A Compositional Framework for Preference-Aware Agents *

Tobias Kappé
LIACS, Leiden University
Leiden, The Netherlands
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
tkappe@liacs.nl

Farhad Arbab
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
LIACS, Leiden University
Leiden, The Netherlands
farhad@cwi.nl

Carolyn Talcott
SRI International
Menlo Park, CA 94025, USA
carolyn.talcott@sri.com

A formal description of a Cyber-Physical system should include a rigorous specification of the computational and physical components involved, as well as their interaction. Such a description, thus, lends itself to a compositional model where every module in the model specifies the behavior of a (computational or physical) component or the interaction between different components. We propose a framework based on Soft Constraint Automata that facilitates the component-wise description of such systems and includes the tools necessary to compose subsystems in a meaningful way, to yield a description of the entire system. Most importantly, Soft Constraint Automata allow the description and composition of components' preferences as well as environmental constraints in a uniform fashion. We illustrate the utility of our framework using a detailed description of a patrolling robot, while highlighting methods of composition as well as possible techniques to employ them.

1 Introduction

As the complexity of tasks for Cyber-Physical Systems grows, so does the need for the ability to decompose the specification of a system into multiple components. Such a decomposition generally eases the design of the specification and furthermore improves reusability. Formal verification tools, like model checkers such as Vereofy [2], for instance, can then separately verify the properties of individual components, simplifying verification of properties claimed to emerge from their composition.

To make such descriptions robust against obstructions, it makes sense to enrich them with a notion of preference [15]. For instance, if a patrolling robot finds its path obstructed, it may settle for a slightly adjusted patrol path. More generally, the presence of lower-preference actions enables a component (and the system as a whole) to be more flexible when its most-preferred action is incompatible with the actions allowed by other components or, especially, its (less predictable) environment. Endowment of preferences and "alternative" behavior to components raises the question of how to go about composing them in a meaningful way. In particular, not all concerns may be equally important in a composition.

The contribution of this paper is a framework that enables component-wise, preference-aware descriptions of Cyber-Physical Systems, based on Soft Constraint Automata (SCAs) [1], a generalisation of Constraint Automata [3]. Because SCAs express preferences using well-studied structures called c-semirings [6], we can rely on existing results to develop methods to compose SCAs.

Our notion of preference differs from earlier investigations of priority in the context of concurrency theory [8]. First, priority is usually assigned to favor specifically prioritized actions over non-prioritized ones in otherwise nondeterministic choices. Preferences represent more abstract constraints that express considerations and compromises among a broad spectrum of concerns that often cannot even be directly related with each other. Moreover, priority assignments usually induce a statically determined partial order

*The work was partially supported by ONR grant N00014–15–1–2202.

Kargahi and Trivedi: Verification and Validation of
Cyber-Physical Systems 2016 (V2CPS-16)
EPTCS 232, 2016, pp. 21–35, doi:10.4204/EPTCS.232.6

on concrete actions. As more abstract constraints, preferences of an agent need not specify concrete actions, and compose with other concerns and constraints that arise from a dynamically changing environment to select a suitable action for the agent. We believe both of these factors contribute towards a more compositional framework.

The remainder of this paper is organised as follows. In Section 2 we review the necessary notation. In Section 3, we describe the components of our framework, after which we expand on the methods for their composition in Section 4. We give a detailed example of modelling a patrolling robot using the proposed framework in Section 5. We list our conclusions in Section 6 and discuss further work in Section 7.

2 Preliminaries

To introduce the components of our framework, we first discuss some necessary notions. Most importantly, we review Constraint Semirings and Soft Constraint Satisfaction Problems later in this section.

Let S be a set. When S_1 and S_2 are sets such that $S \subseteq S_1 \times S_2$, we write $\mathsf{Pr}_i(S)$, $i \in \{1,2\}$, for the *projection* to the i-th component of S, i.e., $\mathsf{Pr}_i(S) = \{s_i : \langle s_1, s_2\rangle \in S\}$. Also, we use S^ω to denote the set of *streams* [13] of elements in S, i.e., the set of functions from $\mathbb{N}$ to S. For $\sigma \in S^\omega$, we write $\mathsf{tail}(\sigma)$ for the unique element of S^ω such that $\mathsf{tail}(\sigma)(n) = \sigma(n+1)$. We may abbreviate the repeated application of tail by tail^n; more precisely, tail^0 is the identity function, while $\mathsf{tail}^{n+1} = \mathsf{tail} \circ \mathsf{tail}^n$ for $n \geq 0$.

We often need to work with sets of symbols that may have associated values from some domain $\mathbb{D}$. Throughout this paper, we consider $\mathbb{D}$ to be fixed. Let V be a finite set (of symbols). An *assignment* of V is a function $\alpha : V \to \mathbb{D}$. We denote the set of all assignments of V by $\mathsf{Assign}(V)$, while $\mathsf{Assign}_{\subseteq}(V)$ denotes the set of all assignments of subsets of V, i.e., $\mathsf{Assign}_{\subseteq}(V) = \bigcup\{\mathsf{Assign}(V') : V' \subseteq V\}$.

If α is an assignment of V and $V' \subseteq V$, then the restriction of α to V', written $\alpha{\restriction_{V'}}$, is the unique assignment of V' that coincides with α on V'. If α and β are assignments of V and U respectively and both agree on $U \cap V$, then their composition, written $\alpha + \beta$, is the unique assignment of $U \cup V$ such that $(\alpha+\beta){\restriction_V} = \alpha$ and $(\alpha+\beta){\restriction_U} = \beta$.

2.1 Preferences

In order to work with preferences, we need a domain for expressing them. Additionally, we require an operator to select the best preference value from a set of preference values (if such a best value exists) and an operator to compose preference values. It turns out that a Constraint Semiring provides the right kind of structure for such a domain. The exact definition below is due to Bistarelli [5]. Here, the operator $\bigoplus$ models the selection of the best preference, while $\otimes$ is used to obtain the preference of a composed action, given the preferences of the component actions.

Definition 1 (c-semiring)**.** *A* Constraint Semiring *(or* c-semiring*, for short) is a tuple* $\langle E, \bigoplus, \otimes, \mathbf{0}, \mathbf{1}\rangle$ *where E is a set (called the* carrier *of the semiring) with* $\mathbf{0}, \mathbf{1} \in E$*, while* $\bigoplus : \mathscr{P}(E) \to E$ *and* $\otimes : E \times E \to E$ *are operators such that for every $e \in E$ and every family of subsets of E indexed by some set I, $\{E_i\}_{i\in I}$:*

- $\bigoplus$ *obeys the following restrictions:*
  - $\bigoplus\{e\} = e$
  - $\bigoplus\emptyset = \mathbf{0}$ *and* $\bigoplus E = \mathbf{1}$
  - $\bigoplus(\bigcup_{i\in I} E_i) = \bigoplus\{\bigoplus_{i\in I} E_i : i \in I\}$ *(the* flattening *property).*
- $\otimes$ *is commutative and associative, with* $\mathbf{1}$ *its unit element and* $\mathbf{0}$ *its absorbing element.*
- $\bigoplus$ *distributes over* $\otimes$*, i.e.,* $e \otimes \bigoplus E_i = \bigoplus\{e \otimes e' : e' \in E_i\}$

Examples of c-semirings include the *boolean semiring* $\mathbb{B} = \langle \{\bot, \top\}, \bigvee, \wedge, \bot, \top \rangle$ the *weighted (or tropical) semiring* $\mathbb{W} = \langle \mathbb{R}_{\geq 0} \cup \{\infty\}, \inf, \hat{+}, \infty, 0 \rangle$ and the *probabilistic semiring* $\mathbb{P} = \langle [0,1], \sup, \cdot, 0, 1 \rangle$. Also interesting is the *set semiring* $\mathbb{S}_\Sigma = \langle \mathscr{P}(\Sigma), \bigcup, \cap, \emptyset, \Sigma \rangle$ where Σ is some finite set of symbols.

We refer to the set E as a c-semiring if the other elements are not relevant for the discussion. When we want to explicitly mention a single operator or unit of a semiring E, we write $\bigoplus_E$, $\mathbf{0}_E$, and so on. Applying $\bigoplus$ on two-element sets, we denote it as a binary infix operator $\oplus$. A c-semiring $\langle E, \bigoplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ induces a relation $\leq_E$ on E, defined for $e, e' \in E$ as $e \leq_E e'$ if and only if $e \oplus e' = e'$. The relation $<_E$ on E is $\leq_E$ with its diagonal excluded. In general, $\leq_E$ is a partial order [5]; one can construct c-semirings (such as $\mathbb{S}_A$ when $|A| > 1$) where the induced ordering is not total. Constraint Semirings have a number of convenient properties. We refer to [5, Section 2.1] for an excellent treatment including proofs.

Particularly noteworthy in a c-semiring are cancellative elements:

Definition 2 (cancellative elements)**.** *Let $\langle E, \bigoplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ be a c-semiring. An element $e \in E$ is* cancellative *if for all e', e'' we have $e' \otimes e = e'' \otimes e$ if and only if $e' = e''$. The set of cancellative elements of E is written $\mathscr{C}(E)$, and its complement within E is denoted by $\overline{\mathscr{C}}(E)$. When $\mathscr{C}(E) \cup \{\mathbf{0}\} = E$, we call E a* cancellative c-semiring.

Of course, one can construct new c-semirings from existing c-semirings E_1 and E_2. For instance, we can impose a c-semiring structure on the Cartesian product of two c-semirings, gaining the product semiring $E_1 \times E_2$ [7]. One can also trim down this product, by filtering out pairs containing $\mathbf{0}_i$:

Definition 3 (join semiring)**.** *Let $\langle E_1, \bigoplus_1, \otimes_1, \mathbf{0}_1, \mathbf{1}_1 \rangle$ and $\langle E_2, \bigoplus_2, \otimes_2, \mathbf{0}_2, \mathbf{1}_2 \rangle$ be two cancellative c-semirings. Their* join semiring*, written $E_1 \odot E_2$, is the c-semiring $\langle E_1 \odot E_2, \bigoplus, \otimes, \langle \mathbf{0}_1, \mathbf{0}_2 \rangle, \langle \mathbf{1}_1, \mathbf{1}_2 \rangle \rangle$ in which $E_1 \odot E_2$ is the set $(\mathscr{C}(E_1) \times \mathscr{C}(E_2)) \cup \{\langle \mathbf{0}_1, \mathbf{0}_2 \rangle\}$ and $\bigoplus$ and $\otimes$ are defined for $E' \subseteq E_1 \odot E_2$ and $e_i, e_i' \in E_i$ as follows:*

$$\bigoplus E' = \left\langle \bigoplus\nolimits_1 \mathsf{Pr}_1(E'), \bigoplus\nolimits_2 \mathsf{Pr}_2(E') \right\rangle \quad \textit{and} \quad \langle e_1, e_2 \rangle \otimes \langle e_1', e_2' \rangle = \langle e_1 \otimes_1 e_1', e_2 \otimes_2 e_2' \rangle$$

One can easily surmise that $E_1 \odot E_2$ is cancellative, and that the order induced on the join semiring $E_1 \odot E_2$ is the product order obtained from the orders induced on its operands, i.e., if $e_i, e_i' \in E_i$ then

$$\langle e_1, e_2 \rangle \leq_{E_1 \odot E_2} \langle e_1', e_2' \rangle \iff e_1 \leq_{E_1} e_2 \text{ and } e_1' \leq_{E_2} e_2'$$

The join semiring can be used to compose preferences from two c-semirings that are deemed equally important, or at least, a selection between the two indications of preference is yet to be made. In Section 5, we use it for our patrolling robot, where we want to defer selection of the most important preference (energy level versus the patrolling mission) to run-time.

Because we can construct a c-semiring that induces the product ordering on the Cartesian product, one may wonder if we can also construct a c-semiring that induces a lexicographic ordering. Indeed, this is possible, but to maintain distributivity of $\otimes$ over $\bigoplus$ we must exclude some elements from the carrier [9]. As a consequence, not all types of c-semirings will be equally suitable as candidates to serve as the most significant component of a lexicographic composition.

Definition 4 (lexicographic product semiring)**.** *Let $\langle E_1, \bigoplus_1, \otimes_1, \mathbf{0}_1, \mathbf{1}_1 \rangle$ and $\langle E_2, \bigoplus_2, \otimes_2, \mathbf{0}_2, \mathbf{1}_2 \rangle$ be two c-semirings*[1]*. Their* lexicographic product semiring*, written $E_1 \triangleright E_2$, is the c-semiring $\langle E, \bigoplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ in*

[1] In [9], E_1 is restricted to have a total ordering. We claim that such a restriction is not necessary for this construction to yield a c-semiring. A proof appears in the appendix of the technical report [11].

which E is the set $(\mathscr{C}(E_1) \times E_2) \cup (\overline{\mathscr{C}}(E_2) \times \{\mathbf{0}_2\})$, while $\mathbf{0} = \langle \mathbf{0}_1, \mathbf{0}_2 \rangle$ (and likewise for $\mathbf{1}$). Lastly, $\bigoplus$ and $\otimes$ are defined for $E' \subseteq E$ and $e_i, e_i' \in E_i$ by

$$\bigoplus E' = \left\langle \bigoplus\nolimits_1 \mathsf{Pr}_1(E'), \bigoplus\nolimits_2 m(E') \right\rangle \quad \text{and} \quad \langle e_1, e_2 \rangle \otimes \langle e_1', e_2' \rangle = \langle e_1 \otimes_1 e_1', e_2 \otimes_2 e_2' \rangle$$

in which $m(E')$ contains precisely the elements e_2 of $\mathsf{Pr}_2(E')$ such that $\langle \bigoplus_1 \mathsf{Pr}_1(E'), e_2 \rangle \in E'$.

The lexicographic product semiring is useful when one concern is objectively more important than another. In Section 5 we use it to mark the preference of patrolling as most important, while the preference to stay on the path is considered of secondary importance.

The intuition behind the second component of the additive operator above is that the best preference is chosen from the elements in the least significant position that co-occur with the best preference value in the most significant position — since any value for which this is not the case occurs in a tuple that cannot be the maximal element of E with respect to the lexicographic ordering.

To move between different domains of preferences in a smooth manner, the notion of a c-semiring homomorphism is useful. Its definition simply follows the familiar pattern from algebra. We will use homomorphisms later on to embed preference values from individual components into composed c-semirings.

Definition 5 (c-semiring homomorphism)**.** *Let E_1 and E_2 be c-semirings. A* c-semiring homomorphism *(or in this paper, simply a* homomorphism*) is a function $h : E_1 \to E_2$ such that*

- *$h(\mathbf{0}_{E_1}) = \mathbf{0}_{E_2}$ and $h(\mathbf{1}_{E_1}) = \mathbf{1}_{E_2}$*
- *for all $E_1' \subseteq E_1$ it holds that $h\left(\bigoplus_{E_1} E_1'\right) = \bigoplus_{E_2} \{h(e) : e \in E_1'\}$*
- *for all $e, e' \in E_1$ it holds that $h(e \otimes_{E_1} e') = h(e) \otimes_{E_2} h(e')$*

We call h order-reflecting *[12] if for all $e, e' \in E$, it holds that $h(e) \leq_{E_2} h(e')$ implies $e \leq_{E_1} e'$.*

It can easily be shown that if $h : E_1 \to E_2$ is an order-reflecting homomorphism, then for all $e, e' \in E_1$ it holds that $e \leq_{E_1} e'$ if and only if $h(e) \leq_{E_2} h(e')$.

Let E_1 and E_2 be c-semirings. The following mappings are particularly useful:

$$h_L(e) = \begin{cases} \langle \mathbf{0}_{E_1}, \mathbf{0}_{E_2} \rangle & e = \mathbf{0}_{E_1} \\ \langle e, \mathbf{1}_{E_2} \rangle & \text{otherwise} \end{cases} \qquad h_R(e) = \begin{cases} \langle \mathbf{0}_{E_1}, \mathbf{0}_{E_2} \rangle & e = \mathbf{0}_{E_2} \\ \langle \mathbf{1}_{E_1}, e \rangle & \text{otherwise} \end{cases}$$

If E_1 (respectively E_2) is cancellative, then one can observe that h_L (respectively h_R) is an order-reflecting homomorphism from E_1 (respectively E_2) to $E_1 \odot E_2$ as well as $E_1 \triangleright E_2$. In the sequel, we refer to these mappings as *canonical injections*. Their domain and codomain c-semirings will always be made explicit. Lastly, the injection from $E_1 \odot E_2$ into $E_1 \times E_2$ is also an order-reflecting homomorphism.

2.2 Soft Constraint Satisfaction Problems

The notion of a Soft Constraint Satisfaction Problem elegantly captures [6] a number of generalizations of Constraint Satisfaction Problems aimed at attaching preference values to candidate solutions. Our definitions below are compatible with those of [6, 5] but differ slightly for the sake of subsequent representation.

Definition 6 (Soft Data Constraint)**.** *A* Soft Data Constraint *(SDC) is a tuple $\langle U, E, [\cdot] \rangle$ such that U is a finite set of symbols, E is a c-semiring (called the* underlying semiring*) and $[\cdot]$ is a function from $\mathsf{Assign}(V)$ to E. We write $\mathsf{SDC}(V, E)$ for the set of all SDCs that involve a subset of V and have E as their underlying semiring.*

When $\langle U, E, [\cdot]\rangle$ is an SDC such that $[\cdot]$ has a range of $\{\mathbf{0}, e\} \subseteq E$ with $e \neq \mathbf{0}_E$, we call $\langle U, E, [\cdot]\rangle$ a *binary constraint*. It is often more convenient to denote binary constraints in an abbreviated fashion. We write such constraints as $\langle U, \phi, e\rangle$, where ϕ is some first-order logic expression with U as variables that is satisfied by $\alpha \in \mathsf{Assign}(U)$ if and only if $[\alpha] = e$. The c-semiring E will always be clear from the context when we use this abbreviation. When $E = \mathbb{B}$, we omit e, for necessarily $e = \mathbf{1}_{\mathbb{B}} = \top$.

In a Soft Constraint Satisfaction Problem, one can use Soft Data Constraints to express preferences for assignments of subsets of variables. The total preference is then given by composing the preference values obtained from the SDCs into one.

Definition 7 (Soft Constraint Satisfaction Problem)**.** *A* Soft Constraint Satisfaction Problem *(SCSP) is a tuple $\langle V, E, C\rangle$ such that V is a finite set of symbols, E is a c-semiring (called the* underlying semiring*) and C is a finite subset of $\mathsf{SDC}(V, E)$. We write $\mathsf{SCSP}(V, E)$ for the set of all SCSPs that involve a subset of V and have E as their underlying semiring.*

As an example of an SCSP, let $\mathbb{D} = \mathbb{R}_{\geq 0}$ and consider $K = \langle \{x, y\}, \mathbb{P}, C\rangle$ with C containing precisely the (abbreviated) SDCs $\langle \{x\}, x < 1, 0.9\rangle$ and $\langle \{x, y\}, x + y = 2, 0.4\rangle\}$.

The *preference function* induced by an SCSP $P = \langle V, E, C\rangle$, with $C = \{\langle U_1, E, [\cdot]_1\rangle, \ldots, \langle U_n, E, [\cdot]_n\rangle\}$ is the function $[\cdot]_P : \mathsf{Assign}(V) \to E$ defined by $[\alpha]_P = [\alpha{\restriction_{U_1}}]_1 \otimes \cdots \otimes [\alpha{\restriction_{U_n}}]_n$ when $n > 0$ and $[\alpha]_P = \mathbf{1}$ otherwise. It is not hard to see that every SCSP can be reduced to have exactly one SDC, namely $\langle V, E, [\cdot]_P\rangle$. The advantage of SCSPs containing multiple SDCs is that they can often express preferences more concisely by only considering values of symbols relevant to the concern at hand.

Definition 8 (SCSP solutions)**.** *Let $P = \langle V, E, C\rangle$ be an SCSP. A* solution *to P is an assignment α of V such that $[\alpha]_P \neq \mathbf{0}_E$ and there exists no other assignment β of V with $[\alpha]_P <_E [\beta]_P$. We write $\mathsf{Sol}(P)$ for the set of solutions to P.*

For the example SCSP K mentioned above, $\alpha \in \mathsf{Assign}(\{x, y\})$ such that $\alpha(x) = 0.5$ and $\alpha(y) = 1.5$, with preference value $0.9 \cdot 0.4 = 0.36$ qualifies for a solution.

Observe that the wording of Definition 8 accommodates for the possibility that $\leq_E$ may not be total. We can also compose SCSPs that share an underlying semiring, simply by taking the union of their variables and SDCs.[2]

Definition 9 (SCSP composition)**.** *Let $P_1 = \langle V_1, E, C_1\rangle$ and $P_2 = \langle V_2, E, C_2\rangle$ be SCSPs. Their* composition*, written $P_1 \otimes P_2$, is the SCSP $\langle V_1 \cup V_2, E, C_1 \cup C_2\rangle$.*

By this definition, we immediately see that the operator $\otimes$ on SCSPs is commutative and associative. At this point, it is worth noting that classical CSPs are captured by SCSPs; by the definitions above, a CSP is simply an SCSP over the boolean semiring. Conversely, one can translate an SCSP into a CSP by simply having a single constraint encode the requirement for a solution given in Definition 8.

Lastly, we can move SCSPs between c-semirings using homomorphisms:

Definition 10 (homomorphisms for SCSPs)**.** *Let $P = \langle V, E_1, C\rangle$ be an SCSP and $h : E_1 \to E_2$ be a homomorphism. Then $h(P)$ is the SCSP $\langle V, E_2, C'\rangle$ where $C' = \{\langle U, E_2, h \circ [\cdot]\rangle : \langle U, E_1, [\cdot]\rangle \in C\}$.*

One can prove [12] that if h is order-reflecting, then h will preserve the solutions to an SCSP, or more precisely, $\mathsf{Sol}(P) = \mathsf{Sol}(h(P))$. Moreover, it follows immediately from the above definition that homomorphisms are compatible with SCSP composition, in the sense that if P_1 and P_2 are SCSPs that have E as underlying semiring and $h : E \to E'$ is a homomorphism, then $h(P_1 \otimes P_2) = h(P_1) \otimes h(P_2)$.

[2] Similar to [5], we assume that no two SCSPs being composed share the exact same SDC.

2.3 Separating SCSPs from CSPs

A watershed between SCSPs and CSPs becomes clear when we consider the composition operator. Let $P_1 = \langle V_1, \mathbb{B}, C_1 \rangle$ and $P_2 = \langle V_2, \mathbb{B}, C_2 \rangle$ be CSPs. If α is a solution to $P_1 \otimes P_2$, then $\alpha{\restriction_{V_i}}$ is a solution to P_i for $i \in \{1,2\}$. Because of the way preferences compose, this property need not hold when the semiring is something other than the boolean semiring. Loss of this property can be useful: it exhibits the possibility of SCSPs *compromising* when higher-preference assignments turn out to be incompatible.[3]

3 Components

We now introduce Soft Constraint Automata, as the most fundamental building blocks of our framework. Soft Constraint Automata were first proposed as a generalization of Constraint Automata [3] in [1], to enable service discovery based on non-crisp preferences. In contrast, Soft Constraint Automata as used in this paper employ their preferences purely as a means to select their next transition.

Definition 11 (Soft Constraint Automaton)**.** *A* Soft Constraint Automaton *(SCA) is a tuple* $\langle Q, V, E, \rightarrow, q^0 \rangle$ *where*

- Q *is a finite set of states, with* $q^0 \in Q$
- V *is a finite set of port symbols*
- E *is a c-semiring, referred to as the* underlying semiring
- $\rightarrow \subseteq Q \times \mathsf{SCSP}(V,E) \times Q$*, the* transition relation

When $\langle q, P, q' \rangle \in \rightarrow$, we write $q \xrightarrow{P} q'$; if P contains only the binary constraint $\langle U, \phi, e \rangle$, we write $q \xrightarrow{U,\ \phi,\ e} q'$. Note that, unlike [1], we truly label transitions with SCSPs rather than a set of ports and a single data constraint (the set of ports is implied by the SCSP labels in our notation). Analogous to SCSPs, one can observe that because Constraint Automata (CAs) [3] have their transitions labeled with CSPs (which are simply SCSPs over the boolean c-semiring), they can be obtained as SCAs with the boolean semiring as underlying semiring [1].

We can also lift homomorphisms to operate on SCAs, as they did on SCSPs.

Definition 12 (homomorphisms for SCAs)**.** *Let* $A = \langle Q, V, E, \rightarrow, q^0 \rangle$ *be an SCA and let* $h : E \rightarrow E'$ *be a homomorphism. Then* $h(A)$ *is the SCA* $\langle Q, V, E', \rightarrow_h, q^0 \rangle$*, where* $\rightarrow_h$ *is the smallest relation such that* $q \xrightarrow{h(P)}_h q'$ *whenever* $q \xrightarrow{P} q'$.

The semantics of an SCA is obtained by means of its execution model, as the stream of solutions to the SCSPs that label the transitions. As a consequence, each element of this stream is a (partial) assignment of the set of port symbols of the automaton, V.

Definition 13 (SCA semantics)**.** *Let* $A = \langle Q, V, E, \rightarrow, q^0 \rangle$ *be an SCA. Its* execution relation *is the smallest binary relation* $\Rightarrow_A$ *on* $Q^\omega \times \mathsf{Assign}_{\subseteq}(V)^\omega$ *satisyfing the rule*

$$\frac{\lambda(0) \xrightarrow{P} \lambda(1) \qquad \pi(0) \in \mathsf{Sol}(P)}{\langle \lambda, \pi \rangle \Rightarrow_A \langle \mathsf{tail}(\lambda), \mathsf{tail}(\pi) \rangle}$$

The language *accepted by A, written* $L(A)$*, consists of the elements* $\pi \in \mathsf{Assign}_{\subseteq}(V)^\omega$ *such that there exists a stream* $\lambda \in Q^\omega$ *with* $\lambda(0) = q^0$ *and* $\langle \lambda, \pi \rangle \Rightarrow_A \langle \mathsf{tail}(\lambda), \mathsf{tail}(\pi) \rangle \Rightarrow_A \langle \mathsf{tail}^2(\lambda), \mathsf{tail}^2(\pi) \rangle \Rightarrow_A \ldots$

[3] Conversely, if α_1 and α_2 are solutions to CSPs P_1 and P_2 respectively, and both agree on common symbols, then $\alpha_1 + \alpha_2$ is necessarily a solution to $P_1 \otimes P_2$. This property also does not hold for SCSPs; we refer to the technical report [11] for details.

It is easy to observe that for any SCA, A, there exists an SCA A' with the boolean semiring as its underlying semiring, such that $\Rightarrow_A$ is the same relation as $\Rightarrow_{A'}$, and, by extension, $L(A) = L(A')$ (refer to Section 5.1 for an example). Moreover, if h is an order-reflecting homomorphism, one can quickly see that $L(A) = L(h(A))$. By the latter observation, we can consider a CA to be an SCA over any c-semiring E, as finding an order-reflecting homomorphism $h : \mathbb{B} \to E$ is trivial: simply map the units of $\mathbb{B}$ to those of E. In the sequel, we therefore regard a CA as an SCA over some semiring E whenever convenient.

Lastly, a graphical representation of SCAs is appropriate for the remainder of this paper. We depict SCAs as transition systems; the labels on the edges of our transition systems will in all cases be the abbreviated representation of binary SDCs mentioned in the previous section. Because we restrict graphical depictions to use this abbreviation, we will not be able to draw all SCAs, but the ones we can draw will have less convoluted representations and will suffice for our purposes in Section 5. For an example of an SCA with the probabilistic c-semiring $\mathbb{P}$ as the underlying semiring, refer to Figure 1. We note that, in this representation, the set of port symbols of the automaton, V, is left implicit as the union of the occuring variable sets; i.e., in Figure 1, $V = \{V_1, V_2\}$.

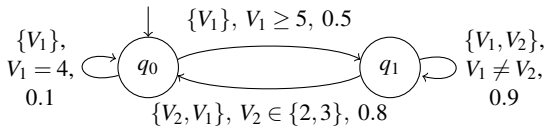


Figure 1: An example of our graphical representation of SCAs

4 Composition

We now turn our attention to composition of SCAs. Firstly and most obviously, we can lift the composition operator of CAs [3] to SCAs, as is done in [1]. Intuitively, this composition operator composes transitions that agree on common ports. A side-condition here is that the operands have the same underlying semiring.

Definition 14 (product composition of SCAs)**.** *Let* $A_1 = \langle Q_1, V_1, E, \to_1, q_1^0 \rangle$ *and* $A_2 = \langle Q_2, V_2, E, \to_2, q_2^0 \rangle$ *be two SCAs. Their* product composition*, written* $A_1 \otimes A_2$*, is the SCA* $\langle Q_1 \times Q_2, V_1 \cup V_2, E, \to, \langle q_1^0, q_2^0 \rangle \rangle$*, in which* $\to$ *is the smallest relation satisyfing the rule*

$$\frac{q_1 \xrightarrow{\langle V_1',\, E,\, C_1 \rangle}_1 q_1' \qquad q_2 \xrightarrow{\langle V_2',\, E,\, C_2 \rangle}_2 q_2' \qquad V_1' \cap V_2 = V_2' \cap V_1}{\langle q_1, q_2 \rangle \xrightarrow{\langle V_1',\, E,\, C_1 \rangle \otimes \langle V_2',\, E,\, C_2 \rangle} \langle q_1', q_2' \rangle}$$

By the above definition, it is obvious that the $\otimes$-operator for SCAs is commutative and associative, modulo a simple relabeling of the states. In the sequel, we abstract from this relabeling, as it has no bearing on the semantics of the automaton. Furthermore, application of a homomorphism to SCAs (as in Definition 12) is compatible with product composition in the sense that for SCAs A_1 and A_2 that share an underlying semiring E and a homomorphism $h : E \to E'$, we have $h(A_1 \otimes A_2) = h(A_1) \otimes h(A_2)$.

One can now see that the same observations that distinguish composition of SCSPs from that of CSPs in Section 2.3, also distinguish composition of SCAs from that of CAs. At this level, the advantage of these properties for describing Cyber-Physical Systems is more clear; when the transitions preferred by the operands A_1 and A_2 are incompatible, $A_1 \otimes A_2$ may still have transitions composed of lower-preference transitions found in A_1 and A_2. Informally, one may say that by providing alternative (lower-preference) transitions, SCAs can be made resillient in their composition with unforeseen automata.

One notable difference of Definition 14 with the definition of composition for CAs in [1] is that we exclude rules for independent transitions, i.e., those of the form

$$\frac{q_i \xrightarrow{\langle V_i',\, E,\, C_i\rangle}_i q_i' \qquad q_j \in Q_j \qquad V_i' \cap V_j = \emptyset}{\langle q_i, q_j\rangle \xrightarrow{\langle V_i',\, E,\, C_j\rangle} \langle q_i', q_j\rangle} \tag{1}$$

This is due to the fact that, if these rules are included, the product composition will be biased towards independent transitions in terms of preference (due to the fact that for $e_1, e_2 \in E$ we have $e_1 \otimes e_2 \leq_E e_1$; c.f. [5, Theorem 2.1.3]), unless the preference value of the original transition is composed with some other preference value e. If we choose $e = \mathbf{1}_E$ for this preference value, the composed preference is unchanged (and we are essentially applying (1) as-is). If on the other hand we choose $e = \mathbf{0}_E$, then we are in fact prohibiting independent transitions, as the composed preference value will be $\mathbf{0}_E$. Having established that neither $\mathbf{0}$ nor $\mathbf{1}$ is a suitable candidate, we leave it to the designer of the SCA to include self-transitions, each with an appropriate preference value $e \in E$, to states of an automaton where other automata are permitted to make independent transitions, and vice versa. For a concrete use of this technique, we refer to Section 5. Note that e need not be the same for all states; a component may have a different preference regarding independent moves by other components depending on context. For example, it is conceivable that in some situations, a component may want to (almost) completely inhibit behavior of other components until some task is completed.

The product composition operator of Definition 14 to CAs now coincides with the product composition operator for CAs of [3], provided every state q has a self-transition $q \xrightarrow{\emptyset,\, \top,\, \mathbf{1}} q'$ [10].

Of course, it may also occur that we want to compose SCAs that model different concerns. In this case, it may not make sense to use the product composition operator proposed above. Moreover, if the preferences of the operands are expressed in different semirings, the product composition operator in does not apply. To deal with such cases, we introduce another variant of composition, based on the join product. However, for such an operator to make sense, we need to be able to encode the preferences expressed by the operands into the composed preference domain. Therefore, we must add a side condition guaranteeing that such an embedding is possible.

Definition 15 (join composition of SCAs)**.** *Let A_1 and A_2 be SCAs with cancellative underlying semirings E_1 and E_2 respectively. Their* join composition*, written $A_1 \odot A_2$, is the SCA $h_L(A_1) \otimes h_R(A_2)$, in which h_L and h_R are the canonical injections from E_1 to $E_1 \odot E_2$ and E_2 to $E_1 \odot E_2$ respectively.*

The operator $\odot$ on SCAs is also commutative and associative, up to a trivial order-reflecting homomorphism, i.e., if A_1, A_2 and A_3 are SCAs, then $A_1 \odot A_2 = h(A_2 \odot A_1)$ and $A_1 \odot (A_2 \odot A_3) = h'((A_1 \odot A_2) \odot A_3)$ for some order-reflecting homomorphisms h and h'. In the sequel, we abstract from this homomorphism and simply regard $\odot$ as commutative and associative. Going by the fact that $A_1 \odot A_2$ has $E_1 \odot E_2$ as underlying semiring, we can surmise that the transitions allowed in $A_1 \odot A_2$ are those that are Pareto-optimal with respect to the preferences of both automata. In other words, the join composition of two SCAs will exclude behavior that "unnecessarily inconveniences" one of the operands.

We can also use the lexicographic composition described in Definition 4 to compose automata where the preferences expressed by one are subsidiary to the other. Again, we must include a side-condition guaranteeing that we can embed the preferences from the operands.

Definition 16 (lexicographic composition of SCAs)**.** *Let A_1 and A_2 be SCAs with underlying cancellative c-semirings E_1 and E_2 respectively. Their* lexicographic composition*, written $A_1 \triangleright A_2$, is the SCA $h_L(A_1) \otimes h_R(A_2)$, in which h_L and h_R are the canonical injections from E_1 and E_2 to $E_1 \triangleright E_2$ respectively.*

We observe that, unlike $\otimes$ and $\odot$, the operator $\triangleright$ is (by design) not commutative. However, it is associative, in part due to the fact that if c-semirings E_1 and E_2 are cancellative, then so is $E_1 \triangleright E_2$.

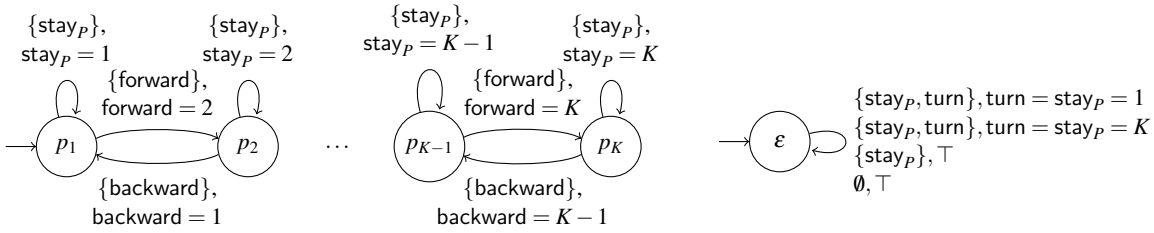
5 Example: Patrolling Agent

In this example, we consider an agent that is tasked with patrolling along a predefined path. We start with the simplest description possible and subsequently extend our model to include deviation from the path as well as energy usage. At every step, we describe the environment in terms of Constraint Automata, after which we express the relevant preferences of the agent using Soft Constraint Automata.

Composite states are denoted by (flattened) tuples; i.e., if p, q and r are states of the automata A_1, A_2 and A_3 respectively, then $\langle p,q,r\rangle$ is a state of the automaton $A_1 \odot A_2 \odot A_3$. As a special case, we omit the state symbol of single-state automata from the state tuple; their sole state is also written as ε to reflect this. Finally, all c-semirings used in this section are tacitly assumed to be cancellative.

5.1 Movement back and forth

The path of the agent is broken up into K discrete positions, each of which is represented by a state; positions 1 and K are the endpoints of the path. At every position, the agent may move forward or backward along the path. The agent can also opt to stay at its current position; these self-transitions allow other components to make transitions independently, i.e., make moves while the agent remains stationary. The constraint automaton A_{path}, which represents these possibilities, is given in Figure 2a. To provide a turning point for the agent, we use the CA $A_{\mathsf{turn},P}$ in Figure 2b; note that in $A_{\mathsf{patrol}} \otimes A_{\mathsf{turn},P}$ the agent is required to stay put when turning: if stay_P does not fire, then neither can turn.



(a) A sketch of the CA A_{path}, modelling the position along the path. (b) The CA $A_{\mathsf{turn},P}$, modelling turning points.

Figure 2: CAs pertaining to the path.

The preference of the agent to keep patrolling is expressed by the two-state SCA in Figure 3. In this description, the transitions are labeled with preferences from some semiring E. To ensure that the agent does indeed patrol, we stipulate that $e_{\mathsf{backtrack}} <_E e_{\mathsf{stay},P} <_E e_{\mathsf{turn}}, e_{\mathsf{progress}}$. In other words, the agent will prefer to start another lap or progress (in its current target direction); failing that, it would rather stay at its current position than backtrack to the previous one.

The final system as presented up to this point is now formed by the following expression (recall that we can interpret the CAs A_{path} and $A_{\mathsf{turn},P}$ as SCAs with the same underlying semiring as A_{patrol}):

$$A_{\mathsf{move}} = A_{\mathsf{path}} \otimes A_{\mathsf{turn},P} \otimes A_{\mathsf{patrol}}$$

By itself, this automaton is not particularly interesting; it will exhibit precisely the desired patrolling behavior. As a matter of fact, it is almost trivial to use a Constraint Automaton and achieve the exact same

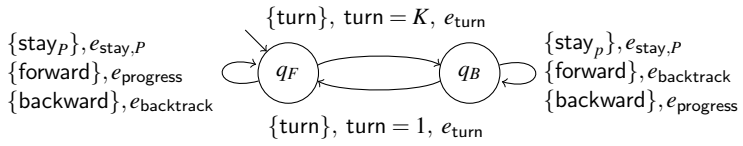


Figure 3: The SCA A_{patrol}, expressing patrolling preferences.

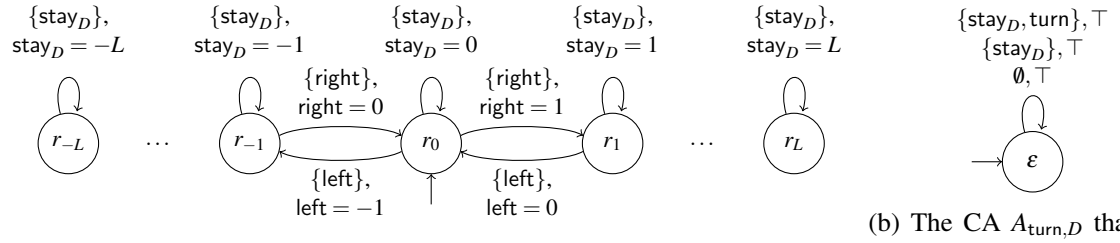
behavior, by pruning the transitions whose preference is dominated by that of some other transition, and setting the preference value of the remaining transitions to $\mathbf{1}$. In this case, this would mean that we delete:

- the transitions involving $e_{\mathsf{stay},P}$ and $e_{\mathsf{backtrack}}$ in A_{patrol} from both states
- the self-transition that fires backward (respectively forward) from q_F (respectively q_B) from A_{patrol}
- the self-transition with stay_P in states $p_2, \ldots, p_{K-1}$ from A_{path}.

We will see in the sequel that allowing these "alternative" transitions to remain enables us to extend our model more easily later on.

5.2 Deviating from the path

We now add in the possibility for our agent to stray from its predetermined path to the left or the right by L steps, while still preferring that the agent stays on the path. To model this in the environment, we construct the CA sketched in Figure 4a. In this CA, the state r_0 represents that the agent is on the path, while the other states r_i represent that the agent is located $|i|$ steps beside the current position on the path towards the next turning point, with the sign of i indicating the direction of divergence (negative for deviation to the left, positive otherwise).



(a) A sketch of the CA A_{stray}, modelling deviation from the path in any position.

(b) The CA $A_{\mathsf{turn},D}$ that fixes the turning point on the path.

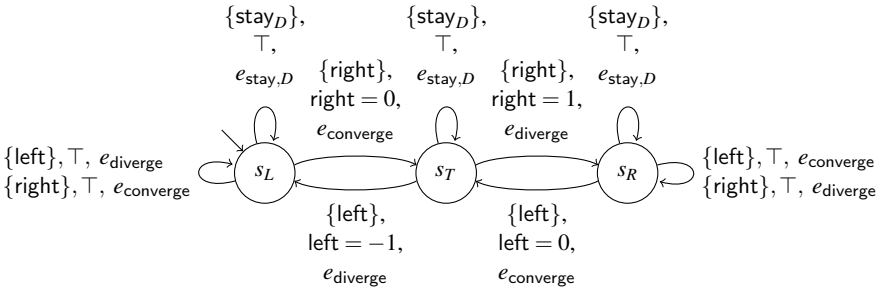
Figure 4: CAs pertaining to deviation from the path.

We now encourage the reader to consider what the automaton $A_{\mathsf{path}} \otimes A_{\mathsf{deviate}}$ (c.f. Figures 2a and 4a) looks like, as it will be a subcomponent of our final system. For instance, when this automaton is in state $\langle p_i, q_j \rangle$, with $i < n$ and $j < L$, it can transition to state $\langle p_{i+1}, q_{j+1} \rangle$ by firing $\{\mathsf{forward}, \mathsf{right}\}$.

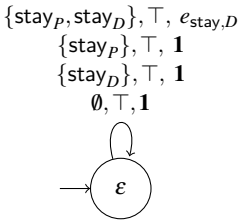
Moving on, we need the CA in Figure 4b to make sure that the agent cannot turn and diverge (or turn and converge) at the same time. If we want to require that the agent turns only when it is on the path, we can add the constraint $\mathsf{stay}_D = 0$ to the self-transition that fires turn in $A_{\mathsf{turn},D}$.

As indicated, we prefer for the agent to stay on the path as much as possible. The preferences expressed to this end are of a different form than those of Figure 3; after all, when the agent has strayed from the path to the right, we prefer to go to the left, and vice versa. If the agent is still on the path, it prefers to remain that way. Such preferences are expressed in the SCA in Figure 5a, where we require that $e_{\mathsf{diverge}} <_E e_{\mathsf{stay},D} <_E e_{\mathsf{converge}}$. In this SCA, state s_L represents that the agent has deviated to the left,

while s_R models a deviation to the right. We can see that in s_T (where the agent is on the path), the most preferred transition is the self-transition firing $e_{\mathsf{stay},D}$.



(a) The SCA A_{center}, expressing the preference of the agent to stay on the path.



(b) The SCA A_{drive}, that encodes our preference to keep moving.

Figure 5: SCAs pertaining to movement preferences.

To encode the preference that the agent should not stay in the exact same state in both A_{stray} and A_{path} (i.e., it is better to diverge than not to move at all), we compose A_{center} with the SCA depicted in Figure 5b. This SCA expresses its preferences over the same semiring E as A_{center}, and we postulate that $e_{\mathsf{stay},D} \otimes e_{\mathsf{stay},D} <_E e_{\mathsf{diverge}}$. Informally, this SCA penalizes firing port stay_D in concert with port stay_P by the additional "cost" $e_{\mathsf{stay},D}$ and leaves all other preferences intact.

By expressing this preference in a separate SCA we achieve a (desirable) separation of the concern "the agent should stay on the path" from "the agent should keep moving". More importantly, however, by creating a separate SCA for the latter concern we also save ourselves the error-prone effort of manually working out the possible combinations of firing right with or without firing stay_D concurrently.

The component that describes deviation from the path is now given by

$$A_{\mathsf{deviate}} = A_{\mathsf{stray}} \otimes A_{\mathsf{turn},D} \otimes A_{\mathsf{center}} \otimes A_{\mathsf{drive}}$$

Now, to compose A_{move} with A_{deviate}, we have several options. First of all, if they share their underlying semiring, we can simply calculate their product composition $A_{\mathsf{move}} \otimes A_{\mathsf{deviate}}$. There is, however, an objection to using the product composition operator in this particular case. Unless we tune our preferences carefully, we may find that $e_{\mathsf{progress}} \otimes e_{\mathsf{stay},D} <_E e_{\mathsf{stay},P} \otimes e_{\mathsf{converge}}$. If our primary concern is for the agent to patrol, then such a preference may be undesirable, for it will cause the agent to choose returning to the path over continuing its patrol beside the path.

Since patrolling and staying on the path are separate concerns, one may propose to use the join composition on these components, i.e., $A_{\mathsf{move}} \odot A_{\mathsf{deviate}}$. However, this causes $\langle e_{\mathsf{progress}}, e_{\mathsf{stay},D} \rangle$ and $\langle e_{\mathsf{stay},P}, e_{\mathsf{converge}} \rangle$ be unordered, meaning that in a state where both transitions are available, neither would be preferred over the other. Indeed, the only method to enforce the importance of one concern over the other is to use the lexicographic join operator and calculate $A_{\mathsf{move}} \triangleright A_{\mathsf{deviate}}$.

To illustrate the resulting automaton, we consider the following transitions:

$$\langle p_1, q_F, r_{-1}, s_L \rangle \xrightarrow{\{\mathsf{forward},\ \mathsf{right}\}, \mathsf{right}=0 \wedge \mathsf{forward}=2} \langle p_2, q_F, r_0, s_T \rangle \tag{1}$$

$$\langle p_1, q_F, r_{-1}, s_L \rangle \xrightarrow{\{\mathsf{forward},\ \mathsf{stay}_D\}, \mathsf{stay}_D=-1 \wedge \mathsf{forward}=2} \langle p_2, q_F, r_{-1}, s_L \rangle \tag{2}$$

$$\langle p_n, q_B, r_0, s_T \rangle \xrightarrow{\{\mathsf{stay}_P,\ \mathsf{right}\}, \mathsf{stay}_P=L \wedge \mathsf{left}=1} \langle p_n, q_B, r_1, s_R \rangle \tag{3}$$

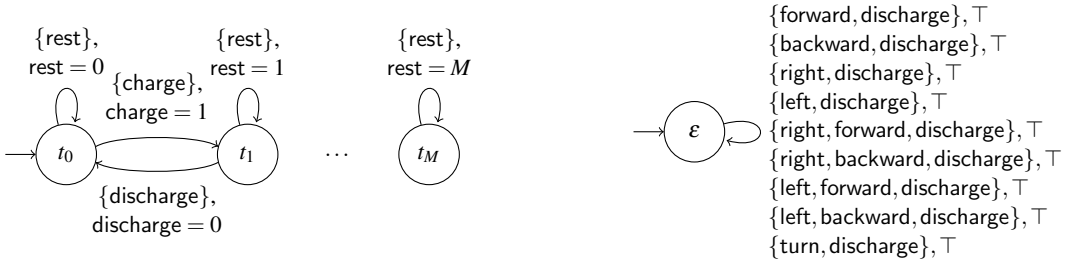
$$\langle p_n, q_B, r_0, s_T \rangle \xrightarrow{\{\mathsf{stay}_P,\ \mathsf{stay}_D\}, \mathsf{stay}_P=L \wedge \mathsf{stay}_D=0} \langle p_n, q_B, r_0, s_T \rangle \tag{4}$$

We can see that transition (2) is preferred over transition (1) since $\langle e_{\mathsf{progress}}, e_{\mathsf{stay},D}\rangle$ is a value preferred over $\langle e_{\mathsf{progress}}, e_{\mathsf{converge}}\rangle$. This means that our agent would rather move forward and converge at the same time than just move forward. Also, transition (3) is preferred over transition (4) for $\langle e_{\mathsf{stay},P}, e_{\mathsf{stay},D} \otimes e_{\mathsf{stay},D}\rangle <_E \langle e_{\mathsf{stay},P}, e_{\mathsf{diverge}}\rangle$; we can take this to mean that, if the agent cannot progress, it prefers to deviate from its path (and possibly find a path forward) over staying in the same location.

As a final observation in this section, we note that because A_{drive} is in the second component of the lexicographical composition, the agent is still allowed to turn while stationary at the endpoints of the path.

5.3 Energy usage

We now consider that the agent may be supplied with a finite amount of energy units M, in what we will for the purpose of this discussion refer to as a battery. The agent can also recharge at a designated position in the environment, but has to remain stationary to do so. To model the amount of energy left, we use the CA depicted in Figure 6a. The ports charge and discharge allow manipulation of the energy state and present the new energy level as their values. To attach an energy cost to actions from the model, we need to connect this CA to other ports; the CA that takes care of this is found in Figure 6b. For the sake of simplicity, we assume a unit energy cost for all actions that require energy.

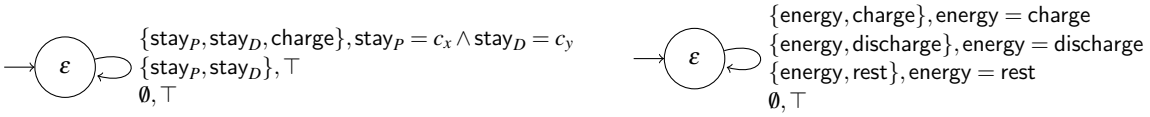


(a) A sketch of the CA A_{battery}, keeping track of energy. (b) The CA A_{usage}, modelling energy consumption

Figure 6: CAs pertaining to the battery level.

We assume that the charging station is located at c_y unit steps to the right of path position c_x. To model this placement, we use a CA similar to the one from Figure 2b, found in Figure 7a. In order to act on the change in energy conveniently, we use the CA in Figure 7b, which fires the port energy whenever a port of A_{battery} fires, and provides the energy level at that port.

At this point, we can incorporate the CAs described in this section into the description of the full system. By doing so, we would obtain a system that keeps track of its battery level. If the battery is empty (state t_1 of A_{battery}), no further energy-consuming moves are possible (for lack of a transition that fires discharge). However, in the absence of a mechanism that "plans" the moves of the agent, such a system



(a) The CA A_{charge}, modelling the charging station. (b) The CA A_{energy}, which provides the energy level.

Figure 7: More energy-related CAs.

would most likely become stuck after it has depleted its energy. The reason for this is that there is no preference expressing that the agent likes to recharge when its energy level drops below a certain level ℓ.

To accommodate for this, we first need to design a different regime of preferences in which the agent prefers to move towards the charging station. This can be done using a pair of SCAs, both similar to A_{center} (refer to Figure 5a), one to express preferences about movement along the path and another to express preferences about movement deviating from the path. For the sake of brevity, we assume the construction of this SCA to be obvious and refer to it as A_{return}. We can now calculate the join composition

$$A_{\mathsf{position}} = (A_{\mathsf{return}} \otimes A_{\mathsf{battery}} \otimes A_{\mathsf{usage}} \otimes A_{\mathsf{charge}} \otimes A_{\mathsf{energy}}) \odot (A_{\mathsf{move}} \triangleright A_{\mathsf{deviate}})$$

to obtain an automaton that expresses preferences on both the regular movement and the movement toward the charging station. Finally, we must express that the preference originating from A_{return} should be considered whenever the target state of the transition within A_{battery} is t_i with $i < \ell$, while the preference from $A_{\mathsf{move}} \triangleright A_{\mathsf{deviate}}$ takes precedence in all other cases. Let $E_{\mathsf{return}} \odot E_{\mathsf{patrol}}$ be the underlying semiring of A_{position}, and let h be the injection from $E_{\mathsf{return}} \odot E_{\mathsf{patrol}}$ into $E_{\mathsf{return}} \times E_{\mathsf{patrol}}$. Then the SCA A_{select} in Figure 8, with $E_{\mathsf{return}} \times E_{\mathsf{patrol}}$ as the underlying semiring, selects a preference between the two under product composition with $h(A_{\mathsf{position}})$, i.e., $A_{\mathsf{agent}} = A_{\mathsf{select}} \otimes h(A_{\mathsf{position}})$.

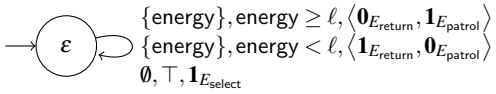


Figure 8: The SCA A_{select}, selecting between the preference regimes based on the current energy level.

By composing the preference value in the first position with $\mathbf{0}_{E_{\mathsf{return}}}$ and the one in the second position with E_{patrol} when $\mathsf{energy} \geq \ell$, A_{select} essentially silences the preferences expressed by A_{return}. As a consequence, the only preferences that matter when $\mathsf{energy} \geq \ell$ are those expressed by $A_{\mathsf{move}} \triangleright A_{\mathsf{deviate}}$.

5.4 Discussion

Needless to say, the final automaton can become rather large. To be precise, the automaton A_{agent} has $54K(2L+1)(M+1)$ states. However, in spite of syntactically available transitions, the pertinent constraints imply that not all of those states are actually reachable; for instance, positions that cannot be reached using the amount of energy M available to the agent do not appear in the reachable state space. A sketch of the state space for small parameters appears in the technical report [11]. In this sketch, we can see that even for these small parameters the size of the state space can become quite large, with a dense transition structure. This further drives home our point that the addition of preferences is only useful when we consider preferences as first-class citizens of a compositional framework; adding preference after the fact (i.e., to the composed state space) is simply not feasible.

The vastness of this state space also indicates the difficulty of the task that an engineer faces to produce a monolithic specification of the behavior of such a system, in terms of an automaton, an LTS, a set of constraints, logic, or any other formalism. Our compositional framework based on SCAs supports separate specification of multitudes of components, agents, aspects, features, and modules that comprise such a system — a much more manageable task. The specification of the whole system can subsequently be constructed using the composition operators in our framework to combine those smaller specifications. Of course, for this approach to work, one needs a verification method that is compatible with the composition operators.

6 Conclusion

In this paper, we argued that components with preference values associated with their actions generally aid in making a system robust, by providing alternatives when the most preferred option is not available. This property extends to composition, in the sense that such preference-aware components are capable of falling back to lower-preference actions when the composed preference dominates that of actions otherwise preferred by the component. We considered an existing formalism for preference-aware components, namely Soft Constraint Automata, and proposed a number of operators that can be used to compose such preference-aware components; each of these has its use depending on the agent being described.

By means of an example, we then illustrated the utility of SCAs and their different composition operators to design a reasonably robust autonomous agent. Most importantly, in this example we demonstrated that the use of SCAs can promote separation of concerns and extensibility. In the process, we highlighted some techniques using the composition operators, arguing in favor of their generality.

7 Further work

For our framework to have any practical value, we of course need to be able to simulate the behavior of SCAs. This is slightly more complicated than simulating CAs, because instead of CSPs we now need to solve SCSPs to discover which transitions are available. Luckily, a number of SCSP-solving algorithms exist [5]. We have developed a tentative implementation of the ideas expressed in this paper, inspired in part by the SCSP-solving approach in [14], using Gecode[4] as a CSP-solver. We plan to continue work on this implementation, as a tool to explore and demonstrate the specification capabilities of SCAs. In particular, our tool can still be extended to include the techniques from [10], which enable state-by-state evaluation of composition operators, saving computation of the unreachable state space.

We limited the example in Section 5 to a single-agent system. Nevertheless, it seems intuitively clear that the same formalism can be used to describe multi-agent systems. We also suspect that in the context of multi-agent systems, coordination between all agents for every transition (to find out the most-preferred transition) is not always necessary or even possible. For instance, when the inter-agent distance is large, it may occur that transitions available to agents are never mutually exclusive and a local calculation of a most-preferred transition would suffice. Because these situations need not persist throughout the operation, it seems that some sort of mechanism to distinguish between these situations is necessary, like the minimum contact distance in [15].

While this work concentrates on a method to specify Cyber-Physical Systems, our framework as of yet lacks a method allowing users to verify and reason about properties of the system. To obtain such a method, one can look into existing verification techniques for CAs [2]. Since SCAs constitute a generalization of CAs, we expect that model checking SCAs is at least as hard as model checking CAs. A method based on rewriting logic may also be feasible, especially because it has already been shown that solving SCSPs using rewriting logic is possible [16].

In this work, we restricted ourselves to agents that make single-step decisions about their movements. As a consequence, the agents modeled are unable to handle situations in which a transition with very low preference must be taken in order for a subsequent transition with a very high preference to become available. Further work may look into accommodating a method capable of planning a multi-step path, perhaps by means of simulation as proposed by Belzner et al. in [4].

[4] See `http://www.gecode.org`

References

[1] Farhad Arbab & Francesco Santini (2012): *Preference and Similarity-Based Behavioral Discovery of Services*. In: *Proc. Web Services and Formal Methods (WS-FM)*, pp. 118–133. Available at http://dx.doi.org/10.1007/978-3-642-38230-7_8.

[2] Christel Baier, Tobias Blechmann, Joachim Klein, Sascha Klüppelholz & Wolfgang Leister (2010): *Design and Verification of Systems with Exogenous Coordination Using Vereofy*. In: *Proc. International Symposium on Leveraging Applications, ISoLA 2010*, pp. 97–111. Available at http://dx.doi.org/10.1007/978-3-642-16561-0_15.

[3] Christel Baier, Marjan Sirjani, Farhad Arbab & Jan Rutten (2006): *Modeling component connectors in Reo by constraint automata*. *Science of Computer Programming* 61, pp. 75–113. Available at http://dx.doi.org/10.1016/j.scico.2005.10.008.

[4] Lenz Belzner, Rolf Hennicker & Martin Wirsing (2015): *OnPlan: A Framework for Simulation-Based Online Planning*. In: *Proc. Formal Aspects of Component Software*, pp. 1–30. Available at http://dx.doi.org/10.1007/978-3-319-28934-2_1.

[5] Stefano Bistarelli (2004): *Semirings for Soft Constraint Solving and Programming*. *Lecture Notes in Computer Science* 2962, Springer. Available at http://dx.doi.org/10.1007/b95712.

[6] Stefano Bistarelli, Ugo Montanari & Francesca Rossi (1995): *Constraint Solving over Semirings*. In: *Proc. International Joint Conference on Artificial Intelligence, IJCAI 95*, pp. 624–630. Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.212.6733&rep=rep1&type=pdf.

[7] Stefano Bistarelli, Ugo Montanari & Francesca Rossi (1997): *Semiring-based constraint satisfaction and optimization*. *J. ACM* 44(2), pp. 201–236. Available at http://dx.doi.org/10.1145/256303.256306.

[8] Rance Cleaveland & Matthew Hennessy (1990): *Priorities in Process Algebras*. *Inf. Comput.* 87(1/2), pp. 58–77. Available at http://dx.doi.org/10.1016/0890-5401(90)90059-Q.

[9] Fabio Gadducci, Matthias M. Hölzl, Giacoma Valentina Monreale & Martin Wirsing (2013): *Soft Constraints for Lexicographic Orders*. In: *Advances in Artificial Intelligence and Its Applications, Mexican International Conference on Artificial Intelligence, MICAI*, pp. 68–79. Available at http://dx.doi.org/10.1007/978-3-642-45114-0_6.

[10] Sung-Shik T.Q. Jongmans, Tobias Kappé & Farhad Arbab (2015): *Composing Constraint Automata, State-by-State*. In: *Proc. Formal Aspects of Component Software*, pp. 263–280. Available at http://dx.doi.org/10.1007/978-3-319-28934-2_12.

[11] Tobias Kappé, Farhad Arbab & Carolyn Talcott (2016): *A Compositional Framework For Preference-Aware Agents*. CWI Technical Report FM-1603. Available at https://repository.cwi.nl/noauth/search/fullrecord.php?publnr=24625.

[12] Sanjiang Li & Mingsheng Ying (2008): *Soft constraint abstraction based on semiring homomorphism*. *Theor. Comput. Sci.* 403(2-3), pp. 192–201. Available at http://dx.doi.org/10.1016/j.tcs.2008.03.029.

[13] Jan J. M. M. Rutten (2005): *A coinductive calculus of streams*. *Mathematical Structures in Computer Science* 15(1), pp. 93–147. Available at http://dx.doi.org/10.1017/S0960129504004517.

[14] Martin Sachenbacher & Brian C. Williams (2006): *Conflict-Directed A* Search for Soft Constraints*. In: *Proc. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR*, pp. 182–196. Available at http://dx.doi.org/10.1007/11757375_16.

[15] Carolyn L. Talcott, Farhad Arbab & Maneesh Yadav (2015): *Soft Agents: Exploring Soft Constraints to Model Robust Adaptive Distributed Cyber-Physical Agent Systems*. In: *Software, Services, and Systems — Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering*, pp. 273–290. Available at http://dx.doi.org/10.1007/978-3-319-15545-6_18.

[16] Martin Wirsing, Grit Denker, Carolyn L. Talcott, Andy Poggio & Linda Briesemeister (2007): *A Rewriting Logic Framework for Soft Constraints*. *Electr. Notes Theor. Comput. Sci.* 176(4), pp. 181–197. Available at http://dx.doi.org/10.1016/j.entcs.2007.06.015.