A Component-Oriented Framework for Autonomous Agents

Kappé, T.; Arbab, F.; Talcott, C.L.; Proença J., Lumpe M.

A Component-oriented Framework for Autonomous Agents

Tobias Kappé*[1], Farhad Arbab[2,3], and Carolyn Talcott[4]

[1]University College London, London, United Kingdom
[2]Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
[3]LIACS, Leiden University, Leiden, The Netherlands
[4]SRI International, Menlo Park, USA

Abstract

The design of a complex system warrants a compositional methodology, i.e., composing simple components to obtain a larger system that exhibits their collective behavior in a meaningful way. We propose an automaton-based paradigm for compositional design of such systems where an *action* is accompanied by one or more *preferences*. At run-time, these preferences provide a natural fallback mechanism for the component, while at design-time they can be used to reason about the behavior of the component in an uncertain physical world. Using structures that tell us how to compose preferences and actions, we can compose formal representations of individual components or agents to obtain a representation of the composed system. We extend Linear Temporal Logic with two unary connectives that reflect the compositional structure of the actions, and show how it can be used to diagnose undesired behavior by tracing the falsification of a specification back to one or more culpable components.

1 Introduction

Consider the design of a software package that steers a crop surveillance drone. Such a system (in its simplest form, a single drone agent) should survey a field and relay the locations of possible signs of disease to its owner. There are a number of concerns at play here, including but not limited to maintaining an acceptable altitude, keeping an eye on battery levels and avoiding birds of prey. In such a situation, it is best practice to isolate these separate concerns into different modules — thus allowing for code reuse, and requiring the use of well-defined protocols in case coordination between modules is necessary. One would also like to verify that the designed system satisfies desired properties, such as "even on a conservative energy budget, the drone can always reach the charging station".

In the event that the designed system violates its verification requirements or exhibits behavior that does not conform to the specification, it is often useful to have an example of such behavior. For instance, if the surveillance drone fails to maintain its target altitude, an example of behavior where this happens could tell us that the drone attempted to reach the far side of the field and ran out of energy. Additionally, failure to verify an LTL-like formula typically comes with a counterexample — indeed, a counterexample arises from the automata-theoretic verification approach quite naturally [33]. Taking this idea of *diagnostics* one step further in the context of a compositional design, it would also be useful to be able to identify the components responsible for allowing a behavior that deviates from the specification, whether this behavior comes from a run-time observation or a design-time counterexample to a desired property. The designer then knows which components should be adjusted (in our example, this may turn out to be the route planning component), or, at the very least, rule out components that are not directly responsible (such as the wildlife evasion component).

*tkappe@cs.ucl.ac.uk

In this paper, we propose an automata-based paradigm based on Soft Constraint Automata [1, 20], called Soft Component Automata (SCAs[1]). An SCA is a state-transition system where transitions are labeled with actions and preferences. Higher-preference transitions typically contribute more towards the goal of the component; if a component is in a state where it wants the system to move north, a transition with action north has a higher preference than a transition with action south. At run-time, preferences provide a natural fallback mechanism for an agent: in ideal circumstances, the agent would perform only actions with the highest preferences, but if the most-preferred actions fail, the agent may be permitted to choose a transition of lower preference. At design-time, preferences can be used to reason about the behavior of the SCA in suboptimal conditions, by allowing all actions whose preference is bounded from below by a threshold. In particular, this is useful if the designer wants to determine the circumstances (i.e., threshold on preferences) where a property is no longer verified by the system.

Because the actions and preferences of an SCA reside in well-defined mathematical structures, we can define a composition operator on SCAs that takes into account the composition of actions as well as preferences. The result of composition of two SCAs is another SCA where actions and preferences reflect those of the operands. As we shall see, SCAs are amenable to verification against formulas in Linear Temporal Logic (LTL). More specifically, one can check whether the behavior of an SCA is contained in the behavior allowed by a formula of LTL.

Soft Component Automata are a generalization of Constraint Automata [3]. The latter can be used to coordinate interaction between components in a verifiable fashion [2]. Just like Constraint Automata, the framework we present blurs the line between *computation* and *coordination* — both are captured by the same type of automata. Consequently, this approach allows us to reason about these concepts in a uniform fashion: coordination is not separate in the model, it is effected by components which are inherently part of the model.

We present two contributions in this paper. First, we propose an compositional automata-based design paradigm for autonomous agents that contains enough information about actions to make agents behave in a robust manner — by which we mean that, in less-than-ideal circumstances, the agent has alternative actions available when its most desired action turns out to be impossible, which help it achieve some subset of goals or its original goals to a lesser degree. We also put forth a dialect of LTL that accounts for the compositional structure of actions and can be used to verify guarantees about the behavior of components, as well as their behavior in composition. Our second contribution is a method to trace errant behavior back to one or more components, exploiting the algebraic structure of preferences. This method can be used with both run-time and design-time failures: in the former case, the behavior arises from the action history of the automaton, in the latter case it is a counterexample obtained from verification.

In Section 2, we mention some work related to this paper; in Section 3 we discuss the necessary notation and mathematical structures. In Section 4, we introduce Soft Component Automata, along with a toy model. We discuss the syntax and semantics of the LTL-like logic used to verify properties of SCAs in Section 5. In Section 6, we propose a method to extract which components bear direct responsibility for a failure. Our conclusions comprise Section 7, and some directions for further work appear in Section 8.

Acknowledgements The authors would like to thank Vivek Nigam and the anonymous FACS-referees for their valuable feedback. This work was partially supported by ONR grant N00014–15–1–2202.

2 Related Work

The algebraic structure for preferences called the *Constraint Semiring* was proposed by Bistarelli et al. [5, 4]. Further exploration of the compositionality of such structures appears in [12, 15, 20]. The structure we propose for modeling actions and their compositions is an algebraic reconsideration of *static constructs* [16].

[1]Here, we use the abbreviation *SCA* exclusively to refer to Soft *Component* Automata.

The automata formalism used in this paper generalizes *Soft Constraint Automata* [3, 1]. The latter were originally proposed to give descriptions of Web Services [1]; in [20], they were used to model fault-tolerant, compositional autonomous agents. Using preference values to specify the behavior of autonomous agents is also explored from the perspective of rewriting logic in the *Soft Agent Framework* [31, 32]. Recent experiments with the Soft Agent Framework show that behavior based on soft constraints can indeed contribute robustness [22].

Sampath et al. [28] discuss methods to detect unobservable errors based on a model of the system and a trace of observable events; others extended this approach [11, 25] to a multi-component setting. Casanova et al. [9] wrote about fault localisation in a system where some components are inobservable, based on which computations (tasks involving multiple components) fail. In these paradigms, one tries to find out where a *runtime fault* occurs; in contrast, we try to find out which component is responsible for *undesired behavior*, i.e., behavior that is allowed by the system but not desired by the specification.

A general framework for fault ascription in concurrent systems based on *counterfactuals* is presented in [13, 14]. Formal definitions are given for failures in a given set of components to be necessary and/or sufficient cause of a system violating a given property. Components are specified by sets of sets of events (analogous to actions) representing possible correct behaviors. A parallel (asynchronous) composition operation is defined on components, but there is no notion of composition of events or explicit interaction between components. A system is given by a global behavior (a set of event sets) together with a set of system component specifications. The global behavior, which must be provided separately, includes component events, but may also have other events, and may violate component specifications (hence the faulty components). In our approach, global behavior is obtained by component composition. Undesired behavior may be local to a component or emerge as the result of interactions.

In LTL, a counterexample to a negative result arises naturally if one employs automata-based verification techniques [24, 33]. In this paper, we further exploit counterexamples to gain information about the component or components involved in violating the specification. The application of LTL to Constraint Automata is inspired by an earlier use of LTL for Constraint Automata [2].

Some material in this paper appeared in the first author's master's thesis [18].

3 Preliminaries

If Σ is a set, then 2^Σ denotes the set of subsets of Σ, i.e., the *powerset* of Σ. We write Σ^* for the set of *finite words* over Σ, and if $\sigma \in \Sigma^*$ we write $|\sigma|$ for the *length* of σ. We write $\sigma(n)$ for the n-th letter of σ (starting at 0). Furthermore, let Σ^ω denote the set of functions from $\mathbb{N}$ to Σ, also known as *streams* over Σ [26]. We define for $\sigma \in \Sigma^\omega$ that $|\sigma| = \omega$ (the smallest infinite ordinal). Concatenation of a stream to a finite word is defined as expected. We use the superscript ω to denote infinite repetition, writing $\sigma = \langle 0, 1 \rangle^\omega$ for the parity function; we write Σ^π for the set of *eventually periodic* streams in Σ^ω, i.e., $\sigma \in \Sigma^\omega$ such that there exist $\sigma_h, \sigma_t \in \Sigma^*$ with $\sigma = \sigma_h \cdot \sigma_t^\omega$. We write $\sigma^{(k)}$ with $k \in \mathbb{N}$ for the *k-th derivative* of σ, which is given by $\sigma^{(k)}(n) = \sigma(k + n)$.

If S is a set and $\odot : S \times S \to S$ a function, we refer to $\odot$ as an *operator on* S and write $p \odot q$ instead of $\odot(p, q)$. We always use parentheses to disambiguate expressions if necessary. To model composition of actions, we need a slight generalization. If $R \subseteq S \times S$ is a relation and $\odot : R \to S$ is a function, we refer to $\odot$ as a *partial operator on S up to R*; we also use infix notation by writing $p \odot q$ instead of $\odot(p, q)$ whenever pRq. If $\odot : R \to S$ is a partial operator on S up to R, we refer to $\odot$ as *idempotent* if $p \odot p = p$ for all $p \in S$ such that pRp, and *commutative* if $p \odot q = q \odot p$ whenever $p, q \in S$, pRq and qRp. Lastly, $\odot$ is *associative* if for all $p, q, r \in S$, pRq and $(p \odot q)Rr$ if and only if qRr and $pR(q \odot r)$, either of which implies that $(p \odot q) \odot r = p \odot (q \odot r)$. When $R = S \times S$, we recover the canonical definitions of idempotency, commutativity and associativity.

A *constraint semiring*, or *c-semiring*, provides a structure on preference values that allows us to *compare* the preferences of two actions to see if one is preferred over the other as well as *compose* preference values of component actions to find out the preference of their composed action. A c-semiring [5, 4] is a tuple $\langle \mathbb{E}, \bigoplus, \otimes, \mathbf{0}, \mathbf{1} \rangle$ such that (1) $\mathbb{E}$ is a set, called the *carrier*, with $\mathbf{0}, \mathbf{1} \in \mathbb{E}$, (2) $\bigoplus : 2^\mathbb{E} \to \mathbb{E}$ is a function such that for $e \in \mathbb{E}$ we have that $\bigoplus \emptyset = \mathbf{0}$ and $\bigoplus \mathbb{E} = \mathbf{1}$, as well as $\bigoplus \{e\} = e$, and for $\mathcal{E} \subseteq 2^\mathbb{E}$, also $\bigoplus \{\bigoplus(E) : E \in \mathcal{E}\} = \bigoplus \bigcup \mathcal{E}$ (the *flattening property*), and

(3) $\otimes : \mathbb{E} \times \mathbb{E} \to \mathbb{E}$ is a commutative and associative operator, such that for $e \in \mathbb{E}$ and $E \subseteq \mathbb{E}$, it holds that $e \otimes \mathbf{0} = \mathbf{0}$ and $e \otimes \mathbf{1} = e$ as well as $e \otimes \bigoplus E = \bigoplus\{e \otimes e' : e' \in E\}$. We denote a c-semiring by its carrier; if we refer to $\mathbb{E}$ as a c-semiring, associated symbols are denoted $\bigoplus_{\mathbb{E}}, \mathbf{0}_{\mathbb{E}}$, et cetera. We drop the subscript when only one c-semiring is in context.

The operator $\bigoplus$ of a c-semiring $\mathbb{E}$ induces an idempotent, commutative and associative binary operator $\oplus : \mathbb{E} \times \mathbb{E} \to \mathbb{E}$ by defining $e \oplus e' = \bigoplus(\{e, e'\})$ The relation $\leq_{\mathbb{E}} \subseteq \mathbb{E} \times \mathbb{E}$ is such that $e \leq_{\mathbb{E}} e'$ if and only if $e \oplus e' = e'$; $\leq_{\mathbb{E}}$ is a partial order on $\mathbb{E}$, with $\mathbf{0}$ and $\mathbf{1}$ the minimal and maximal elements [4]. All c-semirings are complete lattices, with $\bigoplus$ filling the role of the least upper bound operator [4]. Furthermore, $\otimes$ is *intensive*, meaning that for any $e, e' \in \mathbb{E}$, we have $e \otimes e' \leq e$ [4]. Lastly, when $\otimes$ is idempotent, $\otimes$ coincides with the greatest lower bound [4].

Models of a c-semiring include $\mathbb{W} = \langle \mathbb{R}_{\geq 0} \cup \{\infty\}, \inf, \hat{+}, \infty, 0 \rangle$ (the *weighted semiring*), where inf is the infimum and $\hat{+}$ is arithmetic addition generalized to $\mathbb{R}_{\geq 0} \cup \{\infty\}$. Here, $\leq_{\mathbb{W}}$ coincides with the obvious definition of the order $\geq$ on $\mathbb{R}_{\geq 0} \cup \{\infty\}$. Composition operators for c-semirings exist, such as product composition [6] and (partial) lexicographic composition [12]. We refer to [20] for a self-contained discussion of these composition techniques.

4 Component Model

We now discuss our component model for the construction of autonomous agents.

4.1 Component Action Systems

Observable behavior of agents is the result of the actions put forth by their individual components; we thus need a way to talk about how actions compose. For example, in our crop surveillance drone, the following may occur:

- The component responsible for taking pictures wants to take a snapshot, while the routing component wants to move north. Assuming the camera is capable of taking pictures while moving, these actions may compose into the action "take a snapshot while moving north". In this case, actions compose *concurrently*, and we say that the latter action *captures* the former two.
- The drone has a single antenna that can be used for GPS and communications, but not both at the same time. The component responsible for relaying pictures has finished its transmission and wants to release its lock on the antenna, while the navigation component wants to get a fix on the location and requests use of the antenna. In this case, the actions "release privilege" and "obtain privilege" compose *logically*, into a "transfer privilege" action.
- The routing component wants to move north, while the wildlife avoidance component notices a hawk approaching from that same direction, and thus wants to move south. In this case, the intentions of the two components are contradictory; these component actions are *incomposable*, and some resolution mechanism (e.g., priority) will have to decide which action takes precedence.

All of these possibilities are captured in the definition below.

Definition 1. A *Component Action System (CAS)* is a tuple $\langle \Sigma, \circledcirc, \boxdot \rangle$, such that Σ is a finite set of *actions*, $\circledcirc \subseteq \Sigma \times \Sigma$ is a reflexive and symmetric relation and $\boxdot : \circledcirc \to \Sigma$ is an idempotent, commutative and associative operator on Σ up to $\circledcirc$ (i.e., $\boxdot$ is an operator defined only on elements of Σ related by $\circledcirc$). We call $\circledcirc$ the *composability relation*, and $\boxdot$ the *composition operator*.

Every CAS $\langle \Sigma, \circledcirc, \boxdot \rangle$ induces a relation $\sqsubseteq$ on Σ, where for $a, b \in \Sigma$, $a \sqsubseteq b$ if and only if there exists a $c \in \Sigma$ such that a and c are composable ($a \circledcirc c$) and they compose into b ($a \boxdot c = b$). One can easily verify that $\sqsubseteq$ is a preorder; accordingly, we call $\sqsubseteq$ the *capture preorder* of the CAS.

As with c-semirings, we may refer to a set Σ as a CAS. When we do, its composability relation, composition operator and preorder are denoted by $\circledcirc_\Sigma$, $\boxdot_\Sigma$ and $\sqsubseteq_\Sigma$. We drop the subscript when there is only one CAS in context.

We model incomposability of actions by omitting them from the composability relation; i.e., if south is an action that compels the agent to move south, while north drives the agent north, we set south $\not\circledcirc$ north. Note that $\circledcirc$ is not necessarily transitive. This makes sense in the scenarios above, where snapshot is composable with south as well as north, but north is incomposable with south. Moreover, incomposability carries over to compositions: if south $\circledcirc$ snapshot and south $\not\circledcirc$ north, also (south $\boxdot$ snapshot) $\not\circledcirc$ north. This is formalized in the following lemma.

Lemma 1. *Let $\langle \Sigma, \circledcirc, \boxdot \rangle$ be a CAS and let $a, b, c \in \Sigma$. If $a \circledcirc b$ but $a \not\circledcirc c$, then $(a \boxdot b) \not\circledcirc c$. Moreover, if $a \not\circledcirc c$ and $a \sqsubseteq b$, then $b \not\circledcirc c$.*

Proof. For the first claim, suppose that $(a \boxdot b) \circledcirc c$. Then, since $\boxdot$ is associative up to $\circledcirc$, it follows that $b \circledcirc c$ and $a \circledcirc (b \boxdot c)$, which contradicts the premise that $b \not\circledcirc c$. We thus conclude that $(a \boxdot b) \not\circledcirc c$.

For the second claim, suppose that $a \not\circledcirc c$ and $a \sqsubseteq b$. Then there exists a $d \in \Sigma$ such that $a \circledcirc d$ and $a \boxdot d = b$. By the above, $b = (a \boxdot d) \not\circledcirc c$. □

The composition operator facilitates concurrent as well as logical composition. Given actions obtain, release and transfer, with their interpretation as in the second scenario, we can encode that obtain and release are composable by stipulating that obtain $\circledcirc$ release, and say that their (logical) composition involves an exchange of privileges by choosing obtain $\boxdot$ release = transfer. Furthermore, the capture preorder describes our intuition of capturing: if snapshot and move are the actions of the first scenario, with snapshot $\circledcirc$ north, then snapshot, north $\sqsubseteq$ snapshot $\boxdot$ north.

Port Automata [21] contain a model of a CAS. Here, actions are sets of symbols called *ports*, i.e., elements of 2^P for some finite set P. Actions $\alpha, \beta \in 2^P$ are compatible when they agree on a fixed set $\gamma \subseteq P$, i.e., if $\alpha \cap \gamma = \beta \cap \gamma$, and their composition is $\alpha \cup \beta$. Similarly, we also find an instance of a CAS in *(Soft) Constraint Automata* [3, 1]; see [18] for a full discussion of this correspondence.

4.2 Soft Component Automata

Having introduced the structure we impose on actions, we are now ready to discuss the automaton formalism that specifies the sequences of actions that are allowed, along with the preferences attached to such actions.

Definition 2. A *Soft Component Automaton (SCA)* is a tuple $\langle Q, \Sigma, \mathbb{E}, \rightarrow, q^0, t \rangle$ where Q is a finite set of *states*, with $q^0 \in Q$ the *initial state*, Σ is a CAS and $\mathbb{E}$ is a c-semiring with $t \in \mathbb{E}$, and $\rightarrow \subseteq Q \times \Sigma \times \mathbb{E} \times Q$ is a finite relation called the *transition relation*. We write $q \xrightarrow{a,e} q'$ when $\langle q, a, e, q' \rangle \in \rightarrow$.

An SCA models the actions available in each state of the component, how much these actions contribute towards the goal and the way actions transform the state. The threshold value restricts the available actions to those with a preference bounded from below by the threshold, either at run-time, or at design-time when one wants to reason about behaviors satisfying some minimum preference.

We stress here that the threshold value is purposefully defined as part of an SCA, rather than as a parameter to the semantics in Section 4.4. This allows us to speak of the preferences of an individual component, rather than a threshold imposed on the whole system; instead, the threshold of the system arises from the thresholds of the components, which is especially useful in Section 6.

We depict SCAs in a fashion similar to the graphical representation of finite state automata: as a labeled graph, where vertices represent states and the edges transitions, labeled with elements of the CAS and c-semiring. The initial state is indicated by an arrow without origin. The CAS, c-semiring and threshold value will always be made clear where they are germane to the discussion.

An example of an SCA is A_e, drawn in Figure 1; its CAS contains the incomposable actions charge, $\mathsf{discharge}_1$ and $\mathsf{discharge}_2$, and its c-semiring is the weighted semiring $\mathbb{W}$. This particular SCA can model the component of the crop surveillance drone responsible for keeping track of the amount of energy remaining in the system; in state q_n (for $n \in \{0, 1, \ldots, 4\}$), the drone has n units of energy left, meaning that in states q_1 to q_4, the component can spend one unit of energy through $\mathsf{discharge}_1$, and in states q_2 to q_4, the drone can consume two units of energy through $\mathsf{discharge}_2$.

In states q_0 to q_3, the drone can try to recharge through charge.[2] Recall that, in $\mathbb{W}$, higher values reflect a lower preference (a higher *weight*); thus, charge is preferred over $\mathsf{discharge}_1$.

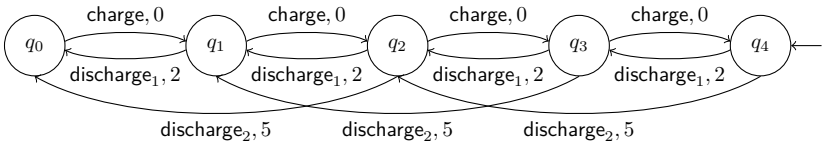


Figure 1: A component modeling energy management, A_{e}.

Here, A_{e} is meant to describe the possible behavior of the energy management component only. Availability of the actions within the *total model* of the drone (i.e., the composition of all components) is subject to how actions compose with those of other components; for example, the availability of charge may depend on the state of the component modelling position. Similarly, preferences attached to actions concern energy management only. In states q_0 to q_3, the component prefers to top up its energy level through charge, but the preferences of this component under composition with some other component may cause the composed preferences of actions composed with charge to be different. For instance, the total model may prefer executing an action that captures $\mathsf{discharge}_2$ over one that captures charge when the former entails movement and the latter does not, especially when survival necessitates movement.

Nevertheless, the preferences of A_{e} affect the total behavior. For instance, the weight of spending one unit of energy (through $\mathsf{discharge}_1$) is lower than the weight of spending two units (through $\mathsf{discharge}_2$). This means that the energy component prefers to spend a small amount of energy before re-evaluating over spending more units of energy in one step. This reflects a level of care: by preferring small steps, the component hopes to avoid situations where too little energy is left to avoid disaster.

4.3 Composition

Composition of two SCAs arises naturally, as follows.

Definition 3. Let $A_i = \langle Q_i, \Sigma, \mathbb{E}, \to_i, q_i^0, t_i \rangle$ be an SCA for $i \in \{0, 1\}$. The *(parallel) composition* of A_0 and A_1 is the SCA $\langle Q, \Sigma, \mathbb{E}, \to, q^0, t_0 \otimes t_1 \rangle$, denoted $A_0 \bowtie A_1$, where $Q = Q_0 \times Q_1$, $q^0 = \langle q_0^0, q_1^0 \rangle$, $\otimes$ is the composition operator of $\mathbb{E}$, and $\to$ is the smallest relation satisfying

$$\frac{q_0 \xrightarrow{a_0,\ e_0}_0 q_0' \qquad q_1 \xrightarrow{a_1,\ e_1}_1 q_1' \qquad a_0 \circledcirc a_1}{\langle q_0, q_1 \rangle \xrightarrow{a_0 \boxdot a_1,\ e_0 \otimes e_1} \langle q_0', q_1' \rangle}$$

In a sense, composition is a generalized product of automata, where composition of actions is mediated by the CAS: transitions with composable actions manifest in the composed automaton, as transitions with composed action and preference.

Composition is defined for SCAs that share CAS and c-semiring. Absent a common CAS, we do not know which actions compose, and what their compositions are. However, composition of SCAs with different c-semirings does make sense when the components model different concerns (e.g., for our crop surveillance drone, "minimize energy consumed" and "maximize covering of snapshots"), both contributing towards the overall goal. Earlier work on Soft Constraint Automata [20] explored this possibility. The additional composition operators proposed there can easily be applied to Soft Component Automata.

A state q of a component may become unreachable after composition, in the sense that no state composed of q is reachable from the composed initial state. For example, in the total model of our drone, it may occur that any state representing the drone at the far side of the field is unreachable, because the energy management component prevents some transition for lack of energy.

[2] This is a rather simplistic description of energy management. We remark that a more detailed description is possible by extending SCAs with *memory cells* [17] and using a memory cell to store the energy level. In such a setup, a state would represent a *range* of energy values that determines the components disposition regarding resources.

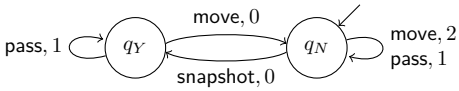


Figure 2: A component modeling the desire to take a snapshot at every location, A_{s}.

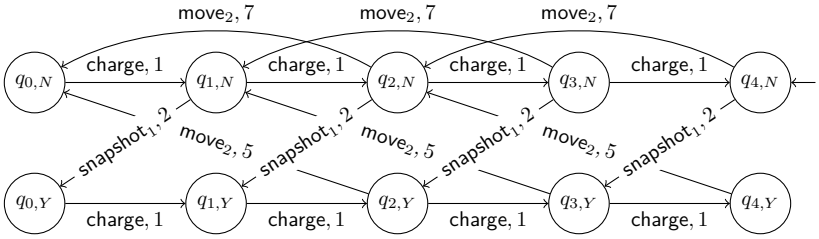


Figure 3: The composition of the SCAs A_{e} and A_{s}, dubbed $A_{\mathsf{e,s}}$: a component modeling energy and snapshot management. We abbreviate pairs of states $\langle q_i, q_j \rangle$ by writing $q_{i,j}$.

To discuss an example of SCA composition, we introduce the SCA A_{s} in Figure 2, which models the concern of the crop surveillance drone that it should take a snapshot of every location before moving to the next. The CAS of A_{s} includes the pairwise incomposable actions pass, move and $\mathsf{snapshot}$, and its c-semiring is the weighted c-semiring $\mathbb{W}$. We leave the threshold value t_{s} undefined for now. The purpose of A_{s} is reflected in its states: q_Y (respectively q_N) represents that a snapshot of the current location was (respectively was not) taken since moving there. If the drone moves to a new location, the component moves to q_N, while q_Y is reached by taking a snapshot. If the drone has not yet taken a snapshot, it prefers to do so over moving to the next spot (missing the opportunity).[3]

We grow the CAS of A_{e} and A_{s} to include the actions move, move_2, $\mathsf{snapshot}$ and $\mathsf{snapshot}_1$ (here, the action α_i is interpreted as "execute action α and account for i units of energy spent"), and $\circledcirc$ is the smallest reflexive, commutative and transitive relation such that the following hold: $\mathsf{move} \circledcirc \mathsf{discharge}_2$ (moving costs two units of energy), $\mathsf{snapshot} \circledcirc \mathsf{discharge}_1$ (taking a snapshot costs one unit of energy) and $\mathsf{pass} \circledcirc \mathsf{charge}$ (the snapshot state is unaffected by charging). We also choose $\mathsf{move} \boxdot \mathsf{discharge}_2 = \mathsf{move}_2$, $\mathsf{snapshot} \boxdot \mathsf{discharge}_1 = \mathsf{snapshot}_1$ and $\mathsf{pass} \boxdot \mathsf{charge} = \mathsf{charge}$. The composition of A_{e} and A_{e} is depicted in Figure 3.

The structure of $A_{\mathsf{e,s}}$ reflects that of A_{e} and A_{s}; for instance, in state $q_{2,Y}$ two units of energy remain, and we have a snapshot of the current location. The same holds for the transitions of $A_{\mathsf{e,s}}$; for example, $q_{2,N} \xrightarrow{\mathsf{snapshot}_1,\ 2} q_{1,Y}$ is the result of composing $q_2 \xrightarrow{\mathsf{discharge}_1,\ 2} q_1$ and $q_N \xrightarrow{\mathsf{snapshot},\ 0} q_Y$.

Also, note that in $A_{\mathsf{e,s}}$ the preference of the move_2-transitions at the top of the figure is lower than the preference of the diagonally-drawn move_2-transitions. This difference arises because the component transition in A_{s} of the former is $q_N \xrightarrow{\mathsf{move},\ 2} q_N$, while that of the latter is $q_Y \xrightarrow{\mathsf{move},\ 0} q_N$. As such, the preferences of the component SCAs manifest in the preferences of the composed SCA.

The action $\mathsf{snapshot}_1$ is not available in states of the form $q_{i,Y}$, because the only action available in q_Y is pass, which does not compose into $\mathsf{snapshot}_1$.

4.4 Behavioral semantics

The final part of our component model is a description of the behavior of SCAs. Here, the threshold determines which actions have sufficient preference for inclusion in the behavior. Intuitively, the threshold is an indication of the amount of flexibility allowed. In the context of composition, lowering the threshold of a component is a form of compromise: the component potentially gains behavior available for composition. Setting a lower threshold makes a component more permissive, but may also make it harder (or impossible) to achieve its goal.

The question of where to set the threshold is one that the designer of the system should answer

[3] A more detailed description of such a component could count the number of times the drone has moved without taking a snapshot first, and assign the preference of doing so again accordingly.

based on the properties and level of flexibility expected from the component; Section 5 addresses the formulation of these properties, while Section 6 talks about adjusting the threshold.

Definition 4. Let $A = \langle Q, \Sigma, \mathbb{E}, \to, q^0, t \rangle$ be an SCA. We say that a stream $\sigma \in \Sigma^\omega$ is a *behavior* of A when there exist streams $\mu \in Q^\omega$ and $\nu \in \mathbb{E}^\omega$ such that $\mu(0) = q^0$, and for all $n \in \mathbb{N}$, $t \leq \nu(n)$ and $\mu(n) \xrightarrow{\sigma(n),\ \nu(n)} \mu(n+1)$. The set of behaviors of A, denoted by $L(A)$, is called the *language* of A.

We note the similarity between the behavior of an SCA and that of Büchi-automata [8]; we elaborate on this in Appendix A.

To account for states that lack outgoing transitions, one could include implicit transitions labelled with halt (and some appropriate preference) to an otherwise unreachable "halt state", with a halt self-loop. Here, we set for all $\alpha \in \Sigma$ that $\mathsf{halt} \circledcirc \alpha$ and $\mathsf{halt} \boxdot \alpha = \mathsf{halt}$. To simplify matters, we do not elaborate on this.

Consider $\sigma = \langle \mathsf{snapshot}, \mathsf{move}, \mathsf{move} \rangle^\omega$ and $\tau = \langle \mathsf{snapshot}, \mathsf{move}, \mathsf{pass} \rangle^\omega$. We can see that when $t_\mathsf{s} = 2$, both are behaviors of A_s; when $t_\mathsf{s} = 1$, τ is a behavior of A_s, while σ is not, since every second move-action in σ has preference 2. More generally, if A and A' are SCAs over c-semiring $\mathbb{E}$ that only differ in their threshold values $t, t' \in \mathbb{E}$, and $t \leq t'$, then $L(A') \subseteq L(A)$. In the case of A_e, the threshold can be interpreted as a bound on the amount of energy to be spent in a single action; if $t_\mathsf{e} < 5$, then behaviors with $\mathsf{discharge}_2$ do not occur in $L(A_\mathsf{e})$.

Interestingly, if A_1 and A_2 are SCAs, then $L(A_1 \bowtie A_2)$ is not uniquely determined by $L(A_1)$ and $L(A_2)$. For example, suppose that $t_\mathsf{e} = 4$ and $t_\mathsf{s} = 1$, and consider $L(A_{\mathsf{e},\mathsf{s}})$, which contains $\langle \mathsf{snapshot} \rangle \cdot \langle \mathsf{move}, \mathsf{snapshot}, \mathsf{charge}, \mathsf{charge}, \mathsf{charge} \rangle^\omega$ even though the corresponding stream of component actions in A_e, i.e., the stream $\langle \mathsf{discharge}_1 \rangle \cdot \langle \mathsf{discharge}_2, \mathsf{discharge}_1, \mathsf{charge}, \mathsf{charge}, \mathsf{charge} \rangle^\omega$ is not contained in $L(A_\mathsf{e})$. This is a consequence of a more general observation for c-semirings, namely that $t \leq e$ and $t' \leq e'$ is sufficient but not necessary to derive $t \otimes t' \leq e \otimes e'$.

5 Linear Temporal Logic

We now turn our attention to verifying the behavior of an agent, by means of a simple dialect of Linear Temporal Logic (LTL). The aim of extending LTL is to reflect the compositional nature of the actions. This extension has two aspects, which correspond roughly to the relations $\sqsubseteq$ and $\circledcirc$: reasoning about behaviors that *capture* (i.e., are composed of) other behaviors, and about behaviors that are *composable* with other behaviors. For instance, consider the following scenarios:

(i) We want to verify that under certain circumstances, the drone performs a series of actions where it goes north before taking a snapshot. This is useful when, for this particular property, we do not care about other actions that may also be performed while or as part of going north, for instance, whether or not the drone engages in communications while moving.

(ii) We want to verify that every behavior of the snapshot-component is composable with some behavior that eventually recharges. This is useful when we want to abstract away from the action that allows recharging, i.e., it is not important which particular action composes with charge.

Our logic aims to accommodate both scenarios, by providing two new connectives: $\succ \phi$ describes every behavior that captures a behavior validating ϕ, while $\circledcirc \phi$ holds for every behavior composable with a behavior validating ϕ.

5.1 Syntax and semantics

The syntax of the LTL dialect we propose for SCAs contains atoms, conjunctions, negation, and the "until" and "next" connectives, as well as the unary connectives $\circledcirc$ and $\succ$. Formally, given a CAS Σ, the language $\mathcal{L}_\Sigma$ is generated by the grammar

$$\phi, \psi ::= \top \mid a \in \Sigma \mid \phi \wedge \psi \mid \phi \, U \, \psi \mid X \, \phi \mid \neg \phi \mid \succ \phi \mid \circledcirc \phi$$

As a convention, unary connectives take precedence over binary connectives. For example, $\succ \phi \, U \, \neg\psi$ should be read as $(\succ \phi) \, U (\neg\psi)$. We use parentheses to disambiguate formulas where this convention does not give a unique bracketing.

The semantics of our logic is given as a relation $\models_\Sigma$ between Σ^ω and $\mathcal{L}_\Sigma$; to be precise, $\models_\Sigma$ is the smallest such relation that satisfies the following rules

$$\frac{\sigma \in \Sigma^\omega}{\sigma \models_\Sigma \top} \qquad \frac{\sigma \in \Sigma^\omega}{\sigma \models_\Sigma \sigma(0)} \qquad \frac{\sigma \models_\Sigma \phi \qquad \sigma \models_\Sigma \psi}{\sigma \models_\Sigma \phi \wedge \psi}$$

$$\frac{n \in \mathbb{N} \qquad \forall k < n.\ \sigma^{(k)} \models_\Sigma \phi \qquad \sigma^{(n)} \models_\Sigma \psi}{\sigma \models_\Sigma \phi \, U \, \psi} \qquad \frac{\sigma^{(1)} \models_\Sigma \phi}{\sigma \models_\Sigma X \, \phi}$$

$$\frac{\sigma \not\models_\Sigma \phi}{\sigma \models_\Sigma \neg\phi} \qquad \frac{\sigma \models_\Sigma \phi \qquad \sigma \sqsubseteq^\omega \tau}{\tau \models_\Sigma \succ \phi} \qquad \frac{\sigma \models_\Sigma \phi \qquad \sigma \circledcirc^\omega \tau}{\tau \models_\Sigma \circledcirc \phi}$$

in which $\sqsubseteq^\omega$ and $\circledcirc^\omega$ are the pointwise extensions of the relations $\sqsubseteq$ and $\circledcirc$, i.e., $\sigma \sqsubseteq^\omega \tau$ when, for all $n \in \mathbb{N}$, it holds that $\sigma(n) \sqsubseteq \tau(n)$, and similarly for $\circledcirc^\omega$.

Although the atoms of our logic are formulas of the form $\phi = a \in \Sigma$ that have an exact matching semantics, in general one could use predicates over Σ. We chose not to do this to keep the presentation of examples simple.

The semantics of $\circledcirc$ and $\succ$ match their descriptions: if $\sigma \in \Sigma^\omega$ is described by ϕ (i.e., $\sigma \models_\Sigma \phi$) and $\tau \in \Sigma^\omega$ captures this σ at every action (i.e., $\sigma \sqsubseteq^\omega \tau$), then τ is a behavior described by $\succ \phi$ (i.e., $\tau \models_\Sigma \succ \phi$). Similarly, if $\rho \in \Sigma^\omega$ is described by ϕ (i.e., $\rho \models_\Sigma \phi$), and this ρ is composable with $\sigma \in \sigma^\omega$ at every action (i.e., $\sigma \circledcirc^\omega \rho$), then ρ is described by $\circledcirc \phi$ (i.e., $\rho \models_\Sigma \circledcirc \phi$).

As usual, we obtain disjunction ($\phi \vee \psi$), implication ($\phi \to \psi$), "always" ($\Box \phi$) and "eventually" ($\Diamond \phi$) from these connectives. For example, $\Diamond \phi$ is defined as $\top \, U \, \phi$, meaning that, if $\sigma \models_\Sigma \Diamond \phi$, there exists an $n \in \mathbb{N}$ such that $\sigma^{(n)} \models_\Sigma \phi$. The operator $\circledcirc$ has an interesting dual that we shall consider momentarily.

We can extend $\models_\Sigma$ to a relation between SCAs (with underlying c-semiring $\mathbb{E}$ and CAS Σ) and formulas in $\mathcal{L}_\Sigma$, by defining $A \models_\Sigma \phi$ to hold precisely when $\sigma \models_\Sigma \phi$ for all $\sigma \in L(A)$. In general, we can see that fewer properties hold as the threshold t approaches the lowest preference in its semiring, as a consequence of the fact that decreasing the threshold can only introduce new (possibly undesired) behavior. Limiting the behavior of an SCA to some desired behavior described by a formula thus becomes harder as the threshold goes down, since the set of behaviors exhibited by that SCA is typically larger for lower thresholds.

We view the tradeoff between available behavior and verified properties as essential and desirable in the design of robust autonomous systems, because it represents two options available to the designer. On the one hand, she can make a component more accommodating in composition (by lowering the threshold, allowing more behavior) at the cost of possibly losing safety properties. On the other hand, she can restrict behavior such that a desired property is guaranteed, at the cost of possibly making the component less flexible in composition.

Example: no wasted moves Suppose we want to verify that the agent never misses an opportunity to take a snapshot of a new location. This can be expressed by

$$\phi_{\mathsf{w}} = \succ \Box(\mathsf{move} \to X(\neg \mathsf{move} \, U \, \mathsf{snapshot}))$$

This formula reads as "every behavior captures that, at any point, if the current action is a move, then it is followed by a sequence where we do not move until we take a snapshot". Indeed, if $t_e \otimes t_s = 5$, then $A_{\mathsf{e,s}} \models_\Sigma \phi_{\mathsf{w}}$, since in this case every behavior of $A_{\mathsf{e,s}}$ captures that between move-actions we find a $\mathsf{snapshot}$-action. However, if $t_e \otimes t_s = 7$, then $A_{\mathsf{e,s}} \not\models_\Sigma \phi_{\mathsf{w}}$, since $\langle \mathsf{move}_2, \mathsf{move}_2, \mathsf{charge}, \mathsf{charge}, \mathsf{charge}, \mathsf{charge} \rangle^\omega$ would be a behavior of $A_{\mathsf{e,s}}$ that does not satisfy ϕ_{w}, as it contains two successive actions that capture move.[4] This shows the primary use of $\succ$, which is to verify the behavior of a component in terms of the behavior contributed by subcomponents.

[4]Recall that move_2 is the composition of move and $\mathsf{discharge}_2$, i.e., $\mathsf{move} \sqsubseteq \mathsf{move}_2$.

Example: verifying a component interface Another application of the operator $\circledcirc$ is to verify properties of the behavior composable with a component. Suppose we want to know whether all behaviors composable with a behavior of A validate ϕ. Such a property is useful, because it tells us that, in composition, A filters out the behaviors of the other operand that do not satisfy ϕ. Thus, if every behavior that composes with a behavior of A indeed satisfies ϕ, we know something about the behavior *imposed* by A in composition. Perhaps surprisingly, this use can be expressed using the $\circledcirc$-connective, by checking whether $A \models_\Sigma \neg \circledcirc \neg\phi$ holds; for if this is the case, then for all $\sigma, \tau \in \Sigma^\omega$ with σ a behavior of A and $\sigma \circledcirc^\omega \tau$, we know that $\sigma \not\models_\Sigma \circledcirc \neg\phi$, thus in particular $\tau \not\models_\Sigma \neg\phi$ and therefore $\tau \models_\Sigma \phi$.

More concretely, consider the component A_{e}. From its structure, we can tell that the action charge must be executed at least once every five moves. Thus, if τ is composable with a behavior of A_{e}, then τ must also execute some action composable with charge once every five moves. This claim can be encoded by

$$\phi_{\mathsf{c}} = \neg \circledcirc \neg \Box \left(X \circledcirc \mathsf{charge} \vee X^2 \circledcirc \mathsf{charge} \vee \cdots \vee X^5 \circledcirc \mathsf{charge} \right)$$

where X^n denotes repeated application of X. If $A_{\mathsf{e}} \models_\Sigma \phi_{\mathsf{c}}$, then every behavior of A_{e} is incomposable with behavior where, at some point, one of the next five actions is not composable with with charge. Accordingly, if $\sigma \in \Sigma^\omega$ is composable with some behavior of A_{e}, then, at every point in σ, one of the next five actions must be composable with charge. All behaviors that fail to meet this requirement are excluded from the composition.

5.2 Decision procedure

We developed a procedure to decide whether $A \models_\Sigma \phi$ holds for a given SCA A and $\phi \in \mathcal{L}_\Sigma$. The full details of this procedure are given in Appendix A; the main results are summarized below.

Proposition 1. *Let $\phi \in \mathcal{L}_\Sigma$. Given an SCA A and CAS Σ, the question whether $A \models_\Sigma \phi$ is decidable. In case of a negative answer, we obtain a stream $\sigma \in \Sigma^\pi$ such that $\sigma \in L(A)$ but $\sigma \not\models_\Sigma \phi$. The total worst-case complexity is bounded by a stack of exponentials in $|\phi|$, i.e., $2^{\cdot^{\cdot^{\cdot^{|\phi|}}}}$, whose height is the maximal nesting depth of $\succ$ and $\circledcirc$ in ϕ, plus one.*

This complexity is impractical in general, but we suspect that the nesting depth of $\succ$ and $\circledcirc$ is at most two for almost all use cases. We exploit the counterexample in Section 6.

6 Diagnostics

Having developed a logic for SCAs as well as its decision procedure, we investigate how a designer can cope with undesirable behavior exhibited by the agent, either as a run-time behavior σ, or as a counterexample σ to a formula found at design-time (obtained through Proposition 1). The tools outlined here can be used by the designer to determine the right threshold value for a component given the properties that the component (or the system at large) should satisfy.

6.1 Eliminating undesired behavior

A simple way to counteract undesired behavior is to see if the threshold can be raised to eliminate it — possibly at the cost of eliminating other behavior. For instance, in Section 5.1, we saw a formula ϕ_{w} such that $A_{\mathsf{e,s}} \not\models_\Sigma \phi_{\mathsf{w}}$, with counterexample $\sigma = \langle \mathsf{move}_2, \mathsf{move}_2, \mathsf{charge}, \mathsf{charge}, \mathsf{charge}, \mathsf{charge} \rangle^\omega$, when $t_{\mathsf{e}} \otimes t_{\mathsf{s}} = 7$. Since all move_2-labeled transitions of $A_{\mathsf{e,s}}$ have preference 7, raising[5] $t_{\mathsf{e}} \otimes t_{\mathsf{s}}$ to 5 ensures that σ is not present in $L(A_{\mathsf{e,s}})$; indeed, if $t_{\mathsf{e}} \otimes t_{\mathsf{s}} = 5$, then $A_{\mathsf{e,s}} \models_\Sigma \phi_{\mathsf{w}}$. We should be careful not to raise the threshold too much: if $t_{\mathsf{e}} \otimes t_{\mathsf{s}} = 0$, then $L(A_{\mathsf{e,s}}) = \emptyset$, since every behavior of $A_{\mathsf{e,s}}$ includes a transition with a non-zero weight — with threshold $t_e \otimes t_{\mathsf{s}} = 0$, $A_{\mathsf{e,s}} \models_\Sigma \psi$ holds for *any* ψ.

In general, since raising the threshold does not add new behavior, this does not risk adding additional undesired behavior. The only downside to raising the threshold is that it possibly

[5] Recall that $7 \leq_{\mathbb{W}} 5$, so 5 is a "higher" threshold in this context.

eliminates desirable behavior. We define the *diagnostic preference* of a behavior as a tool for finding such a threshold.

Definition 5. Let $A = \langle Q, \Sigma, \mathbb{E}, \to, q^0, t\rangle$ be an SCA, and let $\sigma \in \Sigma^\pi \cup \Sigma^*$. The *diagnostic preference* of σ in A, denoted $d_A(\sigma)$, is calculated as follows:

1. Let Q_0 be $\{q^0\}$, and for $n < |\sigma|$ set $Q_{n+1} = \{q' : q \in Q_n,\ q \xrightarrow{\sigma(n),\ e} q'\}$.
2. Let $\xi \in \mathbb{E}^\pi \cup \mathbb{E}^*$ be the stream such that $\xi(n) = \bigoplus\{e : q \in Q_n,\ q \xrightarrow{\sigma(n),\ e} q'\}$.
3. $d_A(\sigma) = \bigwedge\{\xi(n) : n \leq |\sigma|\}$, with $\bigwedge$ the greatest lower bound operator of $\mathbb{E}$.

Since σ is finite or eventually periodic, and Q is finite, ξ is also finite or eventually periodic. Consequently, $d_A(\sigma)$ is computable.

Lemma 2. *Let $A = \langle Q, \Sigma, \mathbb{E}, \to, q^0, t\rangle$ be an SCA, and let $\sigma \in \Sigma^\pi \cup \Sigma^*$. If $\sigma \in L(A)$, or σ is a finite prefix of some $\tau \in L(A)$, then $t \leq_\mathbb{E} d_A(\sigma)$.*

Proof. If $\sigma \in L(A)$, there exist streams $\mu \in Q^\omega$ and $\nu \in \mathbb{E}^\omega$ such that $\mu(n) = q^0$, and for all $n \in \mathbb{N}$, $t \leq \nu(n)$ and $\mu(n) \xrightarrow{\sigma(n),\ \nu(n)} \mu(n+1)$. It is not hard to see that $\mu(n) \in Q_n$ for $n \in \mathbb{N}$. Then also $t \leq_\mathbb{E} \nu(n) \leq_\mathbb{E} \xi(n)$ for all $n \in \mathbb{N}$. Thus, $t \leq_\mathbb{E} d_A(\sigma)$. Likewise, if σ is a finite prefix of some $\tau \in L(A)$, then $t \leq_\mathbb{E} d_A(\tau)$ by the above, and $d_A(\tau) \leq_\mathbb{E} d_A(\sigma)$ by definition of d_A, thus $t \leq_\mathbb{E} d_A(\sigma)$. □

Since $d_A(\sigma)$ is a necessary upper bound on t when σ is a behavior of A, it follows that we can exclude σ from $L(A)$ if we choose t such that $t \not\leq_\mathbb{E} d_A(\sigma)$. In particular, if we choose t such that $d_A(\sigma) <_\mathbb{E} t$, then $\sigma \notin L(A)$. Note that this may not always be possible: if $d_A(\sigma)$ is $\mathbf{1}$ then such a t does not exist.

Note that there may be another threshold (i.e., not obtained by Lemma 2), which may also eliminate fewer desirable behaviors. Thus, while this lemma gives helps to choose a threshold to exclude some behaviors, it is not a definitive guide. We refer to Appendix B for a concrete example.

6.2 Localizing undesired behavior

One can also use the diagnostic preference to identify the components that are involved in allowing undesired behavior. Let us revisit the first example from Section 5.1, where we verified that every pair of move-actions was separated by at least one snapshot action, as described in ϕ_w. Suppose we choose $t_\mathsf{e} = 10$ and $t_\mathsf{s} = 1$; then $t_\mathsf{e} \otimes t_\mathsf{s} = 11$, thus $\sigma = \langle \mathsf{move}_2, \mathsf{charge}, \mathsf{charge}\rangle^\omega \in L(A_\mathsf{s})$, meaning $A_{\mathsf{e},\mathsf{s}} \not\models_\Sigma \phi_\mathsf{w}$. By Lemma 2, we find that $11 = t_{\mathsf{e},\mathsf{s}} = t_\mathsf{e} \otimes t_\mathsf{s} \leq_\mathbb{W} d_{A_{\mathsf{e},\mathsf{s}}}(\sigma) = 7$. Even if A_s's threshold were as strict as possible (i.e., $t_\mathsf{s} = 0 = \mathbf{1}_\mathbb{W}$), we would find that $t_\mathsf{e} \otimes t_\mathsf{s} \leq_\mathbb{W} d_{A_{\mathsf{e},\mathsf{s}}}(\sigma)$, meaning that we cannot eliminate σ by changing t_s only. In some sense, we could say that t_e is responsible for σ.[6]

More generally, let $(A_i)_{i\in I}$ be a finite family of automata over the c-semiring $\mathbb{E}$ with thresholds $(t_i)_{i\in I}$. Furthermore, let $A = \bowtie_{i\in I} A_i$ and let ψ be such that $A \not\models_\Sigma \psi$, with counterexample behavior σ. Suppose now that for some $J \subseteq I$, we have $\bigotimes_{i\in J} t_i \leq_\mathbb{E} d_A(\sigma)$. Since $\otimes$ is intensive, we furthermore know that $\bigotimes_{i\in I} t_i \leq_\mathbb{E} \bigotimes_{i\in J} t_i$. Therefore, at least one of t_i for $i \in J$ must be adjusted to exclude the behavior σ from the language of $\bowtie_{i\in I} A_i$.

We call $(t_i)_{i\in J}$ *suspect* thresholds: *some* t_i for $i \in I$ must be adjusted to eliminate σ; by extension, we refer to J as a *suspect subset* of I. Note that I may have distinct and disjoint suspect subsets. If $J \subseteq I$ is disjoint from every suspect subset of I, then J is called *innocent*. If J is innocent, changing t_j for some $j \in J$ (or even t_j for all $j \in J$) alone does not exclude σ. Finding suspect and innocent subsets of I thus helps in finding out which thresholds need to change in order to exclude a specific undesired behavior.

Algorithm 1 gives pseudocode to find minimal suspect subsets of a suspect set I; we argue correctness of this algorithm in Theorem 1; for a proof, see [19].

[6] Arguably, A_e as a whole may not be responsible, because modifying the preference of the move-loop on q_N in A_s can help to exclude the undesired behavior as well. In our framework, however, the threshold is a generic property of any SCA, and so we use it as a handle for talking about localizing undesired behaviors to component SCAs.

```
Function FindSuspect (I):
    M := ∅;
    foreach i ∈ I do
        if I \ {i} is suspect then
            M := M ∪ FindSuspect(I \ {i});
        end
    end
    if M = ∅ then
        return {I};
    else
        return M;
    end
end
```

Algorithm 1: Algorithm to find minimal suspect subsets.

Theorem 1. *If I is suspect and $d_A(\sigma) < \mathbf{1}$, then* `FindSuspect`(I) *contains exactly the minimal suspect subsets of I.*

Proof. First, note that it is easy to see that `FindSuspect` never returns $\emptyset$.

The proof proceeds by induction on I. In the base, where $I = \{i\}$, we can see that $\bigotimes \emptyset = \mathbf{1}$, thus, since $d_A(\sigma) < \mathbf{1}$, it follows that $I \setminus \{i\} = \emptyset$ is not suspect. The first branch of the subsequent **if** is selected, which returns $\{I\}$ itself. This matches the fact that I is the only suspect subset of I.

In the inductive step, we assume the claim holds for all strict subsets of I. We consider two cases. On the one hand, if there exists an $i \in I$ such that $I \setminus \{i\}$ is suspect, then we know that the **foreach**-loop will modify M (since `FindSuspect` never returns an empty set). Moreover, I itself is not minimally suspect. The algorithm then returns

$$\bigcup \{\texttt{FindSuspect}(I \setminus \{i\}) : i \in I,\ I \setminus \{i\} \text{ suspect}\}$$

By induction, `FindSuspect`$(I \setminus \{i\})$ returns all minimal suspect subsets of $I \setminus \{i\}$. Since each of these is also a minimal suspect subset of I, and since very minimal suspect subset of I that is not equal to I is contained in one of these, the claim follows by the fact that we ruled out I as a minimal suspect subset. □

In the case where $d_A(\sigma) = \mathbf{1}$, it is easy to see that $\{\{i\} : i \in I\}$ is the set of minimal suspect subsets of I.

In the worst case, every subset of I is suspect, and therefore the only minimal suspect subsets are the singletons; in this scenario, there are $O(|I|!)$ calculations of a composed threshold value. Using memoization to store the minimal suspect subsets of every $J \subseteq I$, the complexity can be reduced to $O(2^{|I|})$.

While this complexity makes the algorithm seem impractical (I need not be a small set), we note that the case where all components are individually responsible for allowing a certain undesired behavior should be exceedingly rare in a system that was designed with the violated concern in mind: it would mean that *every component* contains behavior that ultimately composes into the undesired behavior — in a sense, *facilitating* behavior that counteracts their interest.

7 Discussion

In this paper, we proposed a framework that facilitates the construction of autonomous agents in a compositional fashion. We furthermore considered an LTL-like logic for verification of the constructed models that takes their compositional nature into account, and showed the added value of operators related to composition in verifying properties of the interface between components. We also provided a decision procedure for the proposed logic.

The proposed agents are "soft", in that their actions are given preferences, which may or may not make the action feasible depending on the threshold preference. The designer can *decrease*

this threshold to allow for more behavior, possibly to accommodate the preferences of another component, or *increase* it to restrict undesired behavior observed at run-time or counterexamples to safety assertions found at design-time. We considered a simple method to raise the threshold enough to exclude a given behavior, but which may overapproximate in the presence of partially ordered preferences, possibly excluding desired behavior.

In case of a composed system, one can also find out which component's thresholds can be thought of as *suspect* for allowing a certain behavior. This information can give the designer a hint on how to adjust the system — for example, if the threshold of an energy management component turns out to be suspect for the inclusion of undesired behavior, perhaps the component's threshold needs to be more conservative with regard to energy expenses to avoid the undesired behavior. We stress that responsibility may be assigned to a *set* of components as a whole, if their composed threshold is suspect for allowing the undesired behavior, which is possible when preferences are partially ordered.

8 Further Work

Throughout our investigation, the tools for verification and diagnosis were driven by the compositional nature of the framework. As a result, they apply not only to the "grand composition" of all components of the system, but also to subcomponents (which may themselves be composed of sub-subcomponents). What is missing from this picture is a way to "lift" verified properties of subcomponents to the composed system, possibly with a side condition on the interface between the subcomponent where the property holds and the subcomponent representing the rest of the system, along the lines of the interface verification in Section 5.1.

If we assume that agents have low-latency and noiseless communication channels, one can also think of a multi-agent system as the composition of SCAs that represent each agent. As such, our methods may also apply to verification and diagnosis of multi-agent systems. However, this assumption may not hold. One way to model this could be to insert "glue components" that mediate the communication between agents, by introducing delay or noise. Another method would be to introduce a new form of composition for loosely coupled systems.

Finding an appropriate threshold value also deserves further attention. In particular, a method to adjust the threshold value *at run-time*, would be useful, so as to allow an agent to relax its goals as gracefully as possible if its current goal appears unachievable, and raise the bar when circumstances improve.

Lastly, the use soft constraints for autonomous agents is also being researched in a parallel line of work [31], which employs rewriting logic. Since rewriting logic is backed by powerful tools like Maude, with support for soft constraints [34], we aim to reconcile the automata-based perspective with rewriting logic.

A Decision Procedure

In this appendix, we work out the details of a decision procedure for the logic proposed in Section 5, i.e., a procedure to decide whether $A \models_\Sigma \phi$ holds for a given A and ϕ. This method follows [33], i.e., we do the following:

1. Translate A to a Büchi-automaton A_M with the same language as A.
2. Translate ϕ to a Büchi-automaton A_ϕ that accepts the streams verified by ϕ.
3. Check whether language of A_M is contained in that of A_ϕ.

The last step is an instance of checking ω-regular language containment, which can be decided in $O(2^{|A_\phi|})$, where $|A_\phi|$ is the number of states of A_ϕ [33]. Moreover, in case of a negative answer, this method provides a $\sigma \in \Sigma^\pi$ such that $\sigma \in L(A_M)$ but $\sigma \notin L(A_\phi)$, and therefore $\sigma \in L(A)$ but $\sigma \not\models_\Sigma \phi$.

We give the details for the first two steps step below, but first we briefly recall the details of Büchi-automata.

A.1 Büchi-automata

A *(non-deterministic) Büchi-automaton* [8] (BA) is a tuple $A = \langle Q, \Sigma, \to, q^0, F \rangle$ such that Q is a finite set of *states*, with $q^0 \in Q$ the *initial state* and $F \subseteq Q$ the set of *accepting states*, Σ is a finite set called the *alphabet* and $\to \subseteq Q \times \Sigma \times Q$ is a relation called the *transition relation*. We write $q \xrightarrow{a} q'$ whenever $\langle q, a, q' \rangle \in \to$.

A stream $\lambda \in Q^\omega$ is a *trace* of a stream $\sigma \in \Sigma^\omega$ in A if $\lambda(n) \xrightarrow{\sigma(n)} \lambda(n+1)$ holds for all $n \in \mathbb{N}$. A trace λ is *accepting* if $\lambda(n) \in F$ for infinitely many $n \in \mathbb{N}$. A stream $\sigma \in \Sigma^\omega$ is *accepted* by A if it has an accepting trace λ such that $\lambda(0) = q^0$. The set of streams accepted by A is the *language* of A and denoted by $L(A)$.

An *Alternating Büchi-automaton* (ABA) is a tuple $A = \langle Q, \Sigma, \to, q^0, F \rangle$ such that Q is a finite set of *states* with $q^0 \in Q$ the *initial state* and $F \subseteq Q$ the set of *accepting states*, Σ is a finite set called the *alphabet* and $\to \subseteq Q \times \Sigma \times 2^Q$ is a (finite) relation called the transition relation. Unlike a BA, a single transition in an ABA can have multiple destinations. We write $q \xrightarrow{a} P$ when $\langle q, a, P \rangle \in \to$. A *run* of a stream $\sigma \in \Sigma^\omega$ in A is a labeled tree T such that the root of T is labeled with q^0, and when q is the label of a node of T at depth n and the set of labels of children of said node is P, $q \xrightarrow{\sigma(n)} P$ is a transition of A. A run T is *accepting* if every infinite branch of T is labeled by an accepting state infinitely often.

ABAs accept the same languages as their non-deterministic cousins [23]: given an ABA A, we can construct a BA A' such that $L(A) = L(A')$.

A.2 SCAs to a BAs

The translation of an SCA to a BA is relatively straightforward.

Lemma 3. *Let A be an SCA. We can construct a BA A' such that $L(A) = L(A')$.*

Proof. Choose $A' = \langle Q, \Sigma, \to_t, q^0, Q \rangle$, where $\to_t$ is the relation in which $q \xrightarrow{a}_t q'$ if and only if $q \xrightarrow{a, e} q'$ and $t \leq e$. We can now use the witness streams for $\sigma \in L(A)$ to show that $\sigma \in L(A')$ and vice versa. Indeed, for the inclusion from right to left we can infer the existence of a stream $\nu \in \mathbb{E}^\omega$, while for the inclusion from left to right we simply discard the stream of preferences. □

A.3 Formulas to BAs

We present two methods to translate a formula into a BA that accepts precisely the streams validated by the formula. The first method is an extension of the recursive translation by Sherman et al. [29]. We also propose an extension to the approach of Muller et al. [24], which has a different complexity bound.

A.3.1 Recursive method

One easily constructs BAs that represent atomic formulas $\top$ and σ. Moreover, we can recreate the effect of logical connectives using BAs; for example, we can construct $A_\succ$ such that $\tau \in L(A_\succ)$ if and only if there exists a $\sigma \in L(A)$ with $\sigma \sqsubseteq^\omega \tau$ easily: simply choose $A_\succ = \langle Q, \Sigma, \to_\succ, q^0, F \rangle$ where $q \xrightarrow{b,\, e}_\succ q'$ if and only if $q \xrightarrow{a,\, e} q'$ with $a \sqsubseteq b$. Similar constructions exist for the other connectives, including $\circledcirc$. This is formalized in the following lemma.

Lemma 4. *Let A_1 and A_2 be BAs over alphabet Σ and let $a \in \Sigma$. One can construct BAs A_a, $A_\wedge$, A_U, A_X, $A_\neg$, $A_\succ$ and $A_\circledcirc$ such that the following are true*

(i) $\sigma \in L(A_a)$ *if and only if* $\sigma(0) = a$

(ii) $\sigma \in L(A_\wedge)$ *if and only if* $\sigma \in L(A_1)$ *and* $\sigma \in L(A_2)$ *[10]*

(iii) $\sigma \in L(A_U)$ *if and only if there exists an* $n \in \mathbb{N}$ *such that for all* $k < n$, $\sigma^{(k)} \in L(A_1)$ *and* $\sigma^{(n)} \in L(A_2)$

(iv) $\sigma \in L(A_X)$ *if and only if* $\sigma' \in L(A_1)$

(v) $\sigma \in L(A_\neg)$ *if and only if* $\sigma \notin L(A_1)$ *[8]*

(vi) $\sigma \in L(A_\succ)$ *if and only if* $\tau \sqsubseteq^\omega \sigma$ *for some* $\tau \in L(A_1)$

(vii) $\sigma \in L(A_\circledcirc)$ *if and only if* $\sigma \circledcirc^\omega \tau$ *for some* $\tau \in L(A_1)$

Proof. We treat the claims one by one.

(i) The BA $A = \langle \{q^0, q^1\}, \Sigma, \to, q^0, \{q^1\} \rangle$, where $\to$ is smallest relation such that $q^0 \xrightarrow{a} q^1$ and $q^1 \xrightarrow{b} q^1$ for $b \in \Sigma$, suffices.

(ii) Refer to [33, Proposition 6] for a proof.

(iii) Let $A_i = \langle Q_i, \Sigma, \to_i, q_i^0, F_i \rangle$ for $i \in \{1, 2\}$ and assume (without loss of generality) that Q_1 and Q_2 are disjoint. We choose $Q = Q_1 \cup Q_2 \cup \{q^0\}$ and $F = F_1 \cup F_2$, and let $\to$ be the smallest relation in $Q \times \Sigma \times 2^Q$ satisfying the rules

$$\frac{i \in \{1,2\} \qquad q \xrightarrow{a}_i q'}{q \xrightarrow{a} \{q'\}} \qquad \frac{q_1^0 \xrightarrow{a}_1 q'}{q^0 \xrightarrow{a} \{q', q^0\}} \qquad \frac{q_2^0 \xrightarrow{a}_2 q''}{q^0 \xrightarrow{a} \{q''\}}$$

We choose $A_U = \langle Q, \Sigma, \to, q^0, F \rangle$. It remains to show that A_U validates the claim. If $\sigma \in L(A_U)$, then there is an accepting run T of σ in A_U. Let us refer to the nodes of T labeled with q^0 as *pivot nodes*. For every $n \in \mathbb{N}$, there is at most one pivot node at depth n, since every pivot node has a pivot node as its parent, and at most one pivot node among its children. Furthermore, there are two types of pivot nodes:

- nodes with children labeled by q^0 and some $q' \in Q_1$, called *branch nodes*
- nodes with children labeled by some $q'' \in Q_2$, called *stop nodes*

All children of stop nodes must be labeled with states in Q_2; as a consequence, no stop node has a pivot node in its descendants. This shows that there is at most one stop node in T. Furthermore, if there were no stop nodes in T, then every pivot node would have a branch node among its children, rendering an infinite run of pivot nodes, which contradicts that T is an accepting run. We can thus derive that T has exactly one stop node at some depth n, and that all other pivot nodes in T occur as branch node parents of this stop node. One can then show that if $k < n$, we can construct an accepting run for $\sigma^{(k)}$ in A_1 from the subtree of T rooted at the unique branch node with depth k, and that the tree rooted at the stop node gives us an accepting run of $\sigma^{(n)}$ in A_2.

For the other direction, one can combine the accepting runs of $\sigma^{(k)}$ in A_1 and $\sigma^{(n)}$ in A_2 into an accepting run of σ in A_U easily, by starting with a finite tree consisting of n pivot nodes, and attaching the runs.

(iv) Simply add a new state q to A_1 and make this the initial state, then add a transition from q to the initial state of A_1 labeled with a for every $a \in \Sigma$. It follows that $\sigma \in L(A_X)$ if and only if $\sigma' \in L(A_1)$.

(v) Refer to [30] or [27] for a proof.

(vi) Let $A_1 = \langle Q_1, \Sigma, \to_1, q_1^0, F_1 \rangle$. Choose $A_\succ = \langle Q_1, \Sigma, \to_\succ, q_1^0, F_1 \rangle$, where $q \xrightarrow{b}_\succ q'$ if and only if there exists an $a \in \Sigma$ such that $q \xrightarrow{a}_1 q'$ and $a \sqsubseteq b$. It is easily shown that $A_\succ$ validates the claim.

(vii) By a construction analogous to the above. □

We thus obtain a recursive formula-to-automaton translation; for example, if $\phi = \neg\psi$, we construct the automaton A_ψ representing ψ, from which we obtain the automaton A_ϕ representing ϕ by the aforementioned construction.

Corollary 1. *Let* $\phi \in \mathcal{L}_\Sigma$. *Then one can construct an automaton* A_ϕ *such that* $\sigma \models_\Sigma \phi$ *if and only if* $\sigma \in L(A_\phi)$.

Complexity The construction in Corollary 1 sees a sharp rise in the number of states at each recursion. For instance, in the construction of $A_\wedge$ referenced above, if A_1 and A_2 have n and m states respectively, then $A_\wedge$ has $2nm$ states [10]. The situation is worse for negation; here, we see a necessarily exponential rise in the number of states [27]. Thus the recursive translation ends up with the prohibitively unfeasible upper bound of a stack of exponentials, which is as high as the nesting depth of negations in the formula.

A.3.2 Subformula construction

Another approach [24] translates ϕ to a language-equivalent ABA, where each (possibly negated) subformula is a state. Intuitively, a state representing subformula ψ can be seen as a requirement that the remainder of the satisfies ψ. This approach yields an automaton of size linear in the size of the formula, but the translation from ABA to BA is necessarily exponential [7].

To use this method for our logic, we need to incorporate the connectives $\succ$ and $\circledcirc$, i.e., we need to interlink a state representing a subformula of the form $\succ\psi$ to other states, such that the streams accepted starting in that state are the streams that validate $\succ\psi$. The naïve way to do this is to extend the state space of our output automaton to include subformulas of the input formula ϕ under the $\succ$ connective. For instance, a formula of the form $\phi = \succ(\psi_1 \wedge \psi_2)$ would give states for ϕ, $\psi_1 \wedge \psi_2$, ψ_1, ψ_2, $\succ\psi_1$ and $\succ\psi_2$. If $a \sqsubseteq b$ and the state ψ has a transition of the form $\psi \xrightarrow{a} P$, then we add a transition $\succ\psi \xrightarrow{b} \{\succ\rho : \rho \in P\}$.

Unfortunately, this approach is unsound in general. For example, consider a CAS Σ with distinct actions a, b and c such that $a, b \sqsubseteq c$, and set $\phi = \succ X(a \wedge b)$. Note that there exists no $\sigma \in \Sigma^\omega$ with $\sigma \models_\Sigma \phi$. We now try to translate ϕ to a semantically equivalent ABA using the construction above. By the subformula construction, we find a BA $A_{X(a\wedge b)}$ representing $X(a \wedge b)$, with transitions $\phi \xrightarrow{x} \{a, b\}$ for $x \in \Sigma$, as well as $a \xrightarrow{a} \emptyset$ and $b \xrightarrow{b} \emptyset$, as a subautomaton of A_ϕ. For the remainder of A_ϕ, we have states ϕ, $\succ a$ and $\succ b$ with transitions $\phi \xrightarrow{x} \{\succ a, \succ b\}$ for $x \in \Sigma$, and $\succ a \xrightarrow{a} \emptyset$, $\succ a \xrightarrow{c} \emptyset$ as well as $\succ b \xrightarrow{b} \emptyset$ and $\succ b \xrightarrow{c} \emptyset$ (since $a \sqsubseteq a, c$ and $b \sqsubseteq b, c$). We can now construct a tree with a root labeled by ϕ, and two children labeled by $\succ a$ and $\succ b$ respectively, as a run showing that $\langle c\rangle^\omega \in L(A_\phi)$. A similar pathological case exists for the operator $\circledcirc$.

The essence of the problem above is in the use of ABA, where lifting the construction for $\succ$ in BAs is unsound. Specifically, we have an accepting run for the behavior $\langle c\rangle^\omega$ starting at $\succ X(a \wedge b)$ in the form of a tree T, but this run does not give rise to an accepting run T' starting at $X(a \wedge b)$. In general, if $\succ\phi \xrightarrow{z} \succ\phi'$ and $\succ\phi \xrightarrow{z} \succ\phi''$ then the construction only guarantees that there exist x, y such that $\phi \xrightarrow{x} \phi'$ and $\phi \xrightarrow{y} \phi''$ with $z \sqsubseteq x, y$, while x and y may differ.

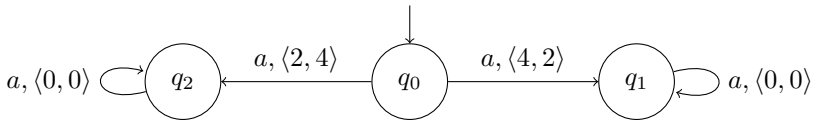
We briefly sketch a method that gets around this problem. Instead of the above, we can use the subformula construction as follows. Given ϕ, find all subformulas of the form $\succ\psi$ or $\circledcirc\chi$ which do not appear below $\succ$ or $\circledcirc$. Recursively construct the ABAs representing A_ψ and A_χ for each of these, and convert them to equivalent BAs A'_ψ and A'_χ, before applying the (sound) conversion to BAs $A_{\succ\psi}$ and $A_{\circledcirc\chi}$ representing $\succ\psi$ and $\circledcirc\chi$ respectively. Now apply the subformula construction to ϕ, except that the states representing $\succ\psi$ and $\circledcirc\chi$ are replaced with the states of $A_{\succ\psi}$ and $A_{\circledcirc\chi}$. The resulting automaton A_ϕ represents ϕ; this can be shown by proving that if q is a state of A_ϕ representing a subformula ρ which is not below a $\succ$ or $\circledcirc$, then the streams accepted at q are precisely the streams that validate ρ; one can do this by induction on the structure of ϕ, with atomic formulas $\top$ and a for $a \in \Sigma$ as well as formulas of the form $\succ\psi$ and $\circledcirc\chi$ as the base.

Complexity Due to the intermittent conversion of ABA to BA in the method outlined above, we can surmise that the complexity is bound from above by a stack of exponentials as high as the nesting depth of $\circledcirc$ and $\succ$, plus one for the final translation of ABA to BA.

B Caveat regarding diagnostic preference

In this appendix, we show that applying the method that arises from Lemma 2 does not always give the lowest threshold that excludes a given behavior.

First, we fix the c-semiring $\mathbb{E}$ as $\mathbb{W} \times \mathbb{W}$, that is: the carrier is $(\mathbb{R} \cup \{\infty\})^2$, $\oplus_\mathbb{E}$ is the pairwise minimum and $\otimes_\mathbb{E}$ is the pairwise (affinely extended) sum, and furthermore $\langle\infty, \infty\rangle$ and $\langle 0, 0\rangle$ are



the minimal, respectively maximal elements. As a result, $\leq_{\mathbb{E}}$ is the product order (i.e., $\langle e_1, e_2\rangle \leq_{\mathbb{E}} \langle e_1', e_2'\rangle$ if and only if $e_1 \geq e_1'$ and $e_2 \geq e_2'$), and $\wedge_{\mathbb{E}}$ is the pairwise maximum.

Furthermore, let A be the SCA depicted below, with CAS $\Sigma = \{a\}$.[7]

Suppose we want to choose t such that $\sigma = \langle a\rangle^{\omega}$ is not in $L(A)$. We calculate:

$$d_A(\sigma) = (\langle 2,4\rangle \oplus_{\mathbb{E}} \langle 4,2\rangle) \wedge_{\mathbb{E}} (\langle 0,0\rangle \oplus_{\mathbb{E}} \langle 0,0\rangle) = \langle 2,2\rangle \wedge_{\mathbb{E}} \langle 0,0\rangle = \langle 2,2\rangle$$

By Lemma 2, we know that if $\sigma \in L(A)$, then $t \leq_{\mathbb{E}} \langle 2,2\rangle$. We can thus choose t such that $t \not\leq_{\mathbb{E}} \langle 2,2\rangle$ in order to exclude σ; for example, $t = \langle 1,1\rangle$ would do. However, we can also choose $t = \langle 3,3\rangle \leq_{\mathbb{E}} \langle 2,2\rangle$. In this case, we find that $\sigma \notin L(A)$ as well. In conclusion, the application of Lemma 2 did not give the lowest threshold that excluded the given behavior.

References

[1] Arbab, F., Santini, F.: Preference and Similarity-Based Behavioral Discovery of Services. In: Proc. Web Services and Formal Methods (WS-FM). pp. 118–133 (2012)

[2] Baier, C., Blechmann, T., Klein, J., Klüppelholz, S., Leister, W.: Design and verification of systems with exogenous coordination using Vereofy. In: Proc. Int. Symp. on Leveraging Applications (ISoLA). pp. 97–111 (2010)

[3] Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling component connectors in Reo by constraint automata. Science of Computer Programming 61, 75–113 (2006)

[4] Bistarelli, S.: Semirings for Soft Constraint Solving and Programming, LNCS, vol. 2962. Springer (2004)

[5] Bistarelli, S., Montanari, U., Rossi, F.: Constraint solving over semirings. In: Proc. Int. Joint Conference on Artificial Intelligence (IJCAI). pp. 624–630 (1995)

[6] Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. J. ACM 44(2), 201–236 (1997)

[7] Boker, U., Kupferman, O., Rosenberg, A.: Alternation removal in büchi automata. In: Proc. Int. Colloquium on Automata, Languages and Programming (ICALP). pp. 76–87 (2010)

[8] Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. Logic, Methodology and Philosophy of Science. pp. 1–11. Stanford Univ. Press, Stanford, Calif. (1962)

[9] Casanova, P., Garlan, D., Schmerl, B.R., Abreu, R.: Diagnosing unobserved components in self-adaptive systems. In: Proc. Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 75–84 (2014)

[10] Choueka, Y.: Theories of automata on ω-tapes: A simplified approach. J. Comput. Syst. Sci. 8(2), 117–141 (1974)

[11] Debouk, R., Lafortune, S., Teneketzis, D.: Coordinated decentralized protocols for failure diagnosis of discrete event systems. Discrete Event Dynamic Systems 10(1-2), 33–86 (2000)

[12] Gadducci, F., Hölzl, M.M., Monreale, G.V., Wirsing, M.: Soft constraints for lexicographic orders. In: Proc. Mexican Int. Conference on Artificial Intelligence, MICAI. pp. 68–79 (2013)

[7] The precise choice of $\circledcirc$ and $\boxdot$ does not matter.

[13] Goessler, G., Astefanoaei, L.: Blaming in component-based real-time systems. In: Proc. Embedded Software (EMSOFT). pp. 7:1–7:10 (2014)

[14] Gößler, G., Stefani, J.: Fault ascription in concurrent systems. In: Proc. Trustworthy Global Computing (TGC). pp. 79–94 (2015)

[15] Hölzl, M.M., Meier, M., Wirsing, M.: Which soft constraints do you prefer? Electr. Notes Theor. Comput. Sci. 238(3), 189–205 (2009)

[16] Hüttel, H., Larsen, K.G.: The use of static constructs in a modal process logic. In: Proc. Symp. on Logical Foundations of Computer Science. pp. 163–180 (1989)

[17] Jongmans, S.T., Kappé, T., Arbab, F.: Constraint automata with memory cells and their composition. Sci. Comput. Program. 146, 50–86 (2017)

[18] Kappé, T.: Logic for Soft Component Automata. Master's thesis, Leiden University, Leiden, The Netherlands (2016), `http://liacs.leidenuniv.nl/assets/Masterscripties/CS-studiejaar-2015-2016/Tobias-Kappe.pdf`

[19] Kappé, T., Arbab, F., Talcott, C.: A component-oriented framework for autonomous agents (FM-TODO) (May 2017), `TODO`

[20] Kappé, T., Arbab, F., Talcott, C.L.: A compositional framework for preference-aware agents. In: Proc. Workshop on Verification and Validation of Cyber-Physical Systems (V2CPS). pp. 21–35 (2016)

[21] Koehler, C., Clarke, D.: Decomposing port automata. In: Proc. ACM Symp. on Applied Computing (SAC). pp. 1369–1373 (2009)

[22] Mason, I.A., Nigam, V., Talcott, C., Brito, A.: A framework for analyzing adaptive autonomous aerial vehicles. In: Proc. Workshop on Formal Co-Simulation of Cyber-Physical Systems (CoSim) (2017)

[23] Miyano, S., Hayashi, T.: Alternating finite automata on omega-words. Theor. Comput. Sci. 32, 321–330 (1984)

[24] Muller, D.E., Saoudi, A., Schupp, P.E.: Weak Alternating Automata Give a Simple Explanation of Why Most Temporal and Dynamic Logics are Decidable in Exponential Time. In: Proc. Symp. on Logic in Computer Science (LICS). pp. 422–427 (1988)

[25] Neidig, J., Lunze, J.: Decentralised diagnosis of automata networks. IFAC Proceedings Volumes 38(1), 400–405 (2005)

[26] Rutten, J.J.M.M.: A coinductive calculus of streams. Mathematical Structures in Computer Science 15(1), 93–147 (2005)

[27] Safra, S.: On the complexity of omega-automata. In: Proc. Foundations of Computer Science. pp. 319–327 (1988)

[28] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Failure diagnosis using discrete-event models. IEEE Trans. Contr. Sys. Techn. 4(2), 105–124 (1996)

[29] Sherman, R., Pnueli, A., Harel, D.: Is the interesting part of process logic uninteresting? A translation from PL to PDL. SIAM J. Comput. 13(4), 825–839 (1984)

[30] Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for büchi automata with applications to temporal logic (extended abstract). In: Proc. Automata, Languages and Programming. pp. 465–474 (1985)

[31] Talcott, C.L., Arbab, F., Yadav, M.: 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 (2015)

[32] Talcott, C.L., Nigam, V., Arbab, F., Kappé, T.: Formal specification and analysis of robust adaptive distributed cyber-physical systems. In: Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems - Int. School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016. pp. 1–35 (2016)

[33] Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Proc. Logics for Concurrency - Structure versus Automata (Banff Higher Order Workshop). pp. 238–266 (1995)

[34] Wirsing, M., Denker, G., Talcott, C.L., Poggio, A., Briesemeister, L.: A rewriting logic framework for soft constraints. Electr. Notes Theor. Comput. Sci. 176(4), 181–197 (2007)