Mathematical Structures in Computer Science (2023), 1–20
doi:10.1017/S0960129522000421

PAPER

Weighted synchronous automata[†]

Leandro Gomes[1,*], Alexandre Madeira[2] and Luis Soares Barbosa[3] Q1

[1]Université de Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France, [2]CIDMA, University of Aveiro, Aveiro, Portugal and [3]HASLab INESC TEC, University of Minho, INL, Braga, Portugal Q2
[*]Corresponding author. Email: leandrogomes.moreiragomes@univ-lille.fr

(Received 8 August 2021; revised 25 November 2022; accepted 10 December 2022)

Abstract
This paper introduces a class of automata and associated languages, suitable to model a computational paradigm of fuzzy systems, in which both *vagueness* and *simultaneity* are taken as first-class citizens. This requires a weighted semantics for transitions and a precise notion of a synchronous product to enforce the simultaneous occurrence of actions. The usual relationships between automata and languages are revisited in this setting, including a specific Kleene theorem.

Keywords: Weighted automata; regular languages; synchronous languages

1. Introduction

The notion of an automaton (Kleene 1956), as the *de facto* mathematical abstraction of a computational process over a discrete state space, is constantly revisited to capture different sorts of computational behaviours in the most varied contexts, either prescribed in a program or discovered in Nature. Already in 1997, Milner (2006) emphasised that *from being a prescription for how to do something – in Turing's terms a 'list of instructions', software becomes much more akin to a description of behaviour, not only programmed on a computer, but occurring by hap or design inside or outside it.* Over time different kinds of automata were proposed *generate* (or *recognise*, depending on the perspective) such behaviours (or the languages that express them). Regular expressions, as a basic notation to express languages and behaviours, were first axiomatised by Kozen (1990) as Kleene algebras, which are basically partially ordered, semirings endowed with a closure operator. Several interpretations and variants of this structure are documented in the literature (Hoare et al. 2011; Jipsen and Andrew Moshier 2016; Kozen and Mamouras; Kozen 1997; McIver et al. 2006, 2013; Qiao et al. 2008; Thiemann 2016).

This paper was born out of a challenge: having previously worked with the Fuzzy Arden Syntax (FAS) (Gomes et al. 2021), a fuzzy, imperative language used for medical diagnosis and prescription of medical procedures, our aim was to introduce a specific kind of automata, and corresponding languages, are able to express the behaviour of the underlying fuzzy systems.

Two specific ingredients have to be taken into consideration. The first is *vagueness*, or *uncertainty*, a notion that underlies the interpretation of both variables and predicates in FAS programs. The second is *simultaneity*, i.e. a form of *parallel execution* which is not captured by

[†]This work is financed by the FCT – Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within the project IBEX, with reference PTDC/CCI-COM/4280/2021.

non-deterministic interleaving of elementary steps, as in typical models of concurrency. Consider, for illustration purposes, the following program.

Example 1.1.

```
if (Temperature is in Fever_condition)
then medicine:=5 else medicine:=0
```

The program adjusts the dose of medicine to be administrated to a patient depending on her temperature. The variable `Fever_condition` is a function assigning, to each real value of the temperature measured, a value (e.g. within the range $[0, 1]$) to record how close such temperature is of a 'fever condition'. In a scenario where the predicate `Temperature is in Fever_condition` and its negation have a value greater than 0, let us say, 0.4 and 0.6, respectively, the program executes both the `then` and the `else` blocks, weighted by the value associated to each of them. In practice, this results in a multiplication of the values taken, in each case, by variable `medicine`. Intuitively, the values 0.4 and 0.6 mean that `Temperature` has probably not reached the limit of a fever condition but is close to it.

Summing up, the intended semantics of a conditional statement in FAS does not reduce to a non-deterministic, or even to a probabilistic choice (McIver et al. 2013). Instead, it corresponds to a sort of parallel execution enforcing all branches to run in parallel, with (possibly) different weights associated to the evaluation of each condition. Therefore, as this small program illustrates, *vagueness* and *simultaneity* are the two ingredients our framework needs to deal with.

Vagueness can be captured by a *fuzzy* finite-state automata (FFA), a structure introduced in the 1960's in Wee and Fu (1969) to give a formal semantics to uncertainty and vagueness inherent to several computational systems. Different variants of this idea, e.g. incorporating fuzziness into either states or transitions, or both, are well documented in the literature (Doostfatemeh and Kremer 2005; Li and Pedrycz 2005; Liu et al. 2021; Mateescu et al. 1995). The corresponding *fuzzy* languages (Lee and Zadeh 1969; Zadeh 1996) are recognised by this class of automata only up to a certain membership degree. Applications are transversal to several domains as reported in Lin and Ying (2002), Mordeson and Malik (2002), Pedrycz and Gacek (2001), Ying (2002).

On the other hand, *simultaneity* was suitably formalised in what Milner called the 'synchronous version of CCS' – the SCCS calculus (Milner 1983), a variant of CCS (Milner 1980) where arbitrary actions are allowed to execute *synchronously*. This very same idea of synchronous evolution appears in the work of C. Priscariu on synchronous Kleene algebra (Prisacariu 2010). Models for such structures are given in terms of sets of synchronous strings and finite automata accepting them. These structures found application, for instance, in variants of deontic logic to formalise contract languages (Segerberg 1982; von Wright 1968) and of Hoare logic to reason about parallel synchronous programs with shared variables (Prisacariu 2010).

The aim of this paper is to formalise the behaviour of this class of systems. $\mathscr{H}$-automata are introduced as a variant of fuzzy transition automata in the spirit of reference (Mateescu et al. 1995), where transitions take 'truth' values in a complete Heyting algebra $\mathscr{H}$, and a suitable synchronous product construction is defined. The paper proceeds by generalising synchronous sets (Prisacariu 2010) into a notion of a *$\mathscr{H}$-synchronous language*, defined as a word valuation function over $\mathscr{H}$. Some preliminary results in this direction appeared in the authors' conference paper (Gomes et al. 2020). However, the formal framework was now completely redefined in a very general sense – note, for example, that the need for explicitly introducing $\mathscr{H}$-valued guards in the language, as suggested in that preliminary work, becomes redundant, i.e. implicit in the relevant mathematical strucutre and, thus, in the proposed language semantics.

As a main result it is shown that, for any complete Heyting algebra $\mathscr{H}$, $\mathscr{H}$-synchronous languages equipped with suitable language operators, as proposed here, defines a synchronous Kleene algebra. Moreover, its actions can generate a $\mathscr{H}$-automaton accepting precisely the $\mathscr{H}$-synchronous language that constitutes its interpretation. As in the classical, well-known case, a regular expression can be obtained from a $\mathscr{H}$-automaton by a standard state elimination procedure (Hopcroft et al. 2003). The procedure results in a $\mathscr{H}$-automaton with a single transition from the initial to the final state, labelled by an action α whose interpretation is precisely the language recognised by that $\mathscr{H}$-automaton.

This paper is organised as follows. The remaining of this section sums up related work and some preliminaries to the paper's contribution. Section 2 introduces $\mathscr{H}$-synchronous languages and defines a number of operators over them, proving that, in this way $\mathscr{H}$-synchronous languages forms a synchronous Kleene algebra. Section 3 studies $\mathscr{H}$-automata, including their synchronous product. A few examples of FAS programs involving conditionals are interpreted in this framework. Then, a Kleene theorem for $\mathscr{H}$-automata and $\mathscr{H}$-synchronous languages is proved in Section 4. Finally, Section 5 concludes and enumerates some topics for future research.

1.1 Related work

The construction of a finite fuzzy automata with membership degrees taken in a lattice-ordered monoid $\mathscr{L}$ is studied in Li and Pedrycz (2005) in a context analogous to the one considered here, based on the concept of *$\mathscr{L}$-fuzzy regular expression*. Those are defined as regular expressions from an alphabet X with a scalar $\lambda \in \mathscr{L}$ multiplication, which resorts to the monoid multiplication. It is precisely this scalar that attributes the weight to a transition in the automaton. In our approach, on the other hand, automata are built using standard regular expressions instead of fuzzy regular expressions. Regular expressions are then interpreted as some sort of weighted languages (i.e. functions with values on a complete Heyting algebra) accepted by an automaton with weighted transitions.

Most of the results presented in the context of fuzzy languages are constructed using either the real interval $[0, 1]$ or a generic residuated lattice to model the (possible) many valued membership values. Such is the case of reference (Mateescu et al. 1995). However, one of the main results of this paper, Theorem 1, relies on properties provided by a specific characterisation of the underlying lattice structure. In particular, the operator ';' has to be idempotent and commutative. The definition a $\mathscr{H}$-automaton proposed here differs from the one in Mateescu et al. (1995) with respect to the underlying semantic structure, which is assumed to be as a complete Heyting algebra.

Probabilistic automata (Rabin 1963), another approach to handle uncertainty, weight transi- **Q3**
tion by elements of a probability distribution. An equivalent to Kleene's theorem for these family of automata is presented by Bollig et al. (2012), considering (probabilistic) strings with probabilistic choice, guarded choice, concatenation and the star operator. Extensive surveys on this class of automata are documented in Vidal et al. (2005a,b). In the approach proposed here, however, *uncertainty* can be measured in an arbitrary, either discrete or continuous, domain, depending on the relevant application scenario. This is captured by a complete Heyting algebra introduced as a parameter in the model.

Figure 1 summarises some systems from the literature, highlighting the difference of the approach taken in this paper.

Moreover, as we summarised above, the notion of weight can take different meanings. Figure 2 summarises some of these different approaches.

1.2 Preliminaries

The notion of a synchronous Kleene algebra (SKA) plays in the automata construction introduced by Prisacariu (2010) a role similar to the one played by Kleene algebras in the classical case (Broda et al. 2013). Actually, SKA:

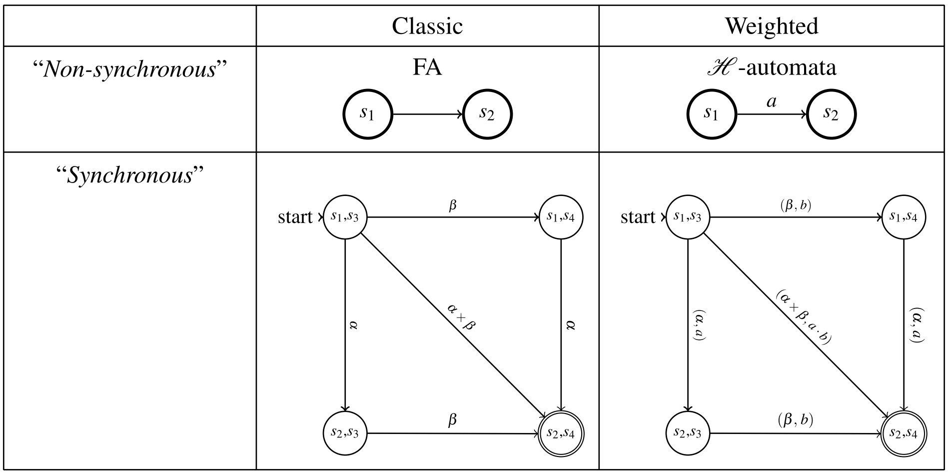


Figure 1. Simple representation of different classes of automata: finite state automata, their version with weights introduced in this paper ($\mathscr{H}$-automata) and the synchronous product (non-weighted Prisacariu 2010 and weighted).

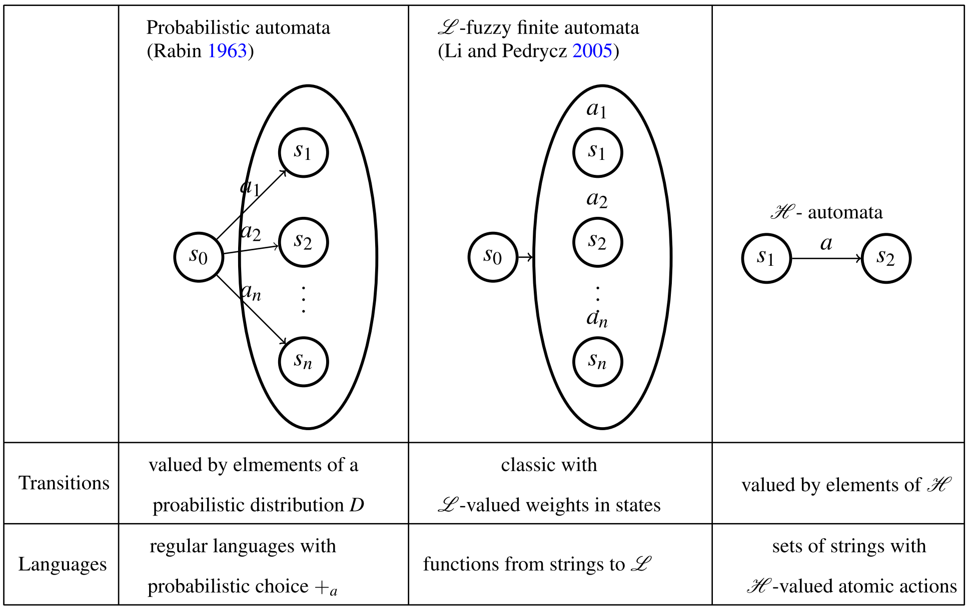


Figure 2. Taxonomy of related work. Values $a, a_1, \ldots a_n$ represent probabilities or weights.

- extends Kleene algebra with a synchronous operator to model synchronous execution of actions;
- has an interpretation over synchronous languages, the equivalent of regular languages to include actions corresponding to the synchronous execution of other actions;
- induce the construction of a class of finite automata, which accept the same languages that defines the interpretation of the SKA actions.

The relevant definitions are recalled below.

Definition 1. (Kleene algebra). *A* Kleene algebra $(A, +, \cdot, ^*, \mathbf{0}, \mathbf{1})$ *is an idempotent semiring with a unary operator '*' satisfying axioms (1)–(13) in Table 1. Partial order $\leq$ is induced by '$+$' as* $\alpha \leq \beta \Leftrightarrow \alpha + \beta = \beta$.

Well-known examples of Kleene algebras include the algebra of binary relations over a set, the set of all languages over an alphabet, and the $(min, +)$-algebra, also known as the tropical algebra, defined over the reals with an additional $+\infty$ constant, as

$$\mathbf{R} = (\mathbb{R}_0^+ \cup +\infty, min, +, ^*, +\infty, 0)$$

Extending this definition with a multiplication '$\times$' to capture the synchronous execution of actions[1] leads to the notion of a *synchronous Kleene algebra (SKA)*, introduced in Prisacariu (2010).

Definition 2. (SKA). *Let B be a set of labels. A* SKA *is a Kleene algebra extended with an operator '$\times$' i.e. a tuple*

$$\mathbf{S} = (A, B, +, \cdot, \times, ^*, \mathbf{0}, \mathbf{1})$$

where $B \subset A$, satisfying the axioms in Table 1.

$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$	(1)	$\alpha \cdot \gamma \leq \gamma \Rightarrow \alpha^* \cdot \gamma \leq \gamma$	(12)
$\alpha + \beta = \beta + \alpha$	(2)	$\gamma \cdot \alpha \leq \gamma \Rightarrow \gamma \cdot \alpha^* \leq \gamma$	(13)
$\alpha + \alpha = \alpha$	(3)	$\alpha \times (\beta \times \gamma) = (\alpha \times \beta) \times \gamma$	(14)
$\alpha + \mathbf{0} = \mathbf{0} + \alpha = \alpha$	(4)	$\alpha \times \beta = \beta \times \alpha$	(15)
$\alpha \cdot (\beta \cdot \gamma) = (\alpha \cdot \beta) \cdot \gamma$	(5)	$\alpha \times \mathbf{1} = \mathbf{1} \times \alpha = \alpha$	(16)
$\alpha \cdot \mathbf{1} = \mathbf{1} \cdot \alpha = \alpha$	(6)	$\alpha \times \mathbf{0} = \mathbf{0} \times \alpha = \mathbf{0}$	(17)
$\alpha \cdot (\beta + \gamma) = (\alpha \cdot \beta) + (\alpha \cdot \gamma)$	(7)	$a \times a = a\,, a \in B$	(18)
$(\alpha + \beta) \cdot \gamma = (\alpha \cdot \gamma) + (\beta \cdot \gamma)$	(8)	$\alpha \times (\beta + \gamma) = (\alpha \times \beta) + (\alpha \times \gamma)$	(19)
$\alpha \cdot \mathbf{0} = \mathbf{0} \cdot \alpha = \mathbf{0}$	(9)	$(\alpha + \beta) \times \gamma = (\alpha \times \gamma) + (\beta \times \gamma)$	(20)
$\mathbf{1} + (\alpha \cdot \alpha^*) = \alpha^*$	(10)	$(\alpha_\times \cdot \alpha) \times (\beta_\times \cdot \beta) = (\alpha_\times \times \beta_\times) \cdot (\alpha \times \beta)$, where	(21)
$\mathbf{1} + (\alpha^* \cdot \alpha) = \alpha^*$	(11)	$\alpha_\times, \beta_\times \in B^\times$, with $B^\times$ *the* $\times$ -closure of B.	

Table 1. Axiomatisation of a SKA (based on Prisacariu 2010)

Following a common practice, we write ab, rather than $a \cdot b$, for $a, b \in B$. Note that axiom (18) applies only to elements of B, instead of any arbitrary action A. This comes from the fact that such a property, being intuitive for atomic actions, is not so, or even desirable, for an arbitrary action in A. Consider, for example, action $(a + b) \times (a + b)$, whose execution may result in $a \times b$ by choosing a from the first entity and b from the second. However, by the axiomatisation above, we have

$$\begin{aligned}
& (a + b) \times (a + b) \\
= & \quad \{ (19) \} \\
& (a + b) \times a + (a + b) \times b \\
= & \quad \{ (20) \} \\
& (a \times a) + (b \times a) + (a \times b) + (b \times b)
\end{aligned}$$

$$= \quad \{ (15) \}$$

$$(a \times a) + (a \times b) + (a \times b) + (b \times b)$$

$$= \quad \{ (3) \text{ and } (18) \}$$

$$a + (a \times b) + b$$

Moreover, axiom (21) provides an exchange like rule to describe interaction between elements in $B^{\times}$ and A. The restriction to actions in $B^{\times}$ relates to the synchrony model, describing the parallelism of sequences of actions by concatenating small synchronous steps.

We will call by *synchronous regular expressions* the terms of a SKAs, i.e., the terms given the grammar

$$\alpha ::= a \mid \mathbf{0} \mid \mathbf{1} \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha \times \alpha \mid \alpha^*$$

where a is a *atomic action*, constituting the set B. Actions $\alpha_{\times}$, $\beta_{\times}$ are built only with operator '$\times$' from B, constituting the set $B^{\times}$ (e.g. a, $a \times b \in B^{\times}$ but $a + b$, $a \times b + c$, $\mathbf{0}$, $\mathbf{1} \notin B^{\times}$). The set of synchronous regular expressions will be denoted by Sreg.

If synchronous execution of actions is captured as above, vagueness, on the other hand, requires the consideration of weighted transitions forming a complete Heyting algebra.

Definition 3. (Complete Heyting algebra)**.** *A Heyting algebra is a bounded distributive lattice*

$$\mathscr{H} = (H, \vee, \wedge, \mathbf{0}, \mathbf{1}, \rightarrow)$$

with join '$\vee$' and meet '$\wedge$' operators, least '0' and greatest '1' elements, equipped with a binary operator '$\rightarrow$' that is right adjoint to '$\wedge$'. Some axioms are listed in Table 2.

$$a \wedge b = b \wedge a \tag{22}$$
$$a \wedge a = a \tag{23}$$
$$a \vee (a \wedge b) = a \tag{24}$$
$$a \wedge b \le c \Leftrightarrow b \le a \rightarrow c \tag{25}$$

Table 2. Part of the axiomatisation of a Heyting algebra

$\mathscr{H}$ *is a* complete Heyting algebra (CHA) *iff it is complete as a lattice, therefore entailing the existence of arbitrary suprema. The usual precedence of the operators, with '*' having the highest precedence, then ';', '$\times$', and finally '+', will be assumed.*

Let us briefly revisit some properties of this structure that will be later used in proofs. Completeness ensures that all suprema exist when characterising operators '$\cdot$', '$\times$' and '*' on $\mathscr{H}$-synchronous languages as (possible) infinite sums. Let us denote by $\bigvee$, $\bigwedge$ the distributed versions of the associative operators '$\vee$' and '$\wedge$', respectively, and by I a (possible infinite) index set. Axiom (25) ensures that every suprema distributes, on both sides, over arbitrary infima, i.e.

$$a \wedge \left(\bigvee_{i \in I} a_i\right) = \bigvee_{i \in I} (a \wedge a_i) \tag{26}$$

$$\left(\bigvee_{i \in I} a_i\right) \wedge a = \bigvee_{i \in I} (a_i \wedge a) \tag{27}$$

Instances of a complete Heyting algebra are enumerated in the following examples.

Example 1.2. (**2**- the Boolean algebra)**.** A first example is the well-known binary structure

$$\mathbf{2} = (\{\top, \bot\}, \vee, \wedge, \bot, \top, \rightarrow)$$

with the standard interpretation of Boolean connectives.

Example 1.3. A second example is the three-valued Gödel chain, which introduces an explicit denotation u for 'unknown' (or 'undefined'). $\mathbf{3} = (\{\top, u, \bot\}, \vee, \wedge, \bot, \top, \rightarrow)$ where

$\vee$	$\bot$	u	$\top$
$\bot$	$\bot$	u	$\top$
u	u	u	$\top$
$\top$	$\top$	$\top$	$\top$

$\wedge$	$\bot$	u	$\top$
$\bot$	$\bot$	$\bot$	$\bot$
u	$\bot$	u	u
$\top$	$\bot$	u	$\top$

$\rightarrow$	$\bot$	u	$\top$
$\bot$	$\top$	$\top$	$\top$
u	$\bot$	$\top$	$\top$
$\top$	$\bot$	u	$\top$

Example 1.4. (Gödel algebra)**.** Another example is given by the standard Gödel algebra $\mathbf{G} = ([0, 1], \max, \min, 0, 1, \rightarrow)$ where

$$x \rightarrow y = \begin{cases} 1, & \text{if } x \leq y \\ y, & \text{if } y < x \end{cases}$$

2. $\mathcal{H}$-Synchronous Languages

In order to capture both *synchronous execution* and *vagueness* in transitions, their interpretation is made over synchronous languages with embedded weights. The latter are taken, as explained above, from a complete Heyting algebra $\mathcal{H}$. In a sense, this generalises the work of Prisacariu (2010) which considers non-weighted, but synchronous languages. A number of operators over these languages, referred to as *$\mathcal{H}$-synchronous languages*, are introduced below, structuring this domain as a SKA, parametric on the set of weights.

Definition 4. ($\mathcal{H}$-synchronous languages)**.** *Let B be a set of symbols and $\mathcal{H}$ a complete Heyting algebra over a carrier H. $\mathcal{H}$-synchronous actions are pairs associating a non-empty set of symbols in B to a weight in H. Formally, $\Sigma = \mathcal{P}_{ne}(B) \times H \setminus \{\mathbf{0}\}$, where $\mathcal{P}_{ne}(X)$ denotes the non-empty powerset of X. For each action, functions $b : \Sigma \longrightarrow \mathcal{P}_{ne}(B)$ and $h : \Sigma \longrightarrow H \setminus \{\mathbf{0}\}$, denote the corresponding projections. $\mathcal{H}$-synchronous words are elements of Σ^*. $\mathcal{H}$-synchronous languages are sets of such words, i.e. elements of $\mathcal{P}(\Sigma^*)$.*

The weight of a word is computed by

$$hs : \Sigma^* \longrightarrow H, \;\; hs(u) = \bigwedge_{x \leftarrow u} h(x)$$

Clearly, $hs(\varepsilon) = \mathbf{1}$, for ε the empty string.

As an illustration, consider a finite set of labels $B = \{a, b\}$ and take the Gödel algebra $\mathbf{G}$, from Example 1.4, as a domain for weights. Thus, representing a sequence by the juxtaposition of its elements, $hs((\{a\}, 0.6)(\{a, b\}, 0.5)) = 0.6 \wedge 0.5 = 0.5$. Thus, one may turn a language $\mathcal{L} \in \mathcal{P}(\Sigma^*)$ into a function of synchronous words to weights through a translation function

$$t : \mathcal{P}(\Sigma^*) \longrightarrow H^{\mathcal{P}(B)^*}$$

such that

$$t(\emptyset) = ()$$

$$t(\mathscr{L}) = \left(\pi_1^*(w) \mapsto hs(w)\right)_{w \in \mathscr{L}}$$

Of course, t is not injective, thus two characterisations of a language, i.e. as an element of $\mathscr{P}(\Sigma^*)$ or of $H^{\mathscr{P}(B)^*}$, are not isomorphic. Function hs is particularly relevant to state, as we will do later, that an automata recognises a word w if it does so with a weight equal or bigger than $hs(w)$. **Q4**

The standard operators from regular language theory can be defined over $\mathscr{H}$-synchronous languages, as follows.

Definition 5. *The following operations are defined over $\mathscr{H}$-synchronous languages $\mathscr{L}$, $\mathscr{L}_1$, $\mathscr{L}_2$, for any complete Heyting algebra $\mathscr{H}$:*

- $\varnothing = \emptyset$ *(the empty language)*
- $1 = \{\varepsilon\}$ *(the language containing only the empty string)*
- $\mathscr{L}_1 + \mathscr{L}_2 = \mathscr{L}_1 \cup \mathscr{L}_2$
- $\mathscr{L}_1 \cdot \mathscr{L}_2 = \{uv |\ u \in \mathscr{L}_1, v \in \mathscr{L}_2\}$
- $\mathscr{L}_1 \times \mathscr{L}_2 = \{u \times v |\ u \in \mathscr{L}_1, v \in \mathscr{L}_2\}$*, where* $u \times v$ *is defined by*
  - $u \times \varepsilon = u = \varepsilon \times u$
  - $u \times v = (b(x) \cup b(y), h(x) \wedge h(y))(u' \times v')$ *where* $u = xu'$ *and* $v = yv'$.
- $\mathscr{L}^*$ *is the least fixed point of equation* $X = 1 + \mathscr{L} \cdot X$.

With respect to the product of languages, note that $\{a, b\} \in \mathscr{L}_1 \times \mathscr{L}_2$ may correspond to any of the following situations: $\{a\} \in \mathscr{L}_1$ and $\{b\} \in \mathscr{L}_2$, $\{b\} \in \mathscr{L}_1$ and $\{a\} \in \mathscr{L}_2$, or, finally, $\{a, b\}$ belongs just to one of the languages, and ε to the other. Note also that if $a \in \mathscr{L}_1$ and $bc \in \mathscr{L}_2$, then $\{a, b\}c \in \mathscr{L}_1 \times \mathscr{L}_2$.

Definition 6. (Atomic languages)**.** *Let B be a set of symbols and $\mathscr{H}$ a complete Heyting algebra over a carrier H and $\Sigma = \mathscr{P}_{ne}(B) \times H \setminus \{\mathbf{0}\}$ a set of synchronous actions. The* set of atomic actions of Σ *is given by $\Sigma_0 = \{a \in \Sigma \mid |b(a)| = 1\}$. For any atomic action $a \in \Sigma_0$, the language $\mathscr{L}_a = \{a\}$ is called* atomic language. *We denote by $\mathscr{B}_\Sigma$ the class of atomic languages of Σ and by $\mathscr{B}_\Sigma^\times$ the $\times$-closure of $\mathscr{B}$. We use also $\mathscr{L}_\Sigma$ to denote the class of the languages of Σ.*

Theorem 1. *Let B be a set of symbols and $\mathscr{H}$ a complete Heyting algebra over a carrier H and $\Sigma = \mathscr{P}_{ne}(B) \times H \setminus \{\mathbf{0}\}$ a set of synchronous actions. The structure*

$$L = (\mathscr{L}_\Sigma, \mathscr{B}_\Sigma, +, \cdot, \times, ^*, \varnothing, 1)$$

defines a SKA.

Proof. We detail the verification of axioms (1), (13), (20) and (18) making repeated use of Definition 5. The remaining cases follow a similar argument. For axiom (1) observe:

$$\begin{aligned}
& w \in \mathscr{L}_1 \cdot (\mathscr{L}_2 \cdot \mathscr{L}_3) \\
\Leftrightarrow\ & w = u \cdot v \text{ such that } u \in \mathscr{L}_1 \text{ and } v \in \mathscr{L}_2 \cdot \mathscr{L}_3 \\
\Leftrightarrow\ & w = u \cdot v \text{ such that } u \in \mathscr{L}_1 \text{ and } v = s \cdot t \text{ such that } s \in \mathscr{L}_2 \text{ and } t \in \mathscr{L}_3 \\
\Leftrightarrow\ & w = u \cdot s \cdot t \text{ such that } u \cdot s \in \mathscr{L}_1 \cdot \mathscr{L}_2 \text{ and } t \in \mathscr{L}_3 \\
\Leftrightarrow\ & w \in (\mathscr{L}_1 \cdot \mathscr{L}_2) \cdot \mathscr{L}_3
\end{aligned}$$

Regarding axiom (18), consider the atomic language $\mathscr{L}_a$. We have

$$\begin{aligned}
& w \in \mathscr{L}_a \times \mathscr{L}_a \\
\Leftrightarrow\ & w = a \times a \text{ such that } a \in \mathscr{L}_a \\
\Leftrightarrow\ & w = (b(a) \cup b(a), h(a) \wedge h(a)) \text{ such that } a \in \mathscr{L}_a
\end{aligned}$$

$$\begin{aligned} &\Leftrightarrow\ w = a \text{ such that } a \in \mathscr{L}_a \\ &\Leftrightarrow\ w \in \mathscr{L}_a \end{aligned}$$

For axiom (20),

$$\begin{aligned} & w \in (\mathscr{L}_1 + \mathscr{L}_2) \times \mathscr{L}_3 \\ \Leftrightarrow\ & u \in (\mathscr{L}_1 + \mathscr{L}_2) \text{ and } v \in \mathscr{L}_3 \text{ for some } u \text{ and } v \text{ such that } w = u \times v \\ \Leftrightarrow\ & u \in (\mathscr{L}_1 \cup \mathscr{L}_2) \text{ and } v \in \mathscr{L}_3 \text{ for some } u \text{ and } v \text{ such that } w = u \times v \\ \Leftrightarrow\ & u \in \mathscr{L}_1 \text{ or } u \in \mathscr{L}_2, \text{ and } v \in \mathscr{L}_3, \text{ for some } u \text{ and } v \text{ such that } w = u \times v \\ \Leftrightarrow\ & w \in \mathscr{L}_1 \times \mathscr{L}_3 \text{ or } w \in \mathscr{L}_1 \times \mathscr{L}_3 \\ \Leftrightarrow\ & w \in (\mathscr{L}_1 \times \mathscr{L}_3) + (\mathscr{L}_2 \times \mathscr{L}_3) \end{aligned}$$

For axiom (21), consider the Σ-languages $\mathscr{L}_1^\times, \mathscr{L}_2^\times \in \mathscr{B}^\times$. Then,

$$\begin{aligned} & w \in (\mathscr{L}_1^\times \cdot \mathscr{L}_1) \times (\mathscr{L}_2^\times \cdot \mathscr{L}_2) \\ \Leftrightarrow\ & w = (x \cdot u) \times (y \cdot v) \text{ such that } x \in \mathscr{L}_1^\times, u \in \mathscr{L}_1, y \in \mathscr{L}_2^\times, v \in \mathscr{L}_2 \\ \Leftrightarrow\ & (b(x) \cup b(y), h(x) \wedge h(y)) \cdot (u \times v) \text{ such that } x \in \mathscr{L}_1^\times, u \in \mathscr{L}_1, y \in \mathscr{L}_2^\times, v \in \mathscr{L}_2 \\ \Leftrightarrow\ & (x \times y) \cdot (u \times v) \text{ such that } (x \times y) \in \mathscr{L}_1^\times \times \mathscr{L}_2^\times, u \times v \in \mathscr{L}_2 \times \mathscr{L}_2 \\ \Leftrightarrow\ & w \in (\mathscr{L}_1^\times \times \mathscr{L}_2^\times) \cdot (\mathscr{L}_1 \times \mathscr{L}_2) \end{aligned}$$

□

Similarly to the homomorphism used to interpret Sreg as synchronous sets (Prisacariu 2010), we define a map to interpret actions of $\alpha \in$ Sreg as $\mathscr{H}$-synchronous languages.

Definition 7. (Sreg-interpretation)**.** *The function* $I : \text{Sreg} \to \mathscr{P}(\Sigma^*)$*, called a* Sreg-interpretation, *is defined as follows:*

$I(a) = \mathscr{L}_a,\ a \in B \times H$
$I(\mathbf{0}) = \varnothing$
$I(\mathbf{1}) = \chi$
$I(\alpha + \beta) = I(\alpha) \cup I(\beta)$
$I(\alpha \cdot \beta) = I(\alpha) \cdot I(\beta)$
$I(\alpha \times \beta) = I(\alpha) \times I(\beta)$
$I(\alpha^*) = I(\alpha)^*$

3. $\mathscr{H}$-Automata

This section presents the automata construction for $\mathscr{H}$-synchronous languages. First we define a class of automata on top of a complete Heyting algebra $\mathscr{H}$, referred as $\mathscr{H}$-automata. An appropriate notion of a synchronous product for these automata is then presented.

Definition 8. ($\mathscr{H}$-Automata)**.** *Let* $\mathscr{H}$ *be a complete Heyting algebra and* B *a set of symbols. A* $\mathscr{H}$-automaton *is a tuple*

$$\mathscr{M} = (S, \Sigma, s_0, F, \delta)$$

where:

- *S is a finite set of states;*
- *$\Sigma = \mathscr{P}_{ne}(B) \times H$ is the input alphabet;*

- *$s_0 \in S$ is the initial state;*
- *$F \subseteq S$ is the set of final states;*
- *$\delta : S \times \Sigma \times S \to H$ is the transition function.*

Intuitively, $\delta(s_1, x, s_2)$, for $x \in \Sigma$, can be interpreted as the truth degree of '*input x causing a transition from s_1 to s_2*'. In a graphical representation of a $\mathscr{H}$-automaton, the weight of a transition from s_1 to s_2 caused by an action a is represented explicitly as follows:

$$s_1 \xrightarrow{(a,\delta(s_1,a,s_2))} s_2$$

The transition function can be inductively extended to sequences Σ^* by defining $\delta^* : S \times \Sigma^* \times S \to H$ such that, for any $s_1, s_2 \in S$,

$$\delta^*(s_1, \varepsilon, s_2) = \begin{cases} \mathbf{1} & \text{if } s_1 = s_2 \\ \mathbf{0} & \textit{otherwise} \end{cases}$$

and, for any $s_1, s_2 \in S$, $w \in \Sigma^*$ and $x \in \Sigma$,

$$\delta^*(s_1, xw, s_2) = \bigvee_{s' \in S} (\delta(s_1, x, s') \wedge \delta^*(s', w, s_2))$$

Clearly, for any states $s_1, s_2 \in X$ and any word $w \in \Sigma^*$, $\delta^*(s_1, w, s_2)$ can be interpreted as the truth degree of '*word w causes a transition from s_1 to s_2*'.

A *recognising function* for a particular automaton $\mathscr{M}$ succeeds in recognising a word if, for each label $x \in \Sigma$ appearing in the word, the weight associated to the corresponding transition $\delta(s_1, x, s_2)$ is such that $h(x) \leq \delta(s_1, x, s_2)$. Formally,

Definition 9. (Recognising function)**.** *Let $\mathscr{H}$ be a complete Heyting algebra and $\mathscr{M} = (S, \Sigma, s_0, F, \delta)$ a $\mathscr{H}$-automata. The* recognising function *for $\mathscr{M}$, $\rho_{\mathscr{M}} : S \times \Sigma^* \times S \to H$, is recursively defined by*

$$\rho_{\mathscr{M}}(s_1, xw, s_2) = \begin{cases} \delta(s_1, x, s') \wedge \rho_{\mathscr{M}}(s', w, s_2) & \textit{if } h(x) \leq \delta(s_1, x, s') \\ \mathbf{0} & \textit{otherwise} \end{cases}$$

and

$$\rho_{\mathscr{M}}(s_1, \varepsilon, s_2) = \begin{cases} \mathbf{1} & \textit{if } s_1 = s_2 \\ \mathbf{0} & \textit{otherwise} \end{cases}$$

Definition 10. *Let $\mathscr{H}$ be a complete Heyting algebra, $\mathscr{M} = (S, \Sigma, s_0, F, \delta)$ be an $\mathscr{H}$-automaton, and $\rho_{\mathscr{M}}$ a recognising function for $\mathscr{M}$. The $\mathscr{H}$-synchronous language recognised by $\mathscr{M}$ is defined as follows:*

$$\mathscr{L}(\mathscr{M}) = \{w \in \Sigma^* | \rho_{\mathscr{M}}(s_0, w, s) > 0 \textit{ for some } s \in F\}$$

Theorem 2. *Let $\mathscr{H}$ be a complete Heyting algebra, $\mathscr{M} = (S, \Sigma, s_0, F, \delta)$ be an $\mathscr{H}$-automaton, and $\rho_{\mathscr{M}}$ a recognising function for $\mathscr{M}$ and $w \in \Sigma^*$. Then,*

$$w \in \mathscr{L}(\mathscr{M}) \textit{ iff } \rho_{\mathscr{M}}(s_0, w, s) \geq hs(w)$$

Proof. First observe that from the definitions of $\rho_{\mathscr{M}}$ and hs, for any $s, s' \in S$ and $w \in \Sigma^*$, either $\rho_{\mathscr{M}}(s, w, s') = 0$ or $\rho_{\mathscr{M}}(s, w, s') \geq hs(w)$. For the converse direction, since $\rho_{\mathscr{M}}(s, w, s') \geq hs(w)$, we have $\rho_{\mathscr{M}}(s, w, s') \geq 0$ and hence $w \in \mathscr{L}(\mathscr{M})$. $\square$

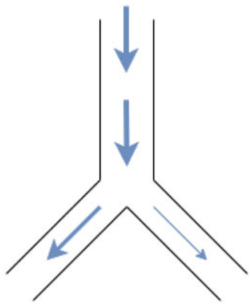


Figure 3. Conditional in FAS, illustrated by a liquid flowing through a 'Y-shaped' pipe.

We end this section defining and exemplifying a notion of synchronous product of $\mathscr{H}$-automata, which corresponds to the automata counterpart to synchronous composition in Sreg. It is based on the parallel product of labelled transition systems with shared actions. Formally,

Definition 11. (Synchronous product of $\mathscr{H}$-automata)**.** *Let* $\mathscr{M}_\alpha = (S^\alpha, \Sigma^\alpha, s_0^\alpha, F^\alpha, \delta^\alpha)$ *and* $\mathscr{M}_\beta = (S^\beta, \Sigma^\beta, s_0^\beta, F^\beta, \delta^\beta)$ *be two* $\mathscr{H}$*-automata. Let* $\Sigma^{\alpha\times\beta} = \Sigma^\alpha \cup \Sigma^\beta \cup (\Sigma^\alpha \times \Sigma^\beta)$*, with*

$$\Sigma^\alpha \times \Sigma^\beta = \{a \times b \mid a \in \Sigma^\alpha, b \in \Sigma^\beta\},$$

The synchronous product of $\mathscr{M}_\alpha$ and $\mathscr{M}_\beta$ *is the* $\mathscr{H}$*-automaton*

$$\mathscr{M}_{\alpha\times\beta} = (S^\alpha \times S^\beta, \Sigma^{\alpha\times\beta}, (s_0^\alpha, s_0^\beta), F^\alpha \times F^\beta, \delta^{\alpha\times\beta})$$

whose transition function

$$\delta^{\alpha\times\beta} : (S^\alpha \times S^\beta) \times \Sigma^{\alpha\times\beta} \times (S^\alpha \times S^\beta) \to H$$

is defined by, for any $p \in \Sigma^{\alpha\times\beta}$*,*

$$\delta^{\alpha\times\beta}((s^\alpha, s^\beta), p, (t^\alpha, t^\beta)) = \begin{cases} \delta^\alpha(s^\alpha, p, t^\alpha) & \text{if } p \in \Sigma^\alpha \setminus \Sigma^\beta \text{ and } s^\beta = t^\beta \\ \delta^\beta(s^\beta, p, t^\beta) & \text{if } p \in \Sigma^\beta \setminus \Sigma^\alpha \text{ and } s^\alpha = t^\alpha \\ \bigvee\limits_{\substack{a,b \\ p = a\times b}} (\delta^\alpha(s^\alpha, a, t^\alpha) \wedge \delta^\beta(s^\beta, b, t^\beta)) & \text{otherwise} \end{cases}$$

As discussed in the Introduction, $\mathscr{H}$-automata provide a suitable semantic structure for FAS programs. Let us illustrate such a potential through the discussion of two concrete examples.

Example 3.1. Our fist example, already mentioned in the introduction as Example 1.1, is that of a conditional in FAS which involves the simultaneous execution of its branches. An intuitive metaphor to this behaviour is represented as a pipe as depicted in Figure 3. The liquid, represented by blue arrows, reaches a point where it flows through both channels in parallel (capturing simultaneity), with different volumes going through each channel, represented by the different thicknesses of arrows (representing different truth degrees modelling vagueness).

The execution of this program, which involves the multiplication of the values assigned to variable `medicine` in different branches, may lead to two distinct outcomes:

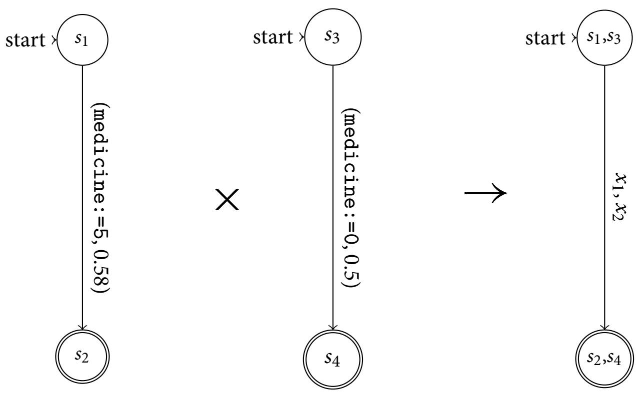
(A) The branches remain separated, and further instructions are executed in parallel. The information from different branches is taken into account by the user;
(B) The information is combined, which results in a single (crisp) output.

Option (B) enforce the consolidation of multiple variable values, which is achieved through instruction `aggregate`, as in the program:

```
if (Temperature is in Fever_condition)
then medicine:=5 else medicine:=0
aggregate;
```

This behaviour can be modelled by the synchronous product of the automata, assuming, for illustration purposes, that the weight of each branch are 0.58 and 0.50, respectively. We also assume that weights are taken from a Gödel algebra (Example 1.4).

Since we are assuming that the truth degrees associated to the evaluation of both branches are strictly positive, actions `medicine:=5` and `medicine:=0` run in parallel. Formally, the two automata are combined through '$\times$' giving rise to



where $x_1 = \big(\{\texttt{medicine:=5}, \texttt{medicine:=0}\}, \delta((s_1, s_3), \{\texttt{medicine:=0}, \texttt{medicine:=5}\}$ and $x_2 = (s_2, s_4)$

The truth degree associated to aggregated variable `medicine`, after execution depends on the truth degrees of both branches of the conditional, which corresponds to the second projection of the actions $(\texttt{medicine:=5}, \delta^{\texttt{medicine:=5}}(s_1, \texttt{medicine:=5}, s_2))$ and $(\texttt{medicine:=5}, \delta^{\texttt{medicine:=0}}(s_3, \texttt{medicine:=0}, s_4))$. Choosing the minimum function as the aggregation operator, leads to the following computation:

$$
\begin{aligned}
& \delta\big((s_1, s_3), \{\texttt{medicine:=5}, \texttt{medicine:=0}\}, (s_2, s_4)\big) \\
= & \big(\delta^{\texttt{medicine:=5}}(s_1, \texttt{medicine:=5}, s_2) \wedge \delta^{\texttt{medicine:=0}}(s_3, \texttt{medicine:=0}, s_4)\big) \\
= & \min\{0.58, 0.5\} \\
= & \ 0.5
\end{aligned}
$$

Example 3.2. As a second example consider an excerpt of a FAS program representing a control system intended to adjust the peak inspiratory pressure (PIP) of a patient depending on her levels of O_2 and CO_2, after a cardiac surgery de Bruin et al. (2018) .

```
if (O2 is in  O2_normal) and (CO2 is in  CO2_very_high)
then PIP:=5
elseif (O2 is in  O2_ low) and (CO2 is in  CO2_very_high)
then PIP:=2
```

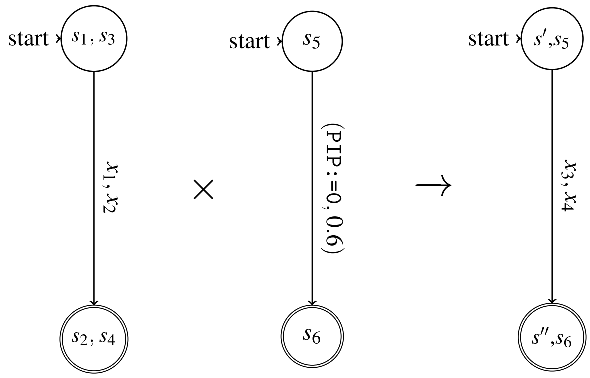
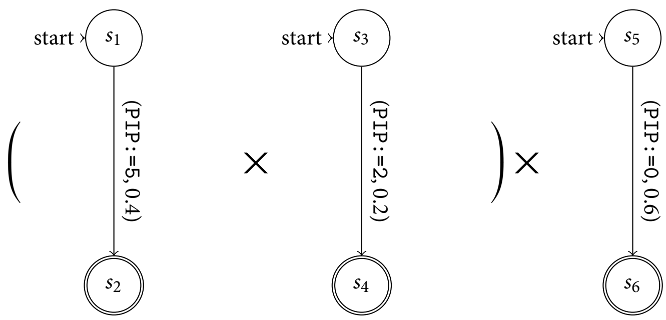


Figure 4. The synchronous product interpreting the FAS conditional.

```
if (O2 is in  O2_ low) and (CO2 is in  CO2_high)
then PIP:=0
aggregate;
```

The set of labels in this example is $B = \{\texttt{PIP:=5}, \texttt{PIP:=2}, \texttt{PIP:=0}\}$. Suppose that the truth degrees of the predicates in each of the branches of the conditional are 0.4, 0.2 and 0.6, respectively, and again assume the Gödel algebra as the domain for weights. The three branches of the conditional are modelled by the automata below

Again, the values of the three predicates are strictly positive and thus the three branches of the conditional are executed in parallel, corresponding to action $\texttt{PIP:=5} \times \texttt{PIP:=2} \times \texttt{PIP:=0}$. Operator '$\times$' being associative, such behaviour is modelled by the synchronous product of the three automata above, yielding



An intermediate step is represented in Figure 4, with $s' = (s_1, s_3)$, $s'' = (s_2, s_4)$, and (x_1, x_2) abbreviating $\big(\{\texttt{PIP:=5}, \texttt{PIP:=2}\}, \delta^{\texttt{PIP:=5}\times\texttt{PIP:=5}}\big((s_1, s_3), \{\texttt{PIP:=5}, \texttt{PIP:=2}\}, (s_2, s_4)\big)\big)$. Similarly, (x_3, x_4) abbreviates

$$\big(\{\texttt{PIP:=5}, \texttt{PIP:=2}, \texttt{PIP:=0}\}, \delta^{\texttt{PIP:=5}\times\texttt{PIP:=2}\times\texttt{PIP:=0}}\big((s', s_5), \{\texttt{PIP:=5}, \texttt{PIP:=2}, \texttt{PIP:=0}\}, (s'', s_6)\big)\big)$$

The truth degree corresponding to the combined values taken by variable PIP depends on three other truth degrees: the second projections of $(\texttt{PIP:=5}, \delta^{\texttt{PIP:=5}}(s_1, \texttt{PIP:=5}, s_2))$, $(\texttt{PIP:=2}, \delta^{\texttt{PIP:=2}}(s_3, \texttt{PIP:=2}, s_4))$ and $(\texttt{PIP:=0}, \delta^{\texttt{PIP:=0}}(s_5, \texttt{PIP:=0}, s_6))$. It is computed as follows:

$$\begin{aligned}
&\delta^{\texttt{PIP:=5}\times\texttt{PIP:=2}\times\texttt{PIP:=0}}\big((s_1, s_3), \{\texttt{PIP:=5}, \texttt{PIP:=2}, \texttt{PIP:=0}\}, (s'', s_6)\big)\\
=&\ \big(\delta^{\texttt{PIP:=5}}(s_1, \texttt{PIP:=5}, s_2) \wedge \delta^{\texttt{PIP:=2}}(s_3, \texttt{PIP:=2}, s_4) \wedge \delta^{\texttt{PIP:=0}}(s_5, \texttt{PIP:=0}, s_6)\big)\\
=&\ \min\{0.4, 0.2, 0.6\}\\
=&0.2
\end{aligned}$$

4. A Kleene Theorem for $\mathscr{H}$-Synchronous Languages

This section establishes a Kleene theorem for $\mathscr{H}$-automata and $\mathscr{H}$-synchronous languages. To proceed in such a direction, however, entails the need for showing that, as it happens in the classic case, the introduction of non-determinism and transitions labelled by the empty string does not compromise the expressiveness of finite $\mathscr{H}$-automata. Such is the aim of the following subsection.

4.1 $\mathscr{H}$-Automata with ε-moves

In standard finite automata theory, it is well-known that the introduction of non-determinism and the presence of ε-moves, i.e. spontaneous transitions labelled by the empty word, do not change the expressiveness of finite automata, since given a non-deterministic automaton with ε-moves, there is a standard procedure to build an equally finite and deterministic automaton recognising exactly the same language (see e.g. Hopcroft and Ullman 1979).

This subsection develops an analogous result for $\mathscr{H}$-automata. Firstly, we notice that the non-determinism is inherent to the very definition of $\mathscr{H}$-automata. For example, the non-deterministic transition $\delta(s, a) = \{w, v\}$ can be represented in a $\mathscr{H}$-automaton by $\delta(s, a, w) = 1$ and $\delta(s, a, v) = 1$. Of course, it is also easy to characterise the class of finite deterministic automata as the subclass of $\mathscr{H}$-automata such that, for each $s, v, w \in S$ and for any symbol a, if $\delta(s, a, v) = 1 = \delta(s, a, w)$ then $v = w$. This clarified, let us consider the effect of ε-moves.

Definition 12. ($\mathscr{H}$-Automata with ε-moves). *Let $\mathscr{H}$ be a complete Heyting algebra and B a set of symbols. A $\mathscr{H}$-*automata with ε-moves*, $\varepsilon\mathscr{H}$-automaton for short, is a tuple*

$$\mathscr{E} = (S, \Gamma, s_0, F, \delta)$$

where

- *S is a finite set of states;*
- *$\Gamma \subseteq \mathscr{P}(B) \times H$ such that, for any $a \in \Gamma$, if $b(a) = \emptyset$, $h(a) = 1$ (by a slight abuse of notation, the empty set of symbols will be represented by ε, originating transitions $(\varepsilon, 1)$);*
- *$s_0 \in S$ is the initial state;*
- *$F \subseteq S$ is the set of final states;*
- *$\delta : S \times \Gamma \times S \to H$ is the transition function such that*
    - *for any $s \in S$, $\delta(s, \varepsilon, s) = 1$*
    - *for any $s, s' \in S$, $\delta(s, \varepsilon, s') = 1$ or $\delta(s, \varepsilon, s') = 0$.*

Definition 13. *The language recognised by an $\varepsilon\mathscr{H}$-automaton $\mathscr{E} = (S, \Gamma, s_0, F, \delta)$ is given by*

$$\mathscr{L}^{\varepsilon}(\mathscr{E}) = \{w \in (\Gamma \setminus \{(\varepsilon, 1)\})^* \mid \rho^{\varepsilon}(s_0, w, s) > 0,\ \textit{for some } s \in F\} \tag{28}$$

where

$$\rho^{\varepsilon}(s_1, xw, s_2)) = \begin{cases} \delta^{\varepsilon}(s_1, x, s') \wedge \rho^{\varepsilon}(s', w, s_2) & \textit{if } h(x) \leq \delta^{\varepsilon}(s_1, x, s') \\ \mathbf{0} & \textit{otherwise} \end{cases} \tag{29}$$

$$\rho^{\varepsilon}(s_1, \varepsilon, s_2) = \begin{cases} \mathbf{1} & \textit{if } s_1 = s_2 \\ \mathbf{0} & \textit{otherwise} \end{cases} \tag{30}$$

with

$$\delta^{\varepsilon}(s, a, v) = \bigvee_{s_1, s_2 \in S} \big(\delta^*(s, \varepsilon^*, s_1) \wedge \delta(s_1, a, s_2) \wedge \delta^*(s_2, \varepsilon^*, v)\big) \tag{31}$$

for any $a \in \Gamma \setminus \{(\varepsilon, 1)\}$.

Definition 14. *Let* $\mathscr{E} = (S, \Gamma, s_0, F, \delta)$*, be a* $\varepsilon\mathscr{H}$*-automaton with* $\mathscr{E} \subseteq \mathscr{P}(B) \times H$*. The* ε*-closure of* $\mathscr{E}$ *is the* $\mathscr{H}$*-automaton*

$$\hat{\mathscr{E}} = (\hat{S}, \Sigma, \hat{s_0}, \hat{F}, \hat{\delta}) \tag{32}$$

where

- $\hat{S} = \{\hat{v} \mid v \in S\}$ *where* $\hat{v} = \{w \mid \delta^*(v, \varepsilon, w) = 1\}$
- $\Sigma = \Gamma \setminus \{(\varepsilon, 1)\}$
- $\hat{F} = \{P \in \hat{S} \mid P \cap F \neq \emptyset\}$
- *for any* $\hat{s}, \hat{v} \in \hat{S}$ *and* $a \in \Sigma$, $\hat{\delta}(\hat{s}, a, \hat{v}) = \bigvee_{s \in \hat{s}, v \in \hat{v}} \delta^{\varepsilon}(s, a, v)$*, where*

$$\delta^{\varepsilon}(s, a, v) = \bigvee_{s_1, s_2 \in S} \big(\delta^*(s, \varepsilon^*, s_1) \wedge \delta(s_1, a, s_2) \wedge \delta^*(s_2, \varepsilon^*, v)\big)$$

Theorem 3. *Let* $\mathscr{E} = (S, \Gamma, s_0, F, \delta)$*, be a* $\varepsilon - \mathscr{H}$*-automaton. Then*

$$\mathscr{L}^{\varepsilon}(\mathscr{E}) = \mathscr{L}(\hat{\mathscr{E}}) \tag{33}$$

Proof. First, observe that, for any $a \in \Gamma \setminus \{(\varepsilon, 1)\}$ and for all $s, v \in S$,

$$\hat{\delta}(\hat{s}, a, \hat{v}) \geq h(a) \Leftrightarrow \delta^{\varepsilon}(s, a, v) \geq h(a) \tag{34}$$

since

$$\begin{aligned}
& \hat{\delta}(\hat{s}, a, \hat{v}) \geq h(a) \\
\Leftrightarrow \quad & \{\ \hat{\delta} \text{ defn.}\} \\
& \bigvee_{s \in \hat{s}, v \in \hat{v}} \delta^{\varepsilon}(s, a, v) \geq h(a) \\
\Leftrightarrow \quad & \{\ \delta^{\varepsilon} \text{ defn.}\} \\
& \bigvee_{s \in \hat{s}, v \in \hat{v}} \Big(\bigvee_{s_1, s_2 \in S} \big(\delta^*(s, \varepsilon^*, s_1) \wedge \delta(s_1, a, s_2) \wedge \delta^*(s_2, \varepsilon^*, v)\big)\Big) \geq h(a) \\
\Leftrightarrow \quad & \{\ \hat{s} \text{ defn.}\} \\
& \bigvee_{s_1, s_2 \in S} \big(\delta^*(s, \varepsilon^*, s) \wedge \delta^*(s, \varepsilon^*, s_1) \wedge \delta(s_1, a, s_2) \wedge \delta^*(s_2, \varepsilon^*, v) \wedge \delta^*(s_2, \varepsilon^*, v)\big) \geq h(a) \\
\Leftrightarrow \quad & \{\ \delta^* \text{ defn.}\}
\end{aligned}$$

$$\bigvee_{s_1,s_2\in S} \left(\delta^*(s,\varepsilon^*,s_1)\wedge\delta(s_1,a,s_2)\wedge\delta^*(s_2,\varepsilon^*,v)\right)\geq h(a)$$

$\Leftrightarrow$ $\{\ \delta^\varepsilon$ defn.$\}$

$$\delta^\varepsilon(s,a,v)\geq h(a)$$

Then, the result follows by induction on the structure of words. For a basic word $a\in\Gamma\setminus\{(\varepsilon,1)\}$,

$$a\in\mathscr{L}^\varepsilon(\mathscr{E})$$

$\Leftrightarrow$ $\{\ (28)\}$

$$\rho^\varepsilon(s_0,a,s)>0, s\in F$$

$\Leftrightarrow$ $\{\ (29)\}$

$$\delta^\varepsilon(s_0,a,s)\geq h(a), s\in F$$

$\Leftrightarrow$ $\{\ (34)\}$

$$\hat{\delta}(\hat{s_0},a,\hat{s})\geq h(a), s\in\hat{F}$$

$\Leftrightarrow$ $\{\ (29)\}$

$$\rho_{\hat{\mathscr{E}}}(\hat{s_0},a,\hat{s})>0, \hat{s}\in\hat{F}$$

$\Leftrightarrow$ $\{\ (28)\}$

$$a\in\mathscr{L}(\hat{\mathscr{E}})$$

For composed words $aw\in(\Gamma\setminus\{(\varepsilon,1)\})^*$,

$$aw\in\mathscr{L}^\varepsilon(\mathscr{E})$$

$\Leftrightarrow$ $\{\ (28)\}$

$$\rho^\varepsilon(s_0,aw,s)>0, s\in F$$

$\Leftrightarrow$ $\{\ (29)\}$

$$\delta^\varepsilon(s_0,a,s')\geq h(a) \text{ and } \rho^\varepsilon(s',w,s)>0, s\in F$$

$\Leftrightarrow$ $\{\ (34) + \text{I.H.}\ (\rho^\varepsilon(s',w,s)>0\Leftrightarrow w\in\mathscr{L}^\varepsilon(\mathscr{E}) \text{ and } \rho_{\hat{\mathscr{E}}}(\hat{s_0},aw,\hat{s})>0\Leftrightarrow w\in\mathscr{L}(\hat{\mathscr{E}}))\}$

$$\hat{\delta}(\hat{s_0},a,\hat{s'})\geq h(a) \text{ and } \rho_{\hat{\mathscr{E}}}(\hat{s'},w,\hat{s_f})>0, \hat{s}\in\hat{F}$$

$\Leftrightarrow$ $\{\ (29)\}$

$$\rho_{\hat{\mathscr{E}}}(\hat{s_0},aw,\hat{s})>0, \hat{s}\in\hat{F}$$

$\Leftrightarrow$ $\{\ (28)\}$

$$aw\in\mathscr{L}(\hat{\mathscr{E}})$$

□

4.2 The theorem

The setting is now ready to establish a Kleene theorem for $\mathscr{H}$-automata and $\mathscr{H}$-synchronous languages. Thus, for any synchronous regular expression $\alpha\in$ Sreg, we will provide a method to

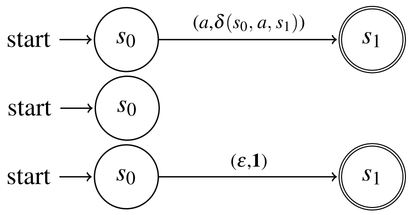


Figure 5. Automata representing actions $a \in \Sigma$, $\mathbf{0}$ and $\mathbf{1}$.

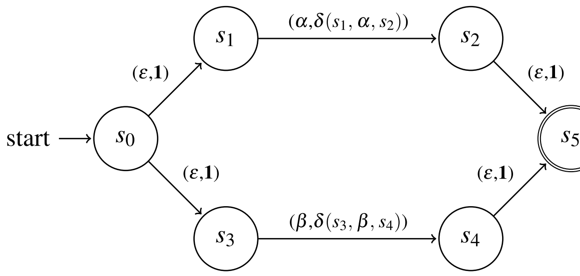


Figure 6. Automaton representing expression $\alpha + \beta$.

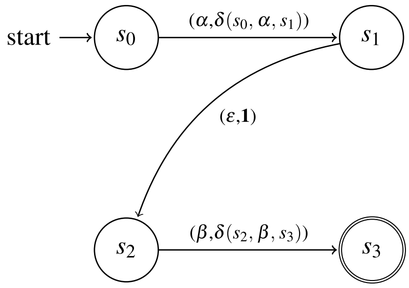


Figure 7. Automaton representing expression $\alpha \cdot \beta$.

build a $\varepsilon\mathscr{H}$-automaton (translatable to a $\mathscr{H}$-automaton, as discussed above) $\mathscr{M}_\alpha$ such that $I(\alpha) = \mathscr{L}(\mathscr{M}_\alpha)$.

For regular expressions built from atomic actions $a \in \Sigma = \mathscr{P}_{ne}(B) \times H$ without resorting to the synchronous product operator, the construction follows the classical recipe, as presented e.g. in Hopcroft and Ullman (1979). This is then extended to synchronous regular expressions, by generalising a construction in Prisacariu (2010) for the synchronous operator '$\times$'.

Theorem 4. *For any* $\alpha \in \mathrm{Sreg}$, *there exists a* $\mathscr{H}$*-automaton* $\mathscr{M}_\alpha$ *such that*

$$I(\alpha) = \mathscr{L}(\mathscr{M}_\alpha)$$

Proof. The automata corresponding to $a \in \Sigma$, $\mathbf{0}$ and $\mathbf{1}$, denoted, respectively, by $\mathscr{M}_a$, $\mathscr{M}_{\mathbf{0}}$ and $\mathscr{M}_{\mathbf{1}}$, are depicted in Figure 5. From Definitions 10 and 7, observe that $I(a) = \mathscr{L}_a = \mathscr{L}_{\mathscr{M}_a}$, $I(\mathbf{0}) = \{\} = \varnothing = \mathscr{L}_{\mathscr{M}_{\mathbf{0}}}$ and that $I(1) = \{\varepsilon\} = \mathscr{L}_{\mathscr{M}_{\mathbf{0}}}$. Then, assuming there exist automata for arbitrary regular actions α and β, we inductively build an $\varepsilon\mathscr{H}$-automaton for Sreg expressions $\alpha + \beta$, $\alpha \cdot \beta$ and α^*. The resulting automata, denoted by $\varepsilon - \mathscr{H}$-automata $\mathscr{E}_{\alpha+\beta}$, $\mathscr{E}_{\alpha\cdot\beta}$, $\mathscr{E}_{\alpha^*}$ and $\mathscr{E}_{\alpha\times\beta}$, are depicted in Figures 6, 7, 8 and 9, respectively. Clearly, Definition 13 entails $I(\alpha + \beta) = \mathscr{L}^\varepsilon(\mathscr{E}_{\alpha+\beta})$, $I(\alpha \cdot \beta) = \mathscr{L}^\varepsilon(\mathscr{E}_{\alpha\cdot\beta})$, $I(\alpha^*) = \mathscr{L}^\varepsilon(\mathscr{E}_{\alpha^*})$ and $I(\alpha \times \beta) = \mathscr{L}^\varepsilon(\mathscr{E}_{\alpha\times\beta})$. Then, by Theorem 3, we conclude that $I(\alpha + \beta) = \mathscr{L}(\hat{\mathscr{E}}_{\alpha+\beta})$, $I(\alpha \cdot \beta) = \mathscr{L}^{(}\hat{\mathscr{E}}_{\alpha\cdot\beta})$, $I(\alpha^*) = \mathscr{L}(\hat{\mathscr{E}}_{\alpha^*})$ and $I(\alpha \times \beta) = \mathscr{L}(\hat{\mathscr{E}}_{\alpha\times\beta})$. $\square$

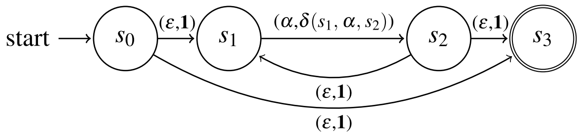


Figure 8. Automaton representing expression α^*.

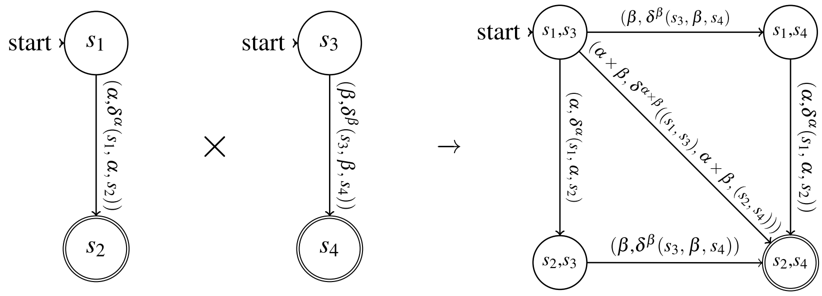


Figure 9. Automaton representing the expression $\alpha \times \beta$.

5. Conclusions

The paper introduced a new class of automata, and corresponding languages, able to capture both *vagueness*, through transitions weighted over a complete Heyting algebra, and *synchronous execution*, through a specific product operator. The work was motivated by the quest for a suitable demantic structure for FAS programs.

To model other situations, for example, in face of a requirement to compute the number of steps involved in an execution, or the resources consumed by a computational process, exploring other structures to parametrise the construction would be a possibility. The tropical semiring

$$R = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0, \rightarrow)$$

with $x \rightarrow y = \max\{y - x, 0\}$, $\forall_{x,y \in \mathbb{R}_+ \cup \{\infty\}}$ would be worth to consider, although it fails idempotency and, therefore Theorem 1.

Finally, a detailed comparison with other possible semantic structures is in order. Probabilistic concurrent Kleene algebra (PCKA), introduced in McIver et al. (2013), is an obvious choice. Such an approach embodies two distinct operators: the concurrency operator '||', from concurrent Kleene algebra of Hoare et al. (2011), to describe the parallel execution of two crisp actions, and a probabilistic choice operator '$\oplus$', to capture uncertainty in the execution of actions.

For reasoning about concurrent programs with some form of uncertainty, PCKA can model Jone's rely/guarantee style calculus with probabilistic behaviour, resorting to a probabilistic event structure semantics (McIver et al. 2016). On the other hand, SKA encodes reasoning in the style of Qwicki and Gries Owicki and Gries (1976) calculus. A possible direction for future work will investigate whether and how this can be extended to the weighted case.

Conflicts of interest

The authors declare none.

Note

1 Following Prisacariu (2010), the symbol '×' stands for the synchronous product; any possible confusion with the same symbol used for Cartesian product is disambiguated by context.

References

Bollig, B., Gastin, P., Monmege, B. and Zeitoun, M. (2012). A probabilistic Kleene theorem. In: Chakraborty, S. and Mukund, M. (eds.) *Automated Technology for Verification and Analysis*, LNCS, vol. 7561, Springer Berlin Heidelberg, 400–415.

Broda, S., Machiavelo, A., Moreira, N. and Reis, R. (2013). On the average size of glushkov and equation automata for KAT expressions. In: Gasieniec, L. and Wolter, F. (eds.) *Fundamentals of Computation Theory - 19th International Symposium, FCT 2013, Liverpool, UK, August 19–21, 2013. Proceedings*, Lecture Notes in Computer Science, vol. 8070, Springer, 72–83.

de Bruin, J. S., Schuh, C. J., Rappelsberger, A. and Adlassnig, K.-P. (2018). Medical fuzzy control systems with fuzzy arden syntax. In: Kacprzyk, J., Szmidt, E., Zadrozny, S., Atanassov, K. T. and Krawczak, M. (eds.) *Advances in Fuzzy Logic and Technology 2017 - Proceedings of: EUSFLAT-2017 - The 10th Conference of the European Society for Fuzzy Logic and Technology, 2017, Warsaw, Poland IWIFSGN'2017 - The Sixteenth International Workshop on Intuitionistic Fuzzy Sets and Generalized Nets, 2017, Warsaw, Poland, Volume 1*, Advances in Intelligent Systems and Computing, vol. 641, Cham, Springer, 574–584.

Doostfatemeh, M. and Kremer, S. C. (2005). New directions in fuzzy automata. *International Journal of Approximate Reasoning* **38** (2) 175–214.

Gomes, L., Madeira, A. and Barbosa, L. S. (2020). Introducing synchrony in fuzzy automata. *ENTCS* **348** 43–60. (LSFA 2019).

Gomes, L., Madeira, A. and Barbosa, L. S. (2021). A semantics and a logic for fuzzy arden syntax. *Soft Computing* **25** (9) 6789–6805.

Hoare, T., Möller, B., Struth, G. and Wehrman, I. (2011). Concurrent Kleene algebra and its foundations. *Journal of Logical and Algebraic Methods in Programming* **80** (6) 266–296.

Hopcroft, J. E., Motwani, R. and Ullman, J. D. (2003). *Introduction to Automata Theory, Languages, and Computation*, int. ed. 2nd ed., Addison-Wesley. **Q5**

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Company. **Q6**

Jipsen, P. and Andrew Moshier, M. (2016). Concurrent kleene algebra with tests and branching automata. *Journal of Logical and Algebraic Methods in Programming* **85** (4) 637–652.

Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In: Shannon, C. and McCarthy, J. (eds.) *Automata Studies*, Princeton, NJ, Princeton University Press, 3–41.

Kozen, D. and Mamouras, K. Kat + b! (Theorem 2). **Q7**

Kozen, D. (1990). On Kleene algebras and closed semirings. In: Rovan, B. (ed.) *Mathematical Foundations of Computer Science 1990, Czechoslovakia, 1990, Proc.*, LNCS, vol. 452, Springer, 26–47.

Kozen, D. (1997). Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems* **19** (3) 427–443.

Lee, E. T. and Zadeh, L. A. (1969). Note on fuzzy languages. *Information Sciences* **1** (4) 421–434.

Li, Y. and Pedrycz, W. (2005). Fuzzy finite automata and fuzzy regular expressions with membership values in lattice-ordered monoids. *Fuzzy Sets and Systems* **156** (1) 68–92.

Liu, A., Wang, S., Barbosa, L. S. and Sun, M. (2021). Fuzzy automata as coalgebras. *Mathematics* **9** (3). **Q8**

Lin, F. and Ying, H. (2002). Modeling and control of fuzzy discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* **32** (4) 408–415.

Mateescu, A., Salomaa, A., Salomaa, K. and Yu, S. (1995). Lexical analysis with a simple finite-fuzzy-automaton model. *Journal of UCS* **1** (5) 292–311.

McIver, A., Cohen, E. and Morgan, C. (2006). Using probabilistic kleene algebra for protocol verification. In: Schmidt, R. A. (ed.) *Relations and Kleene Algebra in Computer Science, 9th International Conference on Relational Methods in Computer Science and 4th International Workshop on Applications of Kleene Algebra, RelMiCS/AKA 2006, Manchester, UK, August 29–September 2, 2006, Proceedings*, Lecture Notes in Computer Science, vol. 4136, Springer, 296–310.

McIver, A., Rabehaja, T. M. and Struth, G. (2013). Probabilistic concurrent Kleene algebra. In: Bortolussi, L. and Wiklicky, H. (eds.) *QAPL 2013, Italy*, EPTCS, vol. 117, 97–115.

McIver, A., Rabehaja, T. and Struth, G. (2016). Probabilistic rely-guarantee calculus. *Theoretical Computer Science* **655** 120–134. QAPL (2013-14).

Milner, R. (1980). *A Calculus of Communicating Systems*. Springer Lecture Notes in Computer Science, vol. 92. **Q9**

Milner, R. (1983). Calculi for synchrony and asynchrony. *Theoretical Computer Science* **25** 267–310.

Milner, R. (2006). *Turing, Computing and Communication*. Springer Berlin Heidelberg, 1–8.

Mordeson, J. and Malik, D. (2002). *Fuzzy Automata and Languages: Theory and Applications*, 1st ed., Chapman and Hall/CRC. **Q10**

Owicki, S. and Gries, D. (1976). An axiomatic proof technique for parallel programs I. *Acta Informatica* **6** (4) 319–340.

Pedrycz, W. and Gacek, A. (2001). Learning of fuzzy automata. *International Journal of Computational Intelligence and Applications* **1** (1) 19–33.

Prisacariu, C. (2010). Synchronous Kleene algebra. *Journal of Logical and Algebraic Methods in Programming* **79** (7) 608–635.

Qiao, R., Wu, J., Wang, Y. and Gao, X. (2008). Operational semantics of probabilistic Kleene algebra with tests. In: *Proceedings - IEEE Symposium on Computers and Communications*, 706–713.

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

Segerberg, K. (1982). A deontic logic of action. *Studia Logica* **41** (2) 269–282.

Thiemann, P. (ed.) (2016). *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2–8, 2016, Proceedings*, Lecture Notes in Computer Science, vol. 9632, Springer.

Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F. and Carrasco, R. C. (2005a). Probabilistic finite-state machines-part I. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **27** (7) 1013–1025.

Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F. and Carrasco, R. C. (2005b). Probabilistic finite-state machines - part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **27** (7) 1026–1039.

von Wright, G. H. (1968). *An Essay in Deontic Logic and the General Theory of Action with a Bibliography of Deontic and Imperative Logic.* –. North-Holland Pub. Co. **Q11**

Wee, W. and Fu, K. S. (1969). A formulation of fuzzy automata and its application as a model of learning systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **5** 215–223.

Ying, M. (2002). A formal model of computing with words. *IEEE Transactions on Fuzzy Systems* **10** (5) 640–652.

Zadeh, L. A. (1996). Fuzzy languages and their relation to human and machine intelligence. In: *Proceedings of International Conference on Man and Computer*, 148–179.

Cite this article: Gomes L., Madeira A. and Barbosa LS. Weighted synchronous automata. *Mathematical Structures in Computer Science*. https://doi.org/10.1017/S0960129522000421