Journal of Logical and Algebraic Methods in Programming 118 (2021) 100615

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Soft constraint automata with memory

Kasper Dokter [a], Fabio Gadducci [b,1], Benjamin Lion [c], Francesco Santini [d,*,2]

[a] Leiden University, Leiden, the Netherlands
[b] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
[c] Centrum Wiskunde & Informatica, Amsterdam, the Netherlands
[d] Dipartimento di Matematica e Informatica, Università di Perugia, Perugia, Italy

ARTICLE INFO

Article history:
Received 10 May 2019
Received in revised form 30 September 2020
Accepted 1 October 2020
Available online 21 October 2020

Keywords:
Constraint automata
Automata with memory
Reo language
Soft constraints

ABSTRACT

We revise *soft constraint automata*, wherein transitions are weighted and each action has an associated preference value. We first relax the underlying algebraic structure to allow bipolar preferences. We then equip automata with memory locations, that is, with an internal state to remember and update information from transition to transition. We furthermore revise automata operators, such as composition and hiding, providing examples on how such memory locations interact with preferences. We finally apply our framework to encode context-sensitive behaviour.

1. Introduction

Many languages have been proposed for the specification and implementation of systems of interacting processes. Such languages include process calculi, concurrent objects, actors, agents, shared memory, and message passing. A distinctive feature of these languages is that they are all primarily *action-based*: the protocol by which all processes interact must be encoded in the actions of individual processes. Consequently, the interaction protocol becomes implicit, which makes it practically impossible to analyse the current protocol or to reuse previously developed protocols.

In contrast, the Reo language [1] treats interaction protocols as an explicit first-class concept, and is therefore referred to as an *interaction-based* language. Reo represents the interaction protocol as a graph-like structure, called a *connector* or *circuit*. Intuitively, such graphs describe networks of channels that facilitate data flow among all processes in the system. Components can perform I/O operations on the boundary nodes of the circuit to which they are connected. In this way, a Reo connector imposes constraints on the order in which the components exchange data items with each other.

Reo comes with a powerful composition operator that allows for the specification of complex interaction protocols by combining simpler (possibly primitive) ones. Even though the basic primitive channels are simple, Reo connectors can actually describe rather complex protocols.

In our work, we focus the attention on the formal models used to describe the behaviour of the interaction protocol. The literature offers several semantic formalisms to express the behaviour of Reo connectors. The work in [2] collects, classifies, and surveys around thirty semantics based on *co-algebraic* or *colouring* techniques, and other models based on, for instance,

* Corresponding author.
E-mail addresses: K.P.C.Dokter@cwi.nl (K. Dokter), fabio.gadducci@unipi.it (F. Gadducci), B.Lion@cwi.nl (B. Lion), francesco.santini@unipg.it (F. Santini).

1 Partially supported by the MIUR PRIN 2017FTXR7S "IT-MaTTerS".
2 Partially supported by GNCS-INdAM ("Gruppo Nazionale per il Calcolo Scientifico").

https://doi.org/10.1016/j.jlamp.2020.100615
2352-2208/

constraints and *Petri nets*. The *operational* models (i.e., automata) are probably the most popular approaches: the main classes are represented by *constraint automata*, and (several) related variants, and *context-sensitive automata*.

The aim of the current work is to generalise *soft constraint automata* [3] or *soft component automata* [4,5] (SCA in both cases), which is a variant of constraint automata that has been developed after the publication of the survey [2]. An SCA is a state-transition system where transitions are labelled with actions and preferences. Higher-preference transitions typically contribute more towards the goal of the component.

The contribution of this paper is twofold. First, we relax the definition of the underlying structure that models preferences. Instead of absorptive semirings (as in [3–5]), we use complete lattice monoids (see Section 2.1) to allow both positive and negative preference values, as well as possibly infinite ones.

Second, we extend SCA with a notion of memory (SCAM), as already accomplished for (non-soft) constraint automata [6]. Each transition of a SCAM can also impose a condition on the current data assigned to a finite set of *memory locations*, and update their respective values. Therefore, together with states, memory locations determine the configuration of a connector, and influence its observable behaviour.

This paper extends the work in [7], a Festschrift in honour of Farhad Arbab. We here improve the original paper by extensively revising the formalism (e.g., the definition of *weighted data streams* [7] into *CLM streams*, Section 2.1); simplifying definitions (e.g., the SCAM semantics in Definition 16); providing a case study that explains the working details of composition and hiding (Section 4); and applying SCAMs in a novel encoding of context-sensitive behaviour (Section 5).

The outline of the paper is as follows: Section 2 defines soft constraints and shows that they can be compared and composed, and their variables renamed and hidden. Section 3 introduces soft constraint automata with memory (SCAMs) and their interpretation as soft constraints. We define their composition and hiding, and show their correctness with respect to the soft constraint semantics. Section 4 presents a case study illustrating the composition and hiding operations on SCAMs. Section 5 offers a novel encoding of context-sensitive behaviour based on SCAMs. Finally, Section 6 summarises related works on different semantics for CA, and Section 7 wraps up the paper with conclusive thoughts and hints about future research.

2. Preliminaries on soft constraints

In contrast to Boolean constraints, a *soft constraint* is a constraint that does not need to be fully satisfied [8]. Instead, such a constraint assigns a *preference value* that measures the degree of satisfaction of a solution. The goal is to find a solution that maximises this preference value.

The structure of this section is as follows: Section 2.1 proposes *complete lattice monoids* (CLMs) to serve as a structured domain of preference values, which allows us to compare and compose such values. Section 2.2 develops a complete lattice monoid of streams equipped with a lexicographic order. We use this construction in the semantics of soft constraint automata in Section 3. Section 2.3 presents our personal take on cylindric and diagonal operators [9]: they are mostly drawn with minor adjustments from [10]. Section 2.4 shows that the set of soft constraints is itself a complete lattice monoid that admits cylindric and diagonal operators. As such, soft constraints can be compared and composed, and variables in a soft constraint can be renamed and hidden.

2.1. Complete lattice monoids

The first step is to define an algebraic structure that models preference values. We refer to [11] for the missing proofs as well as for an introduction to bipolar preferences and a comparison with other proposals.

Definition 1 *(Partial order).* A partial order (PO) is a pair $\langle A, \le\rangle$ such that A is a set and $\le \subseteq A \times A$ is a reflexive, transitive, and anti-symmetric relation. A complete lattice (CL) is a PO such that any subset of A has a least upper bound (LUB).

The LUB of a subset $X \subseteq A$ is denoted as $\bigvee X$, and it is unique. Note that $\bigvee A$ and $\bigvee \emptyset$ correspond respectively to the *top*, denoted as $\top$, and to the *bottom*, denoted as $\bot$, of the CL.

Definition 2 *(Complete lattice monoid).* A (commutative) monoid is a triple $\langle A, \otimes, \mathbf{1}\rangle$ such that $\otimes : A \times A \to A$ is a commutative and associative operation and $\mathbf{1} \in A$ is its identity element.

A partially ordered monoid (POM) is a 4-tuple $\langle A, \le, \otimes, \mathbf{1}\rangle$ such that $\langle A, \le\rangle$ is a PO and $\langle A, \otimes, \mathbf{1}\rangle$ a commutative monoid. A complete lattice monoid (CLM) is a POM such that its underlying PO is a CL.

As usual, we use the infix notation: $a \otimes b$ stands for $\otimes(a, b)$. According to Definition 2, the partial order $\le$ and the product $\otimes$ can be unrelated. This is not so for monotone CLMs.

Definition 3 *(Monotonicity).* A CLM $\langle A, \le, \otimes, \mathbf{1}\rangle$ is monotone if for all $a, b, c \in A$, we have that $a \le b$ implies $a \otimes c \le b \otimes c$.

Our framework (Lemma 3) requires a condition that is stronger than monotonicity.

Definition 4 *(Distributivity).* A CLM $\langle A, \leq, \otimes, \mathbf{1} \rangle$ is distributive if for all $a \in A$ and subset $X \subseteq A$, we have

$$a \otimes \bigvee X = \bigvee \{a \otimes x \mid x \in X\}.$$

Note that $a \leq b$ is equivalent to $\bigvee\{a, b\} = b$. Hence, a distributive CLM is monotone and $\perp$ is its zero element (i.e., $a \otimes \perp = \perp$, for all $a \in A$).

Example 1 *(Boolean CLM).* The Boolean CLM $\mathbb{B} = \langle \{0, 1\}, \leq, \times, 1 \rangle$, with the usual order and multiplication, is distributive.

Distributive CLMs generalise *tropical* semirings: it suffices to define their (idempotent) sum operator as $a \oplus b = \bigvee\{a, b\}$, for all $a, b \in A$. If, moreover, $\mathbf{1}$ is the top of the CL, we end up with *absorptive* semirings [12] (in the algebraic literature) or *c*-semirings [8] (in the soft constraint literature). See [13] for a brief survey on residuation for such semirings. Together with monotonicity, imposing $\mathbf{1}$ to coincide with $\top$ means that preferences are *negative* (i.e., $a \leq \mathbf{1}$, for all $a \in A$). Since we allow the top of the CL to be strictly *positive* (i.e., $\mathbf{1} < \top$), our approach falls into the category of *bipolar* approaches.

Example 2 *(Bipolar CLM).* The CLM $\mathbb{K} = \langle \{0, 1, \infty\}, \leq, \times, 1 \rangle$, with the usual order and multiplication (extended to ∞ by defining $0 \times \infty = 0$ and $1 \times \infty = \infty \times \infty = \infty > 1$), is distributive.

Example 3 *(Power set).* Given a (possibly infinite) set V of variables, we consider the monoid $\langle 2^V, \cup, \emptyset \rangle$ of (possibly empty) subsets of V, with union as the monoidal operator. Since the operator is idempotent ($a \otimes a = a$, for all $a \in A$), the natural order ($a \leq b \Leftrightarrow a \otimes b = b$, for all $a, b \in A$) is a partial order, and it coincides with subset inclusion: in fact, the power set $\langle 2^V, \subseteq, \cup, \emptyset \rangle$ is a CLM. Moreover, since both $\bigvee$ and $\otimes$ model set-union, the power set CLM is distributive.

Example 4 *(Extended integers).* The extended integers $\langle \mathbb{Z} \cup \{\pm\infty\}, \leq, +, 0 \rangle$, where $\leq$ is the natural order, such that, for all $k \in \mathbb{Z}$

$$-\infty \ \leq \ k \ \leq \ +\infty,$$

$+$ is the natural addition, such that, for all $k \in \mathbb{Z} \cup \{+\infty\}$

$$\pm\infty + k \ = \ \pm\infty, \qquad\qquad +\infty + (-\infty) \ = \ -\infty,$$

and 0 the identity element, constitutes a distributive CLM. Here, $+\infty$ and $-\infty$ are respectively the top and the bottom element of the CL.

In the following definition we propose a construction that allows us to compose primitive CLMs (such as those in Examples 1 to 4) into more complex ones.

Definition 5 *(Cartesian product).* Let $\langle A_1, \leq_1, \otimes_1, \mathbf{1}_1 \rangle$ and $\langle A_2, \leq_2, \otimes_2, \mathbf{1}_2 \rangle$ be CLMs. Their Cartesian product is the CLM $\langle A_1 \times A_2, \leq, \otimes, (\mathbf{1}_1, \mathbf{1}_2) \rangle$ such that, for all $(a_1, a_2), (b_1, b_2) \in A_1 \times A_2$

1. $(a_1, a_2) \leq (b_1, b_2)$ if $a_1 \leq_1 b_1$ and $a_2 \leq_2 b_2$,
2. $(a_1, a_2) \otimes (b_1, b_2) = (a_1 \otimes_1 b_1, a_2 \otimes_2 b_2)$.

It is easy to see that the Cartesian product of distributive CLMs is also distributive.

2.2. Streams of preferences

We now introduce the CLM of streams, which we use for our semantics of SCAMs in Definition 16. Here we generalise the results in [14] on binary lexicographic operators. In the following, we denote by A^ω the set of streams (infinite sequences) of elements of A.

Definition 6 *(Lexicographic order).* Let $\langle A, \leq \rangle$ be a PO. The lexicographic order $\leq_l$ on A^ω is given by

$$a_0 a_1 \cdots \leq_l b_0 b_1 \cdots \quad \text{if} \quad \begin{cases} \forall i.\, a_i = b_i & \vee \\ \exists j.\, a_j < b_j \wedge \forall i < j.\, a_i = b_i \end{cases}$$

We write $<_l$ for the usual strict version of the lexicographic order $\leq_l$. The following lemma provides a recursive description of LUBs in lexicographically ordered streams.

Lemma 1. *Let $\langle A, \leq\rangle$ be a CL and $X \subseteq A^\omega$ a subset of the PO $\langle A^\omega, \leq_l\rangle$. Then, $\bigvee X = x_0x_1x_2\cdots \in A^\omega$ exists and satisfies, for all $i \geq 0$, the recursion*

$$x_i = \bigvee\{b_i \mid x_0x_1\cdots x_{i-1}b_ib_{i+1}\cdots \in X\}.$$

Proof. Define $x_0x_1\cdots \in A^\omega$ using the recursion in the lemma.

We prove that $x_0x_1\cdots$ is an upper bound of X. Let $a_0a_1\cdots \neq x_0x_1\cdots$ be in X. Find the smallest $i \geq 0$, such that $a_i \neq x_i$. Then, we have $a_i \in \{b_i \mid x_0x_1\cdots x_{i-1}b_ib_{i+1}\cdots \in X\}$ and $a_i < x_i$. Thus, $a_0a_1\cdots \leq_l x_0x_1\cdots$.

We prove that $x_0x_1\cdots$ is minimal. Let $u_0u_1\cdots \neq x_0x_1\cdots$ be an upper bound of X. Find the smallest $i \geq 0$, such that $u_i \neq x_i$. For all $x_0\cdots x_{i-1}b_i\cdots \in X$, we have $x_0\cdots x_{i-1}b_i\cdots \leq_l u_0u_1\cdots = x_0\cdots x_{i-1}u_i\cdots$, which implies $b_i \leq u_i$. Hence, $x_i = \bigvee\{b_i \mid x_0\cdots x_{i-1}b_i\cdots \in X\} \leq u_i$, and $x_0x_1\cdots \leq_l u_0u_1\cdots$.

We conclude that $\bigvee X = x_0x_1\cdots$, which proves the result. □

Let $\otimes^\omega$ be the operator on data stream given by the point-wise application of $\otimes$ and $\mathbf{1}^\omega$ the data stream composed just by $\mathbf{1}$. Lemma 1 shows that $\langle A^\omega, \leq_l, \otimes^\omega, \mathbf{1}^\omega\rangle$ is a CLM. However, it turns out that this CLM may not be distributive due to the presence of *non-cancellative* (or *collapsing*) elements in A.

Definition 7 *(Cancellative elements).* An element c in a CLM $\langle A, \leq, \otimes, \mathbf{1}\rangle$ is cancellative if $a \otimes c = b \otimes c$ implies $a = b$, for all $a, b \in A$.

In any distributive CLM, $\perp$ is a collapsing element. The presence of such elements prevents the CLM of streams to be distributive, and forces to restrict its carrier, A^ω, to a suitable subset. Let A_ι be the set of cancellative elements, and let A_c be the set of collapsing ones. The following example shows that the subset $A_\iota^\omega \subseteq A^\omega$ of streams of cancellative elements is not a suitable domain for the CLM of streams, as it is not closed under LUBs.

Example 5. Let $\langle A, \leq, +, 0\rangle$, with $A = \mathbb{Z} \cup \{\pm\infty\}$, be the CLM of extended integers from Example 4. Observe that $A_\iota = \mathbb{Z}$ and $A_c = \{\pm\infty\}$. Although $\emptyset$ and $\{1^\omega, 2^\omega, 3^\omega, \cdots\}$ are subsets of A_ι^ω, their respective LUBs are (by Lemma 1) equal to $(-\infty)^\omega$ and $(+\infty)(-\infty)^\omega$, thus not included in A_ι^ω.

To ensure that the set of streams is closed under LUBs, we further include elements of the shape $A_\iota^* A_c \perp^\omega$: streams prefixed by a (possibly empty) finite sequence of cancellative elements, then followed by a single occurrence of a collapsing one, and then closed by an infinite sequence of $\perp$.

Theorem 1 *(Lexicographic CLM). If $\mathbb{S} = \langle A, \leq, \otimes, \mathbf{1}\rangle$ is a distributive CLM, then $\mathbb{S}^\omega = \langle A_\iota^\omega \cup A_\iota^* A_c \perp^\omega, \leq_l, \otimes^\omega, \mathbf{1}^\omega\rangle$ is so.*

Proof. The POM $\mathbb{S}^\omega$ is a CLM, because its carrier $B = A_\iota^\omega \cup A_\iota^* A_c \perp^\omega$ is closed with respect to LUBs: let $X \subseteq B$ and $\bigvee X = x_0x_1\cdots$. Suppose that $x_i \in A_c$, for some $i \geq 0$. Let $j > i$ be arbitrary, and consider the set $X_j = \{b \mid x_0\cdots x_i\cdots x_{j-1}b\cdots \in X\}$. Since $X \subseteq B$ and $x_i \in A_c$, we have either $X_j = \emptyset$ or $X_j = \{\perp\}$. By Lemma 1, we have $x_j = \bigvee X_j = \perp$, and we conclude that $\bigvee X \in B$.

Next, we show that the CLM $\mathbb{S}^\omega$ is distributive, i.e., that $a \otimes^\omega \bigvee X = \bigvee\{a \otimes^\omega x \mid x \in X\}$. Let $X \subseteq B$ be a subset of the carrier, and $a \in B$. Let also $p = \bigvee X$ and $q = \bigvee\{a \otimes^\omega x \mid x \in X\}$: we show by induction that $a_i \otimes p_i = q_i$ for all $i \geq 0$. So, let us suppose that $a_j \otimes p_j = q_j$ for all $0 \leq j < i$, which is vacuously true for $i = 0$. Lemma 1 shows that $q_i = \bigvee S$, with $S = \{b \mid q_0\cdots q_{i-1}b\cdots \in \{a \otimes^\omega x \mid x \in X\}\}$. We distinguish two cases:

Case 1: Suppose that some a_j, $0 \leq j < i$, is non-cancellative. Then, $a \in B$ implies that $a_i = \perp$. Hence, we have either $S = \emptyset$ or $S = \{\perp\}$, and we find $q_i = \bigvee S = \perp = a_i \otimes p_i$.

Case 2: Suppose that all a_j, $0 \leq j < i$, are cancellative. Then,

$$q_i = \bigvee S = \bigvee\{a_i \otimes x_i \mid x_0x_1\cdots \in X \wedge a_j \otimes x_j = q_j \text{ for all } j < i\}.$$

Distributivity of the original CLM and the induction hypothesis implies

$$q_i = a_i \otimes \bigvee\{x_i \mid x_0x_1\cdots \in X, a_j \otimes x_j = a_j \otimes p_j, \text{for all } j < i\}.$$

Applying Lemma 1 yields

$$q_i = a_i \otimes \bigvee\{x_i \mid p_0\cdots p_{i-1}x_i\cdots \in X\} = a_i \otimes p_i.$$

In both cases, we find $q_i = a_i \otimes p_i$, which completes the proof. □

Example 6. Looking at the CLM $\langle \mathbb{Z} \cup \{\pm\infty\}, \leq, +, 0\rangle$ of extended integers from Example 4 and Example 5, the set of elements of the associated lexicographic CLM is $\mathbb{Z}^\omega \cup \mathbb{Z}^*\{\pm\infty\}(-\infty)^\omega$.

2.3. Cylindric operators for ordered monoids

We introduce two families of operators on CLMs, which enable hiding and renaming of variables (cf. [10,15]). The first family is parameterised by a cylindric operator, and models existential quantification. The second family is parameterised by a diagonal operator, and models the equality of variables. Cylindric and diagonal operators originate in the context of cylindric algebras [16] and entered the constraint literature via [9].

Definition 8 *(Pomonoid action).* Let $\mathbb{S} = \langle A, \leq, \otimes, \mathbf{1}\rangle$ be a CLM and $\mathbb{P} = \langle E, \leq\rangle$ a PO. An action of $\mathbb{S}$ on $\mathbb{P}$ is a function $\phi : A \times E \to E$, such that, for all $a, b \in A$ and all $e \in E$,

1. $\phi(\mathbf{1}, e) = e$,
2. $\phi(a, \phi(b, e)) = \phi(a \otimes b, e)$,
3. $a \leq b \implies \phi(a, e) \leq \phi(b, e)$.

The first two requirements state that ϕ is a monoid action of $\mathbb{S}$ on E.

Let V be a set of variables, and recall the power set CLM $\langle 2^V, \subseteq, \cup, \emptyset\rangle$ from Example 3. Consider a CLM $\langle A, \leq, \otimes, \mathbf{1}\rangle$, whose elements $a \in A$ can be thought of as expressions with variables from V. The partial order, $\leq$, the product, $\otimes$, and the identity, $\mathbf{1}$, can be thought of as implication, conjunction, and tautology, respectively. The following definition axiomatises existential quantification for these expressions.

Definition 9 *(Cylindric operator and support).* A cylindric operator $\exists$ over a CLM $\langle A, \leq, \otimes, \mathbf{1}\rangle$ and set of variables V is an action $\exists : 2^V \times A \to A$, such that, for all $X \subseteq V$, all $a, b \in A$, and all $C \subseteq A$,

1. $\exists(X, \mathbf{1}) = \mathbf{1}$,
2. $\exists(X, a \otimes \exists(X, b)) = \exists(X, a) \otimes \exists(X, b)$,
3. $\exists(X, \bigvee C) = \bigvee\{\exists(X, c) \mid c \in C\}$.

The *support* of $a \in A$ is the set of variables $\mathrm{supp}(a) = \{x \mid \exists(\{x\}, a) \neq a\}$.

In the following, we use $\exists_X a$ for $\exists(X, a)$ and write $\exists_x a$ when $X = \{x\}$. Item 3 in Definition 9 is required for the correctness proofs of SCAM operations on SCAMs: see Theorems 2 and 3 later on. Item 3 also implies the monotonicity of $\exists$ in the second argument ($a \leq b$ implies $\exists_X a \leq \exists_X b$, for all $X \subseteq V$ and all $a, b \in A$). By Definition 8 it holds that $a = \exists_\emptyset a \leq \exists_X a$ and $X \cap \mathrm{supp}(\exists(X, a)) = \emptyset$.

Next, we axiomatise expressions that equate two variables.

Definition 10 *(Diagonalisation).* Let $\exists$ be a cylindric operator over a CLM $\langle A, \leq, \otimes, \mathbf{1}\rangle$ and V a set of variables. A diagonal operator δ for $\exists$ is a family of idempotent elements $\delta_{x,y} \in A$, indexed by pairs of variables in V, such that, for all $x, y, z \in V$ and $a \in A$

1. $\delta_{x,x} = \mathbf{1}$,
2. $\delta_{x,y} = \delta_{y,x}$,
3. $z \notin \{x, y\} \implies \delta_{x,y} = \exists_z(\delta_{x,z} \otimes \delta_{z,y})$,
4. $x \neq y \implies \delta_{x,y} \otimes \exists_x(a \otimes \delta_{x,y}) \leq a$.

Axioms 1, 2, and 3 plus idempotency of $\delta_{x,y}$ imply $\exists_x \delta_{x,y} = \mathbf{1}$, which in turn implies (using also idempotency of $\exists_X$) $\mathrm{supp}(\delta_{x,y}) = \{x, y\}$ for $x \neq y$. Diagonal operators can be used for modelling variable substitution [10]: substituting y for $x \neq y$ in a yields $\exists_x(a \otimes \delta_{x,y})$.

2.4. Soft constraints (on infinite domains)

We define the notion of soft constraints, following the approach in [17], but generalising the preference structure, as in [10,15]. Soft constraints are expressions that evaluate to a value in a given CLM. They generalise *crisp* constraints, which are expressions that evaluate into the Boolean CLM.

Definition 11 *(Soft constraints).* Let V be a set of variables, D a domain of interpretation, and $\mathbb{S} = \langle A, \leq, \otimes, \mathbf{1}\rangle$ a CLM. A *soft constraint* is a function $c : (V \to D) \to A$ associating a value in A with each assignment $\eta : V \to D$ of the variables. We write $\mathcal{C}(V, D, \mathbb{S})$ for the set of all such soft constraints.

We write $c\eta$ to denote the application of a constraint $c : (V \to D) \to A$ to a variable assignment $\eta : V \to D$.

Definition 11 does not impose any restriction on the number of variables and the size of the domain of interpretation. In fact, our framework requires infinitely many *timed variables*, as introduced in Section 3.1. Different from standard practice in soft constraint literature, we also consider a possibly infinite domain of interpretation, D, which is required by the introduction of memory locations in Definition 14.

In the following example, we introduce notation to view preference values and Boolean constraints as soft constraints.

Example 7 *(Constant constraints).* A preference value $a \in A$ induces a soft constraint $[a]$ defined as $[a]\eta = a$, for every assignment $\eta : V \to D$.

Example 8 *(Boolean constraints).* A Boolean constraint B induces a soft constraint $[B]$ defined, for every assignment $\eta : V \to D$, as

$$[B]\eta = \begin{cases} \mathbf{1} & \text{if } \eta \text{ satisfies } B \\ \perp & \text{otherwise} \end{cases}$$

For example, for a variable $v \in V$, a datum $d \in D$, and an assignment $\eta : V \to D$, we have $[v = d]\eta = \mathbf{1}$ if and only if $\eta(v) = d$. Since conjunction with a tautology should act as the identity, we choose $\mathbf{1}$ instead of $\top$.

The set of constraints forms a CLM, with the structure lifted from $\mathbb{S}$.

Lemma 2 *(The CLM of constraints). The set $\mathcal{C}(V, D, \mathbb{S})$ of soft constraints, endowed with partial order $\leq$, composition $\otimes$, and unit $[\mathbf{1}]$ defined as*

1. *$c_1 \leq c_2$ if $c_1\eta \leq c_2\eta$ for all $\eta : V \to D$*
2. *$(c_1 \otimes c_2)\eta = c_1\eta \otimes c_2\eta$*

is a CLM denoted as $\mathbb{C}(V, D, \mathbb{S})$. The LUB of a subset $C \subseteq \mathbb{C}(V, D, \mathbb{S})$ satisfies $(\bigvee C)\eta = \bigvee\{c\eta \mid c \in C\}$, for all assignments $\eta : V \to D$. The CLM $\mathbb{C}(V, D, \mathbb{S})$ is distributive if $\mathbb{S}$ is so.

Combining constraints using the $\otimes$ operator builds a new constraint whose support involves at most the variables of the original ones. The composite constraint associates, with every assignment, a preference that is equal to the product of the preferences of its constituents.

Note that, in a bipolar setting, we do not have *conjunction elimination*: for constraints $c_1, c_2 \in \mathcal{C}(V, D, \mathbb{S})$, $c_2 > \mathbf{1}$, monotonicity implies $c_1 \otimes c_2 > c_1$, where $\leq$ is interpreted as implication.

Given a function $\eta : V \to D$ and a set $X \subseteq V$, we denote by $\eta|_X : X \to D$ the usual restriction.

Lemma 3 *(Cylindric and diagonal operators for constraints). If $\mathbb{S}$ is a distributive CLM, then the CLM of constraints $\mathbb{C}(V, D, \mathbb{S})$ admits a diagonal operator $\delta_{x,y} = [x = y]$, for all $x, y \in V$, and a cylindric operator $\exists_X$, defined, for all soft constraints $c \in \mathcal{C}(V, D, \mathbb{S})$ and all subsets $X \subseteq V$ of variables, as*

$$(\exists_X c)\eta = \bigvee\{c\rho \mid \rho|_{V\setminus X} = \eta|_{V\setminus X}\}.$$

Proof. Using the definitions from Example 8, Lemma 2, and distributivity of $\mathbb{S}$, it is straightforward to verify that all axioms in Definitions 8 to 10 are satisfied. For example, for all soft constraints $c, d \in \mathcal{C}(V, D, \mathbb{S})$ and all $X \subseteq V$, we have, for all assignments $\eta : V \to D$, that

$$\begin{aligned}
(\exists_X(c \otimes \exists_X d))\eta &= \bigvee\{c\rho \otimes (\exists_X d)\rho \mid \rho|_{V\setminus X} = \eta|_{V\setminus X}\} \\
&= \bigvee\{c\rho \otimes (\bigvee\{d\xi \mid \xi|_{V\setminus X} = \eta|_{V\setminus X}\}) \mid \rho|_{V\setminus X} = \eta|_{V\setminus X}\} \\
&= \bigvee\{c\rho \mid \rho|_{V\setminus X} = \eta|_{V\setminus X}\} \otimes \bigvee\{d\xi \mid \xi|_{V\setminus X} = \eta|_{V\setminus X}\} \\
&= (\exists_X c \otimes \exists_X d)\eta,
\end{aligned}$$

which shows that $\exists_X(c \otimes \exists_X d) = \exists_X c \otimes \exists_X d$. $\square$

Hiding removes variables from the support: $\mathrm{supp}(\exists_X c) \subseteq \mathrm{supp}(c) \setminus X$.[3] Note that both the infinite number of variables and the infinite domain of interpretation necessitate the existence of LUBs in the definition of $\exists_X c$, which motivates the introduction of complete lattices in the previous section.

[3] The operator is called *projection* in the constraints literature, and $\exists_X c$ is denoted $c \Downarrow_{V\setminus X}$.

Although a constraint c evaluates mappings $\eta : V \to D$ that assign a value in D to every variable in V, the evaluation $c\eta$ may depend on the assignment of a (finite) subset of them, called its support. The cylindric operator from Lemma 3, together with Definition 9, provides a precise characterisation of the support. For instance, a binary constraint c with $\mathrm{supp}(c) = \{x, y\}$ is a function $c : (V \to D) \to A$ that depends only on the assignment of variables $\{x, y\} \subseteq V$, meaning that two assignments $\eta_1, \eta_2 : V \to D$ that differ only for the image of variables $z \notin \{x, y\}$ coincide (i.e., $c\eta_1 = c\eta_2$). The notion of support corresponds to the classical notion of constraint *scope*.

3. Soft constraint automata with memory

Constraint automata have been introduced in [18] as a formalism to describe the behaviour and data flow in coordination models (such as the Reo language [18]); they can be considered as acceptors of *timed data streams* [18,7,19]. Constraint automata have been enriched with new features to create more expressive formalisms. On the one hand, *constraint automata with memory* (CAM) enrich constraint automata with a finite set of memory locations [6]. This extension allows one to handle infinite state spaces by enabling the values of each memory location to range over an infinite data domain. On the other hand, *soft constraint automata* (SCA) enrich constraint automata with soft constraints [3]. This extension allows one to express preference amongst different executions.

We present *soft constraint automata with memory* (SCAM): a generic framework that captures both CAM and SCA in a single formalism. Our approach differs significantly from both existing works with respect to its semantics. Originally, SCA are acceptors of tuples of *weighted timed data streams* [3,7]. In the current work, we interpret a SCAM as a special kind of soft constraint, encoding the same information in an alternative way.

3.1. Soft languages

Memory is the capacity to preserve information through time. Therefore, given a finite set of memory locations L, we model the behaviour of a location $v \in L$ as an infinite sequence of *timed variables*

$$(v, 0), (v, 1), \dots, (v, i), \dots,$$

where variable (v, i) represents the value of memory location $v \in L$ at time step $i \in \mathbb{N}_0$.

We write $\widehat{L} = L \times \mathbb{N}_0$ for the set of timed variables. We define the *k-th derivative* of a variable $x = (v, i) \in \widehat{L}$ as $x^k = (v, i)^k = (v, k + i)$. We define the *k-th derivative* of a set of variables $X \subseteq \widehat{L}$ as $X^k = \{x^k \mid x \in X\}$.

For notational convenience, we treat a timed variable $(v, 0) \in \widehat{L}$ and a plain variable v as equal, and we write a prime for the first derivative. For example, the expression $m' = a$ expands to the expression $(m, 1) = (a, 0)$ on timed variables.

Next, we extend the data domain $\mathcal{D}$ with a special symbol $* \notin \mathcal{D}$ that denotes "no data", and write $\mathcal{D}_* = \mathcal{D} \cup \{*\}$. We model a single execution of a SCAM as an assignment to timed variables.

Definition 12 *(Data stream).* A *data stream* is a map $\eta : \widehat{L} \to \mathcal{D}_*$.

Intuitively, $\eta(v, i) \in \mathcal{D}_*$ represents the data observed at location $v \in L$ and time step $i \geq 0$. If $\eta(v, i) = *$, no data is observed at location v and time step i. We define the k-th derivative $\eta^k : \widehat{L} \to \mathcal{D}_*$ of a data stream η as $\eta^k(v, i) = \eta(v, k + i)$, for all $v \in L$ and $i \geq 0$.

We can visualise a data stream η as an infinite table, with columns indexed by variables $v \in L$, rows indexed by non-negative integers $i \in \mathbb{N}_0$, and entries containing either $*$ or data from $\mathcal{D}$. For a time step $i \geq 0$, we can represent the i-th row of η as the partial map $\eta_i : L \rightharpoonup \mathcal{D}$, with $\mathrm{dom}(\eta_i) = \{v \in L \mid \eta(v, i) \neq *\}$ and $\eta_i(v) = \eta(v, i)$, for all $v \in \mathrm{dom}(\eta_i)$. We refer to η_i as the i-th *data assignment*. The empty function $\tau : L \rightharpoonup \mathcal{D}$, with $\mathrm{dom}(\tau) = \emptyset$, is also a valid data assignment. We use τ to represent an explicit silent step.

Definition 13 *(Soft languages).* A soft language over a CLM $\langle A, \leq, \otimes, \mathbf{1}\rangle$ is a function $c : (\widehat{L} \to \mathcal{D}_*) \to A$.

Suppose that we have a morphism $h : A \to B$. Then, we can view any soft language c over A as a soft constraint over B. Indeed, the composition $h \circ c$ that maps a data stream η to the value $h(c\eta) \in B$ constitutes a soft constraint over B. In particular, if B is the Boolean CLM $\mathbb{B}$, we can view a soft constraint as a crisp constraint that defines a set of accepted executions. Such a constraint corresponds naturally to a constraint automaton [18], which is thus subsumed by Definition 13.

Lemma 3 shows that soft constraints form a cylindric algebra. Thus, relevant notions, such as composition and hiding, carry over from soft constraints to SCAMs. It is straightforward to verify that all these notions correspond to their classical definitions in the literature.

3.2. Syntax

We fix a finite set of memory locations $\mathcal{X}$, a data domain $\mathcal{D}$, and a distributive and cancellative CLM $\mathbb{S}$. Recall the CLM of constraints $\mathbb{C}(V, D, \mathbb{S})$ from Lemma 2, for a set of variables V and domain of interpretation D.

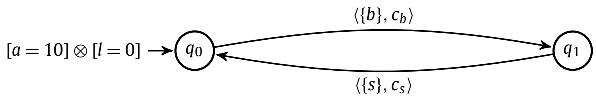

Fig. 1. A SCAM over the data domain $\mathbb{N}_0$ and CLM $\langle \mathbb{Z} \cup \{\pm\infty\}, \leq, +, 0\rangle$, where $c_b = [-b] \otimes [a' = a - b] \otimes [b \leq a] \otimes [l' = b]$ buys an affordable item and saves its price, and $c_s = [s] \otimes [a' = a + s] \otimes [l \leq s]$ sells that item for a higher price.

Definition 14 *(SCAM).* A *soft constraint automaton with memory* over $\mathcal{D}$ and $\mathbb{S}$ is a 6-tuple $\langle \mathcal{Q}, \mathcal{N}, \mathcal{X}, \longrightarrow, \mathcal{Q}_0, c_0\rangle$, such that

1. $\mathcal{Q}$ is a finite set of states,
2. $\mathcal{N}$ is a finite set of port variables,
3. $\mathcal{X}$ is a finite set of memory locations,
4. $\longrightarrow \subseteq \mathcal{Q} \times 2^{\mathcal{N}} \times \mathbb{C}(\widehat{\mathcal{N} \cup \mathcal{X}}, \mathcal{D}_*, \mathbb{S}) \times \mathcal{Q}$ is a finite set of transitions,
5. $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is a set of initial states, and
6. $c_0 \in \mathbb{C}(\widehat{\mathcal{N} \cup \mathcal{X}}, \mathcal{D}_*, \mathbb{S})$ is an initial constraint

such that $\mathcal{X} \cap \mathcal{N} = \emptyset$, $\mathrm{supp}(c_0) \subseteq \mathcal{X}$, and $(q, N, c, p) \in \longrightarrow$ implies that $\mathrm{supp}(c) \subseteq N \cup \mathcal{X} \cup \mathcal{X}'$ (where $\mathcal{X}' = \{x' \mid x \in \mathcal{X}\}$ is the set of first derivatives).

We usually write $q \xrightarrow{N,c} p$ instead of $(q, N, c, p) \in \longrightarrow$ and we call N the synchronisation constraint and c the guard of the transition, respectively. We say that a transition is *invisible* whenever $N = \emptyset$.

Different from [7], the condition $\mathrm{supp}(c) \subseteq N \cup \mathcal{X} \cup \mathcal{X}'$ means that the guards are soft constraints with a single time step look-ahead for memory locations. This is just a simplifying assumption: the following results would carry over smoothly.

Definition 15 *(Runs).* Let $\mathcal{T} = \langle \mathcal{Q}, \mathcal{X}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0, c_0\rangle$ be a SCAM. A *run* λ of $\mathcal{T}$ from $q \in \mathcal{Q}$ is an infinite sequence in $\longrightarrow^\omega$, with $\lambda_i = (p_i, N_i, c_i, q_i) \in \longrightarrow$, such that $p_0 = q$ and $q_i = p_{i+1}$, for all $i \geq 0$. We write $R(\mathcal{T}, q)$ for the set of runs of $\mathcal{T}$ from q and $R(\mathcal{T}) = \bigcup_{q \in \mathcal{Q}_0} R(\mathcal{T}, q)$ for the set of runs of $\mathcal{T}$.

The intuitive meaning of a SCAM $\mathcal{T}$ as an operational model for service queries is similar to the interpretation of transition systems as models for reactive systems. The states represent the configurations of a service. Transitions represent the one-step behaviour, where the meaning of a transition $p \xrightarrow{N,c} q$ is that we can move from configuration p to q, whenever

1. all ports in $n \in N$ perform an I/O operation,
2. all other ports in $\mathcal{N} \setminus N$ perform no I/O operation,
3. all ports/memory locations in $N \cup \mathcal{X} \cup \mathcal{X}'$ satisfy the guard c.

Each assignment to ports in N represents the data exchanged by the I/O operations through these ports, while assignments to variables in $\mathcal{X}$ and $\mathcal{X}'$ represent the data in memory locations before and after the transition.

For example, a transition $p \xrightarrow{\{a,b\},[x=a]\otimes[x'=b]} q$ from state p to state q fires ports a and b, the value at port a is equal to the current value of the memory x, and the next value at x is equal to the current value at port b. If they are not hidden, port variables can be shared with other SCAMs (cf. Definition 17), while memory variables are not shared (cf. Theorem 2).

Example 9 *(A SCAM for buying and selling).* We describe an agent that prefers to buy an item as cheap as possible, and prefers to maximise its profit. We use the set $\mathbb{N}_0$ of natural numbers as a data domain, and we use the extended integers $\langle \mathbb{Z} \cup \{\pm\infty\}, \leq, +, 0\rangle$ as preference values. In particular, every datum can be viewed as a preference value.

Figure 1 shows a (deterministic) SCAM for buying and selling. The set of ports $\mathcal{N}$ is $\{b, s\}$, where the value at b is the purchase price of an item and the value at s is the selling price of an item. The set of variables $\mathcal{X}$ is $\{a, l\}$, where a is the current balance and l is the price of current item. The soft constraint c_b buys $(a' = a - b)$ an affordable $(b \leq a)$ item, and stores its value $(l' = b)$. The preference of $-b$ ensures that maximising preference amounts to minimising purchase price. The soft constraint c_s sells the current item $(a' = a + s)$ for a higher price $(l \leq s)$. The preference of s ensures that maximising preference amounts to maximising selling price.

3.3. Semantics

Recall the lexicographically-ordered CLM of streams $\mathbb{S}^\omega$ from Theorem 1. We interpret a SCAM $\mathcal{T}$ as a soft language

$$L(\mathcal{T}) : (\widehat{\mathcal{N} \cup \mathcal{X}} \to \mathcal{D}_*) \to \mathbb{S}^\omega$$

assigning a stream from $\mathbb{S}^\omega$ to any execution. In this way, the constraint $L(\mathcal{T})$ can be seen as the language of the SCAM $\mathcal{T}$.

x	b	s	a	l
$\eta(x,0)$	6	$*$	10	0
$\eta(x,1)$	$*$	7	4	6
$\eta(x,2)$	$\vdots$	$\vdots$	11	$\vdots$

Fig. 2. Data stream for the SCAM from Example 9, wherein the agent starts with 10 units of money, buys an item for 6 units, and sells it for 7 units.

We describe the intuitive semantics of a SCAM. Let $\eta : \widehat{\mathcal{N} \cup \mathcal{X}} \to \mathcal{D}_*$ be a data stream. First, we define the preference stream $c_\lambda \eta \in \mathbb{S}^\omega$ of η with respect to a run $\lambda = t_0 t_1 t_2 \cdots \in R(\mathcal{T}, q)$ from a state $q \in \mathcal{Q}$. We compute the initial preference $c_0\eta \in \mathbb{S}$ and, for every transition $t_i = (p_i, N_i, c_i, q_i)$ in the run λ, we compute the preference $c_{t_i}\eta^i \in \mathbb{S}$ of transition t_i, where η^i is the ith derivative of η, and c_{t_i} is the soft constraint composed from the guard c_i and the synchronisation constraint N_i. For $i \geq 0$, consider the composition $a_i = c_0\eta \otimes c_{t_i}\eta^i \in \mathbb{S}$. If the initial condition and all transition guards and synchronisation constraints are satisfied (i.e., a_i cancellative, for all $i \geq 0$), then the preference stream of η equals $c_\lambda \eta = a_0 a_1 a_2 \cdots \in \mathbb{S}^\omega$. Otherwise, we set $c_\lambda \eta = \bot^\omega$.

Next, we define the preference value of the data stream η assigned by the SCAM $\mathcal{T}$ as the least upper bound (in the lexicographically-ordered CLM of streams $\mathbb{S}^\omega$) over all the possible runs that start from an initial state. The lexicographic order implies that, at any given state, the SCAM prefers to take the outgoing transition of maximal preference. Indeed, any run that starts with an outgoing transition of suboptimal preference results in a preference stream that is suboptimal in the lexicographic order.

Finally, we hide all memory locations, preventing SCAMs from synchronising on shared memory locations (Theorem 2).

Definition 16 *(SCAM semantics).* Let $\mathcal{T} = \langle \mathcal{Q}, \mathcal{N}, \mathcal{X}, \longrightarrow, \mathcal{Q}_0, c_0 \rangle$ be a SCAM. The semantics of a transition $t = (p, N, c, q) \in \longrightarrow$ is a soft constraint $c_t \in \mathbb{C}(\widehat{\mathcal{N} \cup \mathcal{X}}, \mathcal{D}_*, \mathbb{S})$ defined as

$$c_t = c \otimes \bigotimes_{n \in N} [n \neq *] \otimes \bigotimes_{n \in \mathcal{N} \setminus N} [n = *].$$

The semantics of a run $\lambda = t_0 t_1 t_2 \cdots \in R(\mathcal{T}, q)$ from a state $q \in \mathcal{Q}$ is a soft constraint c_λ that maps a data stream $\eta : \widehat{\mathcal{N} \cup \mathcal{X}} \to \mathcal{D}_*$ to the preference stream $c_\lambda \eta$ defined as

$$c_\lambda \eta = \begin{cases} a_0 a_1 a_2 \cdots & \text{if } a_i = c_0\eta \otimes c_{t_i}\eta^i \text{ is cancellative, for all } i \geq 0, \\ \bot^\omega & \text{otherwise} \end{cases}$$

The accepted language of $\mathcal{T}$ at $q \in \mathcal{Q}$ is defined as

$$L(\mathcal{T}, q) = \bigvee \{ c_\lambda \mid \lambda \in R(\mathcal{T}, q) \}.$$

The language of $\mathcal{T}$ is defined as

$$L(\mathcal{T}) = \exists_{\widehat{\mathcal{X}}} \bigvee \{ L(\mathcal{T}, q) \mid q \in \mathcal{Q}_0 \}.$$

Definition 16 deals exclusively with infinite paths in a SCAM $\mathcal{T}$: if a state q has no outgoing transitions, then $c(\mathcal{T}, q)\eta = \bot$, for every data stream η.

Example 10 *(The language of business).* Let $\mathcal{T}$ be the SCAM from Example 9. Consider a data stream $\eta : \widehat{\{b, s, a, l\}} \to \mathbb{N}_0$ whose prefix is defined in Figure 2. From $\eta(a, 0) = 10$ and $\eta(l, 0) = 0$, it follows that

$$c_0\eta = ([a = 10] \otimes [l = 0])\eta = \mathbf{1},$$

which means that the initial condition is satisfied. There exists only one possible run $\lambda = t_0 t_1 \cdots$ in $\mathcal{T}$ from the initial state q_0. Hence, the stream of preferences associated with η satisfies

$$L(\mathcal{T})\eta = (\exists_{\widehat{\{a,l\}}} \bigvee \{c_\lambda\})\eta = (\exists_{\widehat{\{a,l\}}} c_\lambda)\eta = c_\lambda \eta,$$

where the last equality follows from the fact that the preferences are independent of the memory locations a and l. Concretely, the stream of preferences $L(\mathcal{T})\eta = a_0 a_1 \cdots$ satisfies

$$a_0 = c_0\eta \otimes c_{t_0}\eta^0 = \mathbf{1} \otimes c_b\eta^0 = -\eta^0(b, 0) = -\eta(b, 0) = -6$$
$$a_1 = c_0\eta \otimes c_{t_1}\eta^1 = \mathbf{1} \otimes c_s\eta^1 = \eta^1(s, 0) = \eta(s, 1) = 7$$

The lexicographic order on preference streams ensures that any other data stream $\rho : \widehat{\{b, s, a, l\}} \to \mathbb{N}_0$, for which $\rho(b, 0) < 6$, satisfies $L(\mathcal{T})\eta <_l L(\mathcal{T})\rho$, which means that the data stream ρ is preferred over η. In other words, the SCAM $\mathcal{T}$ prefers to minimise the purchase price.

3.4. SCAM composition

We now introduce the product of automata, extending [3, Definition 5].

Definition 17 *(Soft join).* Let $\mathcal{T}_i = (\mathcal{Q}_i, \mathcal{X}_i, \mathcal{N}_i, \to_i, \mathcal{Q}_{0i}, c_{0i})$ for $i \in \{0, 1\}$ be two SCAMs over $\mathcal{D}$ and $\mathbb{S}$, with $(\mathcal{N}_0 \cup \mathcal{N}_1) \cap (\mathcal{X}_0 \cup \mathcal{X}_1) = \emptyset$. Then, their soft product $\mathcal{T}_0 \bowtie \mathcal{T}_1$ is the tuple $\langle \mathcal{Q}_0 \times \mathcal{Q}_1, \mathcal{X}_0 \cup \mathcal{X}_1, \mathcal{N}_0 \cup \mathcal{N}_1, \longrightarrow, \mathcal{Q}_{00} \times \mathcal{Q}_{01}, c_{00} \otimes c_{01}\rangle$ where $\longrightarrow$ is the smallest relation that satisfies the rule

$$\frac{q_0 \xrightarrow{N_0, c_0}_0 p_0, \quad q_1 \xrightarrow{N_1, c_1}_1 p_1, \quad N_0 \cap \mathcal{N}_1 = N_1 \cap \mathcal{N}_0}{\langle q_0, q_1\rangle \xrightarrow{N_0 \cup N_1, c_0 \otimes c_1} \langle p_0, p_1\rangle}$$

The rule applies when there is a transition in each automaton such that they can fire together. This happens only if the two local transitions agree on the subset of shared ports that *fire* (which is empty, if no ports are shared). The transition in the resulting automaton is labelled with the union of the name sets on both transitions, and the constraint is the conjunction of the constraints of the two transitions.

Note that the new automaton may include asynchronous executions: it suffices that the SCAM is *reflexive*, i.e., every q has an idling transition $q \xrightarrow{\emptyset, \mathbf{1}} q$. To avoid such idling transitions to be of maximal preference, we must use a bipolar CLM of preferences, wherein $\top > \mathbf{1}$.

We now express the composition of SCAM in Definition 17 in terms of composition of languages as defined in Lemma 2.

Theorem 2 *(Correctness of soft join). Let $\mathcal{T}_0$ and $\mathcal{T}_1$ be two SCAMs sharing no memory location. Then, $L(\mathcal{T}_0 \bowtie \mathcal{T}_1) = L(\mathcal{T}_0) \otimes L(\mathcal{T}_1)$.*

Proof. We first show that, for all $(q_0, q_1) \in \mathcal{Q}_0 \times \mathcal{Q}_1$, we have

$$L(\mathcal{T}_0 \bowtie \mathcal{T}_1, (q_0, q_1)) = \bigvee \{c_{\rho_0} \otimes c_{\rho_1} \mid \rho_i \in R(\mathcal{T}_i, q_i), i \in \{0, 1\}\}. \tag{1}$$

Let $\rho \in R(\mathcal{T}_0 \bowtie \mathcal{T}_1, (q_0, q_1))$ be a run from (q_0, q_1). By construction of $\mathcal{T}_0 \bowtie \mathcal{T}_1$, we find runs $\rho_i \in R(\mathcal{T}_i, q_i)$, for $i \in \{0, 1\}$, such that $c_\rho = c_{\rho_0} \otimes c_{\rho_1}$. Hence, c_ρ is less than or equal to the right-hand side of Equation (1). Since, ρ is arbitrary, we conclude the $\leq$ part of Equation (1).

For $i \in \{0, 1\}$, let $\rho_i \in R(\mathcal{T}_i, q_i)$ be, a run from q_i. If ρ_0 and ρ_1 are not compatible according to the rule in Definition 17, we have $c_{\rho_0} \otimes c_{\rho_1} = [\perp^\omega] \leq L(\mathcal{T}_0 \bowtie \mathcal{T}_1, (q_0, q_1))$. If ρ_0 and ρ_1 are compatible according to the rule in Definition 17, we find a run $\rho \in R(\mathcal{T}_0 \bowtie \mathcal{T}_1, (q_0, q_1))$ from (q_0, q_1), such that $c_{\rho_0} \otimes c_{\rho_1} = c_\rho \leq L(\mathcal{T}_0 \bowtie \mathcal{T}_1, (q_0, q_1))$. Since ρ_0 and ρ_1 are arbitrary, we conclude the $\geq$ part of Equation (1), which proves Equation (1).

Using Equation (1) and Theorem 1, we find, for $(q_0, q_1) \in \mathcal{Q}_0 \times \mathcal{Q}_1$, that

$$\begin{aligned} L(\mathcal{T}_0 \bowtie \mathcal{T}_1, (q_0, q_1)) &= \bigotimes_{i=0}^{1} \bigvee \{c_{\rho_i} \mid \rho_i \in R(\mathcal{T}_i, q_i)\} \\ &= L(\mathcal{T}_0, q_0) \otimes L(\mathcal{T}_1, q_1) \end{aligned}$$

Since no memory is shared, we have $\mathcal{X}_0 \cap \mathcal{X}_1 = \emptyset$. Then

$$\begin{aligned} L(\mathcal{T}_0 \bowtie \mathcal{T}_1) &= \exists_{\widehat{\mathcal{X}}} \bigvee \{L(\mathcal{T}_0, q_0) \otimes L(\mathcal{T}_1, q_1) \mid q_0 \in \mathcal{Q}_{00}, q_1 \in \mathcal{Q}_{01}\} \\ &= \exists_{\widehat{\mathcal{X}_0}} \exists_{\widehat{\mathcal{X}_1}} \bigotimes_{i=0}^{1} \bigvee \{L(\mathcal{T}_i, q_i) \mid q_i \in \mathcal{Q}_{0i}\} \\ &= \bigotimes_{i=0}^{1} \exists_{\widehat{\mathcal{X}_i}} \bigvee \{L(\mathcal{T}_i, q_i) \mid q_i \in \mathcal{Q}_{0i}\} = L(\mathcal{T}_0) \otimes L(\mathcal{T}_1), \end{aligned}$$

which proves the result. □

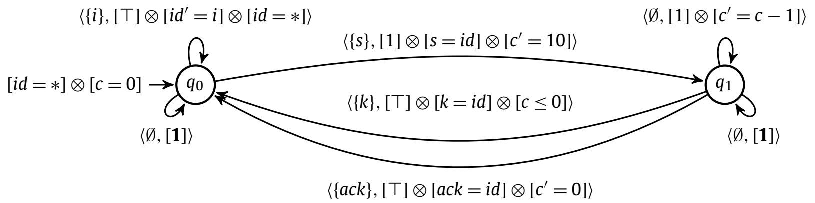

Fig. 3. The operator's SCAM $\mathcal{A}$ over an arbitrary data domain and the CLM $\langle \mathbb{Z}\cup\{\pm\infty\},\le,+,0\rangle$.

3.5. SCAM hiding

The hiding operator [18] abstracts the details of the internal communication in a constraint automaton. For SCA [3, Definition 6], the hiding operator $\exists_O \mathcal{T}$ removes from the transitions all the information about the ports in $O \subseteq \mathcal{N}$, including those in the (support of the) constraints. The definition smoothly extends over SCAMs: in fact, since we allow silent transitions, our definition is much more compact.

Definition 18 *(Soft hiding).* Let $\mathcal{T} = \langle \mathcal{Q}, \mathcal{X}, \mathcal{N}, \longrightarrow, \mathcal{Q}_0\rangle$ be a SCAM and $O \subseteq \mathcal{N}$ a set of ports. Then $\exists_O \mathcal{T}$ is the SCAM $\langle \mathcal{Q}, \mathcal{X}, \mathcal{N}\setminus O \longrightarrow_*, \mathcal{Q}_0\rangle$ where $\longrightarrow_*$ is defined by $q \xrightarrow{N\setminus O,\exists_O c}_* p$ if $q \xrightarrow{N,c} p$.

We express the correctness of hiding in terms of the cylindric operator on soft constraints from Lemma 3.

Theorem 3 *(Correctness of soft hiding). Let $\mathcal{T}$ be a SCAM and O a set of its ports. Then $L(\exists_O \mathcal{T}) = \exists_O L(\mathcal{T})$.*

Proof. We prove that, for all $q \in \mathcal{Q}$, we have

$$L(\exists_O \mathcal{T}, q) = \bigvee\{\exists_O c(\rho) \mid \rho \in R(\mathcal{T}, q)\}. \tag{2}$$

By construction of $\exists_O \mathcal{T}$ in Definition 18, we have a natural 1-1 correspondence between runs $\rho \in R(\exists_O \mathcal{T}, q)$ in $\exists_O \mathcal{T}$ from q and runs $\rho' \in R(\mathcal{T}, q)$ in $\mathcal{T}$ from q, which satisfies $c_\rho = \exists_O c_{\rho'}$. Using the same approach used in the proof of Theorem 2 (i.e., $\le$ and $\ge$ on LUBs), we conclude that Equation (2) holds. For all $q \in \mathcal{Q}$, we now find that $L(\exists_O \mathcal{T}, q) = \exists_O L(\rho, q)$. Hence, $L(\exists_O \mathcal{T}) = \exists_O L(\mathcal{T})$, which proves the result. $\square$

4. A case study

We present an example that illustrates the operations of composition and hiding for SCAMs. The example consists of an interrupt management-system tied to a data flow of information. Even if academic, it is rooted into concepts widely adopted by several real-world examples, e.g., a computer CPU receiving hardware and software interrupts.

We show that, even if the machine has the ability to keep executing a process, in the presence of a *kill* signal sent by the operator, the machine chooses to stop. The given construction could be adapted to express the case where more than one machine is controlled by an operator.

As a carrier for the preferences of the soft constraints, we use the CLM of extended integers $\mathbb{S} = \langle \mathbb{Z}\cup\{\pm\infty\},\le,+,0\rangle$ from Example 4. Recall that a tautology has preference $\mathbf{1}$, which is the element $0 \in \mathbb{S}$, and a false constraint has the least preference $-\infty \in \mathbb{S}$. We refer to $+\infty$ as the element $\top$ and to $-\infty$ as the element $\bot$.

4.1. Operator

Let $\mathcal{A} = \langle \{q_0, q_1\}, \{k, s, i, ack\}, \{c, id\}, \longrightarrow, \{q_0\}, [id = *]\otimes[c = 0]\rangle$ be the SCAM representation of the operator, with transition relation $\longrightarrow$ as in Figure 3. Initially, the memory *id* is empty. The operator, in state q_0, waits to receive a signal i and stores the value carried by the signal in the memory location *id*. Then, the operator waits for a signal $s \neq *$ to arrive, and takes the outgoing transition from q_0 to q_1 only if the value of s equals the current value of memory location *id*. Simultaneously, the operator starts a counter by setting the memory location c to 10. Being in state q_1, the operator repeatedly decreases its counter c. When the value of memory location c becomes negative, the operator sends a kill signal carrying the value stored in memory location *id*. If the operator receives, in state q_1, an acknowledge signal *ack* with the value stored in the memory location *id*, the operator sets the counter memory location c to 0 and returns to its initial state q_0.

The preference values in Figure 3 ensure that, if the guards of the self transition at q_1 and the transitions from q_1 to q_0 are satisfied by the same assignment, the operator prefers to send a kill or acknowledge signal (transitions from q_1 to q_0) instead of decreasing its counter (self transition at q_1).

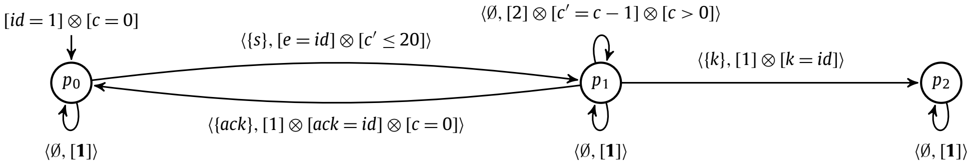

Fig. 4. The machine's SCAM $\mathcal{B}$ over an arbitrary data domain and the CLM $\langle \mathbb{Z} \cup \{\pm\infty\}, \leq, \otimes, \mathbf{1}\rangle$.

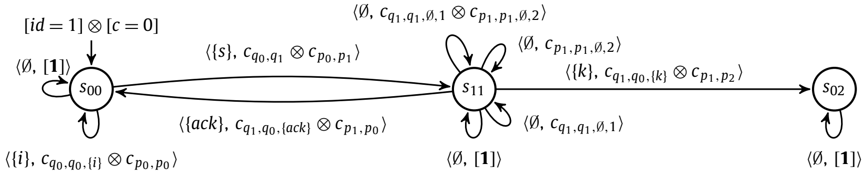

Fig. 5. The product $\mathcal{A} \bowtie \mathcal{B}$ of the machine's SCAM and operator's SCAM, where $s_{ij} = (q_i, p_j)$ and $c_{x,y,N,e}$ is the constraint of the underlying SCAM labelling transition from state x to state y with synchronisation set N and constant preference e. If clear from the context, some elements from x, y, N, e are omitted to identify the constraint.

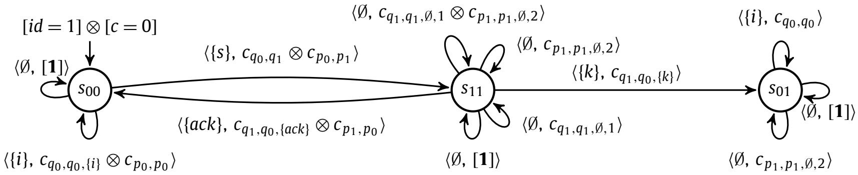

Fig. 6. The product $\mathcal{A} \bowtie \exists_k(\mathcal{B})$ of the machine's SCAM and operator's SCAM after hiding k in the machine, using the same notation for states and guards as in Figure 5.

4.2. Machine

Let $\mathcal{B} = \langle\{p_0, p_1, p_2\}, \{k, s, ack\}, \{c, id\}, \longrightarrow, \{p_0\}, [id = 1] \otimes [c = 0]\rangle$ be the SCAM representation of a machine, whose transition relation $\longrightarrow$ is shown in Figure 4. The machine starts in p_0, with the identity value 1 stored in the memory id. Whenever the value observed on port s corresponds to its identity id, the machine can start executing and moves from state p_0 to p_1. In state p_1, the execution of the machine is simulated by decreasing a counter from a non-deterministically selected initial value of at most 20. Once the counter reaches 0, the machine sends an acknowledgement with its own id value and gets back to state p_0. At any point, however, the machine can be interrupted by a kill signal and goes to state p_2.

The constraints of the machine ensure that the machine terminates, if a kill signal is received. In absence of a kill signal, the machine prefers to execute the process before sending the acknowledgement.

4.3. Composition

Figure 5 shows the SCAM of the composition $\mathcal{A} \bowtie \mathcal{B}$ of the operator $\mathcal{A}$ and the machine $\mathcal{B}$. Their composition synchronises on the shared ports k, ack, and s between $\mathcal{A}$ and $\mathcal{B}$. While the I/O direction of a port is not explicitly mentioned, one should think of the port k as an input port for the machine and an output port for the operator. Similarly, the port ack is used as an output port for the machine and input port for the operator.

If satisfied, the constraint $c_{q_1,q_1,\emptyset,1} \otimes c_{p_1,p_1,\emptyset,2}$ (see the caption of Figure 5 for the notation) evaluates to preference $2 + 1 = 3$, and the constraint $c_{q_1,q_0,\{k\}} \otimes c_{p_1,p_2}$ evaluates to preference $\top$. Since $3 < \top$, when the counter memory of the operator reaches 0, the run where the kill signal is sent has higher preference than the run where the machine keeps executing its process.

4.4. Hiding

We compare the product $\mathcal{A} \bowtie \mathcal{B}$ with the product $\mathcal{A} \bowtie \exists_k(\mathcal{B})$, wherein we hide the port k in $\mathcal{B}$. As displayed in Figure 6, the composite system goes to the state s_{01}, where the transition $\langle \emptyset, c_{p_1,p_1,\emptyset,2}\rangle$ can still be taken (i.e. the machine is still running). Hence, hiding the kill signal in $\mathcal{B}$ does not force the machine to terminate its execution. Note that state s_{01} is a deadlock: the operator cannot receive an acknowledge signal or send a kill signal.

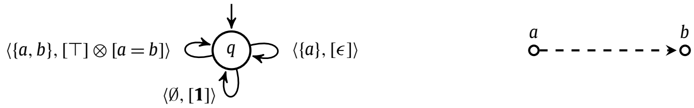

Fig. 7. SCAM representation of the context-sensitive lossysync channel, and its Reo notation. The passing transition, $\langle\{a, b\}, [\top] \otimes [a = b]\rangle$, has priority over the losing transition, $\langle\{a\}, [\epsilon]\rangle$, and the losing transition has priority over the idling transition, $\langle\emptyset, [\mathbf{1}]\rangle$.

5. Application to context-sensitivity

The presented SCAM framework can be applied to model context-sensitivity, which is also known as context-dependency or as context-awareness.

Definition 19. A component is context-sensitive if an I/O request by the environment can disable one of its actions.

One source of context-sensitivity is priority. If an I/O request by the environment enables a high-priority action, then previously enabled actions of lower priority become disabled.

The notion of context-sensitivity received considerable attention in the Reo community. The primal example of a context-sensitive Reo connector is a lossysync channel, which accepts a datum d from its input end, and either atomically offers d at its output end, or loses d if the output is not ready to accept. The literature offers a variety of semantic models that encode context-sensitive behaviour, namely *colouring semantics* [20], *augmented Büchi automata of records* [21], *intentional automata* [22], and *guarded automata* [23]. Context-sensitivity can be encoded in context-insensitive models by adding dual ports [24]. Although we consider here context-sensitivity in the realm of Reo, we stress that context-sensitivity is a fundamental concept that applies to languages other than Reo.

The environment of a connector can be represented in at least two ways. On the one hand, augmented Büchi automata of records, intentional automata, and guarded automata represent the environment as the subset of ports of the connector that have pending requests. On the other hand, colouring semantics and the encoding in [24] represent the environment as another connector of identical type that composes with the current connector.

All context-sensitive models for Reo [20–24] have special syntax to detect the presence or absence of pending I/O requests. Intentional, guarded, and augmented Büchi automata of records query the presence of I/O requests via a Boolean guard. The colouring semantics use two colours for the absence of data flow, allowing the connector to detect the presence of I/O requests. The dual ports in the encoding in [24] serve the same purpose as the two colours in the colouring semantics.

We now propose a context-sensitive semantics without any syntax to detect the presence or absence of pending I/O requests. As such, our approach is arguably simpler than existing approaches. The basic idea is to distinguish four types of transitions, namely

1. *illegal* transitions with unsatisfiable soft constraint,
2. *idling* transition (i.e., a silent self-loop transition),
3. *losing* transition (as in the lossysync),
4. *regular* transitions (i.e., legal, non-idling, non-losing transitions).

Let us consider the distributive CLM $\mathbb{K} \times \mathbb{B}$, and its sub-CLM $\mathcal{E}$ whose set of elements is composed by $\bot = (\bot, \bot)$, $\mathbf{1} = (\mathbf{1}, \top)$, $\epsilon = (\top, \bot)$, and $\top = (\top, \top)$, which are the types respectively of illegal, idling, losing, and regular transitions. It is easy to see that $\mathcal{E}$ is indeed a CLM and it is distributive.

The partial order $\leq$ on $\mathcal{E}$ induces a priority relation on the set of enabled transitions in a SCAM over $\mathcal{E}$. If present, the connector fires any enabled transition of highest priority. The multiplication $\otimes$ propagates the types through composition.

Figure 7 shows the SCAM representation of the lossysync channel. We verify that the SCAM representation of lossysync behaves as desired, if it operates in isolation. Note that $\bot < \mathbf{1} < \top$ and $\bot < \epsilon < \top$, but ϵ and $\mathbf{1}$ are incomparable. If the lossysync has no pending I/O operations on a or b, then the idling transition, $\langle\emptyset, [\mathbf{1}]\rangle$, is the only enabled transition. If there is a pending input at port a, then the losing transition, $\langle\{a\}, [\epsilon]\rangle$, and the idling transition, $\langle\emptyset, [\mathbf{1}]\rangle$, are enabled. Since $\mathbf{1}$ and ϵ are incomparable, the choice between losing and idling is non-deterministic. If there are pending I/O operations on both a and b, then all transitions are enabled. In particular, the passing transition $\langle\{a, b\}, [\top] \otimes [a = b]\rangle$ has priority over the others.

We now verify that the SCAM representation of lossysync behaves as desired, if it operates in a composition. Our approach crucially relies on the correct identification of the three different transition types, namely the illegal, idling, and losing transitions. We define the type of a global transition $\tau = \tau_1 \mid \cdots \mid \tau_n$ as follows

1. τ is illegal if one local transition τ_i is illegal,
2. τ is idling if all local transitions τ_i are idling,
3. τ is losing if τ is not illegal and one transition τ_i is losing,
4. τ is regular if τ is not idling and all local transitions τ_i are idling or regular.

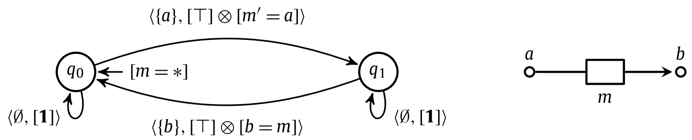

Fig. 8. SCAM representation of the fifo channel, and its Reo notation.

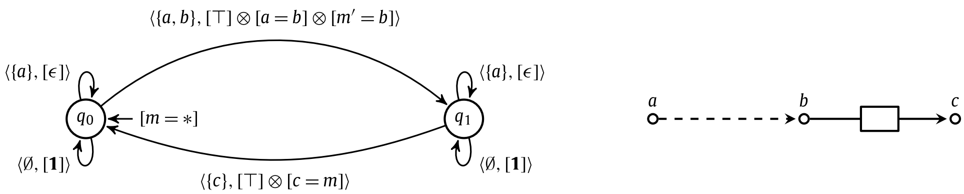

Fig. 9. Composition of the lossysync and fifo in Figs. 7 and 8. If the fifo channel can drain its buffer, then the lossysync channel cannot lose any datum.

The following result ensures that the transition types are correctly propagated through the composition of SCAMs.

Lemma 4. *Let $a_1, \ldots, a_n \in \mathcal{E}$ and $x = a_1 \otimes \cdots \otimes a_n$. Then for $I = \{1, \ldots, n\}$ we have*

1. *$x = \bot$ if and only if $\exists i \in I.\ a_i = \bot$,*
2. *$x = \mathbf{1}$ if and only if $\forall i \in I.\ a_i = \mathbf{1}$,*
3. *$x = \epsilon$ if and only if $(\exists i \in I.\ a_i = \epsilon) \wedge (\forall i \in I.\ a_i \neq \bot)$,*
4. *$x = \top$ if and only if $(\exists i \in I.\ a_i = \top) \wedge (\forall i \in I.\ a_i \in \{\mathbf{1}, \top\})$.*

Proof. Follows immediately from the definition of $\mathcal{E}$. $\square$

Example 11. Consider the composition C of the lossysync channel and the fifo channel, as depicted in Figure 9. Suppose that C is in state q_1, which means that the fifo channel is full. If there is a pending I/O request on port c, then the data can be taken out. And since the fifo channel can drain its buffer, the lossysync channel cannot lose any datum.

It is important to observe that the bipolar approach is essential for the construction of our context-sensitive model. To see this, note that $\mathbf{1}$ is the only sensible value for an idling transition. Otherwise, composition with an idling transition would change the preferences. If $\mathbf{1} = \top$ holds (as in absorptive semirings), then the priority of the losing transition is necessarily lower than the priority of the idling transition. Hence, any component would prefer idling (which is always possible) over losing, which is clearly undesirable.

6. Related work on constraint automata

The closest work to what we present in this paper concerns other extensions of constraint automata (CAs), mostly proposed in the literature about Reo. In the following of this section we survey some of such proposals.

Quantitative CAs (QCAs) are introduced in [25,26] with the aim of describing the behaviour of connectors tied to their *quality of service* (QoS), e.g., a reliability measure or the shortest transmission time. Similarly to CAs, the states of a QCAs correspond to the internal states of the connector it models. The label on a transition consists of a firing set, a data constraint, and a cost that represents a QoS metric. QCAs differ from *timed* [27] and *probabilistic* [28] constraint automata, because these latter two classes of models describe functional aspects of connectors, while QoS represents non-functional aspects.

As applications, SCAs have been already used in [3,29] and [4,5,30]. Different from previous related work, the main motivation behind SCAs is to associate an action with a preference. In [3,29] the authors present a formal framework that is able to discover stateful web services, and to rank the results according to a similarity score expressing the affinities between the query, asked by a user, and the services in a database. Preference for the similarity between the query and each service is modelled through SCAs. In the second group of works instead, the authors advance a framework that facilitates the construction of autonomous agents in a compositional fashion; these agents are 'soft', in that their actions are associated with a preference value, and agents may or may not execute an action depending on a threshold preference. Hence, at design-time, SCAs can be used to reason about the behaviour of the components in an uncertain physical world, i.e., to model and verify the behaviour of cyber-physical systems.

Research on SCAs is currently a trending topic among all the different lines concerning Reo. An example is [31], where the authors describe two complementary approaches to the specification and analysis of robust cyber-physical agent systems: the first one focuses on abstract theoretical concepts based on automata and temporal logics, called *soft component* *automata*; the second approach describes a concrete experimental approach based on executable rewriting logic specifications, simulation, search, and model checking, called *soft agents* [31]. The soft agents framework combines ideas from several previous works: *i)* the use of soft constraints and SCAs for specifying robust and adaptive behaviour, *ii)* partially ordered knowledge sharing for communication in disrupted environments, and *iii)* the real-time Maude approach to timed systems.

The work in [6] extensively presents a kind of CA (there named as `W/MC`) consisting of a finite set of states, a finite set of transitions, three sets of directed ports, and a set of memory cells. The presence of memory cells in `W/MC` allows for explicitly modelling the content of buffers, instead of using states. The main difference is represented by the fact that constraints are crisp, and consequently they do not allow for representing preference values, as needed by application summarised in the following paragraph.

Still concerning cyber-physical systems, the related literature is represented by several works, as for example [30] and [32]. In [30] the authors formalise soft agents in the Maude rewriting logic system [33]. The most important features of this framework are the explicit representation of the physical state, the cyber perception of this state, the robust communication via sharing of partially ordered knowledge, and the robust behaviour based on soft constraints. In [32] the authors address the problem of finding what local properties the agents in a cyber-physical system have to satisfy to guarantee a global required property ϕ; preferences are modelled via semirings on actions, and verified through a model checking function. Note that also all the examples in [4] use SCAs (with preferences) to model the behaviour of cyber-physical systems.

The feature of enhancing automata with memory has roots in the dawn of computer science. In this way, an automaton can base its transition on both the current symbol being read and the values stored in the memory; moreover, it can issue commands to the memory device whenever it makes a transition. For example, *pushdown automata* (*PDAs*) employ a stack through which operations can be determined by the first element on such a data structure; a transition rule optionally pops the top of the stack, and optionally pushes new symbols onto the stack. *Stack automata* allow access to and operations on deeper elements instead, and can recognise a strictly larger set of languages than PDAs [34]. Applications may concern also computational models in biology: e.g., automata can use memory to stabilise the behaviour of modelled proteins [35].

A conclusive related work is represented by [36], where Reo channels are annotated with stochastic values for data arrival rates at channel ends and processing delay rates at channels. Automata are thus stochastically extended in order to compositionally derive a QoS-aware semantics for Reo. The semantics is given by translating a component into *continuous-time Markov chains*. Our approach deals with preferences by using a more general approach: we do not only consider time but different systems of preferences, as long as they can be cast in the algebraic structure we present in Section 3.

7. Conclusions

We have reworked soft constraint automata as originally proposed in [3,4], with the dual purpose of extending the underlying algebraic structure, in order to model both positive and negative preferences, and adding memory locations as originally provided for 'standard' constraint automata [6].

As future work, we have many directions in mind. First, we would like to extend existing Reo compilers [37,38] to a SCAM-based compiler. Our results allow the user to conveniently compile context-sensitive connectors.

Next, we would like to exploit the properties of soft constraints to give additional operators on SCAMs, such as operators for port *renaming* or for *determinising* guards by adding $[m' = m]$, whenever m' is unbound.

Finally, we would like to encode the behaviour of SCAMs into a *concurrent constraint programming* language [10]. Such languages provide agents with actions to *tell* (i.e., add) and *ask* (i.e., query) constraints to a centralised store of information; this store represents a *constraint satisfaction problem*, and standard heuristic-based techniques might be applied to find a solution to complex conditions on *filter channels* [39].

Declaration of competing interest

We declare no conflict of interest.

References

[1] F. Arbab, Reo: a channel-based coordination model for component composition, Math. Struct. Comput. Sci. 14 (3) (2004) 329–366.
[2] S.T.Q. Jongmans, F. Arbab, Overview of thirty semantic formalisms for Reo, Sci. Ann. Comput. Sci. 22 (1) (2012) 201–251.
[3] F. Arbab, F. Santini, Preference and similarity-based behavioral discovery of services, in: M.H. ter Beek, N. Lohmann (Eds.), WS-FM 2012, in: LNCS, vol. 7843, Springer, 2012, pp. 118–133.
[4] T. Kappé, F. Arbab, C.L. Talcott, A compositional framework for preference-aware agents, in: M. Kargahi, A. Trivedi (Eds.), V2CPS@IFM 2016, in: EPTCS, vol. 232, 2016, pp. 21–35.
[5] T. Kappé, F. Arbab, C.L. Talcott, A component-oriented framework for autonomous agents, in: J. Proença, M. Lumpe (Eds.), FACS 2017, in: LNCS, vol. 10487, Springer, 2017, pp. 20–38.
[6] S.T.Q. Jongmans, T. Kappé, F. Arbab, Constraint automata with memory cells and their composition, Sci. Comput. Program. 146 (2017) 50–86.
[7] K. Dokter, F. Gadducci, F. Santini, Soft constraint automata with memory, in: F.S. de Boer, M.M. Bonsangue, J. Rutten (Eds.), It's All About Coordination, in: LNCS, vol. 10865, Springer, 2018, pp. 70–85.
[8] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint satisfaction and optimization, J. ACM 44 (2) (1997) 201–236.
[9] V.A. Saraswat, M.C. Rinard, P. Panangaden, Semantic foundations of concurrent constraint programming, in: D.S. Wise (Ed.), POPL 1991, ACM Press, 1991, pp. 333–352.

[10] F. Gadducci, F. Santini, L.F. Pino, F.D. Valencia, Observational and behavioural equivalences for soft concurrent constraint programming, Log. Algebraic Methods Program. 92 (2017) 45–63.
[11] F. Gadducci, F. Santini, Residuation for bipolar preferences in soft constraints, Inf. Process. Lett. 118 (2017) 69–74.
[12] J. Golan, Semirings and Affine Equations over Them: Theory and Applications, Kluwer, 2003.
[13] S. Bistarelli, F. Gadducci, Enhancing constraints manipulation in semiring-based formalisms, in: G. Brewka, S. Coradeschi, A. Perini, P. Traverso (Eds.), ECAI 2006, in: FAIA, vol. 141, IOS Press, 2006, pp. 63–67.
[14] F. Gadducci, M.M. Hölzl, G.V. Monreale, M. Wirsing, Soft constraints for lexicographic orders, in: F. Castro-Espinoza, A.F. Gelbukh, M. González-Mendoza (Eds.), MICAI 2013, in: LNCS, vol. 8265, Springer, 2013, pp. 68–79.
[15] F. Gadducci, F. Santini, L.F. Pino, F.D. Valencia, A labelled semantics for soft concurrent constraint programming, in: T. Holvoet, M. Viroli (Eds.), COORDINATION 2018, in: LNCS, vol. 9037, Springer, 2015, pp. 133–149.
[16] L. Henkin, J.D. Monk, A. Tarski, Cylindric set algebras and related structures, in: Cylindric Set Algebras, Springer, 1981, pp. 1–129.
[17] S. Bistarelli, U. Montanari, F. Rossi, Soft concurrent constraint programming, ACM Trans. Comput. Log. 7 (3) (2006) 563–589.
[18] C. Baier, M. Sirjani, F. Arbab, J.J.M.M. Rutten, Modeling component connectors in Reo by constraint automata, Sci. Comput. Program. 61 (2) (2006) 75–113.
[19] F. Arbab, J.J.M.M. Rutten, A coinductive calculus of component connectors, in: M. Wirsing, D. Pattinson, R. Hennicker (Eds.), WADT 2002, in: LNCS, vol. 2755, Springer, 2003, pp. 34–55.
[20] D. Clarke, D. Costa, F. Arbab, Connector colouring I: synchronisation and context dependency, Sci. Comput. Program. 66 (3) (2007) 205–225.
[21] M. Izadi, M.M. Bonsangue, D. Clarke, Modeling component connectors: synchronisation and context-dependency, in: A. Cerone, S. Gruner (Eds.), SEFM 2008, IEEE Computer Society, 2008, pp. 303–312.
[22] D. Costa, M. Niqui, J.J.M.M. Rutten, Intentional automata: a context-dependent model for component connectors - (extended abstract), in: F. Arbab, M. Sirjani (Eds.), FSEN 2011, in: LNCS, vol. 7141, Springer, 2012, pp. 335–342.
[23] M.M. Bonsangue, D. Clarke, A. Silva, A model of context-dependent component connectors, Sci. Comput. Program. 77 (6) (2012) 685–706.
[24] S.T.Q. Jongmans, C. Krause, F. Arbab, Encoding context-sensitivity in Reo into non-context-sensitive semantic models, in: W.D. Meuter, G. Roman (Eds.), COORDINATION 2011, in: LNCS, vol. 6721, Springer, 2011, pp. 31–48.
[25] F. Arbab, T. Chothia, S. Meng, Y. Moon, Component connectors with QoS guarantees, in: A.L. Murphy, J. Vitek (Eds.), COORDINATION 2007, in: LNCS, vol. 4467, Springer, 2007, pp. 286–304.
[26] S. Meng, F. Arbab, QoS-driven service selection and composition using quantitative constraint automata, Fundam. Inform. 95 (1) (2009) 103–128.
[27] F. Arbab, C. Baier, F.S. de Boer, J.J.M.M. Rutten, Models and temporal logical specifications for timed component connectors, Softw. Syst. Model. 6 (1) (2007) 59–82.
[28] C. Baier, Probabilistic models for Reo connector circuits, Univers. Comput. Sci. 11 (10) (2005) 1718–1748.
[29] M. Sargolzaei, F. Santini, F. Arbab, H. Afsarmanesh, A tool for behaviour-based discovery of approximately matching web services, in: R.M. Hierons, M.G. Merayo, M. Bravetti (Eds.), SEFM 2013, in: LNCS, vol. 8137, Springer, 2013, pp. 152–166.
[30] C.L. Talcott, V. Nigam, F. Arbab, T. Kappé, Formal specification and analysis of robust adaptive distributed cyber-physical systems, in: M. Bernardo, R.D. Nicola, J. Hillston (Eds.), SFM 2016, in: LNCS, vol. 9700, Springer, 2016, pp. 1–35.
[31] C.L. Talcott, From soft agents to soft component automata and back, in: F.S. de Boer, M.M. Bonsangue, J. Rutten (Eds.), It's All About Coordination, in: LNCS, vol. 10865, Springer, 2018, pp. 189–207.
[32] S. Bistarelli, F. Martinelli, I. Matteucci, F. Santini, A formal and run-time framework for the adaptation of local behaviours to match a global property, in: O. Kouchnarenko, R. Khosravi (Eds.), FACS 2016, in: LNCS, vol. 10231, Springer, 2016, pp. 134–152.
[33] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J.F. Quesada, Maude: specification and programming in rewriting logic, Theor. Comput. Sci. 285 (2) (2002) 187–243.
[34] J.E. Hopcroft, J.D. Ullman, Nonerasing stack automata, Comput. Syst. Sci. 1 (2) (1967) 166–186.
[35] R. Alonso-Sanz, A. Adamatzky, Actin automata with memory, Bifurc. Chaos 26 (1) (2016).
[36] Y. Moon, A. Silva, C. Krause, F. Arbab, A compositional model to reason about end-to-end QoS in stochastic Reo connectors, Sci. Comput. Program. 80 (2014) 3–24.
[37] S.-S.T.Q. Jongmans, Automata-theoretic protocol programming, Ph.D. thesis, Leiden University, 2016.
[38] K. Dokter, F. Arbab, Rule-based form for stream constraints, in: G.D.M. Serugendo, M. Loreti (Eds.), COORDINATION 2018, in: LNCS, vol. 10852, Springer, 2018, pp. 142–161.
[39] F. Arbab, Puff, the magic protocol, in: G. Agha, O. Danvy, J. Meseguer (Eds.), Formal Modeling: Actors, Open Systems, Biological Systems, in: LNCS, vol. 7000, Springer, 2011, pp. 169–206.