

The Crystallizing Substochastic Sequential Machine Extractor – CrySSMEx*

Henrik Jacobsson
henrik.jacobsson@gmail.com

Abstract

This article presents an algorithm, **CrySSMEx**, for extracting minimal finite state machine descriptions of dynamic systems such as recurrent neural networks. Unlike previous algorithms, **CrySSMEx** is parameter free, deterministic, and it efficiently generates a series of increasingly refined models. A novel finite stochastic model of dynamic systems and a novel vector quantization function have been developed to take into account the state space dynamics of the system. The experiments show that (a) extraction from systems that can be described as regular grammars is trivial, (b) extraction from high-dimensional systems is feasible and (c) extraction of approximative models from chaotic systems is possible. The results are promising, but an analysis of shortcomings suggests some possible further improvements. Some largely overlooked connections, of the field of rule extraction from recurrent neural networks, to other fields are also identified.

1 Introduction

Computer simulation is today a widely used method for testing theories. The domains under study through simulation are within fields like physics, chemistry, engineering, economics, ecology, molecular biology, neuroscience, etc. The simulated systems are often very complicated to analyse; partly because of their intrinsic complexity, and partly because of the massive amount of data stemming from numerous instances of models and automated repetitions of simulations (e.g. for multiple alternative scenarios). Many analytical and numerical analysis tools, generic as well as domain specific ones, have been devised to understand simulated models better. For recurrent neural networks (RNNs) (e.g. Kremer, 2001; Kolen & Kremer, 2001) it has been natural to analyse them as finite state machines (FSMs), partly due to their common source of origin (McCulloch & Pitts, 1943), and partly due to the fact that they have often been trained to perform regular language recognition (e.g. Cleeremans, McClelland & Servan-Schreiber, 1989; Christiansen & Chater, 1999). This has led to the development of algorithms for transforming one model into another, i.e. from RNNs into FSMs. Similar approximative transformations are central issues in numerous fields of which some will be brought up and related to explicitly in the end of this article.

Unlike previous approaches, the novel algorithm presented here, **CrySSMEx**¹ (Crystallizing Substochastic Sequential Machine Extractor), is parameter free, handles missing data,

*This is a draft version. The final version of this article will be published in *Neural Computation*, Vol. 18, Issue 9, published by the MIT Press (<http://mitpress.mit.edu/NECO>).

¹**CrySSMEx** is to be pronounced somewhat like “Christmas”. Moreover, when the following text refers to the implementation of the algorithm, this can be acquired as an open source distribution which is available on <http://cryssmex.sourceforge.net>. The latest version and associated information can be found on this homepage.

generates approximative rules if the underlying system is chaotic, and returns results at “any-time” (Craven & Shavlik, 1999), i.e. coarse models are initially created and then iteratively refined.

CrySSME_x is partially based on conclusions drawn from a recent review of earlier algorithms (Jacobsson, 2005) for rule extraction from recurrent neural networks (RNN-RE). The underlying idea behind CrySSME_x is to observe the state and output of a dynamic system, quantize the state space, and refine the quantization of the state space such that the resulting machine typically is minimal, deterministic, and equivalent to the RNN.

The aim of the algorithm will now be presented together with a brief discussion and overview of what distinguishes CrySSME_x from earlier RNN-RE algorithms.

1.1 Aim

CrySSME_x differs from many earlier approaches in that it strives for fidelity rather than accuracy of the rules. Fidelity is the degree to which the rules mimic the network, whereas accuracy is related to how well the rules generalize to unseen examples (Andrews, Diederich & Tickle, 1995). When fidelity is the goal and the underlying network makes mistakes, the machine extracted from the network should also replicate those mistakes. Some earlier approaches have been focused on fidelity too (e.g. Vahed & Omlin, 2004), but most work has had accuracy as the prime goal for the rules (e.g. Giles, Miller, Chen, Chen & Sun, 1992; Zeng, Goodman & Smyth, 1993), which makes sense if the network is used as an intermediate step for acquiring symbolic knowledge from data, e.g. for grammar induction. This approach has in some cases been very successful when the extracted rules were equivalent to the symbolic data generator (e.g. Giles, Miller, Chen, Chen & Sun, 1992; Giles, Miller, Chen, Sun et al., 1992).

One reason to strive for fidelity is that it makes the rules useful for analysing erroneous RNNs. One could compare an erroneous RNN with a sick patient and an RNN-RE algorithm with an instrument of a doctor diagnosing the patient. The doctor would not gain much from an accuracy-seeking instrument describing what the patient should be like if completely healthy, which is basically what accuracy-optimizing methods strive for. Instead, the analysis tool should generate an analysis that reflects the *actual* condition of the patient.

Another difference between accuracy and fidelity is that the latter does not presuppose the existence of any task in which errors can be defined. Instead, the quality of extraction is measured on how well the extracted model mimics the underlying system. This allows for the analysis of simulated systems other than just RNNs. Therefore, in this article, the extraction of rules from RNNs will be treated as an interesting special case of extraction from a broad range of dynamic systems (Section 2.1).

1.2 What is new in CrySSME_x?

The three main criteria in a recent taxonomy of RNN-RE methods (Jacobsson, 2005) were: (1) the means of state observation, (2) the type of rules extracted, and (3) the state space quantization method.

The observation of states in CrySSME_x, as in many other approaches (e.g. Watrous & Kuhn, 1992; Manolios & Fanelli, 1994; Tiño & Vojtek, 1998; Tiño & Kőteles, 1999; Tiño, Čerňanský & Beňušková, 2004), is solely based on sampling the system as it behaves in its domain. The novel components of CrySSME_x are the rule type (Section 2.2), and the quantization method (Section 3). But what really distinguishes CrySSME_x from all earlier approaches, is the integration of the four basic elements found in previous approaches (Jacobsson 2005, p. 1230):

- quantization of state,
- observation of the underlying system,

- rule construction and
- rule minimization.

These four subprocedures have typically been quite separable in RNN-RE algorithms. In earlier approaches, the quantization of the state space has been done by traditional clustering techniques with no sensitivity to, nor any integration with, the dynamics of the RNN. Also, the minimization of the rules (when conducted at all) was just a postprocessing of the rules. In **CrySSMEx**, all four constituents are tightly integrated into one system resulting in an empirical loop of model refinement through model based data selection (cf. Section 4).

1.3 Overview

This article is structured such that the main loop of the algorithm (in Section 4.2) could be understood, at an abstract level, without knowing all the details of the constituents. Therefore readers are recommended to have a brief look at Algorithm 3 (on page 20) of Section 4.2, the point of convergence of this paper, before continuing to read the article. To further aid readers, a list of important abbreviations can be found in Table 1 of Appendix B.

The article is otherwise organized as follows: in Section 2 the specific class of dynamic systems that are analysable with **CrySSMEx** is defined together with a discrete stochastic model of these systems. In Section 3 a novel vector quantizer is described. And, as mentioned, Section 4 connects the constituents of **CrySSMEx** into one coherent algorithm. The remaining sections contain experiments, discussion and conclusions.

2 Modelling dynamic systems

This section will introduce a class of dynamic systems (Section 2.1), a finite stochastic model of these systems (Section 2.2) and a means of transforming the dynamic system into the stochastic model through system observation (Section 2.2.3). The translation process of the system into a model is refined by other parts of **CrySSMEx** (Sections 3–4) so that more precise translations can be made.

2.1 Situated Discrete Time Dynamic Systems

The target domain for **CrySSMEx** is a general class of dynamic systems which includes RNNs. Therefore, only properties of RNNs that are of importance for rule extraction are included. Other properties typically associated with neural networks, such as weights, activation functions and learning, are simply omitted.

The resulting class of systems will here be referred to as *situated* discrete time dynamic system, incorporating state, input, output and dynamics of the system. The system is situated in the sense that it has a defined interface with a domain with which it interacts. After this point in the article, the extraction of rules from such dynamic systems rather than from only RNNs will be considered, but the underlying problems are precisely the same.

2.1.1 Definition

Definition 1 A *situated discrete time dynamic system* (SDTDS), is a quadruple $\langle S, I, O, \gamma \rangle$ where $S \subseteq \mathbb{R}^{n_s}$ is a *set of state vectors*, $I \subseteq \mathbb{R}^{n_i}$ is a *set of input vectors*, $O \subseteq \mathbb{R}^{n_o}$ is a *set of output vectors*, $\gamma : S \times I \rightarrow S \times O$ is the *transition function*, and $n_s, n_i, n_o \in \mathbb{N}$ are the *dimensionalities* of the state, input and output spaces respectively.

Interpretation: If the system, at time t , occupies a state $\vec{s}(t)$ and is fed an input $\vec{i}(t)$, then the resulting next state and produced output is determined by $[\vec{s}(t+1), \vec{o}(t+1)] = \gamma(\vec{s}(t), \vec{i}(t))$. The current and initial state of the system are not included in the SDTDS model since it is something imposed on the system (the SDTDS specifies the framework and behaviour for

any arbitrary initial state just as a function specifies the image of any arbitrary member of the domain of the function). To simplify descriptions, the transition function, γ , can be subdivided into two functions $\gamma_s : S \times I \rightarrow S$ and $\gamma_o : S \times I \rightarrow O$.

It should be noted that the functional dependencies are those of a Mealy system rather than a Moore system in that the output is determined by state *and* input rather than a function of state alone (Hopcroft & Ullman, 1979). The reason for this choice is that a Mealy model can subsume a Moore model but not necessarily vice versa².

In its current implementation, **CrySSMEx** also requires the set of input vectors to be finite, which for example is the case for any symbol processing RNN. This restriction is not included in the definition since it is more a matter of what is put into the SDTDS, rather than a restriction of the system itself. Other than that, there are no theoretical restrictions on the SDTDS as defined above for **CrySSMEx** to analyse it.

There are also some other “implicit requirements”, made by a rule extraction algorithm of the underlying SDTDS, that cause some systems of general interest not to fall under the definition above (Jacobsson, 2005). For example, the state, input and output must be distinctly separable as well as fully and unintrusively observable. Moreover, γ must be a noise-free function, i.e. the observed system is assumed to be completely deterministic.

2.1.2 Collection of data from an SDTDS

An RNN-RE algorithm should transform an RNN into a discrete model mimicking the RNN to a satisfactory degree. To do this, a *compositional* approach has typically been adopted where data is gathered from the internal activations of the RNN and then a model is built from this (Tickle, Andrews, Golea & Diederich, 1998). Within the RNN-RE field, two sub-types of the compositional approach exist, one where the RNN-RE algorithm interacts directly with the RNN while performing a breadth first search, and another where the data is collected from the RNN during interaction with the domain in which it was trained (Jacobsson, 2005). In **CrySSMEx**, the latter is chosen for three reasons: (1) the data (and hence the extracted model) will only contain aspects of the RNN relevant for the domain, (2) it is far more efficient since, in effect, the domain is used as a heuristic when searching among all the possible models that describe the behaviour of the system (Jacobsson & Ziemke, 2003b), and (3) it is possible to do the extraction off-line, i.e. pregenerated data can be used in **CrySSMEx** since no direct interaction between extractor and underlying system is needed.

When the SDTDS is set to hold a certain initial state and is then fed a sequence of input vectors from a domain it will generate a sequence of states and outputs as a result. This domain interaction is the basis for the data collection and the result is recorded as a sequence of *transition events*.

Definition 2 An *SDTDS transition event* at a time t , $\omega(t)$, is a quadruple $\langle \vec{s}(t), \vec{i}(t), \vec{o}(t+1), \vec{s}(t+1) \rangle \in \mathbb{R}^{n_s} \times \mathbb{R}^{n_i} \times \mathbb{R}^{n_o} \times \mathbb{R}^{n_s}$ where $\vec{s}(t+1)$ is the state vector reached after the SDTDS received input $\vec{i}(t)$ while occupying state $\vec{s}(t)$, and $\vec{o}(t+1)$ is the output generated in the transition.

Definition 3 A *transition event set*, Ω , consists of selected transition events recorded from the SDTDS with a given set of input sequences.

The reason that Ω is defined to consist of *selected* events is that it is quite possible that some events are not wanted in the model, e.g. when the user has made an explicit reset of the state with no wish to model the transition caused by this. The user may also want to let the system “settle in” before starting data collection.

²A Moore model, and a Moore machine extraction version of **CrySSMEx** has also been implemented, but is not presented here since it involves small changes in many different parts of the descriptions.

2.1.3 Building a stochastic dynamic model from a quantized SDTDS

The most essential part of CrySSMEx, and all earlier RNN-RE algorithms, is the quantization of the state space. The set of possible states in the state space of the SDTDS is uncountable and must be transformed to a finite domain to make the extraction of a finite machine possible.

Definition 4 A *quantizer* $\Lambda : \mathbb{R}^n \rightarrow \{1, 2, \dots, m\}$ is a function that separates an n -dimensional real space into m uniquely labelled disjoint subspaces. The maximum number of subspaces, m (i.e. the cardinality of the codomain of function Λ), will, for pragmatic reasons, be denoted $|\Lambda|$.

Although not explicitly stated in most RNN-RE papers, all three spaces of RNNs (input, state and output) are actually labelled using some form of quantization function. The quantization of the state space is of course of most central concern, but also the input and output need to be labelled into a finite set of symbols to produce the extracted finite machine. The state, input and output quantizers will be denoted Λ_s , Λ_i and Λ_o respectively.

The SDTDS is in itself of course capable of reacting according to any of the possible input vectors (since the SDTDS definition includes the whole vector spaces in the domains of the transition function), but in its current implementation CrySSMEx requires the input domain to be finite (and Λ_i must be invertible).

The frequencies of quantized transitions in the transition event set, Ω , are transformed into a joint probability distribution that will later be used to build a dynamic model which mimics the SDTDS (Section 2.2.3):

Definition 5 A *stochastic dynamic model* of an SDTDS is a joint probability mass function induced from a transition event set Ω and quantizers Λ_o , Λ_i and Λ_s is defined as a function $p_\Omega : [1, |\Lambda_s|] \times [1, |\Lambda_i|] \times [1, |\Lambda_o|] \times [1, |\Lambda_s|] \rightarrow [0, 1]$ where $p_\Omega(i, k, l, j)$ denotes the probability, that if one picks a random transition event from Ω , it would be a transition from a state enumerated i by Λ_s , over an input vector enumerated k by Λ_i , which generated an output enumerated l by Λ_o , and a new state enumerated j by Λ_s ³.

2.2 Substochastic Sequential Machines

Stochastic machines have been extracted earlier (Tiño & Vojtek, 1998; Tiño & Köteles, 1999), but without modelling the output of the system explicitly. In CrySSMEx, however, the output of the system will be modelled as well.

The stochastic dynamic model (p_Ω in Definition 5) collected from the SDTDS in interaction with its domain gives us information about the estimated probabilities of the effect and outcome of transitions in the system as “viewed” through the quantizers. These probabilities are used to build a finite stochastic machine model of the SDTDS. This type of machine resembles *stochastic sequential machines* (Paz, 1971) or *probabilistic automata* (Rabin, 1963) but has some distinguishing features since there is a realistic possibility of model “incompleteness” due to a finite observed set of transition events. This is due to the fact that the sample of input sequences in Ω will not necessarily provide examples of *all* possible input symbols in *all* possible enumerations of the quantized SDTDS space. The choice here is to make a “closed world assumption”, and consequentially, only what is observed in Ω will be included in the model.

Missing data must therefore be handled when the model is built from Ω . This causes the probabilistic model to become a *substochastic sequential machine* (SSM) rather than the *stochastic sequential machines* of Paz (1971). As a consequence, this incompleteness of the model implies that probability can “leak” out from the state of the machine during parsing of input sequences, causing the probability distributions to become substochastic (see Appendix A). The details of what this entails will be clarified in the following sections.

³The awkward order of i , j , k , and l is due to other contexts of the variables of p_Ω later in this article.

First, however, some additional definitions and notational conventions will be introduced, then the full SSM definition will be given.

2.2.1 Notation of probability distributions as vectors

Sometimes a probability distribution is preferably denoted as a vector (cf. Paz, 1971). The probability mass function over a discrete stochastic variable X , is denoted $p(X = x_i)$, or $p(x_i)$ for short. $p(x_i)$ is interpreted as the probability of X having the value x_i . If we want to express this probability as a vector it is convenient to just write $p(x_i)$ as \vec{x}_i . The full vector, representing the full distribution over X is denoted \vec{x} , i.e. with no index. The vector and probability notation of distributions will be used interchangeably since they are more conveniently expressed as one or the other depending on context. Important types of substochastic vectors and operations on them are defined in Appendix A.

2.2.2 SSM definition

Definition 6 A substochastic sequential machine (SSM) is a quadruple $\langle Q, X, Y, \mathcal{P} = \{p(q_j, y_l | q_i, x_k)\} \rangle$ where Q is a finite set of state elements (SEs), X is a finite set of input symbols, Y is a finite set of output symbols, and \mathcal{P} is a finite set of conditional probabilities (cf. explanation of Equation 3) where $q_i, q_j \in Q$, $x_k \in X$ and $y_l \in Y$.

The terminology is here somewhat different from that of conventional finite state machines. The input and output domains of the SSM will still be considered alphabets of *symbols*, whereas the Q of the SSM will instead be denoted *state elements* or SEs to not confuse them with the state of the SDTDS. Also, the actual state of the SSM is more properly described as a (sub)stochastic distribution over these elements. The interpretation of $p(q_j, y_l | q_i, x_k)$ is that it is the probability of the machine entering the SE q_j and in this transition producing symbol y_l given that it occupied only SE q_i and was fed input symbol x_k . A more detailed description of the SSM interpretation is given in Section 2.2.4 where the use of an SSM as a parser of input symbols is described. But first, the construction of an SSM from a model of the SDTDS will be described.

2.2.3 Translation of an SDTDS into an SSM

It is quite straightforward to see the similarities of the SDTDS and the SSM (cf. definitions 1 and 6). The difference lies mainly in the discreteness of the input, state and output domains of the SSM versus the uncountable domains in the SDTDS. In practice, however, the SSM can be seen as a subclass of the set of SDTDSs since a substochastic SE distribution can be subsumed as an SDTDS state and correspondingly for input and output.

When transforming an SDTDS into an SSM-model, the uncountable domains S , I and O of the SDTDS are reduced to the finite domains, Q , X and Y . The SSM is created from a quantized SDTDS such that the domains of the SSM are isomorphic to the codomains of the respective quantizers. In other words, Q of the SSM is isomorphic to $[1, |\Lambda_s|]$ and correspondingly for the input and output symbols. In the following text, an SE denoted $q_i \in Q$ will correspond to the portion of the state space of the SDTDS enumerated i by the Λ_s -quantizer.

The joint probabilities of observed and quantized SDTDS transitions (p_Ω), are translated into joint probabilities of *SSM transitions* according to:

$$p_\Omega \left(\Lambda_s(\vec{s}(t)) = i, \Lambda_i(\vec{v}(t)) = k, \Lambda_o(\vec{o}(t+1)) = l, \Lambda_s(\vec{s}(t+1)) = j \right) = p(q_i, x_k, y_l, q_j) = \quad (1)$$

i.e. the joint probability of SSM transitions are defined such that they correspond to the observed frequency of transitions in the SDTDS. The conditional probability of the SSM,

$p(q_j, y_l | q_i, x_k)$, is calculated from the joint probability according to equations 2 and 3.

$$p(q_i, x_k) = \sum_{j=1}^{|Q|} \sum_{l=1}^{|Y|} p(q_i, x_k, y_l, q_j) \quad (2)$$

$$p(q_j, y_l | q_i, x_k) = \begin{cases} \frac{p(q_i, x_k, y_l, q_j)}{p(q_i, x_k)} & \text{if } p(q_i, x_k) > 0 \\ 0 & \text{if } p(q_i, x_k) = 0 \end{cases} \quad (3)$$

Although conceptually appealing, the distribution $\mathcal{P} = \{p(q_j, y_l | q_i, x_k)\}$, is perhaps a bit haphazardly termed *conditional probabilities* since a conditional probability $p(a|b)$ traditionally is undefined if $p(b) = 0$. But in the SSM we need these to be defined since there might actually be cases where there is *no* transition from a SE q_i over a specific symbol x_k , simply because there are no observations in Ω of any such event.

Definition 7 If $p(q_j, y_l | q_i, x_k) = 0$ for all $q_j \in Q$ and $y_l \in Y$, then the transition from q_i over input x_k will be referred to as a *dead* transition.

Definition 8 The procedure of transforming an SDTDS from Ω , through the stochastic dynamic model, p_Ω of Definition 5, into an SSM as defined above in equations 1–3 will in pseudo-code be denoted as $ssm = \text{create_machine}(\Omega, \Lambda_s, \Lambda_i, \Lambda_o)$ where Λ_s , Λ_i and Λ_o are the state, input and output quantizer respectively, and ssm the resulting SSM.

When an SSM is created with `create_machine`, the SDTDS from which Ω was sampled will be referred to as *the underlying system* of the SSM. Next, the exact calculations of state and output of the SSM will be described. The SSM processes input such that its distributions over Q and Y correspond to the *degree of belief* of the occupied state and output enumeration of the underlying system.

2.2.4 Parsing an input sequence using an SSM

Unlike a “standard” discrete Mealy machine where exactly one state is occupied at a time (Hopcroft & Ullman, 1979), the complete description of the state occupied by an SSM is the substochastic distribution over zero, one, or more SEs. Likewise, the transitions generate substochastic distributions of output symbols rather than individual symbols.

The exact calculations of distributions are as follows: Let $\vec{q}(t) = (\vec{q}_1(t), \vec{q}_2(t), \dots, \vec{q}_n(t))$ be a substochastic vector denoting the distribution over Q at time t and $x_k(t) \in X$ be the input symbol fed to the machine in that time step. The resulting distribution vector over Q , $\vec{q}(t+1)$, is calculated by⁴

$$\vec{q}(t+1) = \mathcal{P}_q(\vec{q}(t), x_k(t)) \quad (4)$$

where each element $\vec{q}_j(t+1)$ of $\vec{q}(t+1)$ (corresponding to a probability of a SE) is calculated by

$$\vec{q}_j(t+1) = \sum_{i=1}^{|Q|} \left(\vec{q}_i(t) \cdot \sum_{l=1}^{|Y|} p(q_j(t+1), y_l | q_i(t), x_k(t)) \right) \quad (5)$$

and concurrently, the distribution of output symbols $\vec{y}(t+1)$ over Y is generated in the transition by

$$\vec{y}(t+1) = \mathcal{P}_y(\vec{q}(t), x_k(t)) \quad (6)$$

⁴Note that this is a case where the notational choice of letting $p(q_i) = \vec{q}_i$ comes into play (cf. Section 2.2.1), i.e. it is implicit that $\vec{q}_i(t+1)$ and $p(q_i(t+1))$ refer to the probability $p(Q = q_i)$ at time $t+1$.

where each element $\bar{y}_l(t+1)$ of $\bar{y}(t+1)$ (corresponding to a probability of an output symbol) is calculated by

$$\bar{y}_l(t+1) = \sum_{i=1}^{|\mathcal{Q}|} \left(\bar{q}_i(t) \cdot \sum_{j=1}^{|\mathcal{Q}|} p(q_j, y_l(t+1) | q_i(t), x_k(t)) \right) \quad (7)$$

Note that if the transition from $q_i(t)$ over $x_k(t)$ is dead and $\bar{q}_i(t) > 0$, then the respective sum of the probabilities of distribution $\bar{q}_j(t+1)$ and $\bar{y}_l(t+1)$, will be less than 1. In such cases, distributions of the machine will become substochastic (cf. Appendix A).

Another possibility of parsing is to, when possible, divide the probabilities with the sum of the probabilities after each symbol. This mode of parsing will be referred to as *normalized parsing*.

$$\hat{\mathcal{P}}_*(\bar{q}(t), x_k(t)) = \text{normalize}(\mathcal{P}_*(\bar{q}(t), x_k(t))) \quad (8)$$

where the '*' is either q or y (**normalize** is defined in Appendix A).

One may argue that instead of the notion of substochastic probabilities and state “leaking” from the machine, it would be better to add an additional state element q_{dead} to which all dead transitions are then made (producing an additional “dead” output symbol, y_{dead}). This would work, and it would also, as far as I can judge, create a machine equivalent to the machines of Paz (1971). It would, however, wreck the otherwise complete semantic connection between underlying system and SSM since there would be no corresponding elements in S and Y of the SDTDS for q_{dead} and y_{dead} .

To illustrate the parsing of symbol sequences with SSMs, some examples will be explored in Section 2.2.7. But first some important types of SEs must be introduced. There are a number of properties of SSMs and SEs that can be used for a deeper analysis of the machines. In this article only the ones that are crucial for CrySSMEx will be mentioned: deterministic and equivalent SSM SEs.

2.2.5 SSM determinism

An SSM will always be deterministic in the sense that the state element and output symbol distributions are always deterministically calculated. Therefore the determinism of an SE is instead defined such that it reflects the degree to which the SSM determines the succeeding occupied state enumerations and output symbols of the underlying dynamic system. For this purpose entropy and, especially, conditional entropy (Cover & Thomas, 1990) are suitable (see Definition 23 in Appendix A).

A conditional entropy $H(Y|X = x)$ can be interpreted as the remaining uncertainty of variable Y given that variable X would be known to have the value x . Here, the conditional SSM-based entropy of the output given an SE q_i and input x_k in an SSM ssm will be denoted $H_{ssm}(Y|Q = q_i, X = x_k)$ and is defined by

$$H_{ssm}(Y|Q = q_i, X = x_k) = H(\mathcal{P}_y(\bar{q}, x_k)) \quad (9)$$

where \bar{q} is here the degenerate (see Appendix A) SE distribution vector with $\bar{q}_i = 1.0$. The conditional entropy of the SE given the previous SE and input symbol is likewise denoted $H_{ssm}(Q|Q = q_i, X = x_k)$ and is here defined by

$$H_{ssm}(Q|Q = q_i, X = x_k) = H(\mathcal{P}_q(\bar{q}, x_k)) \quad (10)$$

with \bar{q} degenerate as in equation 9.

The interpretation of the entropies in equations 9 and 10 are that given that distribution over Q is concentrated to only q_i and input then is x_k , they return the degree of uncertainty of the SSM regarding the succeeding output symbol and occupied state enumeration of the underlying SDTDS, respectively. This is an idealized interpretation due to the substochastic

nature of the model. The conditional entropy will be zero also when the SSM has another type of uncertainty when transition from q_i over x_k is dead.

Definition 9 An SE $q_i \in Q$ of an SSM ssm is *deterministic* iff $H_{ssm}(Y|Q = q_i, X = x_k) = 0$ and $H_{ssm}(Q|Q = q_i, X = x_k) = 0$ for $\forall x_k \in X$.

A deterministic SE has exactly zero or one outgoing transition for each input symbol.

Definition 10 An SSM is deterministic iff all SEs $q_i \in Q$ are deterministic.

If a machine is deterministic, and its initial SE distribution is degenerate, then all subsequent SE and output distributions will both be either degenerate or exhausted (cf. Appendix A). This definition of a deterministic machine differs somewhat from that of traditional deterministic finite automata (Hopcroft & Ullman, 1979), in which states (corresponding to the state elements of the SSM) must have transitions to exactly one state for all input symbols.

It is quite straightforward to see that a deterministic SSM, in which there are no dead transitions, is equivalent to the nonstochastic standard Mealy machines as defined in Hopcroft and Ullman (1979), if a degenerate distribution over Q is defined as initial state. Such a machine must always occupy only one SE at a time and generate one single output symbol at a time.

SSM determinism will be used as a termination criterion in CrySSMEx (see Algorithm 3). The conditional entropies will also be used as a basis for selection of the most informative state vectors of Ω in order to perform optimization of the SDTDS state quantizer (see Algorithm 2).

2.2.6 Equivalence and nonequivalence of SEs

The second important property of SEs is equivalence. In automata theory, two states q_i and q_j of a machine are equivalent if, and only if, the output of the automata would be the same for all possible future input sequences independently of which of the two possible states that are occupied initially. This can be tested quite efficiently in traditional nonstochastic automata (Hopcroft & Ullman, 1979), but for stochastic machines it is a bit more difficult. In fact, it is even impossible in general for substochastic machines since the model would not “know” what the outcome of dead transition would be in the underlying system. It would, for example, be impossible to determine what other state elements an SE with *no* outgoing transition is or is not equivalent with since the outcome of any possible future input sequence is undefined in the model. The only way to determine equivalence of such an SE, to other SEs, is to go back to the underlying SDTDS to record the missing transitions, and thereby make it part of the SSM model. Since this would break the closed world assumption, it is not considered.

It is, however, possible to determine that two SEs are *not* equivalent if they, in their outgoing transitions, share some input symbols and transitions over these lead to discrepancies in the future output of the SSM. So, what we will do is to provide an algorithm that returns true if and only if two SEs are *not decisively inequivalent* (NDI-equivalent for short)⁵. For example, an SE which has no outgoing transitions will be NDI-equivalent with *all* other SEs since there will be no decisive evidence of the opposite. Two SEs with no input symbols in common in their outgoing transitions will also always be NDI-equivalent.

To determine the NDI-equivalence of SEs q_i and q_j , the recursive function `NDI-equivalent`(`ssm`, \vec{u} , \vec{v} , \emptyset) (described in Algorithm 1) is called, where \vec{u} and \vec{v} are the corresponding degenerate SE distributions for q_i and q_j respectively (the need for the empty set is clarified in Algorithm 1), and the result is true or false depending on whether the SE distributions \vec{u} and \vec{v} are NDI-equivalent or not. The algorithm is highly recursive

⁵It is also possible to test if SEs are decisively equivalent as well, i.e. when all subsequent SEs have the same symbols for outgoing transitions. But preliminary studies have shown that more interesting results are achieved using NDI-equivalence simply because dead transitions are quite common.

and uses a “trick” based on the support sets (see Appendix A) of the SE distributions to avoid infinite recursions that otherwise could occur. If we allow ourselves to jump ahead to a later example, consider the testing of equivalence between SEs q_5 and q_6 in Figure 2 (on page 13). When starting in SE q_5 and the SSM is fed symbol b , the SE distribution is gradually approaching a pure SE q_6 , but it will never quite reach it. The algorithm would not stop if it were not for using reencounters of SE support sets as a termination criteria. Since there is just a finite set of possible support sets (2^Q), this guarantees that the algorithm will terminate.

NDI_equivalent(ssm, \vec{u}, \vec{v}, H)

Input: an SSM ssm , SE distributions \vec{u} and \vec{v} , and history of state support sets H .

Output: returns true if \vec{u} and \vec{v} are not decisively inequivalent given possible future input sequences.

```

begin
1 | if  $\exists x_k \in X : (\hat{\mathcal{P}}_y(\vec{u}, x_k) \neq \hat{\mathcal{P}}_y(\vec{v}, x_k) \wedge \text{sup}(\mathcal{P}_q(\vec{u}, x_k)) \neq \emptyset \wedge$ 
   |    $\text{sup}(\mathcal{P}_q(\vec{v}, x_k)) \neq \emptyset)$  then return false;
   |   /*i.e. the output must be the same for both SE distributions for all possible input
   |     symbols. */
2 | else if  $\vec{u} = \vec{v}$  then return true;
   |   /*i.e. if the distributions are identical, they are equivalent. */
3 | else if  $\langle \text{sup}(\vec{u}), \text{sup}(\vec{v}) \rangle \in H$  then return true;
   |   /*i.e. a loop has been encountered. Eventual inequivalence will be encountered in
   |     another branch of the recursion tree. */
4 | else
   |   /*If the equivalence/inequivalence cannot be asserted, then subsequent inputs
   |     must be tested. */
   |    $R := \text{true};$ 
   |    $k := 1;$ 
   |   while  $R = \text{true} \wedge x_k \in X$  do
   |     /*As long as no inequivalence has been shown ... */
   |      $\vec{u}' = \hat{\mathcal{P}}_q(\vec{u}, x_k);$ 
   |      $\vec{v}' = \hat{\mathcal{P}}_q(\vec{v}, x_k);$ 
5 |     if  $\text{sup}(\vec{u}') \neq \emptyset \wedge \text{sup}(\vec{v}') \neq \emptyset$  then
   |       /*... continue testing recursively. */
   |        $H' = H \cup \langle \text{sup}(\vec{u}'), \text{sup}(\vec{v}') \rangle;$ 
   |        $R := \text{NDI\_equivalent}(ssm, \vec{u}', \vec{v}', H');$ 
   |     end
   |      $k := k + 1;$ 
   |   end
   |   return  $R;$ 
end

```

Algorithm 1: The recursive function **NDI_equivalent**($ssm, \vec{u}, \vec{v}, \emptyset$) returns true if and only if there is no evidence that the future ssm output could differ depending on which of SE distributions \vec{u} or \vec{v} are occupied in the SSM. The if-statement on line 2 can actually be logically omitted, since line 3 will catch the equivalence in subsequent levels of recursion (line 4), but it makes the algorithm considerably more efficient in most realistic cases. The empty support set tests on lines 1&5 together with the normalized parsing ($\hat{\mathcal{P}}_q$ and $\hat{\mathcal{P}}_y$) cause the algorithm to return true when assessment of inequivalence cannot be performed.

What is lacking at this moment, is a formal proof that **NDI_equivalent** works as intended for all possible SSMs under all possible conditions. A formal proof for a method for equivalence testing of states in a *stochastic* sequential machines, however, does exist

(Paz, 1971). In that proof, strong similarities with this algorithm do occur, but a formal 1:1 connection is yet to be done. For now, the experiments of Section 5 will be the only indication that the algorithm as a whole works for the presented cases. In addition to these experiments, the algorithm has been successfully tested in a number of hand-made SSMs, with properties making them interesting to analyse with respect to SE equivalence, e.g. the SSM of Figure 2.

For three SEs q_i , q_j and q_k of an SSM it may very well hold that q_i and q_j are NDI-equivalent and likewise for q_j and q_k while q_i and q_k are not. In other words, the relation is not transitive. It is required, by other parts of **CrySSMEx** (see Algorithm 3) that states can be grouped into disjoint equivalence sets which is not possible if the equivalence relation is not transitive (and symmetric and reflexive as well).

Definition 11 Let $\pi(q_i)$ denote the set of SEs which q_i is NDI-equivalent with. Two SEs q_i and q_j are defined as *universally NDI-equivalent* (UNDI-equivalence, for short) if $\pi(q_i) = \pi(q_j)$.

UNDI-equivalence is a transitive relation (symmetry and reflexiveness is inherited from the NDI-equivalence) and therefore can be used to define non-overlapping equivalence sets. There is, however, more than one way of translating the NDI-equivalence into a transitive relation and this issue is brought up again in Section 6.

Definition 12 A set of UNDI-equivalence sets, E , consists of disjoint sets of SEs, $e \in E$ where $e \subseteq Q$ (with all e s together covering all SEs in the SSM) and for all $q_i, q_j \in e$, q_i and q_j are UNDI-equivalent. In the pseudo-code notation, the function call $E = \text{generate_UNDI_equivalence_sets}(ssm)$ will be used to denote the generation of a set of UNDI-equivalence sets E from an SSM ssm .

A machine with equivalent SEs can be collapsed to a smaller machine by collapsing all equivalent sets into new, individual SEs. This collapsing, or merging, will, however, be part of another subalgorithm (the `merge_cvq`-function of Definition 17).

2.2.7 SSM examples and interpretations

Example 1 Consider an SSM with $Q = \{q_1, q_2\}$, $X = \{\mathbf{a}, \mathbf{b}\}$, $Y = \{\mathbf{c}, \mathbf{d}\}$ and transition probabilities $\mathcal{P} = \{p(q_2, \mathbf{c}|q_1, \mathbf{a}) = 1.0, p(q_2, \mathbf{c}|q_1, \mathbf{b}) = 0.1, p(q_1, \mathbf{c}|q_1, \mathbf{b}) = 0.9, p(q_1, \mathbf{c}|q_2, \mathbf{a}) = 0.8, p(q_1, \mathbf{d}|q_2, \mathbf{a}) = 0.2, p(q_2, \mathbf{d}|q_2, \mathbf{b}) = 1.0\}$ (SSM A in Figure 1). All zero probabilities are left out from description, e.g. that $p(q_1, \mathbf{a}|q_1, \mathbf{a}) = 0.0$. If we let the initial SE vector be $\vec{q}(0) = (1.0, 0.0)$ (i.e. that $p(Q = q_1) = 1.0$ at time $t = 0$) and then parse the string **aabbba** with the machine, the sequence of SE and output symbol distribution vectors (where the two elements of vector \vec{y} correspond to probabilities of symbol **c** and **d** respectively) would be as follows:

- (a) $\vec{q}(1) = (0.0, 1.0)$, $\vec{y}(1) = (1.0, 0.0)$,
- (a) $\vec{q}(2) = (1.0, 0.0)$, $\vec{y}(2) = (0.8, 0.2)$,
- (b) $\vec{q}(3) = (0.9, 0.1)$, $\vec{y}(3) = (1.0, 0.0)$,
- (b) $\vec{q}(4) = (0.81, 0.19)$, $\vec{y}(4) = (0.9, 0.1)$,
- (b) $\vec{q}(5) = (0.729, 0.271)$, $\vec{y}(5) = (0.81, 0.19)$,
- (a) $\vec{q}(6) = (0.271, 0.729)$, $\vec{y}(6) = (0.9458, 0.0542)$.

Note that, since the SSM of Example 1 has no dead transitions, the sums of the SE and output probabilities are always one, respectively. In the next example an SSM that has some dead transitions will be shown.

Example 2 Consider an SSM with $Q = \{q_1, q_2\}$, $X = \{\mathbf{a}, \mathbf{b}\}$, $Y = \{\mathbf{c}, \mathbf{d}\}$ and transition probabilities $\mathcal{P} = \{p(q_2, \mathbf{c}|q_1, \mathbf{a}) = 1.0, p(q_1, \mathbf{c}|q_1, \mathbf{b}) = 0.9, p(q_2, \mathbf{c}|q_1, \mathbf{b}) = 0.1, p(q_1, \mathbf{d}|q_2, \mathbf{a}) = 1.0\}$ (SSM

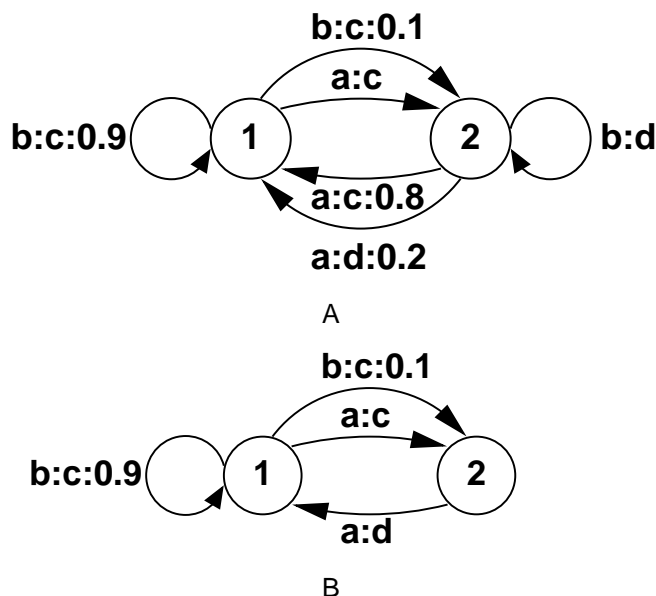


Figure 1: The two SSMs of examples 1 (A) and 2 (B) (with q_i denoted by i). A transition label $\mathbf{x}:\mathbf{y}:p$ is to be read as a transition with \mathbf{x} as input and \mathbf{y} as output and p as the probability of this transition. For example, the transition label “ $\mathbf{b}:\mathbf{c}:0.1$ ” from q_1 to q_2 corresponds to the conditional probability $p(q_2, \mathbf{c}|q_1, \mathbf{b}) = 0.1$. If the p is 1.0, then the probability is omitted from the label.

B in Figure 1). Note that the machine has dead transitions since q_2 has no outgoing transition over symbol \mathbf{b} . SE q_2 is also an example of a deterministic SE. If $\vec{q}(0) = (0.0, 1.0)$ and the SSM is fed symbol \mathbf{b} as input the probabilities of all SEs and outputs would therefore immediately reach zero. In other words, the possibility of being in SE q_2 is eliminated by the symbol \mathbf{b} , and as a consequence of the SSM “observing” \mathbf{b} , the probability of this impossibility vanishes from the machine. If we instead let $\vec{q}(0) = (1.0, 0.0)$ and then parse a sequence of t \mathbf{b} s, the sum of SE probabilities would be $0.9^{(t-1)}$ when $t \geq 1$.

As the example illustrates, the the SSM acts as as an observer of inputs, from which it derives a modelled degree of belief of what the actual enumeration of the state and output of the underlying system would be, given the same input sequence. Typically, if an SSM is given a uniform initial SE distribution, the SE distribution will, for each input symbol, gradually become more and more focused towards a small number of possible SEs (and output symbols). In a way, the SSM can be seen as to “condense”, or “crystallize” to a minimal hypothesis of the factual state of underlying system.

An SSM can be quite different and counterintuitive compared to state machines typically encountered in the literature which will be illustrated in the next example.

Example 3 The SSM of Figure 2 represents a more complex SSM. This machine is not fully connected and also contains a “dead SE” from which there are no transitions (q_{10}). This is a perfectly correct form of SSM and, if provided with an initial SE distribution, this machine can process input sequences just as the SSMs of the previous examples. In this machine, many properties of the UNDI-equivalence become clear. The set of equivalence sets returned by `generate_UNDI_equivalence_sets` is $\{\{q_1\}, \{q_2, q_3, q_4\}, \{q_5, q_6, q_7\}, \{q_8\}, \{q_9\}, \{q_{10}\}\}$. State element q_{10} will, since it has no

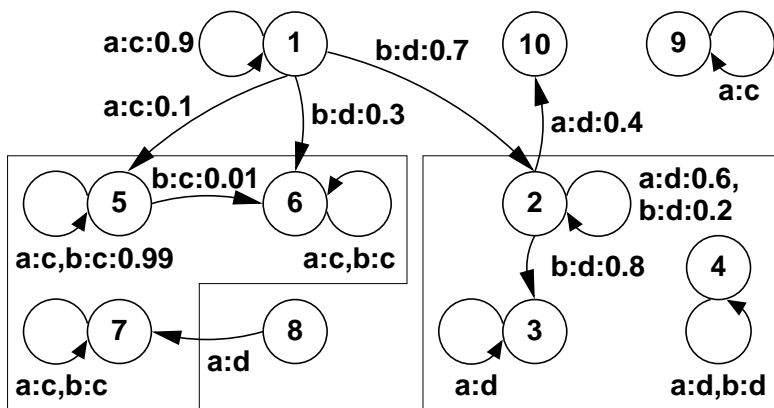


Figure 2: A more complex SSM example where UNDI-equivalence sets have been grouped together. See discussion in text of Example 3.

outgoing transition, be NDI-equivalent with all other SEs. However, since it is the only element with this property, it is not UNDI-equivalent with anything but itself. q_8 is, on the other hand, the only one NDI-equivalent with only itself. One can easily see that the SEs of the equivalence sets $\{q_2, q_3, q_4\}$ and $\{q_5, q_6, q_7\}$ have the output symbols in common, respectively. q_3 is special since it has no outgoing transition over symbol **b**, whereas q_2 and q_4 have. q_3 is, however, NDI-equivalent with SEs q_2 and q_4 since it can not be decided that symbol **b** should result in any different output given any of these three SEs. Then, for the same reason, why is not q_9 UNDI-equivalent to q_5, q_6 and q_7 , although it too, is constantly giving **c** as output? The reason is that SE q_9 is also NDI-equivalent with q_1 , which none of q_5, q_6 and q_7 are, therefore it is not UNDI-equivalent with them the way q_3 is with q_2 and q_4 . If one added q_{11} , NDI-equivalent with q_3 , but not with q_2 and q_4 , then this situation would change (even though q_{11} may seem completely unrelated to q_3). Another thing to notice is that the transition from the equivalence set $\{q_2, q_3, q_4\}$, from q_2 to q_{10} , makes no difference for the assessment of the equivalence of q_2 with q_3 and q_4 since the transitions is to a dead SE from which no decisive inequivalences can be derived.

Now the format of the extracted rules of **CrySSMEx** has been described. The next step is to define a vector quantization function which will later be orchestrated to work in conjunction with these rules.

3 The Crystalline Vector Quantizer

The observation and quantization of the state space of the underlying SDTDS is perhaps the most signifying constituent of RNN-RE algorithms. In previous work, quite traditional clustering algorithms have been used to partition the state space of the RNN (Jacobsson, 2005), e.g. self-organizing maps and k -means clustering. The problem with these clustering algorithms is that they partition the state space solely according to spatial properties, e.g. so that datapoints have low intracluster distances and high intercluster distances (Everitt, Landau & Leese, 2001). In the case of RNNs and other dynamic systems, however, the spatial requirements should give way to *functional requirements*. The spatial (e.g. Euclidean) proximity of two states of the SDTDS is of less importance for deciding if they belong to the same cluster, than the invariance of the apparent behaviour of the SDTDS with respect to these states. Similar problems exist also when clustering internal activations of feedforward networks (Sharkey & Jackson, 1995). This means, among other things, that the quantizer

may need to have varying granularity in different regions of the state space.

A partitioning that is completely guided only by dynamics of the SDTDS should be, however, an idealization (Casey, 1996; Blair & Pollack, 1997; Jacobsson & Ziemke, 2003b). Instead we will have to be content with partitions that are equivalent for a specific and finite set of input sequences (in the finite Ω of Definition 3).

To satisfy the functional requirements, we need a quantizer that allows to generate a division of the state space based on spatial properties but also to split and merge regions into new regions when the functional requirements are not satisfied (details of when exactly it is appropriate to split or merge are covered in Section 4). For this purpose, a novel quantizer is suggested here, a *Crystalline Vector Quantizer* (CVQ). The CVQ has some resemblance with the hierarchical decision tree representation extracted from feed-forward networks by Craven and Shavlik (1996), but differs in the details.

The CVQ is built upon a graph which now will be defined. How the information of this graph is used to quantize a vector space will then be described in Section 3.2 and CVQ training in Section 3.3.

3.1 Definition of CVQ graph

Definition 13 A CVQ graph is a quadruple $CVQ = \langle N_{Leaf}, N_{VQ}, N_{Merged}, n_{root} \rangle$ where $n_{root} \in N_{Leaf} \cup N_{VQ}$ is the root node of the CVQ graph, where the constituents are defined as in definitions 14–16.

The CVQ graph is a directed graph and could thus be described as a set of vertices and edges, but for notational reasons it is easier to omit the edges from the description and instead of edges let nodes have explicit references to other nodes. The first node type, however, has no explicit references to any other nodes.

Definition 14 A leaf node in a CVQ graph $n \in N_{Leaf}$ has only one constituent, $n = \langle ID \rangle$, where $ID \in \mathbb{N}$ is a unique enumeration of the leaf nodes within the CVQ and $1 \leq ID \leq |N_{Leaf}|$.

Definition 15 A Vector Quantizer (VQ) node in a CVQ graph, $n \in N_{VQ}$ is a tuple $n = \langle M, C \rangle$ where M is a list of K model vectors, $[\vec{m}_1, \vec{m}_2, \dots, \vec{m}_K]$, where $[\vec{m}]_i \in \mathbb{R}^d$ and C is a (nonrepetative) list of child nodes $[c_1, c_2, \dots, c_K]$ where $c_i \in N_{Leaf} \cup N_{VQ} \cup N_{Merged}$. $d \in \mathbb{N}$ is the dimensionality of the vector space which the CVQ will be used to quantize.

Definition 16 A merged node in a CVQ graph, $n \in N_{Merged}$, contains only a “link”, $n = \langle n_{group} \rangle$, where $n_{group} \in N_{Leaf} \cup N_{VQ} \cup N_{Merged}$.

The interpretation will be clarified in the next section where the use of a CVQ as quantization function is described in which all CVQ node constituents are of relevance. The example of Figure 3 is also a good help for understanding the interpretation of the CVQ nodes.

The constituents of a CVQ are simply as defined above, but there are of course a number of constraints of how the CVQ graph can be constructed, e.g. that there may be no cycles in the graph. These constraints are not straightforward to formalize, but are quite intuitive. So instead of a lengthy formal descriptions, an example will illustrate a typical CVQ topology (Figure 3). Also, the way the CrySSMEx algorithm builds the CVQ defines the constraints in exact detail (Section 4).

3.2 Quantizing with a CVQ

When a CVQ is used as a quantizer (Definition 4) the corresponding quantization function is denoted Λ_{cvq} and is in turn defined by the recursive function $winner : N_{Leaf} \cup N_{Merged} \cup$

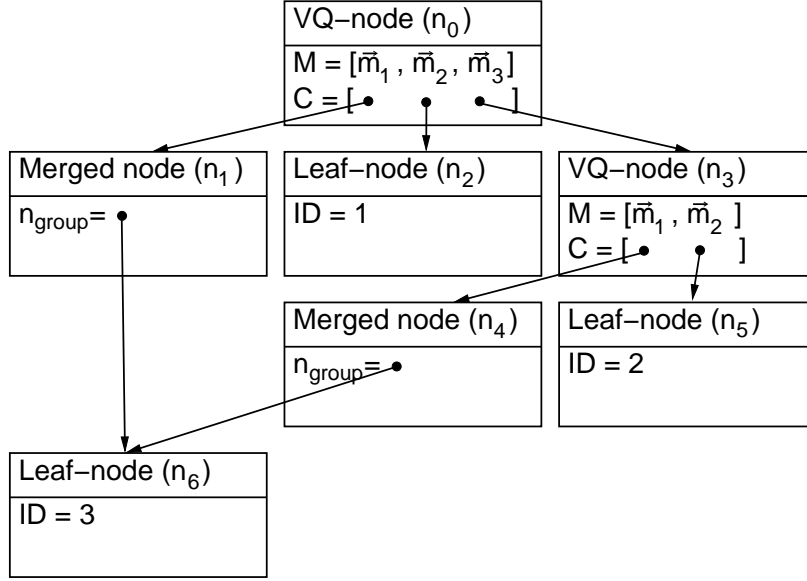


Figure 3: Example of a CVQ with $N_{Leaf} = \{n_2, n_5, n_6\}$, $N_{Merged} = \{n_1, n_4\}$, $N_{VQ} = \{n_0, n_3\}$ and $n_{root} = n_0$.

$N_{VQ} \times \mathbb{R}^d \rightarrow \{1, 2, \dots, m\}$ as defined in equation⁶:

$$\Lambda_{cvq}(\vec{v}) = \text{winner}(n_{root}, \vec{v}) \quad (11)$$

where **winner** is recursively defined as

$$\text{winner}(n, \vec{v}) = \begin{cases} n.ID & \text{if } n \in N_{Leaf} \\ \text{winner}(n.n_{group}, \vec{v}) & \text{if } n \in N_{Merged} \\ \text{winner}(n.c_w, \vec{v}) & \text{if } n \in N_{VQ} \end{cases} \quad (12)$$

where w , the index of the winning child of a VQ-node, is determined according to

$$w = \underset{1 \leq i \leq |n.C|}{\text{argmin}} \|\vec{v} - n.[\vec{m}]_i\| \quad (13)$$

where $\|\vec{v} - n.[\vec{m}]_i\|$ denotes the Euclidean distance between the vector to be quantized and the i th model vector of the VQ-node. If two model vectors have equal distance to the data vector, the smaller of the indices will be returned.

The division of a two-dimensional state space using the CVQ of Figure 3 with exemplified instantiated model vectors is shown in Figure 4.

3.3 CVQ training

The training of the CVQ in **CrySSMEx** is tightly connected with operations of the SSM and sampling of the SDTDS (as discussed in Section 1.2). Here, however, the operations that later will be used to refine the CVQ will be defined as independent of their role in **CrySSMEx** (see Section 4 for this context).

The training consists of replacing leaf nodes with either merged nodes or VQ nodes, add new leaf nodes, and then reenumerate the *IDs* appropriately. Replacement of a leaf node with a VQ-node results in a larger number of leaf nodes and is referred to as *CVQ splitting*.

⁶A object orientation like notation will here be adopted, where $X.Y$ means “The Y of X ”.

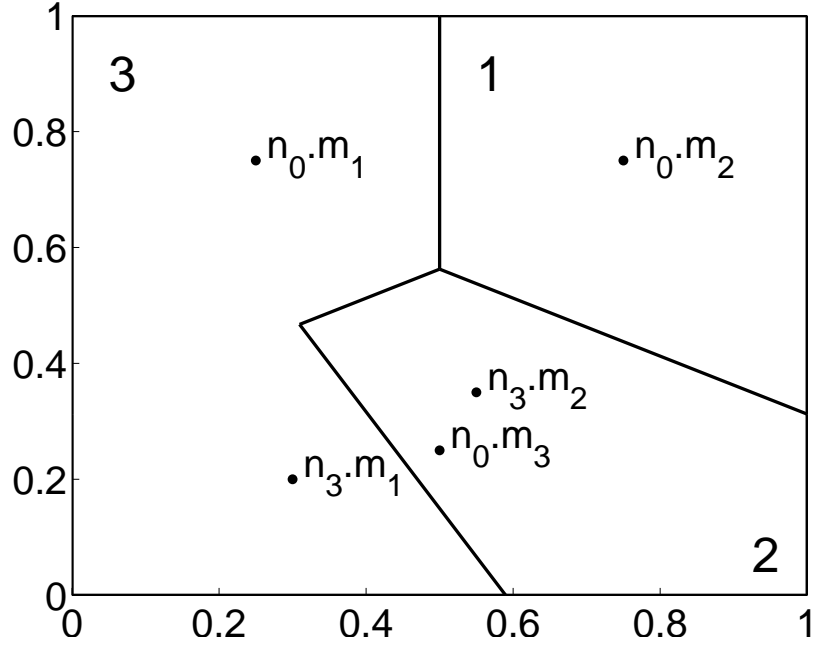


Figure 4: How a two-dimensional space would be quantized if the example CVQ in Figure 3 had model vectors $n_0.M = [(0.25, 0.75), (0.75, 0.75), (0.5, 0.25)]$ and $n_3.M = [(0.30, 0.20), (0.55, 0.35)]$.

Replacement of several leaf nodes with merged nodes results in a smaller number of leaf nodes and is referred to as *CVQ merging*. After completion of each of these operations, leaf nodes will be reenumerated.

First, merging will be described, then basic splitting, then an operation called *complete splitting*. “The user” which will be mentioned in the following descriptions, will be another part of the CrySSMEx algorithm, but it should of course be possible to use CVQ in other contexts.

3.3.1 The initial CVQ

The initial CVQ, denoted cvq^0 , is the simplest possible CVQ consisting only of one leaf node (n_{root}) with $ID = 1$. All vectors will thereby be quantized as $n_{root}.ID = 1$ by the initial CVQ.

3.3.2 Merging

Merging of nodes in a CVQ corresponds to merging regions in the quantized space. This is conveniently described with an example:

Example 4 In the example of Figure 3, nodes n_1 and n_4 have been merged into n_6 . Before this merge, n_1 and n_4 were two separate leaf nodes, but then the “user” discovered that the corresponding regions should not be separated, for some reason. The merge was then conducted by creating a new leaf node, n_6 , and then replace the leaf nodes, n_1 and n_4 with merged nodes connected to n_6 .

In principle, any number of leaf nodes can be merged simultaneously. The merge is an operation on the CVQ graph, not necessarily related to any spatial properties of the

quantized space, i.e. disconnected regions can be merged. The decision of what leaf nodes to merge is also entirely independent from their position in the CVQ graph.

Definition 17 The merging of one or more groups of leaf nodes will be denoted $cvq' := \text{merge_cvq}(cvq, E)$ where E is a set of disjoint sets of ID s covering all leafs of the CVQ. The result, cvq' , is the CVQ where leaves have been merged into one new leaf node per set in E (trivial sets in E , with only one member, are simply ignored). The leaf nodes are also re-enumerated before returning the resulting CVQ.

E will later (in Algorithm 3) be connected to the set of equivalence sets generated from SSMs by the function `generate_UNDI_equivalence_sets` (described in Definition 12).

Example 5 If $cvq' = \text{merge_cvq}(cvq, \{\{1, 3, 5\}, \{2, 4\}, \{6\}\})$ is called, it will replace leaf nodes with ID s 1, 3 and 5 with merged nodes connected to a new leaf node, and correspondingly for 2 and 4. The leaf node with $ID = 6$ will be left unaltered. CVQ-based quantization function $\Lambda_{cvq'}$ will then quantize vectors into the range $[1, 3]$ whereas Λ_{cvq} quantized into the range $[1, 6]$.

3.3.3 Basic splitting

When a CVQ leaf node is split, this corresponds to splitting the corresponding region region enumerated by that leaf. It is easiest to describe also this mechanism with an example:

Example 6 A set of two-dimensional data vectors V are quantized as in Figure 4. Now, the user has discovered that there are actually two types of vectors quantized as 1 (let us call the set of these vectors V_1). The user wants the two classes to be correctly separated by the CVQ. To do this the user collects all data vectors $V_1 = \{\vec{v}_i : \Lambda_{cvq}(\vec{v}_i) = 1\}$ and separates this set into two sets V_1^+ and V_1^- corresponding to the two classes. The node with $ID = 1$ (i.e. n_2 in Figure 3) would then be replaced with a VQ node with two model vectors and two new leaf nodes as children. The model vectors of the VQ node is then set to be the average of the vectors in sets V_1^+ and V_1^- respectively.

Above, a leaf was only split into two new leaves, but in general, a leaf’s corresponding region in the quantized space can be split into any number of regions.

3.3.4 Complete splitting

It is possible that the model vectors will not perfectly separate the data vectors after a basic split, since they might be linearly inseparable or the average vectors of the data sets may not be the perfect model vectors for separating the data⁷. In such cases, `CrySSMEx` would typically re-split the resulting region again automatically (see Section 4 where `CrySSMEx` is described in detail). It is however possible that an imperfect split may cause a non-minimal machine to be extracted and also that `CrySSMEx` will not terminate due to spurious SEs. Therefore a “perfect”, or complete, split mechanism has also been devised.

To perform a complete split, the splitting is first conducted as in Example 6. But if the enumerated data vectors still are not separated, then the new leafs will be re-split using the corresponding subsets of the data vectors until the data vector class can be uniquely inferred from the identity of the involved leaf nodes. After this, all involved leaf nodes “belonging” to the same class are merged.

⁷ In fact, it can be a quite inefficient to use the average as model vectors. The reasons that model vectors are chosen such is basically that it is simple, deterministic and does not require any parameters. Other, more sophisticated methods, such as resource allocating learning vector quantization (Everitt et al., 2001), have also been tested, but it does not really make a big difference apart from longer computation times and somewhat smaller CVQ graphs.

Definition 18 The complete split of several VQ nodes using several data sets at once will be denoted $cvq' := \text{split_cvq}(cvq, D)$ where cvq is the CVQ which to be split and $D = [D_1, D_2, \dots, D_{|\Lambda_{cvq}|}]$ is a list of data sets where D_i is the data set for splitting the leaf node with $ID = i$ (if the node should not be split, then $D_i = \emptyset$). The elements of a data set are pairs $\langle \vec{v}, \ell \rangle$ where $\vec{v} \in \mathbb{R}^n$ is the data vector and $\ell \in \mathbb{N}$ is label, or class, of the data vector. The leaf nodes of cvq' are also re-enumerated immediately after the completion of all splits.

There is also a possibility that the averages of two or more classes are exactly the same in which case the splitting will fail completely. This is very unlikely to occur by chance and no fixes are included in the definition of the algorithm. It has not occurred in any of the experiments (Section 5), and if it did, the implemented algorithm would abort execution and generate a warning. It is of course also very important that there is no pair of differently classified but identical data vectors. This should not happen in the context of **CrySSMEx** due to the way data is collected from a deterministic machine, but it should be kept in mind if the CVQ is to be used in another context.

4 The Crystallizing Substochastic Sequential Machine Extractor

4.1 Data selection from Ω

The perhaps most important point of convergence of the various constituents described so far in this article, is where subsets of Ω , are selected and classified based on properties of the extracted SSM. The goal of **CrySSMEx** is to generate a deterministic SSM from the underlying deterministic SDTDS by dividing the state space into a minimal set of quanta that can be used to describe the SDTDS perfectly in the context of Ω . To do this, indeterministic SEs of the SSM are targeted for being split in the corresponding CVQ using selected state vectors from Ω . The basis for the selection of state vectors is to choose the set which should convey the most information, primarily of the output of the SSM and secondarily, the next state element of the SSM. The entropies $H_{ssm}(Y|Q = q_i, X = x_k)$ and $H_{ssm}(Q|Q = q_i, X = x_k)$ (definitions 9 and 10 respectively) are used for this selection. This basis for selection is not the only one possible, however, and this will be brought up again in Section 6. The entire selection procedure is contained in the function `collect_split_data`, described in Algorithm 2.

4.2 CrySSMEx main loop

The ingredients for **CrySSMEx** have now been presented:

- the SDTDS which represents the class of specimens for **CrySSMEx** to analyse (definition 1),
- the data set, i.e. the SDTDS transition event set Ω (definition 3),
- SSMs, which can be viewed as a subtype of SDTDSs (definition 6),
- SDTDS translation into SSM through quantization of input, output and state (definition 8),
- generation of UNDI-equivalence sets of SEs in SSMs (definition 12),
- CVQ (definitions 13 to 15), used as a quantizer of vectors through the function Λ_{cvq} (equation 11),
- merging and splitting of CVQ leaf nodes (definitions 17 and 18),
- a mechanism for selecting and labelling state vectors of Ω based on properties of the SSM (Algorithm 2).

`collect_split_data`($\Omega, ssm, \Lambda_i, \Lambda_s, \Lambda_o$)

Input: A transition event set, Ω , an SSM, ssm , an input quantizer, Λ_i , a state quantizer, Λ_s , an output quantizer, Λ_o .

Output: A list of data sets D , one data set per $q \in Q$. The element of each data set is a pair $\langle \vec{v}, \ell \rangle$ where $\vec{v} \in \mathbb{R}^n$ is a data vector and $\ell \in \mathbb{N}$ is the assigned label of the vector.

```

begin
   $D := [\emptyset, \emptyset \dots \emptyset]$ ;
  for  $\forall \langle \vec{s}(t), \vec{v}(t), \vec{o}(t+1), \vec{s}(t+1) \rangle \in \Omega$  do
     $q_i := \Lambda_s(\vec{s}(t))$ ;
     $x_k := \Lambda_i(\vec{v}(t))$ ;
     $y_l := \Lambda_o(\vec{o}(t+1))$ ;
     $q_j := \Lambda_s(\vec{s}(t+1))$ ;
    /*If  $q_i$  is indeterministic, the state vector should be stored in  $D$  with an
    appropriate labelling. */
    if  $\exists x_m : H_{ssm}(Y|Q = q_i, X = x_m) > 0$  then
       $x_{max} := \operatorname{argmax}_{x_m \in X} H_{ssm}(Y|Q = q_i, X = x_m)$ ;
      if  $x_k = x_{max}$  then
        /*If output indeterministic with respect to  $q_i$  and  $x_k$ , label the state
        vector with the output symbol,  $y_l$ . */
         $D_i := D_i \cup \langle \vec{s}(t), y_l \rangle$ ;
      end
    else if  $\exists x_m : H_{ssm}(Q|Q = q_i, X = x_m) > 0$  then
      /*If output is uniquely determined from  $q_i$ , but next state is not, label state
      vector using next SE,  $q_j$ . */
       $x_{max} := \operatorname{argmax}_{x_m \in X} H_{ssm}(Q|Q = q_i, X = x_m)$ ;
      if  $x_k = x_{max}$  then
         $D_i := D_i \cup \langle \vec{s}(t), q_j \rangle$ ;
      end
    endif
  end
end
return  $D$ ;
end

```

Algorithm 2: `collect_split_data` selects state vectors from Ω and labels them according to either Λ_o or Λ_s such that they are suitable for use in splitting CVQ nodes. The resulting list of data sets, D consists of one data set of labelled state vectors for each SSM SE, i.e. D_i corresponds to the data set for splitting state q_i .

These constituents are integrated into the CrySSMEx-algorithm as described in Algorithm 3. The principle behind the algorithm is that the SSM should be kept as small as possible through merging of SEs that are UNDI-equivalent while at the same time splitting indeterministic SEs. It is important to decide before an SE is deemed to be indeterministic, that it is not so because it, over one input symbol, transits to two or more SEs which are actually equivalent (or at least UNDI-equivalent). If the machine was not minimized through the merging of equivalent state elements, it would risk resulting in an explosion of SEs due to unjustified splits.

If the algorithm does not converge in due time, additional termination criteria could be added. For example, one may want to limit the number of possible iterations, or put a limit on $|Q|$. The extracted, then possibly indeterministic, SSM will still be a model of the underlying SDTDS, and moreover, the more computational resources dedicated to iterate CrySSMEx, the better a model the SSM will be of the SDTDS, in terms of fidelity.

```

CrySSMEx( $\Omega, \Lambda_o$ )
Input: An SDTDS transition event set,  $\Omega$ , and an output quantization function,  $\Lambda_o$ .
Output: A deterministic SSM mimicking the SDTDS within the domain  $\Omega$  as
described by  $\Lambda_o$ .
begin
  let  $\Lambda_i$  be an invertible quantizer for all  $I$  in  $\Omega$ ;
   $i := 0$ ;
   $ssm^0 := \text{create\_machine}(\Omega, \Lambda_i, \Lambda_{cvq^0}, \Lambda_o)$ ;
  /* $ssm^0$  has  $Q = \{q_1\}$  with all transitions to itself. */
  repeat
     $i := i + 1$ ;
     $D := \text{collect\_split\_data}(\Omega, ssm^{i-1}, \Lambda_i, \Lambda_{cvq^{i-1}}, \Lambda_o)$ ;
     $cvq^i := \text{split\_cvq}(cvq^{i-1}, D)$ ;
     $ssm^i := \text{create\_machine}(\Omega, \Lambda_i, \Lambda_{cvq^i}, \Lambda_o)$ ;
    if  $ssm^i$  has UNDI-equivalent states then
      /*Merge SEs if possible. */
       $E := \text{generate\_UNDI\_equivalence\_sets}(ssm^i)$ ;
       $cvq^i := \text{merge\_cvq}(cvq^i, E)$ ;
       $ssm^i := \text{create\_machine}(\Omega, \Lambda_i, \Lambda_{cvq^i}, \Lambda_o)$ ;
    end
  until  $ssm^i$  is deterministic;
  return  $ssm^i$ ;
end

```

Algorithm 3: The main loop of CrySSMEx.

5 Experiments

The main purpose of the experiments in this article is to show that CrySSMEx manages to extract machines in contexts previously unsolved using RNN-RE algorithms. Another purpose is to identify remaining weaknesses of CrySSMEx by running it on notoriously challenging SDTDSs. Deeper analysis of how and why CrySSMEx behaves as it does and of the resulting SSMs and CVQs will, however, have to be postponed to future work.

5.1 An illustrative example

Most previous work on RNN-RE algorithms has been experimentally tested on regular language domains (Jacobsson, 2005). The aim of this experiment is to demonstrate that this kind of domain is trivial and at the same time illustrate the extraction process. It has

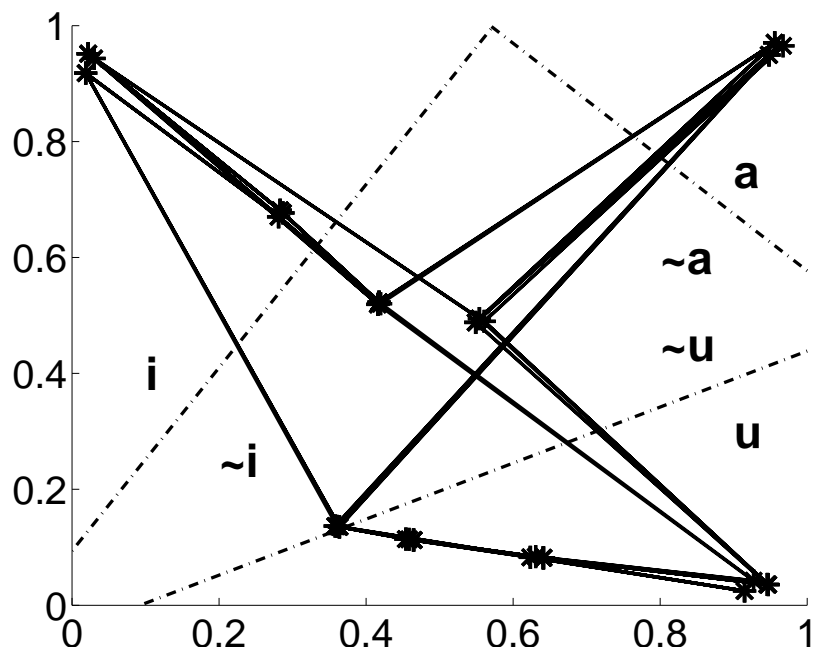


Figure 5: The state space of the RNN in the “badiiguuu”-domain. The states (*) and transitions between states and the decision hyperplanes (Sharkey & Jackson, 1995) for each output unit are plotted (only three are visible).

already been proven that if an RNN is robustly performing a regular language recognition task, then this model can always be extracted (Casey, 1996). But no technique can warrant that such an extraction is possible in practice: there is no guarantee for **CrySSMEx** either, of course, but it does seem to be quite a straightforward process.

Elman (1990) used a simple regular language to train a simple recurrent network (SRN). The domain consisted of three subsequences **ba**, **dii** and **guuu** repeated in random order, e.g. **babadiibaguuudiiguuu...** The task for the RNN was to do next-symbol prediction. In essence, only the vowels were at all predictable. An SRN was here trained with two hidden nodes on this domain with the symbols represented as six dimensional vectors with one node active for respective symbol.

To generate Ω , a string of 10^5 randomly ordered substrings was used on the trained RNN. The state space of the RNN is shown in Figure 5. Three iterations completed the extraction and the sequence of state space divisions, the CVQ graphs and the SSMs are shown in Figure 6. The breadth first technique of Giles, Miller, Chen, Chen and Sun (1992) has also been tested on this domain, and it resulted in an large number of states never visited by the RNN when predicting the actual strings (Jacobsson & Ziemke, 2003b).

Most essential features of **CrySSMEx** are exemplified in this extraction. In the initial SSM, ssm^0 , it can be seen that input symbol **i** generates the maximum amount of uncertainty regarding the output symbol (the output symbol **C** here corresponds to the non-symbol generated by the RNN when it predicts a consonant, with no possibility of predicting the exact symbol due to the random ordering of substrings). For that reason, `collect_split_data` will select state vectors which the RNN occupied when it received **i** as input and label them according to the output label as determined by Λ_o . The CVQ is then split according to the selected data, resulting in cvq^1 . The same procedure is repeated again with ssm^1 and an SSM of three SEs, of which two are UNDI-equivalent, is generated (not shown) This results in two merged nodes in cvq^2 . As can be seen in the state space division, cvq^2 merges two

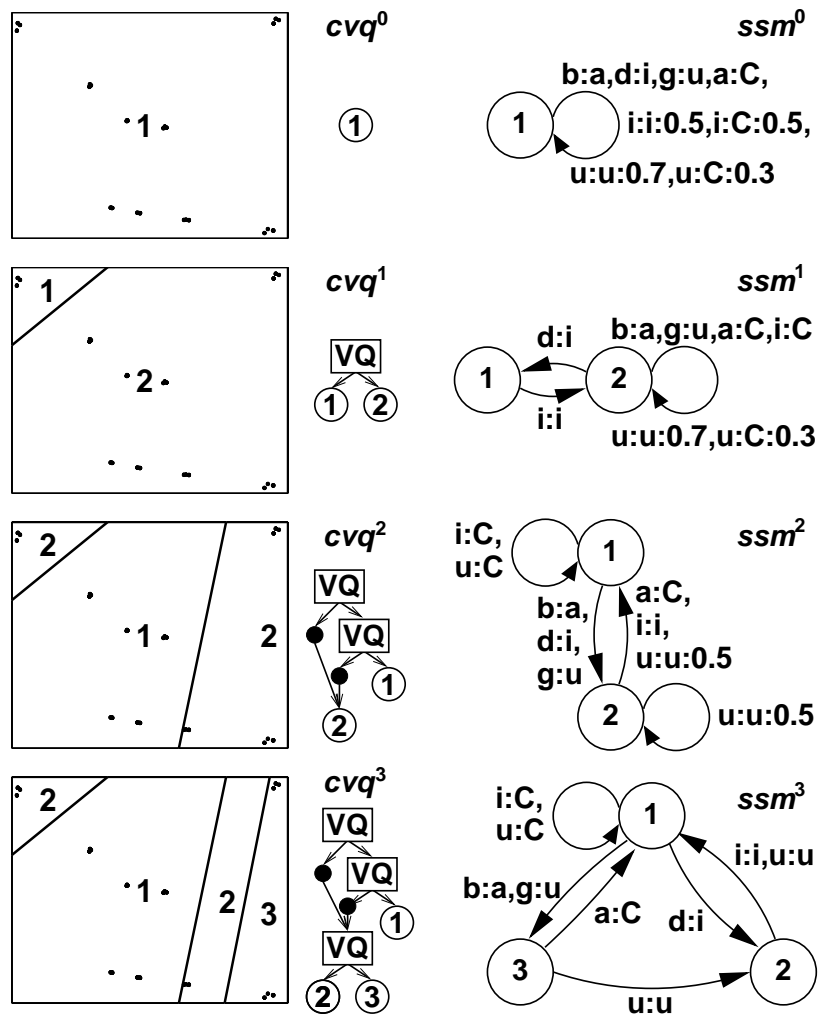


Figure 6: Extraction from RNN predicting in the “badiiguuu”-domain. The state space divisions (cf. Figure 5), CVQ graphs and SSMs are shown for the initial model and all subsequent iterations of CrySSMEx.

(locally) disconnected subspaces. From both SEs of ssm^2 , the output can now be deterministically predicted, but q_2 is still indeterministic since transitions from it over symbol \mathbf{u} is ambiguously mapped to q_1 and q_2 . Therefore `collect_split_data` selects those RNN state vectors in Ω enumerated 2 by Λ_{cvq^2} from which a transition induced by symbol \mathbf{u} was made, and labels them according to the Λ_{cvq^2} -enumeration of the subsequent state vector. After the split of cvq^2 , `CrySSMEx` terminates since the resulting SSM is deterministic and will fully mimic the underlying RNN within Ω .

Note that there are some dead transitions in ssm^3 , e.g. for symbol \mathbf{g} in q_3 . If the underlying RNN is fed a \mathbf{g} while occupying a state in the corresponding subspace, it will certainly react in some manner, but since that event was not recorded in Ω , the resulting SSM does not model it. Also, the resulting SSM is not a model of the input source; for example, although not supported in Ω , ssm^3 models the outcome of infinite sequence of symbols \mathbf{i} and \mathbf{u} . This is due to the fact that `CrySSMEx` does not build a model of the domain, it builds a model of how the underlying system interacts with its domain without guarantees of generalization to situations outside Ω .

5.2 An RNN trained on a context free language

Prediction of symbols in the context free language $\mathbf{a}^n\mathbf{b}^n$ is a challenging domain for RNNs that has been studied quite intensely (e.g. Wiles & Elman, 1995; Bodén & Wiles, 2000; Gers & Schmidhuber, 2001). `CrySSMEx` was here used to analyse 100 successfully trained⁸ SRNs (of one input node, two state nodes and one output node) on predicting the predictable symbols of randomly ordered $\mathbf{a}^n\mathbf{b}^n$ -strings ($1 \leq n \leq 10$). Ω was here generated by exposing the RNN to exactly 200 $\mathbf{a}^n\mathbf{b}^n$ -strings of each length ($1 \leq n \leq 10$) in random order. For all 100 RNNs, extraction was successful, resulting in SSMs of eleven SEs. An example of an extracted machine, together with the CVQ-quantized state space is shown in Figure 7.

The regular grid lattice quantizer of Giles, Miller, Chen, Chen and Sun (1992) was also tested, and it typically did never find any SSM with the same behaviour as the RNN until the state space was divided into at least 40×40 grids. The number of SEs was then between 25 and 70. If the breadth first search of that paper is employed, the number of states becomes even higher (Jacobsson & Ziemke, 2003b) because then many states which would not have been visited when processing $\mathbf{a}^n\mathbf{b}^n$ -strings are also included.

In this domain, some problems for `CrySSMEx` are exposed. First of all, although all extracted SSMs had the same $|Q|$, and all SSMs generated exactly the same outputs as the RNNs (within the sampled domain), actually two types of SSMs were extracted: 90 SSMs of one type and 10 of the other. This is probably due to different forms of dynamics of the underlying RNNs (Tonkes, Blair & Wiles, 1998). The extraction also took either 9 or 10 iterations depending on the underlying RNN. Clearly, `CrySSMEx` is sensitive to internal properties of the RNN that give rise to these differences. This may be a problem, but it may also be a key to a window of analysis of the dynamics of the underlying RNN.

A more serious problem was that if Ω was too small, e.g. with just ten occurrences of each string length, `CrySSMEx` could get stuck in loops where an ssm^i would be exactly equal to ssm^{i-n} , where $n \geq 1$. This was due to merging and splitting of SEs cancelling each other over one or more iteration. The mechanisms behind these loops are not entirely clear and the issue definitely requires more targeted experiments. It seems, however, to be linked with some kind of data starvation since it has only occurred for smaller Ω s. To circumvent this problem, `CrySSMEx` is now implemented to abort execution, by default, if a loop is encountered. Another, also implemented, option is to skip the merge completely if it should result in a loop. This approach is successful in that `CrySSMEx` terminates with a deterministic SSM equivalent with the RNN, but unfortunately with more SEs than the eleven otherwise extracted.

⁸Using a genetic algorithm, see Jacobsson and Ziemke (2003a) for more details and a more detailed list of references.

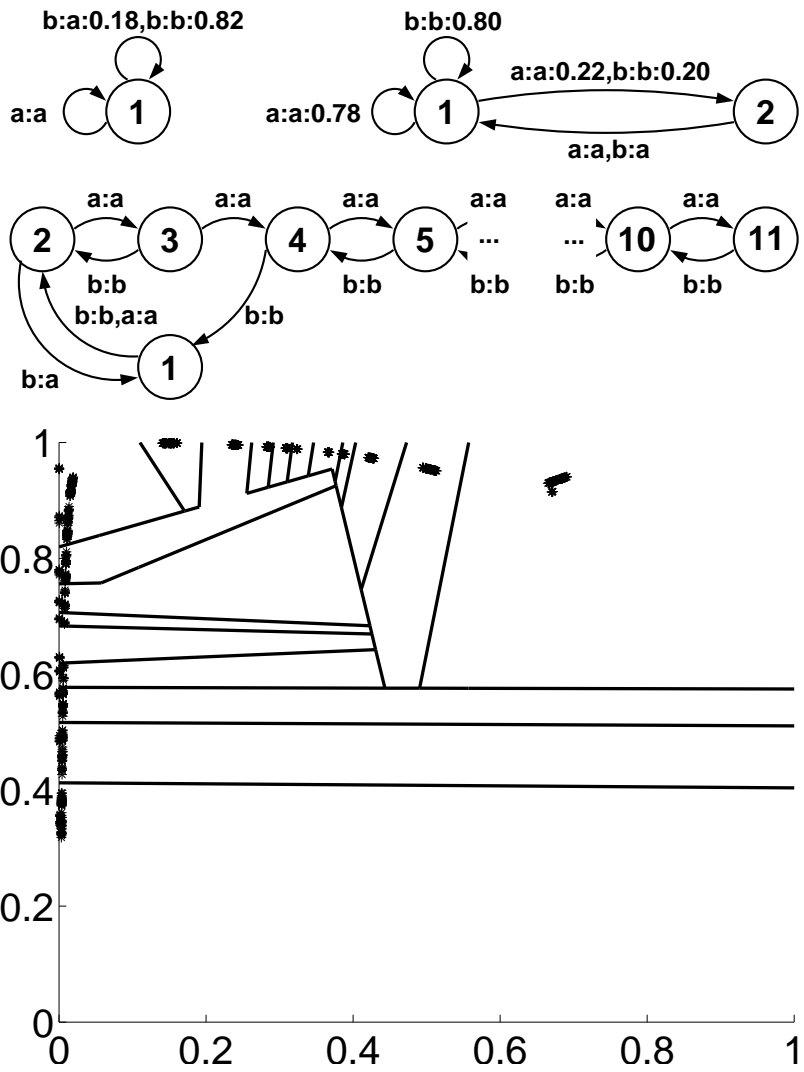


Figure 7: The two first machines (ssm^0 and ssm^1) and the last (ssm^{10}) in the sequence of machines extracted by CrySSMEx in the $\mathbf{a}^n\mathbf{b}^n$ -domain. The state space of the RNN and its cvq^{10} -division of the state space is also shown below the machines. Note that some distant states belong to the same region, while, at the same time, some nearby states are divided due to the functionally driven quantization strategy employed in CrySSMEx. Some of the disjoint regions are also actually merged in the CVQ (that cannot be seen in the diagram).

A third problem occurred sometimes when Ω was generated with longer $\mathbf{a}^n\mathbf{b}^n$ -strings. In some cases, when the RNN generalized perfectly to longer strings, this posed no problem. In other cases, erroneous prediction of the RNN was successfully modelled in the SSM, e.g. that it predicts an \mathbf{a} prematurely if $n = 11$. But, in other cases, the temporal dependencies of the errors are quite complex, and the SSMs seemed to grow indefinitely (without ever exceeding the size of Ω , of course). It is known that RNNs with weights in the vicinity of the correct solution have chaotic error gradients (e.g. Bodén & Wiles, 2000), and perhaps a chaotic RNN may explain the difficulty for CrySSMEx (cf. Section 5.4).

5.3 A large RNN

An SRN of one input node, one output node and 10^3 state nodes (i.e. more than 10^6 weights) was created to test the feasibility of extracting rules from SDTDSs of high dimensionality. The weights were initiated in the interval $[-0.01, 0.01]$ and the network was then exposed to a sequence of 10^4 randomly ordered inputs ($I = \{(0), (1)\}$). The output quantizer used in this case gave three symbols, +, - and 0, corresponding to whether activation of the output node increased, decreased or remained the same. The input symbols **a** and **b** corresponded to the binary activation of the single input unit. The continuous activation function of all nodes, $1/(1 + \exp(-net))$, makes it typically impossible that the output should stabilise completely, i.e. there should be no need for symbol 0, but limits in machine precision made it necessary. This kind of network, with small random weights has been theoretically studied earlier and has been proven to implement definite memory machines (Hammer & Tiño, 2003). This is, however, the first time a large scale network of this type has been studied using RNN-RE.

The extracted machine, with $|Q| = 19$, is shown in Figure 8. The machine perfectly emulated the behaviour of the SRN within Ω as “viewed” through Λ_o . It may be of interest to mention that the generated data took up over 230MB of storage, yet it took only six iterations in the CrySSMEx main loop to extract a machine of 19 states with the same apparent behaviour as the significantly larger RNN.

Some more preliminary experiments were carried out with the same RNN architecture but with larger random weights. For smaller weights, $|Q|$ decreased, and for larger weights, the extraction may become impossible in that the SSM size seemed to grow indefinitely. SSMs were of course still extracted, but no deterministic SSM were found within reasonable time. A high dimensional state space is not needed to make extraction of deterministic SSMs impossible, however, as the next experiment will demonstrate.

5.4 A chaotic system

To do rule extraction from a chaotic system may be considered unreasonable. If a system is chaotic it means that it will never repeat its trajectory in state space and that infinitesimal differences of two states will grow over time (Devaney, 1992). These properties make the system impossible to describe deterministically with a finite set of states. Any attempt to group two distinct SDTDS states into the same subspace will fail since their future trajectories will inevitably diverge if the system is chaotic.

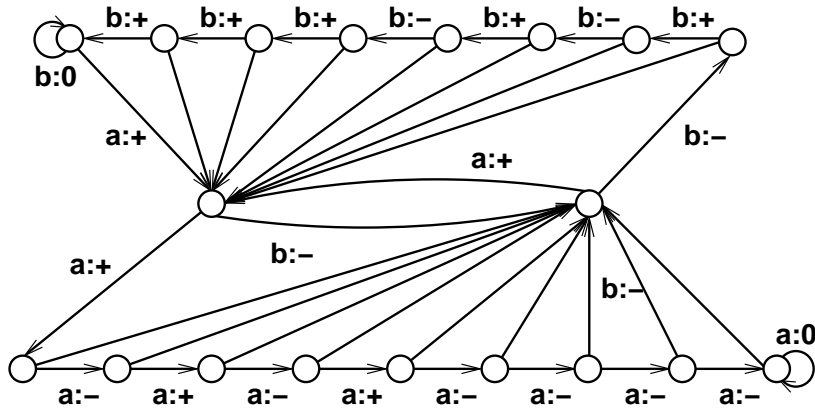


Figure 8: An SSM extracted from an RNN with 10^3 state nodes and random weights. To save space, numbering is omitted and repetitive transition labels are bundled.

It is, however, possible to use **CrySSMEx** to extract indeterministic SSMs from chaotic systems. To demonstrate this an iterated quadratic map was used:

$$s(t+1) = a \cdot s(t) \cdot (1 - s(t)) \quad (14)$$

The constant a , in the interval $[0, 4]$, determines whether the attractor of the system is a fixed point, cyclic or chaotic (Devaney, 1992). This system falls under the SDTDS definition, but with $I = \emptyset$ and $O = \emptyset$. A similar experiment, in the same domain, was conducted by Crutchfield and Young (1990), but their approach was quite different from **CrySSMEx** in that a fixed (unknown) translation from state space into a discrete set of observations was assumed. In **CrySSMEx**, it is precisely this translation that is the target for refinement.

Data was generated by running the system for 10^5 time steps, after an initial 10^5 unmonitored steps to let the system “settle in” on its attractor. The output symbols were, as in the last example, $+$, $-$ and $\mathbf{0}$ corresponding to whether the state increased, decreased or remained unchanged respectively. This choice of output symbols just one of many possible Λ_o , which is why it is part of the input parameters of **CrySSMEx**. Some readers might protest that this contradicts earlier claims in this article that **CrySSMEx** is parameter free. The subtle difference here is that although **CrySSMEx** requires the output quantization as a parameter, this quantization is for RNN applications typically defined *a priori* as a direct consequence of the symbolic domain of the RNN. In the above case, however, a number of output quantizations are conceivable.

The resulting machines are trivial when a is set such that the attractor is fixed or cyclic. If it is fixed, an SSM with one SE, and a transition generating symbol $\mathbf{0}$ are enough for describing the dynamics. If the system is cyclic, a finite set of SEs is enough to describe the system deterministically, e.g. if $a = 3.5$ (having a period four cycle) two SEs is enough, since the system, as “viewed” through Λ_0 , generates the output sequence $\dots + - + - \dots$. If $a = 3.839$, the system has a 3-cycle attractor (Devaney, 1992) and generates the sequence $\dots + + - + + - \dots$ and consequently, the SSM had three SEs.

If a is chosen so that the system is chaotic, **CrySSMEx** will not terminate (at least not until the finite set of Ω is fully accounted for). But the extracted SSMs can nevertheless be argued to account for some of the dynamics of the system. To test the fidelity of the SSM, the extracted machines were initialized with the Λ_s -enumeration of an initial state (chosen within the attractor) of the underlying system, and both the SSM and the system were run in parallel until the SSM failed to predict the output symbol of the system.

Six quadratic map systems, with $a = 3.7$, $a = 3.75$, $a = 3.8$, $a = 3.9$, $a = 3.95$ and $a = 4.0$ respectively, were analysed. The quality of extracted SSMs, in terms of the average time until SSM output deviates from the underlying system, typically increased for higher iterations of **CrySSMEx** (see Figure 9). The number of SEs grew exponentially for all systems, and faster for higher values of a . The number of SEs in relation to the number of correctly predicted symbols reveals that the “cost”, in terms of SSM size, for each correctly predicted symbol also increases exponentially. It is however interesting to note that invested computational time clearly gives revenues in SSM fidelity.

Other values of a were also tested, but if $a = 3.85$, for example, only three SEs were needed to predict the system indefinitely. So the seemingly monotonic relation between a and the number of SEs and prediction difficulty is merely an illusion.

6 Open issues

There are four main directions for future research: to further test, apply, understand (e.g. through mathematical proofs) and improve the algorithm. One obvious line of possible research is to test the outcome of using **CrySSMEx** in applications of previous RNN-RE-algorithms (accounted for in Jacobsson (2005)) How to proceed with testing **CrySSMEx** further and applying it to other systems will not be brought up here in detail, however. There

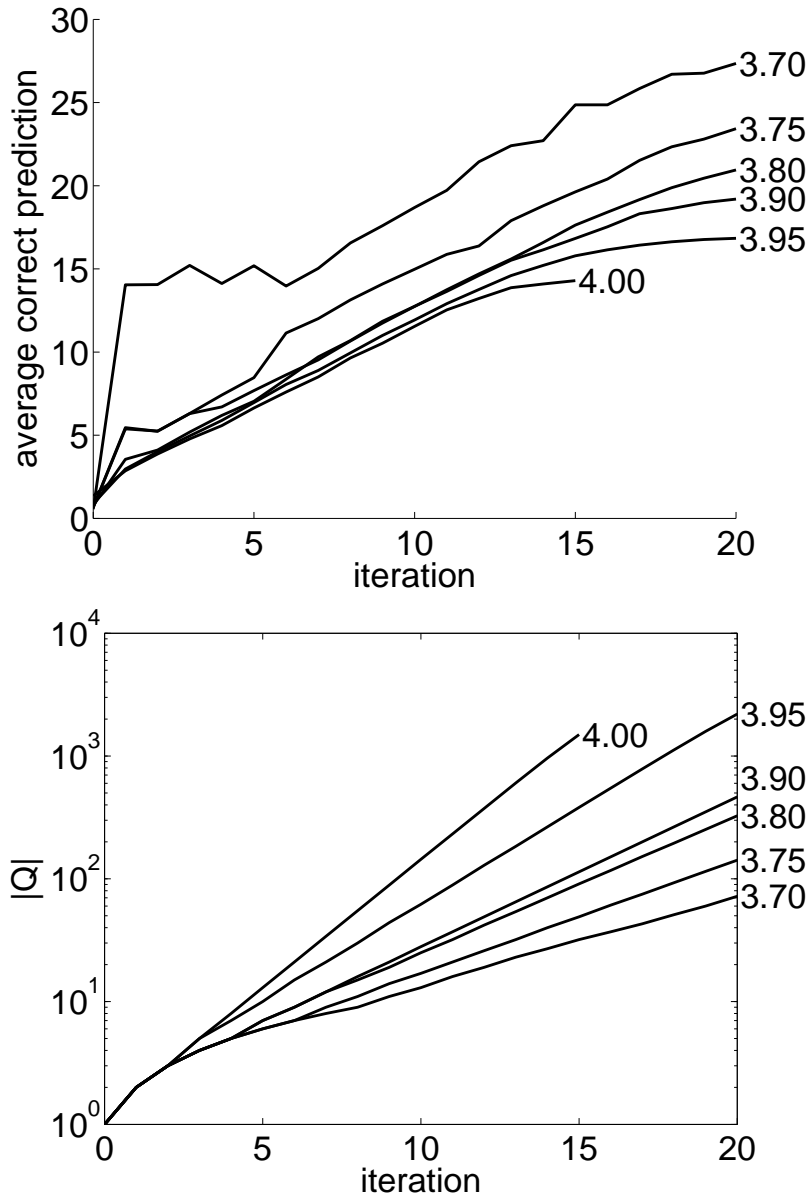


Figure 9: Some results of CrySSMEx modelling chaotic systems from extracted ssm^0 to ssm^{20} . The top diagram shows the average number of correctly predicted symbols before the SSM failed to predict the output symbol of the system. The lower diagram shows the number of SEs of the extracted SSMs. When $a = 4.0$, the extraction was aborted at iteration 15 due to limited memory resources.

are also many possibilities for how to improve `CrySSMEx`, e.g. through CVQ refinement, quantization of SDTDS input and extension to indeterministic SDTDSs, that are here not discussed further.

The most critical issue for now is perhaps the theoretical understanding of the algorithm and there are at least two central decisions in the algorithm that have been made on heuristic grounds:

- NDI-equivalent SEs are now grouped using UNDI-equivalence (Definition 11). There is, however, more than one way to group NDI-equivalent SEs if the relation is non-transitive. The chosen solution is only one possible, quite restrictive, way. For example, if SE pairs q_1 and q_2 and q_2 and q_3 are NDI-equivalent respectively while q_1 and q_3 are not, then hypothesis equivalence sets are $\{\{q_1\}, \{q_2\}, \{q_3\}\}$, $\{\{q_1, q_2\}, \{q_3\}\}$ or $\{\{q_1\}, \{q_2, q_3\}\}$, of which only the first, which results in no merge, is generated by UNDI-equivalence.
- When data is collected in `collect_split_data` (Algorithm 2) a single input symbol is selected based on conditional entropy. It is, however, possible that another symbol should be selected, or that more than one symbol should be included. The selected symbol affects very strongly what the model vectors will be, and even if the seemingly most informative symbol is selected, the selection mechanism includes no heuristics about the underlying geometrical consequences of the decision.

The solution to both these issues should involve more than just finding alternatives to UNDI-equivalence and entropy-based selection of input symbols. I suspect that it may involve systematic testing of merging and splitting in a breadth first search manner. It may even be necessary to split one SE at a time.

In this respect, it is perhaps reasonable to consider `CrySSMEx` a promising first step towards a more generic approach, rather than a final solution to the problem of RNN-RE. Moreover, the fact that the algorithm performs quite well on complex domains while there still are obvious ways to improve it can also be considered quite valuable.

There are at this point no mathematical proofs that `CrySSMEx` will always provide the expected results, and clearly, some of the experiments demonstrate that it will not. A proof should distinguish the set of problems that can be solved by `CrySSMEx` from the ones that cannot. Therefore a proof, or at least a deep theoretical analysis of the algorithm is important. But such an analysis will arguably, since the parts are tightly integrated, require a merge of theories surrounding all `CrySSMEx`-constituents, which brings together ideas from areas such as automata theory (Hopcroft & Ullman, 1979), stochastic machines (Paz, 1971), information theory (Cover & Thomas, 1990) and cluster analysis (Everitt et al., 2001). There are of course also strong connections with the highly developed mathematical field of dynamic systems theory (Devaney, 1992), especially within the context of symbolic dynamics (Crutchfield, 1994). And the whole procedure of generating minimal algorithms (i.e. in this case SSMs) to explain a source of data (i.e. Ω) is of course related to algorithmic information theory (Chaitin, 1987).

The algorithmic complexity remains an open issue too. The experiments clearly shows how evasive this issue is. For example, the analysis of an RNN of 10^3 -dimensional state space ended up in a very modest SSM (Section 5.3) whereas a chaotic one-dimensional autonomous systems generated enormous SSMs (Section 5.4). The SSM size will for chaotic systems be bounded by $|\Omega|$, but such an answer is quite unsatisfying since the algorithm should typically be terminated before it memorizes the entire data set. Given that the system is not chaotic, however, the computational complexity issue will be arguably more interesting, but at the same time very difficult to analyse since there are some deeply challenging factors to include, e.g. properties of γ (of the SDTDS) in combination with the selected input sequences.

7 Discussion and conclusions

7.1 Relation to earlier work

Extraction of deterministic SSMs using `CrySSMEx` has now been shown to be possible for a number of challenging domains. Extraction of stochastic SSMs from a chaotic system was also shown to be possible. The domains on which earlier approaches have been tested have, almost exclusively, been relatively simple binary regular grammars (cf. Jacobsson, 2005). Arguably, the context free domain, the high dimensional SRN and the chaotic system tested in this article, all constitute significantly more difficult problems.

As discussed in the introduction, there are three main differences between `CrySSMEx` and earlier approaches: the SSM, the CVQ and the integration of quantization, observation and minimization. These three ingredients make `CrySSMEx` more efficient than earlier algorithms simply because `CrySSMEx` performs a directed and deterministic search for a minimal quantization of the state space. Earlier approaches have relied on quantizers to find this minimal quantization without any information about the underlying dynamic system context of the state space.

Another difference to most earlier approaches is that `CrySSMEx` is parameter free (apart from Λ_o , which is typically, however, derivable from the RNN domain context). This is quite an advantage, since in the use of the algorithm as an analysis tool, the results are guaranteed not to be coloured by the choice of parameters. Since `CrySSMEx` is also deterministic, there is no need for running it more than once on the same data.

Another main feature is that the algorithm quickly creates an initial coarse stochastic model which it then refines gradually until a deterministic model is found (if possible). This “anytime rule extraction” possibility was considered by Craven and Shavlik (1999) to be an important aspect with respect to the scalability of the algorithms.

The algorithm can also handle missing data due to the substochastic nature of the extracted model, which is important since it uses observation of a system to build models. Observations that may, or may not, include all relevant aspects of the underlying system.

Another distinguishing feature of `CrySSMEx` as compared to most other RE algorithms is that the hypothesis space includes the system space, i.e. that SSMs is a subset of SDTDSs. I believe this is something quite unusual for RE algorithms but could turn out to be very fruitful. I have already used this feature for some verification of `CrySSMEx`; two SSMs, of which one is extracted from the other, should generally be equivalent to each other. One may also, if not satisfied with reading nondeterministic SSMs, always attempt to extract a deterministic description of them. Since the SDTDS definition is so abstract, other interesting systems also fall into this class, e.g. backpropagation learning if $I = \emptyset$, $O = \emptyset$ and S is the weight space (Λ_o could then simply be the enumeration of SSMs, `CrySSMEx`-extracted from the network).

Last, but not least, the extraction results not only in a machine, but also in a hierarchically structured geometrical organization of the state space of the underlying system (cf. the pure black box model of Vahed and Omlin (2004)). Intuitively, the relation between the structure of the CVQ graph, the topology of SDTDS state space, and the SSM should contain important seeds for further development of `CrySSMEx` and deeper analysis of the underlying system.

7.2 Related fields

Following previous arguments (cf. Section 1.1) that a rule extraction algorithm should not even assume that the underlying system is a neural network, I would argue that RNN-RE should *not* be considered a field of neural computation; it should be considered a field of machine learning *applied to* models of neural computation (cf. Craven and Shavlik (1994)). The consequence is that related algorithms are not primarily found in the literature of neural computation (apart from pure RNN-RE algorithms).

One important field, not immediately associated with machine learning, is control theory. Especially when it comes to the system identification aspect of control theory, one suggested definition could as well have been for RE: “System identification deals with the problem of building mathematical models of dynamical systems based on observed data from the system” (Ljung (1999) p. 1). In control theory (e.g. Young & Garg, 1995; Marculescu, Marculescu & Pedram, 1996; Kumar & Garg, 2001), and especially for *discrete event systems*, similar problems as for RNN-RE-algorithms have been dealt with for a long time. There are, however, some distinguishing features that separates **CrySSMEx** from algorithms of control theory. For example: the assumed full observability, discrete time and determinism of the underlying system.

To mature, however, the RNN-RE field needs to take into account the well developed theory of this related field. But once the connection is made to control theory, there are an abundance of other (some partly overlapping) fields that also needs to be taken into account: e.g. inductive logic programming (e.g. Muggleton & Raedt, 1994), grammar induction (e.g. Moore, 1956; Gold, 1967; Lang, 1992; de la Higuera, 2005), computational learning theory (e.g. Valiant, 1984; Angluin, 1987, 2004), symbolic dynamics (Crutchfield, 1994), computational scientific/mathematical discovery (e.g. Simon, 1995/96; Langley, Shrager & Saito, 2002; Colton, Bundy & Walsh, 2000), closed loop discovery (a.k.a. active learning) (e.g. Cohn, Atlas & Ladner, 1994; Bryant, Muggleton, Page & Sternberg, 1999), software testing (Bergadano & Gunetti, 1996), data mining, etc. Taken together, these fields form an almost insurmountable abundance of literature (only very few examples are cited here). Most likely, there are other fields that are also important to take into account (cf. references in Section 6).

Some of the goals of these fields are widely different. For example, the goal for system identification is to better facilitate control of the underlying system whereas for software testing, it is to find errors. The terminology is also very diversified; the underlying systems may be called plants, machines and dynamic systems in control theory, an abstract teacher in computational learning theory, or an interactive user in inductive logic programming. The process is about system identification, model induction, scientific discovery or data mining, etc. The hypothesis space of the induced models also varies from differential equations, finite state machines, statements about systems and ad hoc representations of engineering problems.

After a brief review of the leading papers and books of the field, it becomes obvious from the lack of cross-referencing that the potential connections are not fully exploited. Yet all these fields have one thing in common with rule extraction; one of their central goals is to automatically induce models, conjectures, concepts and predictions based on observations.

The exact nature of the similarities and differences of these fields to each other and to RNN-RE is out of the scope of this paper, however. But, since the goals of these fields overlap with science in general, I would suggest that a natural way to bring these fields closer together could be to build an encompassing theory by taking advantage of the deep insights philosophers of science already have provided us with (e.g. Simon, 1973; Williamson, 2004).

7.3 Future directions and final thoughts

There are three major reasons for the choice of the word “crystallizing” in this work: (1) as mentioned before, when a uniformly initiated SSM parses a sequence of inputs, its SE distribution gradually “crystallizes” into a more ordered distribution, (2) the quantization of the state space geometrically resembles the process of crystallization, and (3) the whole **CrySSMEx** process also conceptually “crystallizes” a model of the underlying system from a rough and unordered model to a more refined model. Another reason is, of course, that one ambition with **CrySSMEx** is to make otherwise opaque dynamic systems *crystal clear* for human observers.

To what degree this last ambition can be fulfilled by **CrySSMEx** remains perhaps unan-

swered in this article. The comprehensibility of extracted rules has always been of central concern for rule extraction techniques (Andrews et al., 1995). But, as a consequence of the fidelity optimization, the rule set (i.e. the number of SEs and transitions) can become very big. This may of course reduce our capacity to comprehend the rules, but this is a consequence of embracing Albert Einstein’s famous motto: “A scientific theory should be as simple as possible, but no simpler”. The extracted machines are just that, as simple as possible, but not so simple that they deviate from the behaviour of the underlying system for the sake of comprehensibility.

I would, however, argue that to sacrifice comprehensibility for the sake of fidelity may be a route towards something more significant for science than current RNN-RE techniques are. As presented in this paper, **CrySSMEx** is postprocessing pregenerated data. It would, however, be quite straightforward to let Ω be resampled based on the extracted SSMs in such a way that data is gathered about dead transitions, or about infrequently visited SEs (cf. active learning (Cohn et al., 1994; Bryant et al., 1999)). The process of model-based data selection is already an ingredient of **CrySSMEx** (Algorithm 2), but by letting the extractor “interrogate” the underlying system, the empirical loop would thereby be completed. I would propose to call this kind of algorithm an “Empirical Machine” (Jacobsson & Ziemke, 2005). The generation of models and selection of data must, however, be based on an efficient strategy in order to minimize the number of queries needed by the machine to build fruitful models. A possible such strategy would be to follow Popper’s ideas that theories must be falsifiable and generate hypotheses regarding the extracted models (of singular or multiple SDTDSs) (Popper, 1990). Of special interest is the relation of falsifiability to “universality” and “precision”, two properties that could probably be quite straightforwardly translated into quantifiable properties of the models. Universality and precision could then be two goals for theories “competing” for empirical data that can falsify them; a framework I would suggest to call a “Popperian Machine” (Jacobsson & Ziemke, 2005).

One thing that distinguishes RNN-RE from most of the suggested related fields makes it especially promising: it focuses on simulated, and thus empirically accessible, systems. Many of the practical problems of induction are simply not present for SDTDSs; e.g. lack of data, noisy data and partial observations. Moreover, since an SDTDS is simulated, it is widely open for active learning since no costly actuators and sensors need to be situated in a real world environment.

The counterargument is of course that the interesting scientific problems are “out there”, in the challenging and noisy reality. Why would we automate scientific discovery within comparably uninteresting simulated universes? There is only one *reality*, whereas there are an infinity of possible simulated realities, so why not focus on the reality that counts? My argument is that the reason is precisely that they *are* individually relatively uninteresting, numerous, and still complex. Some reasons for wanting to automate a process are that it is too exhaustive, too repetitive, too unrewarding and yet too challenging for humans. Another fundamental requirement is that the process should be *possible* to automate, and the properties of these systems clearly make it so. Moreover, since the goal of many of the simulators is to *mimic* the reality, an analysis tool tailored to those systems may at a later stage be ideal for analysing this reality.

As pointed out in the very beginning of this article, there are many important scientific fields with an abundance of simulated models. Models around which the same sound scientific methodology which generated them could be applied. Could the field of machine learning ask for more? These are toy world problems of scientific significance, created by respected researchers in need of automated assistance by, what could possibly be, Empirical and Popperian Machines.

Acknowledgments

I want to thank André Grüning, Andreas Hansson, Gunnar Buason, Amanda Sharkey and Tom Ziemke for many valuable comments on my manuscript. I also must thank the anonymous reviewers for helping me improve the paper significantly. I especially want to thank for the suggested connection of my work to control theory.

References

- Andrews, R., Diederich, J. & Tickle, A. B. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6), 373–389.
- Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, 75, 87–106.
- Angluin, D. (2004). Queries revisited. *Theoretical Computer Science*, 313(2), 175–194.
- Bergadano, F. & Gunetti, D. (1996). Testing by means of inductive program learning. *ACM Transactions on Software Engineering and Methodology*, 5(2), 119–145.
- Blair, A. & Pollack, J. (1997). Analysis of dynamical recognizers. *Neural Computation*, 9(5), 1127–1142.
- Bodén, M. & Wiles, J. (2000). Context-free and context-sensitive dynamics in recurrent neural networks. *Connection Science*, 12(3/4), 196–210.
- Bryant, C. H., Muggleton, S. H., Page, C. D. & Sternberg, M. J. E. (1999). Combining active learning with inductive logic programming to close the loop in machine learning. In *Proceedings of the AISB'99 symposium on AI and scientific creativity*. (informal proceedings)
- Casey, M. (1996). The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. *Neural Computation*, 8(6), 1135–1178.
- Chaitin, G. J. (1987). *Algorithmic information theory*. Cambridge University Press.
- Christiansen, M. H. & Chater, N. (1999). Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23(2), 157–205.
- Cleeremans, A., McClelland, J. L. & Servan-Schreiber, D. (1989). Finite state automata and simple recurrent networks. *Neural Computation*, 1, 372–381.
- Cohn, D. A., Atlas, L. & Ladner, R. E. (1994). Improving generalization with active learning. *Machine Learning*, 15(2), 201–221.
- Colton, S., Bundy, A. & Walsh, T. (2000). On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies*, 53(3), 351–375.
- Cover, T. M. & Thomas, J. A. (1990). *Elements of information theory*. John Wiley, New York.
- Craven, M. W. & Shavlik, J. W. (1994). Using sampling and queries to extract rules from trained neural networks. In W. W. Cohen & H. Hirsh (Eds.), *Machine learning: Proceedings of the eleventh international conference*. San Fransisco, CA: Morgan Kaufmann.
- Craven, M. W. & Shavlik, J. W. (1996). Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*, 8, 24–30.
- Craven, M. W. & Shavlik, J. W. (1999). *Rule extraction: Where do we go from here?* (Tech. Rep. No. Machine Learning Research Group Working Paper 99-1). Department of Computer Sciences, University of Wisconsin.
- Crutchfield, J. P. (1994). The calculi of emergence: Computation, dynamics, and induction. *Physica D*, 75, 11–54.
- Crutchfield, J. P. & Young, K. (1990). Computation at the onset of chaos. In W. Zurek

- (Ed.), *Complexity, entropy and the physics of information*. Addison-Wesley, Reading, MA.
- de la Higuera, C. (2005). A bibliographical study of grammatical inference. *Pattern Recognition*, 38, 1332–1348.
- Devaney, R. L. (1992). *A first course in chaotic dynamical systems*. Addison-Wesley.
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14, 179–211.
- Everitt, B. S., Landau, S. & Leese, M. (2001). *Cluster analysis*. London: Arnold.
- Gers, F. A. & Schmidhuber, J. (2001). LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, 12(6), 1333–1340.
- Giles, C. L., Miller, C. B., Chen, D., Chen, H. H. & Sun, G. Z. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3), 393–405.
- Giles, C. L., Miller, C. B., Chen, D., Sun, G. Z., Chen, H. H. & Lee, Y. C. (1992). Extracting and learning an unknown grammar with recurrent neural networks. In J. E. Moody, S. J. Hanson & R. P. Lippmann (Eds.), *Advances in neural information processing systems* (Vol. 4, pp. 317–324). Morgan Kaufmann Publishers, Inc.
- Gold, M. E. (1967). Language identification in the limit. *Information and Control*, 10(5), 447–474.
- Hammer, B. & Tiño, P. (2003). Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, 15(8), 1897–1929.
- Hopcroft, J. & Ullman, J. D. (1979). *Introduction to automata theory, languages, and compilation*. Addison-Wesley Publishing Company.
- Jacobsson, H. (2005). Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6), 1223–1263.
- Jacobsson, H. & Ziemke, T. (2003a). Improving procedures for evaluation of connectionist context-free language predictors. *IEEE Transactions on Neural Networks*, 14(4), 963–966.
- Jacobsson, H. & Ziemke, T. (2003b). *Reducing complexity of rule extraction from prediction RNNs through domain interaction* (Tech. Rep. No. HS-IDA-TR-03-007). Department of Computer Science, University of Skövde, Sweden.
- Jacobsson, H. & Ziemke, T. (2005). Rethinking rule extraction from recurrent neural networks. In A. d’Avila Garcez, J. Elman & P. Hitzler (Eds.), *IJCAI-05 workshop on neural-symbolic learning and reasoning*.
- Kolen, J. F. & Kremer, S. C. (Eds.). (2001). *A field guide to dynamical recurrent networks*. IEEE Press.
- Kremer, S. C. (2001). Spatiotemporal connectionist networks: A taxonomy and review. *Neural Computation*, 13(2), 248–306.
- Kumar, R. & Garg, V. K. (2001). Control of stochastic discrete event systems modeled by probabilistic languages. *IEEE Transactions on Automatic Control*, 46(4), 593–606.
- Lang, K. J. (1992). Random DFA’s can be approximately learned from sparse uniform examples. In *Proceedings of the fifth ACM workshop on computational learning theory* (pp. 45–52). New York.
- Langley, P., Shrager, J. & Saito, K. (2002). Computational discovery of communicable scientific knowledge. In L. Magnani, N. J. Nersessian & C. Pizzi (Eds.), *Logical and computational aspects of model-based reasoning*. Dordrecht: Kluwer Academic.
- Ljung, L. (1999). *System identification: theory for the user*. Prentice Hall.
- Manolios, P. & Fanelli, R. (1994). First order recurrent neural networks and deterministic finite state automata. *Neural Computation*, 6(6), 1155–1173.
- Marculescu, D., Marculescu, R. & Pedram, M. (1996). Stochastic sequential machine synthesis targeting constrained sequence generation. In *DAC’96: Proceedings of the 33rd annual conference on design automation* (pp. 696–701). New York, NY, USA: ACM Press.

- McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115–133.
- Moore, E. F. (1956). Gedanken-experiments on sequential machines. In C. E. Shannon & J. McCarthy (Eds.), *Annals of mathematical studies* (Vol. 34, pp. 129–153). Princeton University Press.
- Muggleton, S. & Raedt, L. D. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20, 629–679.
- Paz, A. (1971). *Introduction to probabilistic automata*. Academic Press.
- Popper, K. R. (1990). *The logic of scientific discovery* (14 ed.). London: Unwin Hyman. (Originally published 1959)
- Rabin, M. O. (1963). Probabilistic automata. *Information and Control*, 6, 230–245.
- Sharkey, N. E. & Jackson, S. A. (1995). An internal report for connectionists. In R. Sun & L. A. Bookman (Eds.), *Computational architectures integrating neural and symbolic processes* (pp. 223–244). Kluwer, Boston.
- Simon, H. A. (1973). Does scientific discovery have a logic? *Philosophy of Science*, 40, 471–480.
- Simon, H. A. (1995/96). Machine discovery. *Foundations of Science*, 1(2), 171–200.
- Tickle, A. B., Andrews, R., Golea, M. & Diederich, J. (1998). The truth will come to light: directions and challenges in extracting the knowledge embedded within mined artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6), 1057–1068.
- Tiño, P. & Köteles, M. (1999). Extracting finite-state representations from recurrent neural networks trained on chaotic symbolic sequences. *IEEE Transactions on Neural Networks*, 10(2), 284–302.
- Tiño, P., Čerňanský, M. & Beňušková, L. (2004). Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, 15(1), 6–15.
- Tiño, P. & Vojtek, V. (1998). Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences. *Neural Network World*, 8(5), 517–530.
- Tonkes, B., Blair, A. & Wiles, J. (1998). Inductive bias in context-free language learning. In *Proceedings of the ninth Australian conference on neural networks* (pp. 52–56). Brisbane: Department of Computer Science and Electrical Engineering, University of Queensland.
- Vahed, A. & Omlin, C. W. (2004). A machine learning method for extracting symbolic knowledge from recurrent neural networks. *Neural Computation*, 16, 59–71.
- Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11), 1134–1142.
- Watrous, R. L. & Kuhn, G. M. (1992). Induction of finite-state automata using second-order recurrent networks. In J. E. Moody, S. J. Hanson & R. P. Lippmann (Eds.), *Advances in neural information processing systems* (Vol. 4, pp. 309–317). Morgan Kaufmann Publishers, Inc.
- Wiles, J. & Elman, J. L. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent neural networks. In *Proceedings of the seventeenth annual conference of the cognitive science society* (pp. 482–487). Cambridge MA: MIT Press.
- Williamson, J. (2004). A dynamic interaction between machine learning and the philosophy of science. *Minds and Machines*, 14(4), 539–549.
- Young, S. & Garg, V. K. (1995). Model uncertainty in discrete event systems. *SIAM Journal on Control and Optimization*, 33(1), 208–226.
- Zeng, Z., Goodman, R. M. & Smyth, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6), 976–990.

A Substochastic vectors

Some important types of, and operations on, substochastic vectors will here be defined (some of these are also found in Paz (1971)):

Definition 19 A *substochastic vector* \vec{v} is a vector where all elements are nonnegative and the sum of the elements is ≤ 1 .

A special case of the substochastic distribution is where all probabilities are zero:

Definition 20 An *exhausted substochastic vector* \vec{v} is the special case of a substochastic vector where all elements are 0.

And, as another special case, we find vectors with more conventional probabilistic properties:

Definition 21 A *stochastic vector* \vec{v} is the special case of a substochastic vector where the sum of the elements is exactly 1.

And a special case of stochastic vectors is where only one element is probable:

Definition 22 A *degenerate vector* is a stochastic vector one element with probability 1 and the rest 0.

Definition 23 The entropy of an n -dimensional substochastic vector \vec{v} is here denoted as $H(\vec{v})$ and is calculated by

$$H(\vec{v}) = - \sum_{i=1}^n \vec{v}_i \log \vec{v}_i$$

Definition 24 The function `normalize` is used to transform a substochastic vector into a stochastic vector, if possible, according to

$$\text{normalize}(\vec{v}) = \begin{cases} \frac{\vec{v}}{\sum_{i=1}^n \vec{v}_i} & \text{if } \sum_{i=1}^n \vec{v}_i > 0 \\ \vec{v} \cdot 0 & \text{otherwise} \end{cases}$$

Definition 25 The *support set* of a substochastic vector $\vec{v} = (\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n)$ is the set $\{i : \vec{v}_i > 0\}$ and is denoted $\text{sup}(\vec{v})$.

B List of abbreviations

CrySSMEx	Crystallizing SSM Extractor
CVQ	Crystalline Vector Quantizer
NDI-equivalence	Not Decisively-Inequivalent
RE	Rule Extraction
RNN	Recurrent Neural Network
RNN-RE	RNN specific RE
SDTDS	Situated Discrete Time Dynamic System
SE	State Element (of an SSM)
SSM	Substochastic Sequential Machine
UNDI-equivalent	Universally NDI-equivalent
VQ	Vector Quantizer
Λ	Quantizer function
Ω	Transition event set (from an SDTDS)

Table 1: List of important abbreviations.