where} \\ R' &= \{\langle E, L\rangle, \langle 1, 1\rangle\} \cup \{\langle E(1+u)^k,\, u^*au^*\rangle \mid k \geq 1\,\} \end{aligned}$$

(here we use the convention that $(1+u)^0 = 1$). By induction on the number of applications of Φ, we find, for all $n \geq 0$,

$$\Phi^n(R) = R' \cup \{\, \langle (1+u)^k,\, u^* \rangle \mid k \geq n \,\}$$

whence $\tilde{R} = R'$. The language K is represented by the state $\langle E, L \rangle$ in the automaton $\tilde{R}$, which looks like

$$\langle 1,1\rangle \xleftarrow{\ a\ } \langle E, L\rangle \xrightarrow{\ u\ } \langle E(1+u), u^*au^*\rangle \xrightarrow{\ u\ } \langle E(1+u)^2, u^*au^*\rangle \xrightarrow{\ u\ } \cdots$$

Since $\langle E, L\rangle$ and the language $a + u^*0$ in $\mathcal{L}$ are bisimilar, it follows that $K = a + u^*0$. □

10 Non-blocking languages

First the notion of a non-blocking language is introduced, and then the following variation of the problem of Section 9 is studied: Given two languages E and L, we want to find the largest language K in $\mathcal{L}$ such that both $K \leq E$ and $K \sim_U L$ and such that, moreover, K is *non-blocking*. Again the existence of K is proved and a procedure for the construction of a representing automaton is given. This time the procedure is defined for regular languages only (the procedure of Section 9 worked for arbitrary languages but was effective for regular languages).

For a state s in an automaton S, we write $s\!\searrow$ iff there exists a word $w \in A^*$ with $s \xrightarrow{w}$ and $s_w\!\downarrow$. A state s in S is *non-blocking* if

$$\forall\, v \in A^* : \quad \text{if } s \xrightarrow{v} \text{ then } s_v\!\searrow$$

In words: from any state that is reachable from s, one can reach an accepting state. Since the set $\mathcal{L}$ of partial languages carries an automaton structure, we can also speak of *non-blocking languages*. For any language $L = (L^1, L^2)$ in $\mathcal{L}$, the following are equivalent:

1. The language $L = (L^1, L^2)$ is non-blocking
2. For all words $w \in A^*$, if $L \xrightarrow{w}$ then $L_w\!\searrow$
3. For all words $w \in L^2$, there exists a word $v \in A^*$ with $wv \in L^1$
4. $L^2 \subseteq \overline{L^1}$ (the prefix-closure of L_1)
5. $L^2 = \overline{L^1}$

(For the last equivalence, recall that $\overline{L^1} \subseteq L^2$, always.) Because a state s in an automaton S is non-blocking iff the language $l(s) = (L^1_s, L^2_s)$ it represents is non-blocking (since $l : S \to \mathcal{L}$ is a homomorphism), it follows that

$$s \text{ is non-blocking iff } \ \overline{L^1_s} = L^2_s$$

This condition can also be taken as a definition. Accordingly, a state is called *blocking* iff the inclusion $\overline{L^1_s} \subseteq L^2_s$ is strict: $\overline{L^1_s} \neq L^2_s$.

A word of caution about the terminology seems in order here. Let us call a state s in an automaton S a *deadlock* iff $s \not\rightarrow$ and $s\!\uparrow$. A state s is a *livelock* iff the subautomaton $\langle s\rangle$ of S generated by s is non-empty and contains no deadlock states and no accepting states. (A typical example of a livelock state is s with $s\!\uparrow$ and with $s \xrightarrow{a} s$ as the only transition.) It is straightforward to see that a state s is non-blocking iff $\langle s\rangle$ contains no deadlock and no livelock states. One could argue, therefore, that non-*locking* would actually be a better name.

The following is an easy variation on Theorem 9.1.

Theorem 10.1 Let E and L be two languages, and suppose there exists a *non-blocking* language F in $\mathcal{L}$ with $F \leq E$ and $F \sim_U L$. Then there exists a largest *non-blocking* language K in $\mathcal{L}$ satisfying $K \leq E$ and $K \sim_U L$. It is given by

$$K = \bigcup \{ F \in \mathcal{L} \mid F \leq E \text{ and } F \sim_U L \text{ and } F \text{ is non-blocking} \}$$

Proof: The proof is as before, adding the observation that for a family of languages $\{F_i\}_{i\in I}$, if $F_i^2 = \overline{F_i^1}$ for all $i \in I$ then $F^2 = \overline{F^1}$, where $F = \bigcup_{i\in I} F_i$. □

In order to construct an automaton representing K, the fixed point construction of Section 9 will be varied as follows. Consider any two automata S and T with designated (initial) states s in S and t in T, such that the subautomata $\langle s\rangle \subseteq S$ and $\langle t\rangle \subseteq T$ generated by s and t are finite and consider again the relation $R \subseteq S \times T$ defined by $R = \{\langle s_w, t_w\rangle \mid w \in A^* \text{ and } s\xrightarrow{w} \text{ and } t\xrightarrow{w}\}$, which is now finite. Relations $H \subseteq S \times T$ carry an automaton structure as defined in Section 9. Consider the following monotone operation Ψ, which maps a relation $H \subseteq S \times T$ to

$$\Psi(H) = \Phi(H) \cap \{\langle p, q\rangle \in H \mid \langle p, q\rangle \searrow \text{ in } H\}$$

where Φ is the operator of Section 9. Note that $\langle p, q\rangle\searrow$ should be interpreted in the *automaton* H. Let $\hat{R} \subseteq R$ be defined by

$$\hat{R} = \bigcap_{n\geq 0} \Psi^n(R)$$

There are the following variations of Lemma 9.3, Theorem 9.2, and Corollary 9.4.

Lemma 10.2 For Ψ and $\hat{R}$ as defined above,

1. $\hat{R} = \Psi(\hat{R})$
2. For $R' \subseteq R$: if $R' \subseteq \Psi(R')$ then $R' \subseteq \hat{R}$.

Note that $\hat{R}$ is finite because R is.

Proof: Since $R \supseteq \Psi^1(R) \supseteq \Psi^2(R) \supseteq \cdots$ and R is finite, there exists a natural number k (less than the size of R) such that for all $n \geq 0$, $\Psi^k(R) = \Psi^{k+n}(R)$. This implies $\hat{R} = \Psi^k(R) = \Psi^{k+1}(R) = \Psi(\hat{R})$, proving 1. Clause 2 follows, as before, from the monotonicity of Ψ. □

Theorem 10.3 Let the automaton $\hat{R}$ be as defined above.

1. For any *non-blocking* state v in an automaton V: if $v \leq s$ and $v{\sim_U}t$ then $\langle s, t\rangle \in \tilde{R}$ and $v \leq \langle s, t\rangle$.
2. If $\langle s, t\rangle \in \tilde{R}$ then $\langle s, t\rangle$ is *non-blocking*, $\langle s, t\rangle \leq s$, and $\langle s, t\rangle{\sim_U}t$.

Proof: For the proof of 1, let v be a non-blocking state in an automaton V with $v \leq s$ and $v{\sim_U}t$. We now consider the relation

$$P = \{\langle p, q\rangle \in R \mid \exists\, z \in V :\ z \text{ is non-blocking and } z\ H\ p \text{ and } z\ Q\ q\}$$

Again $P \subseteq \Psi(P)$: If $\langle p, q\rangle \in P$ then there is a non-blocking state $z \in V$ with $z\ H\ p$ and $z\ Q\ q$. Because z is non-blocking there exists a word $w \in A^*$ with $z\xrightarrow{w}z_w{\downarrow}$. This implies $p\xrightarrow{w}p_w{\downarrow}$ and $q\xrightarrow{w}q_w{\downarrow}$, because H is a simulation and Q is a partial bisimulation relation. It follows that $\langle p, q\rangle\xrightarrow{w}\langle p_w, q_w\rangle{\downarrow}$, in P, whence $\langle p, q\rangle\searrow$. This argument, in combination with the corresponding proof of Theorem 9.2, proves that $\langle p, q\rangle \in \Psi(P)$. As a consequence,

$$P' = \{\langle z, \langle p, q\rangle\rangle \mid z \text{ is non-blocking and } z\ H\ p \text{ and } z\ Q\ q\}$$

is again a well-defined simulation relation, which proves $v \leq \langle s, t\rangle$. For 2, we need to show in addition to the proof of Theorem 9.2, that if $\langle s, t\rangle$ is an element of $\hat{R}$ then $\langle s, t\rangle$ is non-blocking: Suppose $\langle s, t\rangle\xrightarrow{w}\langle s, t\rangle_w = \langle s_w, t_w\rangle$ in $\hat{R}$, for some word $w \in A^*$. Since $\langle s_w, t_w\rangle \in \hat{R} \subseteq \Psi(\hat{R})$, we have $\langle s_w, t_w\rangle\searrow$, in $\hat{R}$. Thus $\langle s, t\rangle$ is non-blocking, which concludes the proof of 2. □

Corollary 10.4 Let E and L be two regular languages, and suppose there exists a *non-blocking* language F with $F \leq E$ and $F \sim_U L$. The construction above yields an automaton $\tilde{R}$ with $\langle E, L\rangle \in \tilde{R}$ and

$$l(\langle E, F\rangle) = (K =) \bigcup \{F \in \mathcal{L} \mid F \leq E \text{ and } F \sim_U L \text{ and } F \text{ is non-blocking}\}$$

($l : \tilde{R} \to \mathcal{L}$ is the unique homomorphism given by the finality of $\mathcal{L}$).

Proof: As before, the conclusion follows from the lemma and theorem above, which do apply since the regularity of the languages E and L guarantees, by Theorem 7.2, that the subautomata they generate are finite. □

Example 9.6 illustrates that Corollary 10.4 does not hold for languages E and L that are not regular. It is easily seen that, for this particular example, it makes no difference whether Φ or Ψ is repeatedly applied to R. Therefore, $\hat{R} = \tilde{R} = R'$. And the language $K = a + u^*0$ represented by $\langle E, L\rangle$ in R', is *not* non-blocking, indeed.

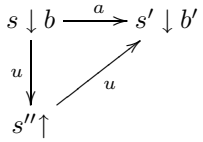
For an example illustrating the construction above, the reader is referred to the appendix in Section 15.

11 Output-control consistency

A coinductive definition is presented of an operation that transforms a (certain type of Moore) automaton into one that is so-called output-control-consistent. The procedural interpretation of this coinductive definition yields an algorithm that is more efficient than those in the literature.

For this, we return to the more general setting of Moore automata $(S, \langle o, t\rangle)$ of Section 3, with inputs in a set A and outputs in a set $B_\tau = B \cup \{\tau\}$, where A and B are arbitrary and τ is a special element not in B. Thus the output function $o : S \to B_\tau$ assigns to each state either an output in B or the so-called *silent* output τ. A state s in S is called *invisible* iff $o(s) = \tau$, also denoted by $s\!\uparrow$. And a state s is called *visible* iff $o(s) = b \in B$, denoted by $s \downarrow b$. The outputs B_τ of the automaton S are intended for the use of a supervisor at some higher level, which will base its decisions about how to restrict the behaviour of S on these outputs. The element τ is used when a (low level) state of S is considered irrelevant for the (high level) supervisor.

There is much more to be said about the context of the present section, which is the hierarchical supervision of discrete-event systems. Here we shall only deal with the following problem: Given a partition of the input symbols A into controllable and uncontrollable ones: $A = C_A + U_A$, define likewise such a partition for the output symbols: $B = C_B + U_B$ which is *consistent* with the partition of A, or in other words, such that the automaton S is *output-control-consistent*. A formal definition will follow in a second, but here is a typical example of what should be avoided. Let $C_A = \{a\}$, $U_A = \{u\}$, and $B = \{b, b'\}$, and consider



For this (fragment of an) automaton, it is impossible to decide whether output symbol b' should be controllable or not: starting in a visible state s, there is both a controllable path $s \xrightarrow{a} s'$ and an uncontrollable path $s \xrightarrow{uu} s'$ from s to s'. The intended observable behaviour on the higher level corresponding to these paths, is in both cases bb'. Therefore it is not possible to decide, irrespective of the choice for b, whether b' should be considered controllable or not. The way out will be to change the automaton into a new one, in which the state s' will occur twice, once with

a controllable version of b' and once with an uncontrollable version of b', as in

$$\begin{array}{ccc} s \downarrow b & \xrightarrow{\ a\ } & s'_c \downarrow b'_c \\ {\scriptstyle u}\big\downarrow & & \\ s''\uparrow & \xrightarrow[\ u\ ]{} & s'_u \downarrow b'_u \end{array}$$

The coinductive definition below will do precisely that. For the formal definition of output-control consistency, let us call a path $s \xrightarrow{w} s'$ in S, with $w \in A^*$, *silent* if both s and s' are visible and all intermediate states (that is, $s_{w'}$ with $w' \leq w$ and $w' \neq w$) are invisible. Furthermore call a silent path w controllable if w contains at least one controllable symbol c in C_A, and uncontrollable otherwise. Formally, now, an automaton S is output-control-consistent if the partitions $A = C_A + U_A$ and $B = C_B + U_B$ are such that the following holds: For any two visible states s with output b and s' with output b', either all of the silent paths between s and s' are controllable and b' is controllable in B: $b' \in C_B$, or all of these paths are uncontrollable and b' is uncontrollable in B: $b' \in U_B$.

Using the coinductive definition technique from Sections 6 and 13, solving the problem turns out to be easier than specifying it. Consider a Moore automaton $(S, \langle o, t\rangle)$ with inputs in A and outputs in $B_\tau = B \cup \{\tau\}$. We define a new Moore automaton $(S', \langle o', t'\rangle)$ as follows. Let

$$S' = \{C(s) \mid s \in S\} \cup \{U(s) \mid s \in S\}$$

The automaton S' has inputs again in A, but the set of outputs is $O_\tau = O \cup \{\tau\}$, with

$$O = C_O + U_O, \quad C_O = \{C(b) \mid b \in B\}, \quad U_O = \{U(b) \mid b \in B\}$$

So for every state s in S, the automaton S' contains two copies $C(s)$ and $U(s)$ and, similarly, the new set of output symbols O_τ contains (τ and) for every output symbol b in B two copies $C(b)$ and $U(b)$. The output function $o' : S' \to O_\tau$ is defined by the following clauses:

$$o'(C(s)) = \begin{cases} \tau & \text{if } o(s) = \tau \\ C(b) & \text{if } o(s) = b \in B \end{cases} \qquad o'(U(s)) = \begin{cases} \tau & \text{if } o(s) = \tau \\ U(b) & \text{if } o(s) = b \in B \end{cases}$$

Writing $(s')_a = t'(s')(a)$ as usual, the transition function $t' : S' \to (1 + S')^A$ is given by

$$(C(s))_a = \begin{cases} U(s_a) & \text{if } s \xrightarrow{a} \text{ and } (a \in U_A \text{ and } s \text{ is visible}) \\ C(s_a) & \text{if } s \xrightarrow{a} \text{ and } (a \in C_A \text{ or } s \text{ is invisible}) \\ \Uparrow & \text{otherwise} \end{cases}$$

$$(U(s))_a = \begin{cases} U(s_a) & \text{if } s \xrightarrow{a} \text{ and } a \in U_A \\ C(s_a) & \text{if } s \xrightarrow{a} \text{ and } a \in C_A \\ \Uparrow & \text{otherwise} \end{cases}$$

The automaton S', we now claim, is output-control-consistent. This can be easily checked, using the formal definition of output-control consistency given above. Rather than giving the proof in detail, let us try to explain the intuition behind the construction of S'. The syntactic annotations $C(s)$ and $U(s)$ on a state s in S can be understood as giving the following information about what happened before that state was reached: Being in a state $C(s)$ means that since the last time that we have passed through a visible state, we have encountered at least one controllable input symbol a in C_A. Dually, if we are in a state $U(s)$ then all of the input symbols that we have encountered since the last time that we passed through a visible state, have been uncontrollable.

Note that for any practical application of the above definition, the automaton S' could be built step for step, starting from a finite Moore automaton S with a silent initial state s in S. The first state to be included in S' would be $U(s)$. Next only those states will be added that correspond to states in S that are reachable from s. It is immediate from the construction of S' that the size of its state space is at most twice that of S. The computational complexity of this algorithmic reading of the definition of S', starting with a Moore automaton with n states and m transitions, will be of order $O(n + m)$: for each state in S there are at most two new states in S', and to each of the transitions $s \xrightarrow{a} s'$ in S will correspond at most four transitions in S', determined by the choices of either C or U for both s and s'.

12 Notes and discussion

As was mentioned in Section 3, the perspective of this paper is essentially coalgebraic, putting the notions of (Moore) automaton (which is a particular coalgebra), homomorphism, and bisimulation at the very heart of the theory, immediately followed by the notions of final automaton (in particular, the automaton of partial languages in Section 5) and coinduction. Most of the definitions and observations in Sections 3 and 4 are instances of more general ones, belonging to a theory called *(universal) coalgebra*. Notably, the definition of a bisimulation relation on Moore automata is an instance of the coalgebraic definition of bisimulation, which at its turn is a categorical generalization, due to Aczel and Mendler [AM89], of Park's [Par81] and Milner's [Mil80] notion of bisimulation for concurrent branching processes. This general categorical definition applies to many different examples, including non-deterministic and probabilistic transition systems, object-based systems, formal power series, infinite data structures, various other types of automata, and dynamical systems. See [Rut96, JR97] and the references therein for more information on coalgebra, and see [JMRR98, JR99] for many recent developments in coalgebra.

Sections 5 through 7 repeat and extend parts of [Rut98]. There the use of a-derivatives in coinductive definition and proof principles for (ordinary) languages $L \subseteq A^*$ was introduced, again following the general coalgebraic approach but also inspired by Brzozowski's paper [Brz64], where a-derivatives seem to appear first, and Conway's book [Con71]. Although [Rut98] briefly mentions partial automata, the present characterization of a final partial automaton in terms of partial languages in Section 5 is new, as is the coinductive definition of the supervised product.

Our main references on the supervisory control of discrete event systems have been Ramadge and Wonham's papers [RW89] and [WR87], and Wonham's lecture notes [Won99]. The notion of controllable sublanguage is taken from there, and so is the problem of the computation of supremal controllable sublanguages. Some of the examples of the present paper have been taken from these references as well. For an easy-going introduction to discrete event systems, containing many examples, see also [CL99].

The notions of control relation and of partial bisimulation in Section 8 are to the best of our knowledge new, as well as their use in the coinductive proof principle for showing controllability. (A word on the terminology: the name 'partial' bisimulation has been used before, at various places, and with different meanings.) In [WR87], the existence of the supremal controllable sublanguage K of a given language L and satisfying another language E is proved for classical (non-blocking) languages K, L, and E. Here this is generalized to partial (possibly blocking) languages. The algorithm for the computation of K, in [WR87, Section 6], works for classical regular languages L and E with $E \leq L$ and L prefix-closed. The present algorithm in Section 9 applies to arbitrary partial languages, and the algorithm in Section 10 works for all partial languages that are regular. Both algorithms are variations on Proposition 3.2 in Section 3, which at its turn is a variation on a similar such fixed-point characterization for bisimilarity on non-deterministic processes [Mil89]. In Wonham's lecture notes, there is an algorithm for the transformation of a finite automaton into an output-control-consistent one [Won99, Section 5.7], with estimated computational complexity $O(nm(n+m))$ (where n is the number of states and m the number of transitions). The coinductive algorithm presented in Section 11 is of order $O(n+m)$.

In [BL98, Thm 3.1], the connection between (Park and Milner's) bisimilarity and controllability is proved for the special case of (finite) partial Moore automata, in which all states are accepting, and for which the (classical) language accepted by the first is a subset of the second. This observation is retrieved here as a special instance of Theorem 8.1, which is formulated using the more general notion of control relation. A more important difference is our use of control relations in coinductive proofs of the controllability of one language with respect to another, as in Examples 8.2. Another contribution of [BL98] is the use of a partition algorithm for the computation of the bisimilarity relation from [Fer90], for solving the supervisory control problem. Their algorithm for the non-blocking case consists of a repeated alternation of an application of this partition algorithm with a trimming step. In contrast, our solution for the same problem (in Section 10), is best characterized as a variation on the original partition algorithm, moving from ordinary bisimilarity to partial bisimilarity. The proof of the correctness of our construction is particularly

simple, because of the use of coinduction. The question whether it will give rise to more efficient algorithms, has not been addressed here.

The general definition and theory of coalgebra is essentially categorical, though it has for our present purposes not been necessary to discuss any category theory at all. We have simply focussed on one concrete category of coalgebras, namely the family of partial Moore automata. Of earlier categorical approaches to control theory, we should mention the work by Arbib and Manes, notably [AM74], in which the authors give a categorical account of the duality between reachability (a simple instance of controllability) and observability. In [AM74], coalgebras are not mentioned but appear implicitly. In some of their later work (notably in [MA86], final coalgebras (called terminal there) do occur explicitly, and are used as models for the behaviour of (total) Moore automata. A crucial difference with recent uses of coalgebras, and with the present paper, is the absence in their work of a logical proof principle for reasoning about final coalgebras. It was not until the afore mentioned introduction of bisimulation into the world of coalgebra by Aczel and Mendler, in the late eighties, that a general coinduction proof principle could be formulated at all. And it is precisely the use of coinductive definitions and proofs that, in our view, makes coalgebra relevant for control theory.

It should be clear, however, that we have only made some very first and preliminary steps regarding the use of coalgebraic and coinductive techniques for the control of discrete event systems. From the coalgebraic perspective, these systems constitute yet another class of examples of interesting coalgebras, which pose new questions and give rise to new variations (such as the notion of partial bisimulation) on standard coalgebraic repertoire. At the same time, the use of coalgebra and coinduction constitutes a new technique that extends the more common algebraic approach to (automata theory and) the control of discrete event systems. In conclusion, we mention one example of a topic for further research. Both the problem of supervisory control with partial observations, and the problem of hierarchical supervision, involve the abstraction from (unobservable or low-level) events. It seems natural to extend the present coalgebraic approach with the use of (variations of) the notion of *weak* bisimulation relations (also called observational equivalences), which have been introduced into the world of communication processes to model the abstraction from so-called *internal* or *silent* moves (cf. [Mil89]).

Acknowledgements

I am grateful to Jan van Schuppen for introducing me to the subject of controllability, for discussions and explanations, and for the many references he has pointed out to me. Many thanks are also due to Jaco de Bakker, Alexandru Baltag, Bart Jacobs, John Thistle, Murray Wonham, and the members of the ACG Colloquium, for pointers to the literature, discussions, and comments.

13 Appendix: coinductive definitions

The unique existence is proved of the operators defined by the behavioural specifications in Section 6. The key ingredient in the proof is the fact that $\mathcal{L}$ is a final automaton. Let for any automaton $(S, \langle o, t\rangle)$ a set $\mathcal{E}_S$ of *expressions* be given by the following syntax:

$$E ::= \underline{s} \mid E + F \mid EF \mid E^* \mid (E \parallel_I F) \mid (E /_I F)$$

where for every state s in S, a *symbol* $\underline{s}$ is included in $\mathcal{E}$, and where $I \subseteq A$. The set $\mathcal{E}_S$ is next supplied with an automaton structure $(\mathcal{E}_S, \langle o_\mathcal{E}, t_\mathcal{E}\rangle)$. The functions $o_\mathcal{E} : \mathcal{E}_S \to 2$ and $t_\mathcal{E} : \mathcal{E}_S \to (1 + \mathcal{E}_S)^A$ are defined by induction on the structure of expressions, using the automaton structure of S for the symbols $\underline{s}$, and following the structure of the behavioural specifications of the operators in Section 6:

$$o_\mathcal{E}(\underline{s}) = o_\mathcal{L}(s), \quad o_\mathcal{E}(E + F) = \max\{o_\mathcal{E}(E), o_\mathcal{E}(F)\}, \quad o_\mathcal{E}(E^*) = 1,$$

$$o_\mathcal{E}(EF) = o_\mathcal{E}(E \parallel_I F) = \min\{o_\mathcal{E}(E), o_\mathcal{E}(F)\}, \quad o_\mathcal{E}(E /_I F) = o_\mathcal{E}(F)$$

Writing E_a for $t_{\mathcal{E}}(E)(a)$, the function $t_{\mathcal{E}} : \mathcal{E} \to (1 + \mathcal{E})^A$ is given by the following clauses:

$$(\underline{s})_a = \underline{s_a}, \;\; (E + F)_a = E_a + F_a, \;\; (E^*)_a = E_a\, E^*$$

$$(EF)_a = \begin{cases} E_a\, F & \text{if } o_{\mathcal{E}}(E) = 0 \\ E_a\, F + F_a & \text{if } o_{\mathcal{E}}(E) = 1 \end{cases}$$

$$(E \parallel_I F)_a = \begin{cases} (E_a \parallel_I F) + (E \parallel_I F_a) & \text{if } a \notin I \\ E_a \parallel_I F_a & \text{if } a \in I \end{cases}$$

$$(E /_I F)_a = \begin{cases} E_a /_I F_a & \text{if } E \xrightarrow{a} \text{ and } F \xrightarrow{a} \\ \underline{0} /_I F_a & \text{if } E \not\xrightarrow{a} \text{ and } F \xrightarrow{a} \text{ and } a \in I \\ \Uparrow & \text{otherwise} \end{cases}$$

Here $(\underline{s})_a$ is defined iff s_a is defined in S. Furthermore the same conventions regarding undefinedness apply that were mentioned in Section 6. The automaton $(\mathcal{E}_S, \langle o_{\mathcal{E}}, t_{\mathcal{E}} \rangle)$ contains for states s and t in S expressions like, for instance, $\underline{s} \parallel_I \underline{t}$, whose behaviour (acceptance and transitions) has been defined precisely according to the behavioural specification of this operator in Section 6. The automaton $\mathcal{E}_{\mathcal{S}}$ is well-behaved in the sense that bisimilarity is a congruence relation with respect to the operators: e.g., if $E \sim E'$ and $F \sim F'$ then $EF \sim E'F'$, and similarly for the other operators. This can be easily proved and is ultimately due to the format of the behavioural specifications we have been using in Section 6.

Next we can define the operators on partial languages that we are after, by choosing for S the automaton of languages $\mathcal{L}$. Writing $\mathcal{E}$ for $\mathcal{E}_{\mathcal{L}}$, there exists a unique homomorphism of automata: $l : \mathcal{E} \to \mathcal{L}$, because $\mathcal{L}$ is a final automaton (Theorem 5.1(2)), which assigns to each expression E the language $l(E)$ it represents. It can be used to define the operators on $\mathcal{L}$ that we are looking for:

$$K + L = l(\underline{K} + \underline{L}), \;\; KL = l(\underline{K}\,\underline{L}), \;\; K^* = l(\underline{K^*}),$$

$$K \parallel_I L = l(\underline{K} \parallel_I \underline{L}), \;\; K /_I L = l(\underline{K} /_I \underline{L})$$

One can show that $l(\underline{K}) = K$ and that l is compositional, e.g., $l(EF) = l(E)l(F)$, using the principle of coinduction (Theorem 5.1(1)) and the afore mentioned fact that bisimilarity on $\mathcal{E}$ is a congruence relation. For instance, the first statement follows by coinduction from the fact that $\{\langle l(\underline{K}),\, K\rangle \mid K \in \mathcal{L}\}$ is a bisimulation relation on $\mathcal{L}$. One can now readily prove that the operators we have defined are solutions of their defining equations, using the fact that l is a compositional homomorphism. Uniqueness of these solutions follows from the uniqueness of l. □

14 Appendix: Nerode equivalence

We briefly mention the connection between the automaton $\mathcal{L}$ of languages (Section 5) and *Nerode equivalence*, which is used in automata theory for the construction of minimal realisations. Nerode equivalence plays no role in the rest of our story.

For a language $L = (L^1, L^2)$ in $\mathcal{L}$, an automaton $(S_L, \langle o, t\rangle)$ is defined as follows. Let $S_L = L^2$ and define $o : S_L \to 2$ and $t : S_L \to (1 + S_L)^A$, for $w \in S_L$, by $o(w) = 1$ iff $w \in L^1$, and by $t(w)(a) = wa$ if $wa \in L^2$, and $t(w)(a)$ is undefined, otherwise. The unique homomorphism $l : S_L \to \mathcal{L}$ maps a word w to the w-derivative of L: $l(w) = L_w$. This follows by coinduction (Theorem 5.1(1)) from the fact that $\{\langle l(w), L_w\rangle \mid w \in L^2\}$ is a bisimulation relation on $\mathcal{L}$. Note that as a consequence, the function l identifies two words v and w in L^2 iff for all $u \in A^*$,

$$(vu \in L^1 \iff wu \in L^1) \text{ and } (vu \in L^2 \iff wu \in L^2)$$

In the case of a classical language $L = (L, A^*)$, the second condition is vacuously true because $L^2 = A^*$. What remains is precisely the definition of Nerode equivalence $\equiv_L$ with respect to L: for words v and w in A^*,

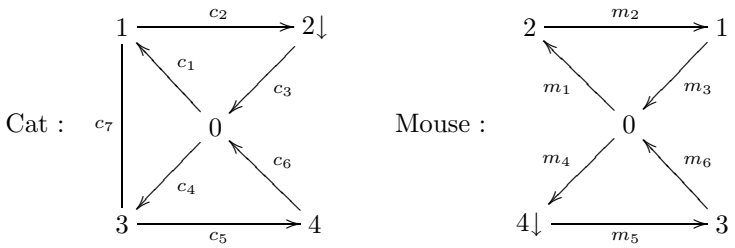
$$v \equiv_L w \text{ iff } (\forall\, u \in A^* :\; vu \in L \iff wu \in L)$$

Thus we have sofar proved that $l(v) = l(w)$ iff $v \equiv_L w$. Since ε generates all of the automaton S_L: $\langle\varepsilon\rangle = A^*$, and since $l(\varepsilon) = L_\varepsilon = L$ and $l(\langle\varepsilon\rangle) \cong \langle l(\varepsilon)\rangle)$ (because l is a homomorphism), we have obtained a characterization of the subautomaton $\langle L\rangle \subseteq \mathcal{L}$ generated by L as the quotient of A^* with respect to Nerode equivalence:

$$(A^*/\equiv_L) \cong l(S_L) = l(\langle\varepsilon\rangle) \cong \langle l(\varepsilon)\rangle = \langle L\rangle$$

15 Appendix: cat and mouse

We illustrate the construction of Section 10 with Wonham and Ramadge's charming example of a cat and a mouse placed in a maze ([RW89]). The maze consists of five rooms numbered 0 through 4, and both the cat and the mouse can move from one room to another according to the transitions in the automata below:



A transition of the form $i \xrightarrow{c_k} j$ means that the cat can move from room i to room j, using doorway c_k, and similarly for the mouse. Different labels for the cat and the mouse are used to indicate that each doorway is for the exclusive use of the cat (c labels) or the mouse (m labels). And all doorways can be traversed in one direction only (as indicated by the transition arrow), but for cat doorway c_7, which may be used by the cat to move both from room 1 to room 3 and vice versa. The home basis for the cat is room number 2 and for the mouse it is room number 4, which are at the same time the only accepting states in the two respective automata.

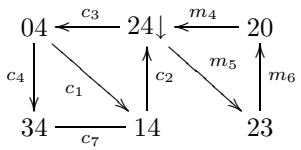
The unrestrained behaviour of cat and mouse together is represented by the shuffle product of both automata, which in our setting can be defined by the subautomaton of $\mathcal{L}$ generated by $l_c(2) \parallel l_m(4)$, where $l_c : \text{Cat} \to \mathcal{L}$ and $l_m : \text{Mouse} \to \mathcal{L}$ (by finality of $\mathcal{L}$). This automaton $T = \langle l_c(2) \parallel l_m(4)\rangle$ is readily seen to consists of 25 states, which we shall number as ij, for i and j between 0 and 4, and with the obvious transitions, such as $02 \xrightarrow{c_1} 12$. The designated state for T is $t = 24$, in which both cat and mouse are in their respective home bases; it is at the same time the only accepting state of T.

Next the free behaviour of cat and mouse should be restrained in such a way that they will never be in the same room together. Moreover, the resulting behaviour should be non-blocking, meaning that from any reachable state, the accepting state 24 should be reachable. This models the requirement that both the cat and the mouse can at all times return to their home bases again. It is assumed, furthermore, that the behaviour of cat and mouse can be constrained by closing at certain moments in time some doorways, and that all doorways can be opened and closed, but for the afore mentioned doorway c_7, which is always open. So let us put $A = C + U$ with $C = \{c_k \mid 0 \le k \le 6\} \cup \{m_k \mid 0 \le k \le 6\}$ and $U = \{c_7\}$. Let the automaton S be defined by $S = T - \{ii \mid 0 \le i \le 4\}$, with transitions as in T as far as they do not involve any of the forbidden states, and with designated state $s = 24$. Now the problem has been formalized in such a way that the construction described in Section 10 can be applied. The relation $R \subseteq S \times T$ used there looks like $R = \{\langle ij, ij\rangle \mid 0 \le i, j \le 4 \text{ and } i \ne j\}$. We find

$$\begin{aligned}
\Psi(R) &= R - \{\langle 13, 13\rangle, \langle 31, 31\rangle\} \\
\Psi^2(R) &= \Psi(R) - \{\langle 01, 01\rangle, \langle 03, 03\rangle\} \\
\Psi^n(R) &= \Psi^2(R) \text{ for all } n \ge 2
\end{aligned}$$

Two elements are removed from R because 13 and 31 can take a c_7-step in T but not in S. And two elements are removed from $\Psi(R)$ because they are not non-blocking: neither from $\langle 01, 01\rangle$

nor from $\langle 03, 03\rangle$ the accepting state $\langle 24, 24\rangle$ can be reached. After this, the situation stabilizes, leaving us with an automaton $\hat{R}$ with 16 states. Looking at the subautomaton in $\hat{R}$ generated by our designated state $\langle s, t\rangle = \langle 24, 24\rangle$, we find (writing ij rather than $\langle ij, ij\rangle$):



References

[Acz88] P. Aczel. *Non-well-founded sets*. Number 14 in CSLI Lecture Notes. Center for the Study of Languages and Information, Stanford, 1988.

[AM74] M.A. Arbib and E.G. Manes. Foundations of system theory: decomposable systems. *Automatica*, 10:285–302, 1974.

[AM89] P. Aczel and N. Mendler. A final coalgebra theorem. In D.H. Pitt, D.E. Ryeheard, P. Dybjer, A. M. Pitts, and A. Poigne, editors, *Proceedings category theory and computer science*, Lecture Notes in Computer Science, pages 357–365, 1989.

[BL98] G. Barrett and S. Lafortune. Bisimulation, the supervisory control problem and strong model matching for finite state machines. *Discrete Event Dynamic Systems*, 8:377–429, 1998.

[Bre98] F. van Breugel. Terminal metric spaces of finitely branching and image finite linear processes. *Theoretical Computer Science*, 202(1-2):223–230, 1998.

[Brz64] J.A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[BV96] J.W. de Bakker and E. de Vink. *Control Flow Semantics*. Foundations of Computing Series. The MIT Press, 1996.

[CL99] C.G. Cassandra and S. Lafortune. *Introduction to discrete event systems*. Kluwer Academic Publishers, 1999.

[Con71] J.H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.

[dB91] J.W. de Bakker. Comparative semantics for flow of control in logic programming without logic. *Information and Computation*, 94(2):123–179, 1991.

[Fer90] J. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13:219–236, 1990.

[JMRR98] B. Jacobs, L. Moss, H. Reichel, and J.J.M.M. Rutten, editors. *Proceedings of the first international workshop on Coalgebraic Methods in Computer Science (CMCS '98)*, volume 11 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B.V., 1998. Available at URL: www.elsevier.nl/locate/entcs.

[JR97] Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, 62:222–259, 1997. Available at URL: www.cwi.nl/ janr.

[JR99] B. Jacobs and J.J.M.M. Rutten, editors. *Proceedings of the second international workshop on Coalgebraic Methods in Computer Science (CMCS '99)*, volume 19 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B.V., 1999. Available at URL: www.elsevier.nl/locate/entcs.

[MA86] E.G. Manes and M.A. Arbib. *Algebraic approaches to program semantics*. Texts and monographs in computer science. Springer-Verlag, 1986.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall International Series in Computer Science. Prentice Hall International, New York, 1989.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings 5th GI conference*, volume 104 of *Lecture Notes in Computer Science*, pages 15–32. Springer-Verlag, 1981.

[Rut96] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. Report CS-R9652, CWI, 1996. Available at URL: www.cwi.nl. To appear in Theoretical Computer Science.

[Rut98] J.J.M.M. Rutten. Automata and coinduction (an exercise in coalgebra). Report SEN-R9803, CWI, 1998. Available at URL: www.cwi.nl. Also in the proceedings of CONCUR '98, LNCS 1466, 1998, pp. 194–218.

[Rut99] J.J.M.M. Rutten. Automata, power series, and coinduction: taking input derivatives seriously (extended abstract). Report SEN-R9901, CWI, 1999. Available at URL: www.cwi.nl. Also in the proceedings of ICALP '99, LNCS 1644, 1999, pp. 645–654.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proc. IEEE*, 77:81–98, 1989.

[Won99] W.M. Wonham. Notes on control of discrete-event systems. These notes are available at http://www.control.utoronto.ca/people/profs/wonham/wonham.html. University of Toronto. 310 pp., 1999.

[WR87] W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3):637–659, 1987.