Symbolic Limited Lookahead Control for Best-effort Dynamic Computing Resource Management

Nicolas Berthier, Hervé Marchand, Eric Rutten

Symbolic Limited Lookahead Control for Best-effort Dynamic Computing Resource Management ⋆

Nicolas Berthier * **Hervé Marchand** ** **Éric Rutten** ***

* *Department of Computer Science, University of Liverpool, UK*
** *INRIA Rennes - Bretagne Atlantique, France*
*** *Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble France*

Abstract: We put forward a new modeling technique for Dynamic Resource Management (DRM) based on discrete events control for symbolic logico-numerical systems, especially Discrete Controller Synthesis (DCS). The resulting models involve state and input variables defined on an infinite domain (Integers), thereby no exact DCS algorithm exists for safety control. We thus formally define the notion of limited lookahead, and associated best-effort control objectives targeting safety and optimization on a sliding window for a number of steps ahead. We give symbolic algorithms, illustrate our approach on an example model for DRM, and report on performance results based on an implementation in our tool ReaX.

Keywords: Discrete Event Systems, Supervisory Control, Infinite-state Systems, Dynamic Resource Management

1. INTRODUCTION & MOTIVATING EXAMPLE

When dealing with automated resource management or scheduling problems, designers of *self-adaptive computing systems* often face the need to ensure some quality of service while meeting intrinsic or application-dependent constraints. Following our previous works on the subject [Berthier et al., 2015, 2016; An et al., 2016], our approach relieves the designers from this error-prone task by delegating some management decisions to a *controlled model* of the system issuing said decisions at *discrete points in time* (also called *execution steps*). The controlled model consists of a given *model* that keeps track of the necessary knowledge about the resources at hand, and an associated *controller*, automatically computed by means of a *Discrete Controller Synthesis* (DCS) algorithm, and whose goal is to enforce given *control objectives*. The controlled model is then paired with other pieces of software to form a *management system*. Yet, the amount of resources to manage may vary during the system's lifetime, yielding the need to handle quantitative, potentially *infinite*, models.

Dynamic Computing Resource Management Observe that, while computing systems generally comprise a *dynamic number of resources*, the latter can often be categorized into a finite number of groups that are of the same kind, sharing some regularity in terms of their status and operations. We say that resources of the same kind belong to the same *family*. Such families may be the threads in a pool, request handlers, computing resources like CPU cores, communication channels, or some type of objects in memory for instance.

⋆ This work was partially supported by the EPSRC through grant EP/M027287/1. Authors' version.

Fig. 1. Example resource automaton for request handlers. Dashed arrows -› denote controllable transitions; ⊢ (resp. ⇥|) denote resource appearance (resp. disposal).

A family of resources can be described using a single *Resource Automaton* (RA), similar to a finite-state automaton where states represent potential statuses of a single one of the resources, and transitions model events (*i.e.,* status change). Transitions that are said *controllable* feature management decisions (*e.g.,* creating or deleting threads in a pool, powering off a CPU core, resuming or suspending a task). The other events are considered *non-controllable* (*e.g.,* end of a task, failure of a CPU core or communication channel).
We depict in Fig. 1 a simple example of such an RA for request handlers, that we use as a basis for the examples throughout the paper. Once it has been created (c — *i.e.,* triggered by the arrival of a request), a handler must be started (s) to be able to process the request; it does so after first entering a waiting stage (W) and then being resumed (r); it can then yield its computation (y) and then be resumed (r) any number of time. A handler disappears from the managed system once it ends its computation (e). Start-up (s) and resume (r) must be triggered by some decision of the management system.

Quantitative Modeling We suggest a quantitative modeling approach to deal with the dynamic aspect of the resources in each family. Instead of modeling each resource instance individually using as many finite-state automata, we represent each state S in which a kind of resource can be, as the quantity s of resources being in S at any time. Further,

discrete changes in resource status manifest as variations between these quantities at discrete instants, and management decisions can be deduced from the variations for controllable transitions, chosen by a synthesized controller. The RA of Fig. 1 features two controllable transitions: at each one of its execution steps, the management system decides how many handlers should be started (s) or resumed (r) based on the other variations it receives as input (c, y, and e). Such modeling results in *linear systems*, for which we give a symbolic representation in Section 2.

Resource Management Goals as Control Objectives Using our modeling approach, interesting quantitative control objectives about one or more family of resources can be specified. Some management concerns relate to the performance of the overall system, *e.g.,* in terms of power or energy consumption. Such goals can be expressed as *optimization* objectives, where the role of the controlled system is to minimize some quantity exhibited by the model, such as the number of non-idle CPU cores at any time.
Useful qualitative properties can also be specified as *safety* objectives on the model. For instance, limiting the quantity of resources of a family into a given state expresses some guarantee of service constraints; *e.g.,* stating that the number of threads in a pool does not exceed the (dynamic) number of CPU cores available in the system.

Towards Best-effort Control The DCS algorithms currently available [Berthier and Marchand, 2014, 2015] for logico-numerical systems focus on safety control. Due to the necessary over-approximations that these algorithms involve to terminate and give conservative results, the degree of freedom (*i.e.,* the domain of controllable variables) given to the synthesized controller must be finite; in the case of symbolic systems, this freedom usually corresponds to the domain of some special input variables, said *controllable*.
In some contexts though, strictly considering a safety control objective might be considered too constraining in terms of the controlled systems' behaviors, or computational power involved in enforcing the objective. Hence, in Section 3 we first formally develop the notion of *symbolic limited lookahead*, inspired by the work of Chung et al. [1992]. We then detail *exact* algorithms for *best-effort bounded safety control* (as opposed to the usual *strict* safety control), accompanied with a transformation of controllers towards *recovery*, where the controlled system still does its best to avoid operating states violating a desired invariant when it reaches states from which it might fail in doing so. Algorithms for *optimization* are also developed, based on the same principles as bounded safety. Sections 4, 5 and 6 give some implementation details and performance results, review related works, and conclude.

2. CONTROL OF SYMBOLIC TRANSITION SYSTEMS

We first fix some notations, and recall in this section the model of Arithmetic Symbolic Transition Systems [Berthier and Marchand, 2014] and the associated control problems.

2.1 Notations

Let $V = \langle v_1, \ldots, v_n \colon \mathcal{D}_V \rangle$, with $\mathcal{D}_V = \prod_{i \in \{1,\ldots,n\}} \mathcal{D}_{v_i}$, be a vector (or tuple) of variables $v_1, \ldots, v_n$, each defined on (infinite) domains $\mathcal{D}_{v_1}, \ldots, \mathcal{D}_{v_n}$. $V \cap W$ is the result of removing from a vector V all variables not belonging to a vector W, and $\varnothing$ is the empty vector; $V \uplus W$ is the concatenation of V and W, defined iff they contain distinct sets of variables ($V \cap W = \varnothing$).

A *valuation* $\nu \in \mathcal{D}_V$ for each variable in V can be seen as the mapping $\nu \colon V \to \mathcal{D}_V$. We denote valuations using tuple notations as in $\nu = (\mathsf{ff}, \ldots, \mathsf{tt})$, to actually denote $\nu = \{v_1 \mapsto \mathsf{ff}, \ldots, v_n \mapsto \mathsf{tt}\}$. Further, given an additional vector W of variables distinct from those in V, and corresponding valuations $\nu \in \mathcal{D}_V$ and $\mu \in \mathcal{D}_W$, the union of ν and μ is $(\nu, \mu) \in \mathcal{D}_{V \uplus W}$. Considering two sequences of $n \in \mathbb{N}^+$ valuations $\boldsymbol{\nu} = a_1 \cdot \ldots \cdot a_n \in \mathcal{D}_V^n$ and $\boldsymbol{\mu} = b_1 \cdot \ldots \cdot b_n \in \mathcal{D}_W^n$, we denote their union by $\boldsymbol{\nu} \cup \boldsymbol{\mu} = (a_1, b_1) \cdot \ldots \cdot (a_n, b_n) \in \mathcal{D}_{V \uplus W}^n$. The empty sequence is noted ε.

Given a vector V_x of variables defined on numerical domains (*e.g.,* Integers or Reals), we denote $\mathcal{L}_{V_x}$ the set of all *linear affine arithmetic expressions* involving variables of V_x only. Given an additional vector V_b of variables defined over finite domains (*e.g.,* Booleans), $\mathcal{P}_{V_b \uplus V_x}$ denotes the set of all *propositional predicates* defined over variables in V_b and a finite set of *linear (in-)equalities* expressed on variables of V_x. Further, $\mathcal{L}^{\mathcal{P}}_{V_b \uplus V_x}$ denotes the set of all *guarded arithmetic expressions*, that are total functions mapping pairwise disjoint predicates of $\mathcal{P}_{V_b \uplus V_x}$ into a finite subset of $\mathcal{L}_{V_x}$.

Note that all numerical expressions involved in predicates and arithmetic expressions are linear, hence any $\mathcal{P}_V$ is closed under quantifier elimination—*i.e.,* any predicate involving existential or universal quantifiers from $\mathcal{P}_V$ is equivalent to a quantifier-free predicate also in $\mathcal{P}_V$. The process of computing such an equivalent quantifier-free predicate is called *quantifier elimination*. For succinctness, we denote in the sequel the set of quantifier-free expressions over variables V as $\mathcal{E}_V = \mathcal{P}_V \cup \mathcal{L}^{\mathcal{P}}_V$.
Given $e \in \mathcal{E}_V$ and a valuation $\nu \in \mathcal{D}_V$, $[\![e]\!]\nu$ is the Boolean or numerical value obtained through the usual evaluation of e after substituting every variable in e with its corresponding value in ν. For a predicate $e \in \mathcal{P}_V$, one writes $\nu \models e$ to denote that ν satisfies e, and thus $[\![e]\!]\nu \equiv (\nu \models e)$.

2.2 Arithmetic Symbolic Transition Systems

An Arithmetic Symbolic Transition System (ASTS) comprises a finite set of (state and input) variables, whose domain can be infinite, and evolves at discrete points in time. An update function indicates the new values for each state variable according to the current values of the state and input variables. This model allows the representation of infinite systems whenever the variables take their values in an infinite domain, while it has a finite structure and offers a compact way to specify systems handling data.

Definition 1. An *Arithmetic Symbolic Transition System* is a tuple $S = \langle X, I, T, A, \Theta_0 \rangle$ where:

- $X = \langle x_1, \ldots, x_n \colon \mathcal{D}_X \rangle$ is a vector of state variables encoding the memory necessary for describing the system's behavior;
- $I = \langle i_1, \ldots, i_m \colon \mathcal{D}_I \rangle$ is a vector of input variables;
- $T \in \mathcal{E}^n_{X \uplus I}$ is the transition function of S, and encodes the evolution of all state variables based on n (well-typed) expressions involving variables in $X \uplus I$;

- $A \in \mathcal{P}_{X \uplus I}$ encodes an assertion on the admissible values of the inputs depending on the current state;
- $\Theta_0 \in \mathcal{P}_X$ is a predicate encoding the initial states.

An ASTS is *linear logico-numerical* if its state and input variables are Boolean variables ($\mathbb{B}$) or numerical variables (typically, $\mathbb{Z}$ or $\mathbb{Q}$), *i.e.,* it is such that $\mathcal{D}_X = \mathbb{B}^k \times \mathbb{Q}^{k'} \times \mathbb{Z}^{k''}$ with $k + k' + k'' = n$ (and similarly for I).

Consider an ASTS $S = \langle X, I, T, A, \Theta_0 \rangle$. The state reached by S starting from any state $x \in \mathcal{D}_X$ with a sequence of m valuations for its input variables $\boldsymbol{\iota} \in \mathcal{D}_I^m$ is denoted by $[S](x, \boldsymbol{\iota}) \in \mathcal{D}_X \cup \{\bot\}$, defined as

$$[S](x, \varepsilon) \stackrel{\text{def}}{=} x$$
$$[S](x, \iota \cdot \boldsymbol{\iota}) \stackrel{\text{def}}{=} [S](T(x, \iota), \boldsymbol{\iota}) \text{ if } (x, \iota) \models A, \bot \text{ otherwise.} \quad (1)$$

$[S](x, \iota) = \bot$ indicates that the valuation of inputs ι does not satisfy the assertion A, and thus S cannot evolve from state x with input ι: we say that ι is not *admissible* (by S) in x. We also write $[S]^i(x) \subseteq \mathcal{D}_X$ the set of states reachable by S from state x whatever the sequence of $i \in \mathbb{N}$ inputs; *i.e.,* $[S]^i(x) \stackrel{\text{def}}{=} \bigcup_{\boldsymbol{\iota} \in \mathcal{D}_I^i} [S](x, \boldsymbol{\iota}) \setminus \{\bot\}$. Further, the *orbit* $\mathcal{O}(S) \subseteq \mathcal{D}_X$ of S (also known as its *reachable state space*) is $\mathcal{O}(S) \stackrel{\text{def}}{=} \bigcup_{(n, x_0) \in \mathbb{N} \times \Theta_0} [S]^n(x_0)$. Note that the orbit might not be computable due to the infiniteness of the state space.

Example 1. Applying the quantitative modeling principles mentioned in the Introduction on the RA described in Fig. 1 provides the ASTS $S_{\text{rh}} = \langle X, I, T, A, \Theta_0 \rangle$ *where each integer state variable* i, w *and* a *represents the number of handlers in states* I, W *and* A*:* $X = \langle \mathsf{i}, \mathsf{w}, \mathsf{a} \colon \mathbb{Z}^3 \rangle$, $I = \langle c, s, r, y, e \colon \mathbb{Z}^5 \rangle$, $\Theta_0(X) = (\mathsf{i} = 0 \wedge \mathsf{w} = 0 \wedge \mathsf{a} = 0)$,

$$T(X, I) = \begin{cases} \mathsf{i} := \ \mathsf{i} - s + c \\ \mathsf{w} := \mathsf{w} + s + y - r \\ \mathsf{a} := \ \mathsf{a} + r - y - e \end{cases}, \text{ and}$$
$$A(X, I) = s \geqslant 0 \wedge r \geqslant 0 \wedge y \geqslant 0 \wedge c \geqslant 0 \wedge e \geqslant 0 \wedge s \leqslant \mathsf{i} \wedge r \leqslant \mathsf{w} \wedge y + e \leqslant \mathsf{a}.$$

The assertion A restricts the domain of admissible inputs and quantities associated with states to natural numbers. Further assumptions on the inputs can be represented in a similar way, e.g., with a conjunction with $c \leqslant 10$ if no more than 10 new handlers can be created between two successive execution steps. We show in Example 2 below how to tune S_{rh} to take controllable transitions into account.
From the (here single) state $x_0 = (0, 0, 0)$ (i.e., such that $\Theta_0(x_0)$), one execution step of S_{rh} with input vector $\iota_1 = (3, 0, 0, 0, 0)$ leads to $x_1 = [S_{\text{rh}}](x_0, \iota_1) = (3, 0, 0)$. One further step with $\iota_2 = (1, 2, 0, 0, 0)$ reaches $x_2 = [S_{\text{rh}}](x_1, \iota_2) = (2, 2, 0)$. With $\iota_3 = (0, 0, 3, 0, 0)$, $[S_{\text{rh}}](x_2, \iota_3) = \bot$ as $(x_2, \iota_3) \models A$ does not hold (due to the falsified condition $r \leqslant \mathsf{w} \equiv 3 \leqslant 2$). A longer, admissible execution trace of S_{rh} based on randomly generated inputs satisfying A, is shown in Fig. 2. ◁

2.3 Control of ASTSs

A *controllable ASTS* is an ASTS whose vector of input variables I is partitioned into *non-controllable* U and *controllable* variables C (*i.e.,* $I = U \uplus C$).

Example 2. Reflecting the controllable transitions of the RA in Fig. 1 on S_{rh} of Example 1 brings the controllable

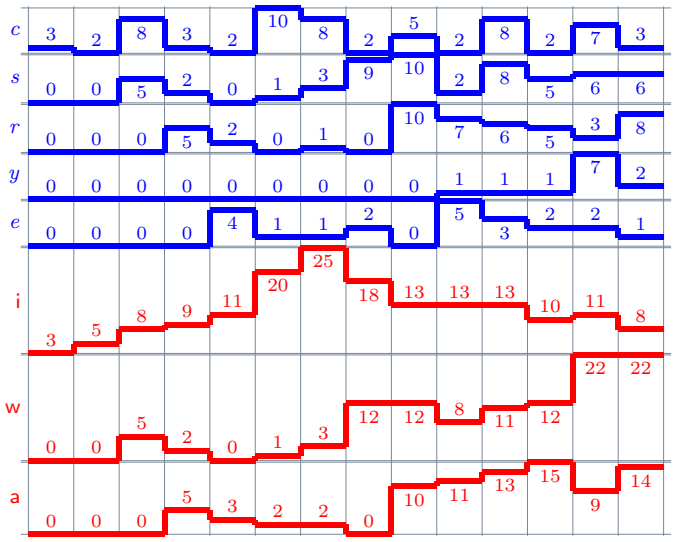

Fig. 2. An execution trace of S_{rh}. Values for variables in I are randomly picked at each step so that A is satisfied. Each column shows the input values at step i and the *new values* (for step $i + 1$) for the state variables; the latter are all null initially.

ASTS $S'_{\text{rh}} = \langle X, U \uplus C, T, A, \Theta_o \rangle$ where $U = \langle c, y, e \colon \mathbb{Z}^3 \rangle$ and $C = \langle s, r \colon \mathbb{Z}^2 \rangle$. Notice $\mathcal{D}_C$ is infinite. ◁

Reactive ASTSs Raw ASTSs shall be *reactive*, meaning that there always exist admissible valuations for their input variables, whatever the current (reachable) state:

Definition 2. An ASTS S is *reactive* iff

$$\forall x \in \mathcal{O}(S), \exists \iota \in \mathcal{D}_I, [S](x, \iota) \neq \bot. \quad (2)$$

This notion is very similar to the usual *deadlock-freeness*, except that we choose to reserve the latter term to controlled ASTSs only; we give a definition for such deadlock-free ASTSs below.

Controlled ASTSs To fulfill its objective, a controller restricts the admissible values for the controllable subset of the input variables of the ASTS it controls.

Definition 3. Given an ASTS $S = \langle X, I, T, A, \Theta_0 \rangle$ with inputs $I = U \uplus C$, a *controller* $K \in \mathcal{P}_{X \uplus I}$ provides a *controlled ASTS* $S/_K = \langle X, I, T, K, \Theta_0 \rangle$ such that $\forall (x, u, c) \in \mathcal{O}(S/_K) \times \mathcal{D}_U \times \mathcal{D}_C$,

$$[S/_K](x, u, c) \neq \bot \Rightarrow [S](x, u, c) \neq \bot. \quad (3)$$

Eq. (3) above states that K is at least as restrictive as A, hence does not allow evolutions that are forbidden in the original system; *i.e.,* $\mathcal{O}(S/_K) \subseteq \mathcal{O}(S)$, and

$$\forall (x, \iota) \in \mathcal{O}(S/_K) \times \mathcal{D}_I, ((x, \iota) \models K \Rightarrow (x, \iota) \models A).$$

In the remainder of the paper, we devise algorithms taking as input a controllable ASTS $S = \langle X, I, T, A, \Theta_0 \rangle$ and some target objective o, and computing a *controller* K for S so that a resulting controlled ASTS $S/_K$ fulfills o.

Deadlock-free ASTSs As our aim is to eventually obtain controlled systems that are executable, we devise algorithms producing *deadlock-free controlled ASTSs*, *i.e.,* for which values for controllable variables always exist whatever the non-controllable inputs admissible by S:

Definition 4. A controlled ASTS $S/_K$ is *deadlock-free* if S is reactive and $\forall (x,u,c) \in \mathcal{O}(S/_K) \times \mathcal{D}_U \times \mathcal{D}_C$,

$$[S](x,u,c) \neq \bot \Rightarrow \exists c' \in \mathcal{D}_C, [S/_K](x,u,c') \neq \bot. \quad (4)$$

3. SYMBOLIC κ-LOOKAHEAD CONTROL

Let us consider given in this Section a controllable ASTS $S = \langle X, U \uplus C, T, A, \Theta_0 \rangle$ and a strictly positive Integer $\kappa \in \mathbb{N}^+$. We first explain the core computations of our algorithms for κ-lookahead control in an explicit way—*i.e.,* using set-theoretic notations—and turn to the related symbolic representations and algorithms. We then focus on the use of such tooling for bounded-safety, best-effort, recovery, and optimization control.

3.1 κ-Lookahead Control

Potential & Admissible i-paths: $\mathcal{R}_i$ & $\mathcal{A}_i$ As ASTSs' transition functions are deterministic by construction, a set of paths starting in a state $x \in \mathcal{D}_X$ and of a finite number i of transitions, can be identified by relating x and a sequence of i valuations for every variable in I:

Definition 5. A (potentially infinite) set of *i-paths* $\mathcal{R}_i \subseteq \mathcal{D}_X \times \mathcal{D}_I^i$, for $i \in \mathbb{N}$, relates states of S with sequences of i successive valuations for its inputs.

Note that we define i-paths based on the transition functions only, *i.e.,* both admissible and non-admissible transitions are considered. To take transitions admissibility into account, we further define the sets $\mathcal{A}_i \stackrel{\text{def}}{=} \left\{(x, \boldsymbol{\iota} \cdot \iota) \in \mathcal{D}_X \times \mathcal{D}_I^i \middle| [S](x,\boldsymbol{\iota}) \neq \bot \Rightarrow [S](x, \boldsymbol{\iota}\cdot\iota) \neq \bot\right\}$, for $i \in \mathbb{N}^+$, that each are the sets of all i-paths whose ith transitions (if any) are admissible by S.

We can now assume given a set of *desirable* κ-paths $\mathcal{R}_\kappa$ for S; we further detail the computation of such sets in Sections 3.4 and 3.5.

Controllable Prefixes of $\mathcal{R}_\kappa$ Given a set $\mathcal{R}_{i+1}$ of $(i+1)$-paths, the set $\mathcal{R}_i$ of all its *direct controllable prefixes* consists of the i-paths whose suffix transitions admissible by S cannot uncontrollably form an $(i+1)$-path that does not belong to $\mathcal{R}_{i+1}$; *i.e.,* after following any i-path in $\mathcal{R}_i$, and given an admissible valuation for all non-controllable variables (U), one can always find a valuation for the controllable variables (C) so that the system remains on an $(i+1)$-path belonging to $\mathcal{R}_{i+1}$.
Formally, $\mathcal{R}_\kappa$ being given, we define $\mathcal{R}_i$ based on $\mathcal{R}_{i+1}$ as

$$\mathcal{R}_i \stackrel{\text{def}}{=} \mathit{Prefix}_{\mathrm{c}}(\mathcal{R}_{i+1}) \qquad (i \in \{0,\dots,\kappa-1\}) \quad (5)$$

where $\mathit{Prefix}_{\mathrm{c}} \colon \mathcal{D}_X \times \mathcal{D}_I^{i+1} \to \mathcal{D}_X \times \mathcal{D}_I^i \stackrel{\text{def}}{=}$

$$\mathcal{R}_{i+1} \mapsto \left\{ (x,\boldsymbol{\iota}) \in \mathcal{D}_X \times \mathcal{D}_I^i \;\middle|\; \begin{array}{l} \forall u \in \mathcal{D}_U, \\ (\nexists c \in \mathcal{D}_C, (x, \boldsymbol{\iota}\cdot(u,c)) \in \mathcal{A}_{i+1}) \cup \\ (\exists c \in \mathcal{D}_C, (x, \boldsymbol{\iota}\cdot(u,c)) \in \mathcal{R}'_{i+1}) \end{array} \right\}$$

with $\mathcal{R}'_{i+1} \stackrel{\text{def}}{=} \mathcal{R}_{i+1} \cap \mathcal{A}_{i+1}$. The left-hand side of the union in the definition of $\mathit{Prefix}_{\mathrm{c}}$ consists in prefixes of $(i+1)$-paths that are not admissible by S whatever the valuation of the controllable variables at step $i+1$. We illustrate this principle in Fig. 3 in which, contrary to i-paths denoted by c, those denoted by a are included into $\mathcal{R}_i$ as there always exists a choice for valuations c to stay on an $(i+1)$-path belonging to $\mathcal{R}_{i+1}$ whatever u. i-paths b belong to

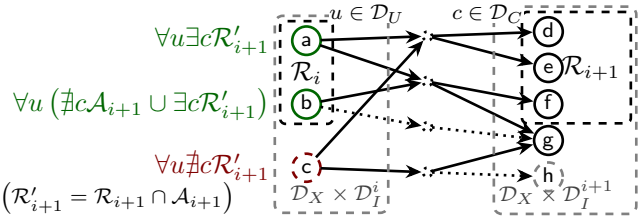

Fig. 3. Direct controllable prefixes $\mathcal{R}_i$ of $\mathcal{R}_{i+1}$. Plain circles ○ represent sets of i-paths (resp. $(i+1)$-paths) belonging to $\mathcal{R}_i$ (resp. $\mathcal{R}_{i+1}$); dashed circles ◌ represent i-paths (resp. $(i+1)$-paths) *not* belonging to $\mathcal{R}_i$ (resp. $\mathcal{R}_{i+1}$); smaller circles ∴ denote transient states, after input values u for the i-th non-controllable variables are given and before the subsequent controllable ones c are chosen; plain ➔ (resp. dotted ⇢) arrows denote sets of transitions satisfying admissible (resp. non-admissible) by S.

$\mathcal{R}_i$ as from them valuations u always leading to g are not admissible at step $i+1$.

Eq. (5) gives an inductive definition for controllable prefixes of any non-negative length $i < \kappa$ of $\mathcal{R}_\kappa$. We now write $\mathcal{R}_i$ to denote all i-paths that are controllable prefixes of $\mathcal{R}_\kappa$.

3.2 Symbolic Computations

In order to symbolically represent valuations of input variables in κ subsequent future steps, we define indexed versions of all input variables of S. From the vector U (resp. C) we define $\kappa - 1$ additional indexed versions $U_2 \dots U_\kappa$ (resp. $C_2 \dots C_\kappa$), and set $U_1 = U$ and $C_1 = C$. We additionally define $I_i \stackrel{\text{def}}{=} U_i \uplus C_i$ for $i \in \{1,\dots,\kappa\}$, as well as $V_{i\rightsquigarrow j} \stackrel{\text{def}}{=} \biguplus_{k\in\{i,\dots,j\}} V_k$ for any vector of indexed variables $V \in \{U, C, I\}$, with $V_{i\rightsquigarrow j} \stackrel{\text{def}}{=} \varnothing$ if $i > j$. Lastly, given two vectors of variables of equal lengths V and W, and any expression $e \in \mathcal{E}_{V\uplus Z}$, $e[V/W] \in \mathcal{E}_{W\uplus Z}$ denotes the substitution in e of all occurrences of any variable belonging to V with its counterpart in W.

Using the above notations, the symbolic representation of a set of i-paths (for $i \in \{0,\dots,\kappa\}$) is a predicate in $\mathcal{P}_{X\uplus I_{1\rightsquigarrow i}}$. More generally, the *i-lookahead* of $e \in \mathcal{E}_X$ is the expression $e|_i \in \mathcal{E}_{X\uplus I_{1\rightsquigarrow i}}$ encoding the value of e after i successive applications of its transition function T, in terms of the current state (variables from X) and any future input sequence of length i (variables from $I_{1\rightsquigarrow i}$): given $i \in \{0,\dots,\kappa\}$, $e|_i$ is such that $\forall (x,\boldsymbol{\iota}) \in \mathcal{D}_X \times \mathcal{D}_{I_{1\rightsquigarrow i}}$,

$$([S](x,\boldsymbol{\iota}) \neq \bot) \Rightarrow ([\![e|_i]\!](x,\boldsymbol{\iota}) = [\![e]\!][S](x,\boldsymbol{\iota})). \quad (6)$$

Let us now turn to the actual construction of such i-lookahead expressions. Assume given an expression $e \in \mathcal{E}_V$ with $V \cap I = \varnothing$. $T^{-1}(e)$ is the (parallel) substitution in e of each state variable in X with its respective expression in T; *i.e.,* $\forall (x,\iota,y) \in \mathcal{D}_X \times \mathcal{D}_I \times \mathcal{D}_{V\setminus(X\uplus I)}$,

$$[\![T^{-1}(e)]\!](x,\iota,y) \stackrel{\text{def}}{=} [\![e]\!](T(x,\iota),y). \quad (7)$$

Proposition 6. $e|_i$ can be recursively built as

$$e|_{i+1} = T^{-1}(e|_i[I_{1\rightsquigarrow i}/I_{2\rightsquigarrow i+1}]), \text{ with } e|_0 = e.$$

Proof. Beyond the base case for $i = 0$ which is obvious, we show that $e|_{i+1}$ is properly built if $e|_i$ is. Given

any $(x, \iota \cdot \boldsymbol{\iota}) \in \mathcal{D}_X \times \mathcal{D}_I^{i+1}$, our recursion hypothesis is $[\![e|_i]\!](x, \boldsymbol{\iota}) = [\![e]\!][S](x, \boldsymbol{\iota})$ (HR). After substituting $e|_{i+1}$ by $T^{-1}(e|_i[I_{1\rightsquigarrow i}/I_{2\rightsquigarrow i+1}])$ and $\boldsymbol{\iota}$ by $\iota \cdot \boldsymbol{\iota}$ in Eq. (6), we assume $[S](x, \iota \cdot \boldsymbol{\iota}) \neq \bot$ (H1) and check the equality

$$
\begin{aligned}
&[\![T^{-1}(e|_i[I_{1\rightsquigarrow i}/I_{2\rightsquigarrow i+1}])]\!](x, \iota \cdot \boldsymbol{\iota}) && (\stackrel{?}{=} [\![e]\!][S](x, \iota \cdot \boldsymbol{\iota})).\\
&= [\![T^{-1}(e|_i)]\!](x, \iota, \boldsymbol{\iota}) && \text{(Def. of } T^{-1} \text{ \& } I \cap I_{2\rightsquigarrow i+1} = \varnothing)\\
&= [\![e|_i]\!](T(x, \iota), \boldsymbol{\iota}) && \text{(by Eq. (7))}\\
&= [\![e]\!][S](T(x, \iota), \boldsymbol{\iota}) && \text{(by HR)}\\
&= [\![e]\!][S](x, \iota \cdot \boldsymbol{\iota}) && \text{(by H1 \& Eq. (1))} \quad \square
\end{aligned}
$$

Example 3. Building upon Example 2, $(\mathsf{a} \leqslant 42)|_1 = \mathsf{a} + r - y - e \leqslant 42$ *symbolically represents all potential paths* $(x, \iota \cdot \boldsymbol{\iota})$ *(i.e., from state* x *with successive inputs* $\iota \cdot \boldsymbol{\iota}$*) for which the value of the state variable* a *after one step is lower or equal than 42: s.t* $[\![\mathsf{a}]\!]T(x, \iota) \leqslant 42$*. Also,* $(\mathsf{a} \leqslant 42)|_1 \wedge (\mathsf{a} \neq 42)|_2 = (\mathsf{a} \leqslant 42)|_1 \wedge \mathsf{a} + r + r_2 - y - y_2 - e - e_2 \neq 42$ *represents all potential paths* $(x, \iota_1 \cdot \iota_2 \cdot \boldsymbol{\iota})$ *for which* $[\![\mathsf{a}]\!]T(x, \iota_1) \leqslant 42 \wedge [\![\mathsf{a}]\!]T(T(x, \iota_1), \iota_2) \neq 42$. ◁

Remark 7. The symbolic computation procedure giving the lookahead of expressions defined on state variables only given in Prop. 6, can be adapted to the case of expressions involving both state and input variables by restricting to $i \in \mathbb{N}^+$ and setting $e|_1 = e$.

As a result of the above remark, the predicate $A|_i$ symbolically describes the set $\mathcal{A}_i$, for $i \in \{1, \ldots, \kappa\}$.

Example 4. Considering S'_{rh} *from Example 2, with future variables* $I_2 = \langle c_2, s_2, r_2, y_2, e_2 \colon \mathbb{Z}^5 \rangle$*,* $A|_2(X, I_{1\rightsquigarrow 2}) =$

$$
\begin{aligned}
&s_2 \geqslant 0 \wedge r_2 \geqslant 0 \wedge y_2 \geqslant 0 \wedge c_2 \geqslant 0 \wedge e_2 \geqslant 0 \wedge\\
&s_2 \leqslant \mathsf{i} - s + c \wedge r_2 \leqslant \mathsf{w} + s + y - r \wedge\\
&y_2 + e_2 \leqslant \mathsf{a} + r - y - e.
\end{aligned}
$$
◁

Symbolic Computation of Controllable Prefixes Let $R_{i+1} \in \mathcal{P}_{X \uplus I_{1\rightsquigarrow i+1}}$ be such that $\mathcal{R}_{i+1} = \left\{ \_ \in \mathcal{D}_X \times \mathcal{D}_I^{i+1} \middle| \_ \models R_{i+1} \right\}$. Mirroring Eq. (5), one can symbolically compute all direct controllable prefixes of the $(i+1)$-paths represented by R_{i+1} using $\text{prefix}_{\text{c}}^i \colon \mathcal{P}_{X \uplus I_{1\rightsquigarrow i+1}} \to \mathcal{P}_{X \uplus I_{1\rightsquigarrow i}}$ defined as

$$\text{prefix}_{\text{c}}^i(R_{i+1}) \stackrel{\text{def}}{=} \forall_{U_i} ((\exists_{C_i} A|_{i+1}) \Rightarrow \exists_{C_i} (A|_{i+1} \wedge R_{i+1}))$$

where $\exists_V e$ (resp. $\forall_V e$) is the existential (resp. universal) elimination of all variables of V from a predicate e.
By extension, $\text{prefix}_{\text{c}}^j(R_{i+1})$, for $j \in \{0, \ldots, i\}$, computes the j-paths that are *controllable prefixes* of R_{i+1}. For instance, $\text{prefix}_{\text{c}}^1(R_3) = \text{prefix}_{\text{c}}^1 \circ \text{prefix}_{\text{c}}^2(R_3)$ denotes all 1-paths that are controllable prefixes of the 3-paths in R_3.

3.3 Building Symbolic κ-lookahead Controllers

We now proceed to define κ-lookahead controllers that enforce *bounded safety* or *κ-optimization* objectives. We first advance means for building controllers strictly allowing a given set of *desirable κ-paths*, and develop controller transformation and composition operations leading to deadlock-free systems with best-effort and recovery control. We detail in Sections 3.4 and 3.5 how to derive desirable κ-paths from target objectives.

State Safety w.r.t. Sets of κ-paths We say that a state $x \in \mathcal{D}_X$ is *safe w.r.t.* $\mathcal{R}_\kappa$ *under non-controllable input* $u \in \mathcal{D}_U$, denoted $x \overset{u}{\rightsquigarrow} \mathcal{R}_\kappa$, whenever there exists a value for controllable variables admissible by S and forming a 1-path that is a controllable prefix of κ-paths belonging to $\mathcal{R}_\kappa$; *i.e.,* $x \overset{u}{\rightsquigarrow} \mathcal{R}_\kappa \stackrel{\text{def}}{=} \exists c \in \mathcal{D}_C, (x, u, c) \in \mathcal{R}_1 \cap \mathcal{A}_1$.

Building Strict κ-lookahead Controllers A κ-lookahead controller *implicitly encodes* all possible relevant outcomes for the next κ steps and achieves its objective by restricting the admissible values of the controllable inputs *for the current step.*

Definition 8. A *strict controller* K_κ *for desirable κ-paths* $\mathcal{R}_\kappa$ is such that $\forall (x, u) \in \mathcal{O}(S) \times \mathcal{D}_U$,

$$\exists c \in \mathcal{D}_C, (x, u, c) \models K_\kappa \Leftrightarrow x \overset{u}{\rightsquigarrow} \mathcal{R}_\kappa. \tag{8}$$

We say that K_κ *induces a deadlock with non-controllable input* u in every reachable state unsafe *w.r.t.* $\mathcal{R}_\kappa$ under u.

Theorem 9. Let $R_\kappa \in \mathcal{P}_{X \uplus I_{1\rightsquigarrow \kappa}}$ be a predicate such that $\mathcal{R}_\kappa = \{ \_ \in \mathcal{D}_X \times \mathcal{D}_I^\kappa \mid \_ \models R_\kappa \}$. Then $K_\kappa = A \wedge \text{prefix}_{\text{c}}^1(R_\kappa)$.

Proof. First, K_κ is a controller for S by construction, (as $\forall P \in \mathcal{P}_{X \uplus I}, (A \wedge P) \Rightarrow A$, leading to Eq. (3) through Eq. (1)). Next, noting that $R_1 = \text{prefix}_{\text{c}}^1(R_\kappa)$ is such that $\mathcal{R}_1 = \left\{ \_ \in \mathcal{D}_X \times \mathcal{D}_I \middle| \_ \models R_1 \right\}$ and $\mathcal{O}(S) \subseteq \mathcal{D}_X$, one can rewrite the left-hand side of Eq. (8) as $\forall (x, u) \in \mathcal{O}(S) \times \mathcal{D}_U, \exists c \in \mathcal{D}_C, (x, u, c) \in \mathcal{R}_1 \cap \mathcal{A}_1$. $\square$

Transformation for Best-effort Control As our aim is to eventually obtain controlled systems that are executable, we now devise a controller transformation producing *deadlock-free controlled systems.*

Definition 10. A *best-effort controller* $\mathsf{be}(K)$ for S obtained from another controller K is such that,
$\forall (x, u, c) \in \mathcal{O}(S) \times \mathcal{D}_U \times \mathcal{D}_C, [S\!\!\!/_{\mathsf{be}(K)}](x, u, c) =$

$$\begin{cases} [S\!\!\!/_K](x, u, c) & \text{if } \exists c \in \mathcal{D}_C, (x, u, c) \models K \\ [S](x, u, c) & \text{otherwise.} \end{cases} \tag{9}$$

In plain words, $S\!\!\!/_{\mathsf{be}(K)}$ behaves as $S\!\!\!/_K$ as long as it does not deadlock, or as S otherwise.

Proposition 11. $\mathsf{be}(K) = A \wedge (\exists_C K \Rightarrow K)$.

Proof. $\mathsf{be}(K)$ is a controller for S for the same reason as for Theorem 9. Then, the rewriting of the definition of $\mathsf{be}(K)$ against Eq. (9) comes directly. $\square$

Although inducing a significant relaxation in terms of the enforced objectives, best-effort controllers bring us deadlock-freeness, hence executable controlled systems:

Theorem 12. If S is reactive, then any best-effort controller for S produces a deadlock-free ASTS.

Proof. Let us write Eq. (4) as $\mathsf{df}(S\!\!\!/_K) \equiv \forall (x, u, c) \in \mathcal{O}(S\!\!\!/_K) \times \mathcal{D}_U \times \mathcal{D}_C, [S](x, u, c) \neq \bot \Rightarrow \exists c' \in \mathcal{D}_C, [S\!\!\!/_K](x, u, c') \neq \bot$. According to Def. 4, we only need to show $\mathsf{df}\left(S\!\!\!/_{\mathsf{be}(K)}\right)$ for any controller K for S.
By definition of $\mathsf{be}(K)$ (Eq. (9) notably), one has $\forall (x, u, c) \in \mathcal{O}(S) \times \mathcal{D}_U \times \mathcal{D}_C, [S](x, u, c) \neq \bot \Rightarrow [S\!\!\!/_{\mathsf{be}(K)}](x, u, c) \neq \bot$ ($\equiv$ R1). Noting that $\mathcal{O}\left(S\!\!\!/_{\mathsf{be}(K)}\right) \subseteq \mathcal{O}(S)$ as $\mathsf{be}(K)$ is also a controller for S, then R1 $\Leftrightarrow \forall (x, u, c) \in \mathcal{O}\left(S\!\!\!/_{\mathsf{be}(K)}\right) \times \mathcal{D}_U \times \mathcal{D}_C, [S](x, u, c) \neq \bot \Rightarrow [S\!\!\!/_{\mathsf{be}(K)}](x, u, c) \neq \bot \equiv \mathsf{df}\left(S\!\!\!/_{\mathsf{be}(K)}\right)$. $\square$

Transformation Towards Best-effort Recovery In addition to the above relaxation of control for strict desirable κ-paths $\mathcal{R}_\kappa$, one can design controllers that also recover from failures to stay in safe states *w.r.t.* $\mathcal{R}_\kappa$ under some non-controllable input in case values for non-controllable variables permit such a recovery. Notice we expose here a way to implement recovery using a single step lookahead, yet this approach can be extended to further steps ahead.

We denote by $\mathit{Safe}(K) \stackrel{\text{def}}{=} \{x \in \mathcal{D}_X \,|\, \forall u \in \mathcal{D}_U, \exists c \in \mathcal{D}_C, (x,u,c) \models K\}$ the set of states from which a controller K is satisfiable whatever the valuation for the non-controllable variables.

Definition 13. A *recovering controller* $\mathsf{re}(K|\mathcal{R}_\kappa)$ for S obtained from a controller K and a strict controller K_κ for desirable κ-paths $\mathcal{R}_\kappa$ such that $K_\kappa \Rightarrow K$, behaves as $S/_K$ in all states $\mathit{Safe}(K_\kappa)$, or reaches such a state in one step whenever possible.

Theorem 14. $\mathsf{re}(K|\mathcal{R}_\kappa) = (\nexists_I E \wedge K) \vee (\exists_U E \wedge A)$ with $E = A \wedge \neg\mathrm{prefix}^0_{\mathrm{c}}(K_\kappa) \wedge \mathrm{prefix}^0_{\mathrm{c}}(K_\kappa)|_1$.

Proof. Note that $\{x \in \mathcal{D}_X | x \models \mathrm{prefix}^0_{\mathrm{c}}(K_\kappa)\} = \mathit{Safe}(K_\kappa)$. Hence, E represents all 1-paths starting in unsafe states *w.r.t.* K_κ and reaching safe ones in one step ($\neg\mathrm{prefix}^0_{\mathrm{c}}(K_\kappa) \wedge \mathrm{prefix}^0_{\mathrm{c}}(K_\kappa)|_1$) that are admissible by S. Further, $\exists_I E$ denotes all unsafe states that are also direct predecessors of safe ones, so $\nexists_I E$ exactly denotes all states that behave according to K in $S/_{\mathsf{re}(K|\mathcal{R}_\kappa)}$. Also, $\exists_U E$ represents all transitions for which some value for non-controllable variables exist to reach a safe state *w.r.t.* $\mathcal{R}_\kappa$. At last, K is a controller for S, hence $\mathsf{re}(K|\mathcal{R}_\kappa) \Rightarrow A$ by construction, and $\mathsf{re}(K|\mathcal{R}_\kappa)$ is a controller for S. □

3.4 Control for Bounded-safety

Given the tooling listed above, computing a bounded-safety controller enforcing some invariant Φ over a sliding window of κ steps, boils down to build a κ-lookahead predicate encoding all κ-paths traversing states satisfying Φ.

Definition 15. Given a predicate $\Phi \in \mathcal{P}_X$, a *strict bounded-safety controller* $K_{\Phi,\kappa}$ is a strict controller for the desirable κ-paths $\mathcal{R}_\kappa = \{\_ \in \mathcal{D}_X \times \mathcal{D}_I^\kappa | \_ \models \bigwedge_{i\in\{1,\ldots,\kappa\}} \Phi|_i\}$.

Example 5. Considering S'_{rh} from Example 2, assume that, for availability purposes, or to accommodate a limited number of hardware resources, one wants to bound the number of already started request handlers at any time. The associated invariant is, e.g., $\Phi(X) = (\mathsf{w}+\mathsf{a} \leqslant 42)$. Enforcing Φ over a two-step window using a strict controller, one obtains $R_2 = \bigwedge_{i\in\{1,2\}} (\mathsf{w}+\mathsf{a} \leqslant 42)|_i = (\mathsf{w}+\mathsf{a}+s-e \leqslant 42) \wedge (\mathsf{w}+\mathsf{a}+s+s_2-e-e_2 \leqslant 42)$. The controller is then $K_{\Phi,2} = A \wedge \mathrm{prefix}^1_{\mathrm{c}}(R_2) = A \wedge s+\mathsf{w}+\mathsf{a}-e \leqslant 42$. ◁

Best-effort and recovering versions of such controllers can be built using $K_{\Phi,\kappa}$ as described in Section 3.3.

3.5 Control for κ-optimization

We now briefly address the control of S for the minimization of some accumulated valuation of a guarded arithmetic expression on state variables over a finite number of steps.

Enforcing some optimization objectives over κ steps ahead actually boils down to build the appropriate set of κ-paths

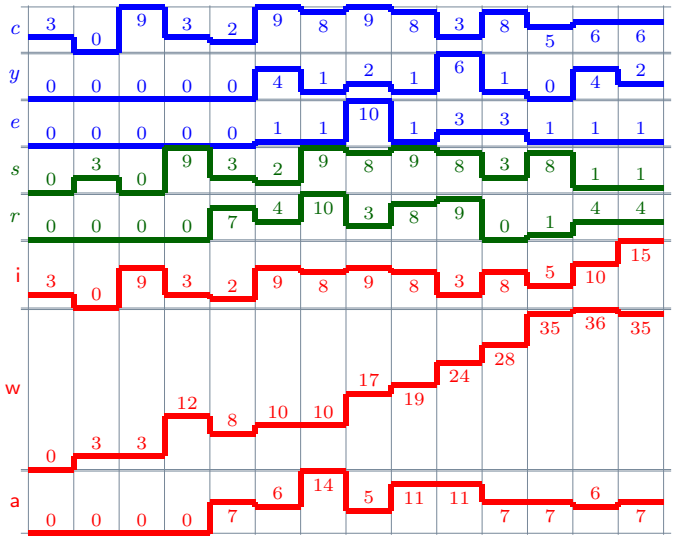

Fig. 4. Example execution trace of $S'_{\mathrm{rh}}/_{K_\omega}$ (built in Example 6). Each step consists in: values for variables in U are randomly picked so that $\exists_C A$ is satisfied; in turn, values for s and r are also randomly picked so that K_ω is satisfied.

$\mathcal{R}_\kappa$ ensuring the "best" choice of values for controllable variables at the current step *w.r.t.* the potential values for the current and future non-controllable ones.

A κ-optimization objective $\min(\Sigma e)_{@\kappa}$ for a guarded linear arithmetic expression $e \in \mathcal{L}^{\mathcal{P}}_X$ targets the minimization of its sum over κ steps. In terms of κ-lookahead expressions, the value to be minimized is then $E_\kappa = \sum_{i\in\{1,\ldots,\kappa\}} e|_i$, and the set $\mathcal{Y} \subseteq \mathcal{D}_X \times \mathcal{D}_I^\kappa$ of κ-paths for which all valuations of controllable variables are guaranteed to minimize E_κ can be defined symbolically as

$$\mathcal{Y} \stackrel{\text{def}}{=} \{\_ \in \mathcal{D}_X \times \mathcal{D}_I^\kappa \,|\, \_ \models \nexists_{C'_{1\rightsquigarrow\kappa}} (A_\kappa \Rightarrow (A'_\kappa \wedge E'_\kappa < E_\kappa))\}$$

with $A_\kappa = \bigwedge_{i\in\{1,\ldots,\kappa\}} A|_i$ and $e' = e[C_{1\rightsquigarrow\kappa}/C'_{1\rightsquigarrow\kappa}]$, where $V' = \{v' \,|\, v \in V\}$ is the set of primed versions of a vector of variables V. In plain words, all paths in $\mathcal{Y}$ are the admissible ones for which there does not exist alternative valuations for current and future controllable variables ($C_{1\rightsquigarrow\kappa}$) such that E_κ evaluates to a strictly greater value (*i.e.*, they give the lowest possible value for E_κ).

Example 6. We continue building up on Example 5. In addition to the limit on started requests, we now consider minimizing the number of handlers waiting to be started, and arbitrarily choose a sliding window of one step for this objective. This previous statement translates into the optimization objective $\omega = \min(\Sigma \mathsf{i})_{@1}$ on $S'_{\mathrm{rh}}/_{K_{\Phi,2}}$ (i.e., now, $A = K_{\Phi,2}$). The arithmetic 1-lookahead expression to minimize is $E_1 = \sum_{i\in\{1\}} \mathsf{i}|_i = \mathsf{i} - s + c$. From E_1, one has $E'_1 < E_1 = s' > s$, and then $Y = \nexists_{\{r',s'\}} (A_1 \Rightarrow (A'_1 \wedge s' > s))$. In the end, the obtained controller is $K_\omega = K_{\Phi,2} \wedge ((s = \mathsf{i} \wedge \mathsf{i}+\mathsf{w}+\mathsf{a}-e \leqslant 41) \vee (s \leqslant \mathsf{i} \wedge s+\mathsf{w}+\mathsf{a}-e = 42))$, that actually starts as many allocated handlers (in state I) as possible, as long as the resulting number of started handlers does not exceed 42. We show in Fig. 4 an execution trace of the resulting controlled system. Remark the value chosen for s always equals that of i (which is indeed minimized), except during the two last steps where the value of $\mathsf{w}+\mathsf{a} = 42$. ◁

The set $\mathcal{Y}$ above can actually be empty, if no choice of values for controllable variables at any instant is guaranteed to lead to the minimal outcome after κ steps *w.r.t.* values for non-controllable variables. In such a case, the resulting system induces deadlocks; yet, a transformation for best-effort control still suffices to eventually obtain a deadlock-free controlled system.

Other sets of κ-paths can also be considered to extend the set of desirable κ-paths. One can for instance consider building a controller for desirable κ-paths $\mathcal{Y} \cup \mathcal{Z}$, where

$$\mathcal{Z} \stackrel{\text{def}}{=} \left\{ \_ \in \mathcal{D}_X \times \mathcal{D}_I^\kappa \;\middle|\; \_ \models \nexists_{U'_{2\rightsquigarrow\kappa}} \left(A_\kappa \Rightarrow (A''_\kappa \wedge E''_\kappa < E_\kappa)\right) \right\}$$

with $e'' = e[U_{2\rightsquigarrow\kappa}/U'_{2\rightsquigarrow\kappa}]$. $\mathcal{Z}$ as defined above consists of the κ-paths for which no future non-controllable input $(U_{2\rightsquigarrow\kappa})$ exist that lead to a lower value for E_κ.

4. IMPLEMENTATION IN REAX & EVALUATIONS

We have implemented all the algorithms detailed in this paper in ReaX [Berthier and Marchand, 2014, 2015] [1] , that is a framework suitable for the symbolic manipulation of logico-numerical systems like ASTSs. We use a combination of (multi-terminal) binary decision diagrams [Billon, 1987] and disjunctions of convex polyhedra [Cousot and Halbwachs, 1978] as symbolic representations of sets of linear arithmetic constraints. This representation allows to implement exact universal and existential eliminations of finite and infinite variables on predicates involving such constraints. Note that our tool fully supports linear *logico*-numerical systems (*i.e.,* linear ASTSs also involving variables defined on finite domains like Booleans and other enumerated types), although as presented above our quantitative modeling technique does not require this feature. Yet, additional finite-state automata can easily be composed with resource families to model a finite number of elements that are deemed relevant *w.r.t.* control objectives in the computing system to manage.

We now give some performance evaluation results based on example models of RAs to assess the practicality of our modeling approach and algorithms. Exploiting the scalability of our RA of Fig. 1, we first constructed a series of "Parallel" systems built by assembling N controllable ASTSs S'_{rh} using parallel composition, for $N \in \{2, 3, 6, 9\}$ [2] . The resulting systems model as many "kinds" of request handlers to be managed. We also built a second series "Alt" obtained by modeling the RA of Fig. 1 with N states A_i and associated additional input/output transitions r_i, y_i and e_i, again for $N \in \{2, 3, 6, 9\}$. For "Parallel", our example objective is to balance the number of active request handlers of each kind. For "Alt" models on the other hand, our objective is to impose an upper-limit on the number of non-started request handlers, similarly to Example 5. We use best-effort control in all cases, and exercise the recovering synthesis for "Alt".

We show in Fig. 5 a simulated execution trace for the

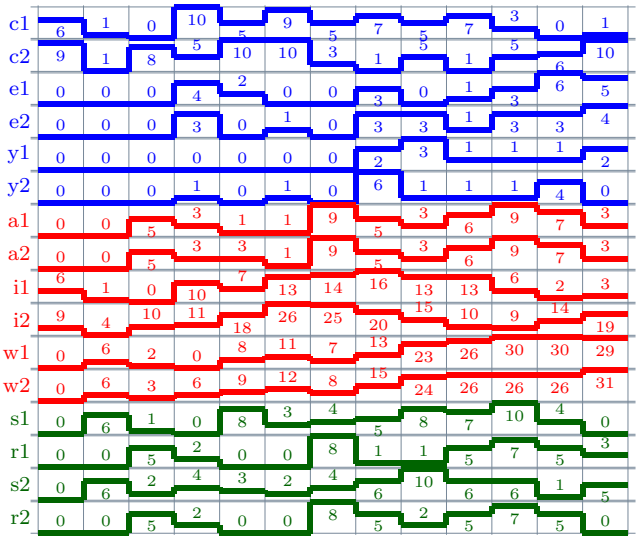

Fig. 5. Example execution trace for two distinct families of request handlers (Parallel, κ=1, $N = 2$). Each column shows the input values at step i along with the *new values* (for step $i + 1$) for the state variables a1, a2, i1, i2, w1, and w2 (all null initially).

Table 1. Synthesis Times (in Seconds)

N:	2	3	6	9
Parallel, κ=1	0.04	0.04	0.28	1.03
Alt, κ=1	0.03	0.04	0.06	0.12
Alt, κ=2	0.10	0.15	0.49	41.28
Alt, κ=1, 1-step recovery	0.05	0.09	0.31	0.16
Alt, κ=2, 1-step recovery	0.14	0.22	0.90	41.32

"Parallel" system with $N = 2$ controlled κ=1-step ahead. Focusing on the 5th execution step, one observes that 3 handlers of type 1 are active, 2 are ending their executions, and none yields (a1 = 3, e1 = 2, and y1 = 0); meanwhile, none of the 3 active handlers of type 2 terminates nor yields (a2 = 3, e2 = 0, and y2 = 0). Yet with no handler of type 1 in waiting stage (w1 = 0), no suitable values for the controllable variables (s1, r1, s2 and r2) exist to achieve the safety objective at this execution step (*i.e.,* balancing the number of handlers in active stage). Therefore, the best-effort controller admits any values that allow the system to progress, and the safety objective is thus not considered for this particular instant: 8 handlers of type 1 and 3 handlers of 2 are started (s1 = 8 and s2 = 3). The failure to strictly impose the safety objective occurs only once in this trace. We report synthesis times in Table 1 [3] . Observe that ReaX's core symbolic manipulation engine may be less suited for performing the computations required by our algorithms than state-of-the-art SMT solvers for instance. Still, these performance results and our simulations show us that ReaX is able to compute controllers imposing useful objectives on system featuring dozens of numerical variables in reasonable time. Therefore, these preliminary experiments convince us that best-effort control and recovery are worth investigating for dynamic computing resource management, in cases where enforcing strict control objectives is not necessary.

5. RELATED WORKS

The control of infinite systems has been subject to several studies based on various models: Timed Automata [Cassez

[1] `http://reatk.gforge.inria.fr/`

[2] Building the parallel composition of ASTSs essentially boils down to concatenating their vectors of variables and transition functions; in our case, we obtain systems with variables of S'_{rh} suffixed with indexes corresponding to their respective family of request handlers.

[3] All results are obtained on a desktop computer equipped with a quad-core Intel i5 (3.3GHz), 16GB of RAM, and a standard GNU/Linux operating system.

et al., 2005], Vector of Discrete Events Systems [Li and Wonham, 1994], Petri Nets [Kumar and Garg, 2005; Holloway et al., 1997] or symbolic transition systems with variables [Kalyon et al., 2011]. Inspired by the work of Chung et al. [1992], we provide here some symbolic devices to compute limited lookahead controllers for logico-numerical reactive systems.

Regarding symbolic modeling for resource control, our quantitative modeling method is loosely related to the one used by Fei et al. [2015] for modeling Resource Allocation Systems: they handle a specific class of RASs where "process types" (that loosely correspond to resource families in our work) are acyclic, and "resource types" required for the execution of processes, are of finite capacity and constant. They seek maximally permissive non-blocking supervisors for deadlock avoidance by modeling the RASs using Extended Finite Automata [Skoldstam et al., 2007]. Although their approach is well suited for controlling plants and discrete systems that can be modeled using finite-state automata and where limited amounts of resources are involved, this modeling method is not fit for systems featuring dynamic amounts of such resources. Lennartson et al. [2014] support dynamic aspects through an encoding similar to ours, and address synthesis for performance optimization (in the sense of make span minimization for Petri Nets), though with a finite alphabet of events and state variables defined on finite domains only.

Few approaches involve discrete event control for self-adaptive computing systems. Dumitrescu et al. [2010] focus on finite-state systems equipped with cost functions and final states, and advance an algorithm for multi-criteria optimal discrete controller synthesis; they practice their approach towards fault-tolerance. Berthier et al. [2015] explore the capabilities of over-approximating symbolic DCS algorithms to introduce quantitative aspects into models involved in the control of dynamically partially reconfigurable architectures investigated by An et al. [2016]. A common issue of these approaches is the need to enumerate the automata of all individual resource to manage: this aspect challenges the computational cost of DCS algorithms involved, and the modeling task *w.r.t.* the control problems at hand. We overcome these limits by allowing more powerful models (*e.g.,* involving numerical controllable variables) and relaxing the control objectives (*e.g.,* accommodating non-strict safety with best-effort control, recovery, and limited lookahead optimization).

6. CONCLUSIONS AND PERSPECTIVES

In this paper, we have exploited the expressivity of logico-numerical models to solve resource management problems occurring in computing systems. Our solution alleviates the need to enumerate the automata of every managed resource, and permits the expression of a wide range of control objectives. To enforce the latter on such systems with infinite control means, we focus on the control over a finite sliding window. We provide exact, effective algorithms for bounded safety and optimization, and illustrate them on an example. We also give means to relax such strict objectives for best-effort control and recovery. We exercise our implementation of the algorithms on some benchmarks. We plan to reuse the resource models put forward in this paper to validate the practicality of the technique, and to transfer them in the framework of ongoing research: on the one side, as a follow-up of the work of An et al. [2016] in model-based control of reconfigurable FPGA-based architectures; on the other side, in control of middleware-level redeployment of computations and reconfigurations of heterogeneous architectures in the context of smart buildings [Sylla et al., 2016]. Further, best-effort and recovery control on finite sliding windows provides a good trade-off in terms of expressiveness of the models and achievable control objectives, these techniques are thus especially interesting in the case of non-critical systems.

REFERENCES

An, X., Rutten, E., Diguet, J.P., and Gamatie, A. (2016). Model-based design of correct controllers for dynamically reconfigurable architectures. *ACM Trans. Emb. Comp. Syst.*

Berthier, N., An, X., and Marchand, H. (2015). Towards Applying Logico-numerical Control to Dynamically Partially Reconfigurable Architectures. In *5th Int. Workshop on Dependable Control of Discrete Systems*, DCDS '15. IFAC.

Berthier, N. and Marchand, H. (2014). Discrete Controller Synthesis for Infinite State Systems with ReaX. In *12th Int. Workshop on Discrete Event Systems*, WODES '14. IFAC.

Berthier, N. and Marchand, H. (2015). Deadlock-Free Discrete Controller Synthesis for Infinite State Systems. In *54th IEEE Conference on Decision and Control*, CDC '15, 1000–10007.

Berthier, N., Rutten, E., De Palma, N., and Gueye, S.M.K. (2016). Designing Autonomic Management Systems by using Reactive Control Techniques. *IEEE Trans. Softw. Eng.*, 42(7), 640–657.

Billon, J. (1987). Perfect normal forms for discrete programs. Technical report, Bull.

Cassez, F., David, A., Fleury, E., Larsen, K., and Lime, D. (2005). Efficient on-the-fly algorithms for the analysis of timed games. In *Conference on Concurrency Theory*, volume 3653 of *LNCS*, 66–80.

Chung, S.L., Lafortune, S., and Lin, F. (1992). Limited lookahead policies in supervisory control of discrete event systems. *IEEE Trans. Autom. Control*, 37(12), 1921–1935.

Cousot, P. and Halbwachs, N. (1978). Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM Symposium on Principles of Programming Languages*, POPL '78, 84–96.

Dumitrescu, E., Girault, A., Marchand, H., and Rutten, E. (2010). Multicriteria optimal discrete controller synthesis for fault-tolerant tasks. In *10th Int. Workshop on Discrete Event Systems*, WODES '10, 356–363. IFAC.

Fei, Z., Reveliotis, S., Miremadi, S., and Åkesson, K. (2015). A BDD-Based Approach for Designing Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems. *IEEE Trans. Autom. Sci. Eng.*, 12(3), 990–1006.

Holloway, L., Krogh, B., and Giua, A. (1997). A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Application*, 7, 151–190.

Kalyon, G., Le Gall, T., Marchand, H., and Massart, T. (2011). Symbolic supervisory control of infinite transition systems under partial observation using abstract interpretation. *Discrete Event Dynamic Systems : Theory and Applications*, 22(2), 121–161.

Kumar, R. and Garg, V. (2005). On computation of state avoidance control for infinite state systems in assignment program model. *IEEE Trans. Autom. Sci. Eng.*, 2(2), 87–91.

Lennartson, B., Wigström, O., Fabian, M., and Basile, F. (2014). Unified Model for Synthesis and Optimization of Discrete Event and Hybrid Systems. In *12th Int. Workshop on Discrete Event Systems*, WODES '14, 86–92. IFAC.

Li, Y. and Wonham, W. (1994). Control of vector discrete-event systems-part ii : controller synthesis. *IEEE Trans. Autom. Control*, 39(3), 512–531.

Skoldstam, M., Åkesson, K., and Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. In *46th IEEE Conference on Decision and Control*, 3387–3392.

Sylla, A.N., Louvel, M., and Rutten, E. (2016). Combining transactional and behavioural reliability in adaptive middleware. In *15th Workshop on Adaptive and Reflective Middleware*, ARM '16.