Can we Communicate? Using Dynamic Logic to Verify Team Automata (Extended Version)

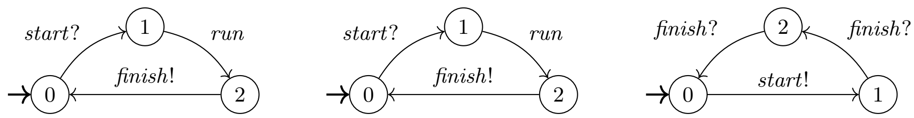
Maurice H. ter Beek[1] (✉), Guillermina Cledou[2], Rolf Hennicker[3], and José Proença[4] (✉)

[1] ISTI–CNR, Pisa, Italy, maurice.terbeek@isti.cnr.it
[2] INESC TEC & Univ. Minho, Portugal, mgc@inesctec.pt
[3] Ludwig-Maximilians-Universität München, Germany
[4] Polytechnic Institute of Porto, Portugal, pro@isep.ipp.pt

Abstract. Team automata describe networks of automata with input and output actions, extended with synchronisation policies guiding how many interacting components can synchronise on a shared input/output action. Given such a team automaton, we can reason over communication properties such as *receptiveness* (sent messages must be received) and *responsiveness* (pending receives must be satisfied). Previous work focused on how to *identify* these communication properties. However, automatically verifying these properties is non-trivial, as it may involve traversing networks of interacting automata with large state spaces. This paper investigates (1) how to *characterise* communication properties for team automata (and subsumed models) using test-free propositional dynamic logic, and (2) how to use this characterisation to *verify* communication properties by model checking. A prototype tool supports the theory, using a transformation to interact with the mCRL2 tool for model checking.

1 Introduction

In automata-based models of Systems of Systems (SoS) that communicate via shared actions, it is of paramount importance to guarantee safe communication, i.e. absence of failures such as message loss (typically of output not received as input, thus violating so called *receptiveness*) or indefinite waiting (typically for input that never arrives, thus violating so called *responsiveness*). This requires knowledge of the adopted communication policy that defines when and which actions are executed (synchronously) and by how many system components. Team automata, originally introduced as an extension of I/O automata [14, 38] in the context of computer supported cooperative work (CSCW) to model groupware systems [29], were formalised as a theoretical framework for studying synchronisation policies in system models [11, 13]. They proved useful also for capturing access control and other security protocols [10,16]. Their distinguishing feature is the variety of synchronisation policies which, in principle, allow any number of interacting (component) automata to participate in the synchronised execution of a shared communicating action, either as a sender or as a receiver.



(a) $\mathcal{A}_{\mathsf{R1}}$ (b) $\mathcal{A}_{\mathsf{R2}}$ (c) $\mathcal{A}_{\mathsf{Ctrl}}$

Fig. 1: The three component automata constituting the Race system

Emblematic synchronisation types were defined to systematise the synchronisation policies realisable in team automata [8] (e.g. multi-cast, broadcast, master-worker) in terms of explicit intervals for the number of sending and receiving components that can participate in a synchronisation. In extended team automata (ETA) [12], synchronisation type specifications (STS) separately assign a synchronisation type to each communicating action. STS uniquely determine a team and induce communication requirements that the team should satisfy. Generic procedures to derive requirements for receptiveness and responsiveness for each synchronisation type were developed, and communication-safety of ETA was defined in terms of compliance with such requirements. A team automaton is called compliant with a set of communication requirements if in each of its reachable states, the requirements are met (i.e. communication is safe); if the required communication cannot occur immediately, but only after some arbitrary other actions have been executed, the team automaton is called weakly compliant (akin to weak compatibility [7,33] or agreement of lazy request actions [5]).

Motivating Example We illustrate the state-of-the-art as schematised in the upper row of Fig. 2. Consider a system ($\mathcal{S}$), called Race, to model competitions of two runner components R1 and R2 under the control of a third component Ctrl. The behaviour of the components is modelled by the component automata (CA) $\mathcal{A}_{\mathsf{R1}}, \mathcal{A}_{\mathsf{R2}}$, and $\mathcal{A}_{\mathsf{Ctrl}}$ in Fig. 1. Both runners have the same behaviour: $\mathcal{A}_{\mathsf{R1}} = \mathcal{A}_{\mathsf{R2}}$. Each runner starts in the initial state 0, indicated by →, in which she is able to receive a *start* signal (input?). Upon reception, she performs the (internal) action *run* and when she reaches the finish line she sends the *finish* signal (output!), after which she is ready for another competition. The controller's task is to start the runners and receive their finish signals. We want to combine these CA in a team such that the controller starts both runners at once, but each runner separately sends her *finish* signal to the controller upon reaching the finish line.

To this aim, ETA use *synchronisation type specifications* (st) to determine the number of senders and receivers allowed to participate in a communication, thus restricting the behaviour of system Race (given by a labelled transition system **lts**($\mathcal{S}$) which contains arbitrary synchronisations of shared actions of the three CA). We specify $([1,1],[2,2])$ for action *start* and $([1,1],[1,1])$ for *finish* such that *start* occurs only as a synchronisation involving exactly *one* component for which it is an output action and exactly *two* for which it is an input action, while *finish* occurs in a *one-to-one* fashion.

In the team's initial state $(0,0,0)$, the controller is in its local state 0 where it can only make progress if its *start* signal is received by a runner. This induces a receptiveness requirement. The ETA **eta**($\mathcal{S}$, **st**) generated over $\mathcal{S}$ by the STS **st** is

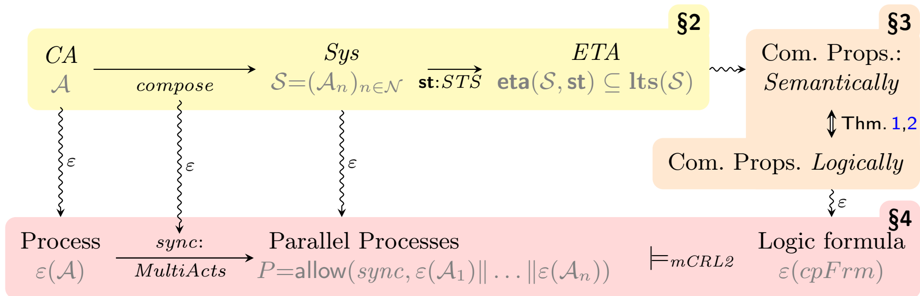


Fig. 2: Overview of this paper; the top row concerns previous work [8, 12]

compliant with this requirement if other team component(s) synchronise by receiving *start* as input in accordance with the synchronisation type of *start*, which is the case. There are other receptiveness and also responsiveness requirements. Requirements and compliance of ETA are called *weak* if the other component(s) may perform intermediate actions before the requirement is satisfied.

Related Work and Challenges Communication safety (mainly receptiveness) and related notions of compatibility have been widely studied to (*semantically*) characterise communication properties [2, 8, 9, 12, 19, 20, 22–25, 27, 28, 34–37], in particular for automata-based system models, but typically limited to pairs of automata or networks with binary, peer-to-peer communication [6, 22, 27, 34–37]. An extension to multi-component communications was first investigated in [23] and then in [8, 9, 12], where the notion of responsiveness was introduced. Only a few approaches come with tool support [1, 3, 7, 9, 17, 26], based on algorithms following the semantic compatibility definitions. The purely semantic nature of communication properties is a serious burden in practice, making it challenging to prove properties in concrete cases: one has to go through all reachable states of a team automaton and check compliance for all requirements at each state.

Contribution In this paper, we pursue a different approach by providing a *logical* characterisation of communication properties, which we believe is interesting by itself, and which has the advantage that it can be checked using available model-checking tools. Our results complete Fig. 2 with three main contributions.

First, after presenting the necessary background on team automata and dynamic logic in Sect. 2, we demonstrate in Sect. 3 that (weak) receptiveness and (weak) responsiveness can be characterised (*logically*) by dynamic logic formulas $(w)rcpFrm$ and $(w)rspFrm$, resp., summarised as $cpFrm$. These results, formulated in Theorems 1 and 2, pave the way for automatically checking these communication properties with tooling available for dynamic logic.

Second, in Sect. 4, we present a transformation (ε) of component automata, systems and ETA into mCRL2 [21] processes and of the characterising dynamic logic formulas $cpFrm$ into μ-calculus formulas. The latter is straightforward, whereas the former makes use of mCRL2's allow operator to suitably restrict the number of multi-action synchronisations such that the semantics of systems of component automata is preserved (up to renaming).

Third, Sect. 4 introduces the open-source prototype tool we developed to perform the transformation into mCRL2 processes and to automatically check communication properties with the model-checking facilities offered by mCRL2, which outputs the result of the formula as well as a witness or counterexample.

To the best of our knowledge, we are the first to provide a logical characterisation of the communication properties of receptiveness and responsiveness.

2 Background on Team Automata and Dynamic Logic

This section summarises the basic notions of (extended) team automata (ETA) following [12], but additionally considering internal actions, and of dynamic logic.

2.1 Component Automata and Systems

A *labelled transition system* (LTS) is a tuple $\mathcal{L} = (Q, q_0, \Sigma, E)$ such that Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite set of labels, and $E \subseteq Q \times \Sigma \times Q$ is a transition relation.

Notation. Given an LTS $\mathcal{L}$, we write $q \xrightarrow{a}_{\mathcal{L}} q'$, or shortly $q \xrightarrow{a} q'$, to denote $(q, a, q') \in E$. Similarly, we write $q \xrightarrow{a}_{\mathcal{L}}$ to denote that a is *enabled* in $\mathcal{L}$ at state q, i.e. there exists $q' \in Q$ such that $q \xrightarrow{a} q'$. For $\Gamma \subseteq \Sigma$, we write $q \xrightarrow{\Gamma}^{*} q'$ if there exist $q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q'$ for some $n \geq 0$ and $a_1, \ldots, a_n \in \Gamma$. A state $q \in Q$ is *reachable by* Γ if $q_0 \xrightarrow{\Gamma}^{*} q$, it is *reachable* if $q_0 \xrightarrow{\Sigma}^{*} q$. The set of reachable states of $\mathcal{L}$ is denoted by $\mathcal{R}(\mathcal{L})$.

A *component automaton* (CA) is an LTS $\mathcal{A} = (Q, q_0, \Sigma, E)$ such that $\Sigma = \Sigma^{?} \uplus \Sigma^{!} \uplus \Sigma^{\tau}$ is a set of *action labels* split into disjoint sets $\Sigma^{?}$ of *input actions*, $\Sigma^{!}$ of *output actions*, and Σ^{τ} of *internal actions*. For easier readability, in graphical representations input actions will be shown with suffix "?", output actions with suffix "!", and internal actions just by their name.

Example 1. Examples of component automata are shown in Fig. 1 of Sect. 1. For $i = 1, 2$, the action labels of $\mathcal{A}_{\mathsf{Ri}}$ are $\Sigma_{\mathsf{Ri}} = \Sigma^{?}_{\mathsf{Ri}} \uplus \Sigma^{!}_{\mathsf{Ri}} \uplus \Sigma^{\tau}_{\mathsf{Ri}}$, where $\Sigma^{?}_{\mathsf{Ri}} = \{\mathit{start}\}$, $\Sigma^{!}_{\mathsf{Ri}} = \{\mathit{finish}\}$, $\Sigma^{\tau}_{\mathsf{Ri}} = \{\mathit{run}\}$. The action labels of $\mathcal{A}_{\mathsf{Ctrl}}$ are $\Sigma_{\mathsf{Ctrl}} = \Sigma^{?}_{\mathsf{Ctrl}} \uplus \Sigma^{!}_{\mathsf{Ctrl}} \uplus \Sigma^{\tau}_{\mathsf{Ctrl}}$ where $\Sigma^{?}_{\mathsf{Ctrl}} = \{\mathit{finish}\}$, $\Sigma^{!}_{\mathsf{Ctrl}} = \{\mathit{start}\}$, $\Sigma^{\tau}_{\mathsf{Ctrl}} = \varnothing$. ▷

A *system* is a pair $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_n)_{n \in \mathcal{N}})$, with $\mathcal{N}$ a finite, nonempty set of component names and $(\mathcal{A}_n)_{n \in \mathcal{N}}$ an $\mathcal{N}$-indexed family of CA $\mathcal{A}_n = (Q_n, q_{0,n}, \Sigma_n, E_n)$.

Example 2. The race system of Sect. 1 is $\mathsf{Race} = (\mathcal{N}_{\mathsf{Race}}, (\mathcal{A}_n)_{n \in \mathcal{N}_{\mathsf{Race}}})$, with $\mathcal{N}_{\mathsf{Race}} = \{\mathsf{R1}, \mathsf{R2}, \mathsf{Ctrl}\}$ and the CA $\mathcal{A}_{\mathsf{R1}}, \mathcal{A}_{\mathsf{R2}}$, and $\mathcal{A}_{\mathsf{Ctrl}}$ from Example 1. ▷

Any system $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_n)_{n \in \mathcal{N}})$ induces an LTS defined by $\mathbf{lts}(\mathcal{S}) = (Q, q_0, \Lambda(\mathcal{S}), E(\mathcal{S}))$, where $Q = \prod_{n \in \mathcal{N}} Q_n$ is the set of *system states*, $q_0 = (q_{0,n})_{n \in \mathcal{N}}$ is the *initial system state*, $\Lambda(\mathcal{S})$ is the set of *system labels*, and $E(\mathcal{S})$ is the set of *system transitions*. Each system state $q \in Q$ is an $\mathcal{N}$-indexed family $(q_n)_{n \in \mathcal{N}}$ of local component states $q_n \in Q_n$. The definitions of $\Lambda(\mathcal{S})$ and $E(\mathcal{S})$ follow below, after the intermediate notion of *system action*.

System actions Σ. The set of *system actions* $\Sigma = \bigcup_{n\in\mathcal{N}} \Sigma_n$ determines actions that will be part of system labels. Within Σ we identify $\Sigma^\bullet = \bigcup_{n\in\mathcal{N}} \Sigma_n^? \cap \bigcup_{n\in\mathcal{N}} \Sigma_n^!$ as the set of *communicating actions*. Hence, an action $a \in \Sigma$ is communicating if it occurs in (at least) one set Σ_n of action labels as an input action and in (at least) one set Σ_m of action labels as an output action. The system is *closed* if all non-communicating actions are internal component actions. For ease of presentation, we assume in this paper that systems are closed.

Example 3. The system actions of the race system are $\Sigma_{\mathsf{Race}} = \{\mathit{start}, \mathit{finish}, \mathit{run}\}$ and its communicating actions are $\Sigma^\bullet_{\mathsf{Race}} = \{\mathit{start}, \mathit{finish}\}$. ▷

System labels $\Lambda(\mathcal{S})$. We use *system labels* to indicate which components participate (simultaneously) in the execution of a system action. There are two kinds of system labels. In a system label of the form $(\mathsf{out}, a, \mathsf{in})$, out represents the set of senders of *outputs* and in the set of receivers of *inputs* that synchronise on the action $a \in \Sigma^\bullet$. Either out or in can be empty, but not both. A system label of the form (n, a) indicates that component n executes an internal action $a \in \Sigma_n^\tau$. Formally, the set $\Lambda(\mathcal{S})$ of system labels of $\mathcal{S}$ is defined as follows:

$$\begin{aligned}\Lambda(\mathcal{S}) = {} & \{\, (\mathsf{out}, a, \mathsf{in}) \mid \varnothing \neq (\mathsf{out} \cup \mathsf{in}) \subseteq \mathcal{N},\ \forall_{n\in\mathsf{out}} \cdot a \in \Sigma_n^!,\ \forall_{n\in\mathsf{in}} \cdot a \in \Sigma_n^? \,\} \\ & \cup \{\, (n, a) \mid n \in \mathcal{N},\ a \in \Sigma_n^\tau \,\}\end{aligned}$$

Note that $\Lambda(\mathcal{S})$ depends only on $\mathcal{N}$ and the sets Σ_n of action labels for each $n \in \mathcal{N}$. As a notational convention, if $\mathsf{out} = \{n\}$ is a singleton, we write (n, a, in) instead of $(\{n\}, a, \mathsf{in})$, and similarly for singleton sets in.

Example 4. The set of system labels of the race system is given by

$$\begin{aligned}\Lambda(\mathsf{Race}) = {} & \{(\mathsf{out}, \mathit{start}, \mathsf{in}) \mid \varnothing \neq (\mathsf{out} \cup \mathsf{in}),\ \mathsf{out} \subseteq \{\mathsf{Ctrl}\}, \mathsf{in} \subseteq \{\mathsf{R1}, \mathsf{R2}\}\}, \\ & \cup \{(\mathsf{out}, \mathit{finish}, \mathsf{in}) \mid \varnothing \neq (\mathsf{out} \cup \mathsf{in}),\ \mathsf{out} \subseteq \{\mathsf{R1}, \mathsf{R2}\}, \mathsf{in} \subseteq \{\mathsf{Ctrl}\}\}, \\ & \cup \{(\mathsf{R1}, \mathit{run}), (\mathsf{R2}, \mathit{run})\}.\end{aligned}$$ ▷

System transitions $E(\mathcal{S})$. System labels provide an appropriate means to describe which components in a system execute, possibly together, a computation step, i.e. a system transition. Formally, a *system transition* $t \in E(\mathcal{S})$ has the form $(q_n)_{n\in\mathcal{N}} \xrightarrow{\lambda}_{\mathsf{lts}(\mathcal{S})} (q'_n)_{n\in\mathcal{N}}$ such that $\lambda \in \Lambda(\mathcal{S})$ and

- either $\lambda = (\mathsf{out}, a, \mathsf{in})$ and:
  - $q_n \xrightarrow{a}_{\mathcal{A}_n} q'_n$ for all $n \in \mathsf{out} \cup \mathsf{in}$ and
  - $q_m = q'_m$ for all $m \in \mathcal{N} \backslash (\mathsf{out} \cup \mathsf{in})$;
- or $\lambda = (n, a)$, $a \in \Sigma_n^\tau$ is an internal action of some component $n \in \mathcal{N}$, and:
  - $q_n \xrightarrow{a}_{\mathcal{A}_n} q'_n$ and
  - $q_m = q'_m$ for all $m \in \mathcal{N} \backslash \{n\}$.

We write Λ and E instead of $\Lambda(\mathcal{S})$ and $E(\mathcal{S})$, resp., if $\mathcal{S}$ is clear from the context. Surely, at most those components that are in a local state in which action a is locally enabled can participate in a system transition for a. Since, by definition of system labels, $(\mathsf{out} \cup \mathsf{in}) \neq \varnothing$, at least one component participates in any system transition. Given a system transition $t = q \xrightarrow{\lambda}_{\mathbf{lts}(\mathcal{S})} q'$, we write $t.\lambda$ for λ.

Example 5. Examples of system transitions of the race system are

$(0,0,0) \xrightarrow{(\mathsf{Ctrl}, start, \varnothing)} (0,0,1)$, $(0,0,0) \xrightarrow{(\mathsf{Ctrl}, start, \{\mathsf{R1},\mathsf{R2}\})} (1,1,1)$,
$(2,2,1) \xrightarrow{(\{\mathsf{R1},\mathsf{R2}\}, finish, \mathsf{Ctrl})} (0,0,2)$, $(2,2,1) \xrightarrow{(\mathsf{R1}, finish, \mathsf{Ctrl})} (0,2,2)$, and
$(1,1,1) \xrightarrow{(\mathsf{R1}, run)} (2,1,1)$.

The LTS of the race system, denoted by $\mathbf{lts}(\mathsf{Race})$, contains all possible system transitions. It can be computed by our tool as shown in Sect. 4.

Note that not all system transitions are really meaningful. For instance, the first transition should not happen, since the controller is supposed to start both runners simultaneously. We also want to reject the third transition, since in our application runners should finish individually. These transitions will be discarded based on synchronisation restrictions for teams considered in the following. ▷

2.2 Team Automata

Synchronisation types specify which synchronisations between components are admissible in a particular system $\mathcal{S}$. A *synchronisation type* $(O, I) \in \mathsf{Intv} \times \mathsf{Intv}$ is a pair of intervals O and I which determine the number of outputs and inputs that can participate in a communication. Each interval has the form $[min, max]$ with $min \in \mathbb{N}$ and $max \in \mathbb{N} \cup \{*\}$ where $*$ denotes 0 or more participants. We write $x \in [min, max]$ if $min \leq x \leq max$ and $x \in [min, *]$ if $x \geq min$.

A *synchronisation type specification* (STS) over $\mathcal{S}$ is a function $\mathbf{st} : \Sigma^{\bullet} \rightarrow \mathsf{Intv} \times \mathsf{Intv}$ that assigns to any communicating action a an individual synchronisation type $\mathbf{st}(a)$. We say that a system label $\lambda = (\mathsf{out}, a, \mathsf{in})$ *satisfies* $\mathbf{st}(a) = (O, I)$, written $\lambda \models \mathbf{st}(a)$, if $|\mathsf{out}| \in O \wedge |\mathsf{in}| \in I$. Each synchronisation type specification $\mathbf{st}$ generates the following subsets $\Lambda(\mathcal{S}, \mathbf{st})$ of system labels and $E(\mathcal{S}, \mathbf{st})$ of corresponding system transitions.

$$\Lambda(\mathcal{S}, \mathbf{st}) = \{\, \lambda \in \Lambda \mid \lambda = (\mathsf{out}, a, \mathsf{in}) \Rightarrow \lambda \models \mathbf{st}(a) \,\}$$
$$E(\mathcal{S}, \mathbf{st}) = \{\, t \in E \mid t.\lambda \in \Lambda(\mathcal{S}, \mathbf{st}) \,\}$$

Thus, for communicating actions, the set of system transitions is restricted to those transitions whose labels respect the synchronisation type of their communicating action. For internal actions no restriction is applied, since an internal action of a component can always be executed when it is locally enabled.

Components interacting in accordance with an STS $\mathbf{st}$ over a system $\mathcal{S}$ are seen as a team whose behaviour is represented by the *(extended) team automaton* (ETA) $\mathbf{eta}(\mathcal{S}, \mathbf{st})$ generated over $\mathcal{S}$ by $\mathbf{st}$ and defined by the LTS

$$\mathbf{eta}(\mathcal{S}, \mathbf{st}) = (Q, q_0, \Lambda(\mathcal{S}, \mathbf{st}), E(\mathcal{S}, \mathbf{st})).$$

We write $\Lambda(\mathbf{st})$ and $E(\mathbf{st})$ instead of $\Lambda(\mathcal{S}, \mathbf{st})$ and $E(\mathcal{S}, \mathbf{st})$, resp., if $\mathcal{S}$ is clear from the context, and assume $\Lambda(\mathbf{st}) \neq \varnothing$. Labels in $\Lambda(\mathbf{st})$ are called *team labels* and transitions in $E(\mathbf{st})$ are called *team transitions*.

Example 6. Recall the race system and its system labels and transitions. We require both runners to *start* simultaneously and to *finish* individually by using the STS $\mathbf{st}_{\mathsf{Race}}$ defined by $start \mapsto ([1,1],[2,2])$ and $finish \mapsto ([1,1],[1,1])$. Then the team labels of the ETA $\mathbf{eta}(\mathsf{Race}, \mathbf{st}_{\mathsf{Race}})$ are given by $\Lambda(\mathbf{st}_{\mathsf{Race}}) = \{\, (\mathsf{Ctrl}, start, \{\mathsf{R1}, \mathsf{R2}\}),\ (\mathsf{R1}, finish, \mathsf{Ctrl}),\ (\mathsf{R2}, finish, \mathsf{Ctrl}), (\mathsf{R1}, run),\ (\mathsf{R2}, run) \,\}$. Example transitions are

$$(0,0,0) \xrightarrow{(\mathsf{Ctrl}, start, \{\mathsf{R1},\mathsf{R2}\})} (1,1,1) \xrightarrow{(\mathsf{R1}, run)} (2,1,1) \xrightarrow{(\mathsf{R1}, finish, \mathsf{Ctrl})} (0,1,2).$$

The full team automaton is computed by our tool, cf. Appendix A. ▷

2.3 Dynamic Logic

We use a (test-free) propositional dynamic logic over a finite set $A \neq \varnothing$ of atomic actions [31]. The set $Act(A)$ of *structured actions* over A is given by the grammar

$$\alpha \ ::= \ a \mid \alpha;\alpha \mid \alpha + \alpha \mid \alpha^* \qquad \text{(actions)}$$

with $a \in A$, sequential composition ;, nondeterministic choice +, and iteration $*$.

Abbreviations If $A = \{a_1, \ldots, a_n\}$, we write *some* for the structured action $a_1 + \cdots + a_n$. Given a nonempty subset of A denoted by B with elements $\{b_1, \ldots, b_m\}$, we write B for the structured action $b_1 + \cdots + b_m$.

The set $Frm(A)$ of *formulas* over A is defined by the grammar

$$\varphi \ ::= \ true \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\alpha\rangle\,\varphi \qquad \text{(formulas)}$$

where $\alpha \in Act(A)$. Formula $\langle\alpha\rangle\,\varphi$ expresses that at the current state it is possible to execute α such that φ holds in the next state. The difference to Hennessy–Milner logic [32] is that actions used as modalities in modal operators can be structured actions, including iteration. This additional power will be crucial to express our communication requirements later on in terms of logic formulas.

Abbreviations We use the usual abbreviations like *false*, $\varphi \wedge \varphi'$, $\varphi \to \varphi'$, and the modal box operator $[\alpha]\,\varphi$ which stands for $\neg\langle\alpha\rangle\,\neg\varphi$ and expresses that whenever in the current state α is executed, then φ holds afterwards. For a finite index set I, we write $\bigvee_{i\in I}$ to denote the generalised '$\vee$', where $\bigvee_{i\in\varnothing}\psi_i = false$ (likewise $\bigwedge_{i\in\varnothing}\psi_i = true$).

Given a set A of atomic actions, we use LTS over A for the semantic interpretation of formulas. Let $\mathcal{L} = (Q, q_0, A, E)$ be an LTS. First we extend the transition relation of $\mathcal{L}$ to structured actions in $Act(A)$ defined inductively by:

$q \xrightarrow{\alpha_1+\alpha_2}_{\mathcal{L}} q'$ if $q \xrightarrow{\alpha_1}_{\mathcal{L}} q'$ or $q \xrightarrow{\alpha_2}_{\mathcal{L}} q'$,
$q \xrightarrow{\alpha_1;\alpha_2}_{\mathcal{L}} q'$ if there exists $\hat{q} \in Q$ such that $q \xrightarrow{\alpha_1}_{\mathcal{L}} \hat{q}$ and $\hat{q} \xrightarrow{\alpha_2}_{\mathcal{L}} q'$,

$q \xrightarrow{\alpha^*}_{\mathcal{L}} q'$ if $q = q'$ or there exists $\hat{q} \in Q$ such that $q \xrightarrow{\alpha}_{\mathcal{L}} \hat{q}$ and $\hat{q} \xrightarrow{\alpha^*}_{\mathcal{L}} q'$.
We write $q \xrightarrow{\alpha}_{\mathcal{L}}$ if there exists q' such that $q \xrightarrow{\alpha}_{\mathcal{L}} q'$.

The *satisfaction* of a formula $\varphi \in \mathit{Frm}(A)$ by an LTS $\mathcal{L} = (Q, q_0, A, E)$ at a state $q \in Q$, written $\mathcal{L}, q \models \varphi$, is inductively defined as follows:

$\mathcal{L}, q \models \mathit{true}$,
$\mathcal{L}, q \models \neg\varphi$ if not $\mathcal{L}, q \models \varphi$,
$\mathcal{L}, q \models \varphi_1 \vee \varphi_2$ if $\mathcal{L}, q \models \varphi_1$ or $\mathcal{L}, q \models \varphi_2$,
$\mathcal{L}, q \models \langle\alpha\rangle\, \varphi$ if there exists $q' \in Q$ such that $q \xrightarrow{\alpha}_{\mathcal{L}} q'$ and $\mathcal{L}, q' \models \varphi$.

For instance, enabledness $q \xrightarrow{\alpha}_{\mathcal{L}}$ is expressed by $\mathcal{L}, q \models \langle\alpha\rangle\, \mathit{true}$.

$\mathcal{L}$ *satisfies* a formula $\varphi \in \mathit{Frm}(A)$, written $\mathcal{L} \models \varphi$, if $\mathcal{L}, q_0 \models \varphi$. Hence, for the satisfaction of a formula by an LTS the non-reachable states are irrelevant.

We deviate from the classical semantics [31], since we use LTS with initial states as models to interpret satisfaction of formulas. This is because we are interested in the formulation of properties of (concurrently running) components, i.e. of process structures. In particular, we can express safety properties (e.g. $[\mathit{some}^*]\, \varphi$) and some kinds of liveness properties (e.g. $[\mathit{some}^*]\, \langle \mathit{some}^*; a\rangle\, \varphi$).

3 Logical Characterisations of Communication Properties

In this section, we first focus on the property of *receptiveness* for team automata, which has been studied before for other automata formalisms mainly in the context of peer-to-peer communication; cf. Introduction. In Sect. 3.1, we summarise the concepts of receptiveness and weak receptiveness and in Sect. 3.2 we show that both notions can be characterised by dynamic logic formulas. Then we turn to (weak) responsiveness, summarising the underlying ideas in Sect. 3.3 and providing logical characterisations in Sect. 3.4. The results form the theoretical basis for automatic checks of communication properties in Sect. 4.

We assume a given system $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_n)_{n\in\mathcal{N}})$ of CA with $\mathbf{lts}(\mathcal{S}) = (Q, q_0, \Lambda, E)$, an STS $\mathbf{st}$, and the generated ETA $\mathbf{eta}(\mathcal{S}, \mathbf{st}) = (Q, q_0, \Lambda(\mathcal{S}, \mathbf{st}), E(\mathcal{S}, \mathbf{st}))$.

3.1 Team Receptiveness

The idea of receptiveness for $\mathbf{eta}(\mathcal{S}, \mathbf{st})$ is as follows. Whenever, in a reachable state q of $\mathbf{eta}(\mathcal{S}, \mathbf{st})$, a group $\{\mathcal{A}_n \mid n \in \mathsf{out}\}$ of CA with $\varnothing \neq \mathsf{out} \subseteq \mathcal{N}$ is (locally) enabled to perform an output action a, i.e. $\forall_{n\in\mathsf{out}} \cdot a \in \Sigma_n^!$ and $q_n \xrightarrow{a}_{\mathcal{A}_n}$, so that (1) the number of CA in out fits the number of allowed senders according to the synchronisation type $\mathbf{st}(a) = (O, I)$, i.e. $|\mathsf{out}| \in O$, and (2) the CA need at least one receiver to join the communication, i.e. $0 \notin I$, we get a *receptiveness requirement*, denoted by $\mathbf{rcp}(\mathsf{out}, a)@q$. If $\mathsf{out} = \{n\}$, we write $\mathbf{rcp}(n, a)@q$ for $\mathbf{rcp}(\{n\}, a)@q$.

Example 7. In the initial state $(0, 0, 0)$ of the race team, there is a receptiveness requirement of the controller who wants to start the competition, expressed by

$\mathsf{rcp}(\mathsf{Ctrl}, \mathit{start})@(0,0,0)$. Later on, when the first runner is in state 2, it wants to send *finish* which leads to three receptiveness requirements:

$$\mathsf{rcp}(\mathsf{R1}, \mathit{finish})@(2,1,1),\ \mathsf{rcp}(\mathsf{R1}, \mathit{finish})@(2,2,1),\ \mathsf{rcp}(\mathsf{R1}, \mathit{finish})@(2,0,2).$$

Similarly, when the second runner is in state 2, we get:

$$\mathsf{rcp}(\mathsf{R2}, \mathit{finish})@(1,2,1),\ \mathsf{rcp}(\mathsf{R2}, \mathit{finish})@(2,2,1),\ \mathsf{rcp}(\mathsf{R2}, \mathit{finish})@(0,2,2).\ \triangleright$$

ETA $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ is compliant with a receptiveness requirement $\mathsf{rcp}(\mathsf{out}, a)@q$ if the group of components (with names in out) can find partners in the team which synchronise with the group by taking (receiving) a as input. If reception is immediate, we talk about receptiveness; if the other components may still perform some intermediate actions before accepting a, we talk about weak receptiveness. Formally, (weak) compliance and (weak) receptiveness are defined as follows: The ETA $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ is *compliant* with $\mathsf{rcp}(\mathsf{out}, a)@q$ if

$$\exists_{\mathsf{in}} \cdot q \xrightarrow{(\mathsf{out}, a, \mathsf{in})}_{\mathsf{eta}(\mathcal{S}, \mathsf{st})}$$

The ETA $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ is *weakly compliant* with $\mathsf{rcp}(\mathsf{out}, a)@q$ if

$$\exists_{\mathsf{in}} \cdot q \xrightarrow{(\Lambda(\mathsf{st})_{\setminus \mathsf{out}})^*\ ;\ (\mathsf{out}, a, \mathsf{in})}_{\mathsf{eta}(\mathcal{S}, \mathsf{st})}$$

where $\Lambda(\mathsf{st})_{\setminus \mathsf{out}}$ denotes the set of team labels in which no component of out participates. Formally, $\Lambda(\mathsf{st})_{\setminus \mathsf{out}} = \{(\mathsf{out}', a, \mathsf{in}) \in \Lambda(\mathsf{st}) \mid (\mathsf{out}' \cup \mathsf{in}) \cap \mathsf{out} = \varnothing\} \cup \{(n, a) \in \Lambda(\mathsf{st}) \mid n \notin \mathsf{out}\}$. Obviously, compliance implies weak compliance.

Definition 1 ((weak) receptiveness). *The ETA* $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ *is* (weakly) receptive *if for all reachable states* $q \in \mathcal{R}(\mathsf{eta}(\mathcal{S}, \mathsf{st}))$*, the ETA* $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ *is (weakly) compliant with* all *receptiveness requirements* $\mathsf{rcp}(\mathsf{out}, a)@q$ *established for* q.

3.2 Logical Characterisations of Receptiveness

Receptiveness notions are of purely semantic nature. To prove receptiveness in concrete cases may be rather cumbersome since one has to go through all reachable states q of a team automaton and check compliance for all receptiveness requirements at q. Therefore we are interested in a syntactic, logical characterisation of receptiveness such that checks can be automated. It turns out that our version of dynamic logic is well suited to express receptiveness.

Example 8. Recall the receptiveness requirement $\mathsf{rcp}(\mathsf{Ctrl}, \mathit{start})@(0,0,0)$ from Example 7. Being a receptiveness requirement implies that the output action *start* is enabled at the local state 0 of the controller, i.e. $0 \xrightarrow{\mathit{start}}_{\mathcal{A}_{\mathsf{Ctrl}}}$. This is equivalent to the fact that in $\mathsf{lts}(\mathsf{Race})$ (cf. Example 5) the *system label* $(\mathsf{Ctrl}, \mathit{start}, \varnothing)$ is enabled at system state $(0,0,0)$, i.e. $(0,0,0) \xrightarrow{\mathit{start}}_{\mathsf{lts}(\mathsf{Race})}$. *Logically*, this is equivalent to $\mathsf{lts}(\mathsf{Race}), (0,0,0) \models \langle(\mathsf{Ctrl}, \mathit{start}, \varnothing)\rangle\, \mathit{true}$. Under this condition, we must prove there is a team transition in the ETA $\mathsf{eta}(\mathsf{Race}, \mathsf{st}_{\mathsf{Race}})$ of the form $(0,0,0) \xrightarrow{(\mathsf{Ctrl}, \mathit{start}, \mathsf{in})}_{\mathsf{eta}(\mathsf{Race}, \mathsf{st}_{\mathsf{Race}})} q'$. This means there is an in so that $(\mathsf{Ctrl}, \mathit{start}, \mathsf{in})$ is a *team label* and $\mathsf{eta}(\mathsf{Race}, \mathsf{st}_{\mathsf{Race}}), (0,0,0) \models \langle(\mathsf{Ctrl}, \mathit{start}, \mathsf{in})\rangle\, \mathit{true}$. The latter is equivalent to $\mathbf{lts}(\mathsf{Race}), (0,0,0) \models \langle(\mathsf{Ctrl}, \mathit{start}, \mathsf{in})\rangle\, \mathit{true}$ since, for team labels, system transitions and team transitions coincide. To check that $\mathbf{eta}(\mathsf{Race}, \mathbf{st}_{\mathsf{Race}})$ satisfies the (only) receptiveness requirement at state $(0,0,0)$ it thus suffices (and it is also necessary) to show that there is an in with $(\mathsf{Ctrl}, \mathit{start}, \mathsf{in})$ being a team label such that the following holds (which is true for $\mathsf{in} = \{\mathsf{R1}, \mathsf{R2}\}$): $\mathbf{lts}(\mathsf{Race}), (0,0,0) \models \langle(\mathsf{Ctrl}, \mathit{start}, \varnothing)\rangle\, \mathit{true} \rightarrow \langle(\mathsf{Ctrl}, \mathit{start}, \mathsf{in})\rangle\, \mathit{true}$. ▷

This example illustrates a key insight in our approach: we cannot capture requests for communication on team level but must consider system transitions with *system labels* which are not team labels, e.g. $(\mathsf{Ctrl}, \mathit{start}, \varnothing)$.

Our general approach to characterise receptiveness properties is as follows. Given system labels $\Lambda(\mathcal{S})$ and synchronisation type specification $\mathbf{st}$ the *"receptiveness formula"* $\mathit{rcpFrm} \in \mathit{Frm}(\Lambda)$ defined below expresses that all receptiveness requirements are fulfilled in any reachable state of the team $\mathbf{eta}(\mathcal{S}, \mathbf{st})$:

$$\begin{aligned} \mathit{rcpReq} &= \{(\mathsf{out}, a, \varnothing) \in \Lambda \mid |\mathsf{out}| \in O, 0 \notin I \text{ for } \mathbf{st}(a) = (O, I)\} \\ \mathit{InCom}(\mathsf{out}, a) &= \{\mathsf{in} \subseteq \mathcal{N} \mid (\mathsf{out}, a, \mathsf{in}) \in \Lambda(\mathbf{st})\} \\ \mathit{rcpFrm} &= [\Lambda(\mathbf{st})^*] \textstyle\bigwedge_{(\mathsf{out}, a, \varnothing) \in \mathit{rcpReq}} \\ &\qquad \big(\langle(\mathsf{out}, a, \varnothing)\rangle\, \mathit{true} \rightarrow \textstyle\bigvee_{\mathsf{in} \in \mathit{InCom}(\mathsf{out}, a)} \langle(\mathsf{out}, a, \mathsf{in})\rangle\, \mathit{true} \big) \end{aligned}$$

Here rcpReq is the set of system labels which correspond to receptiveness requirements (when enabled in a reachable state of the ETA, cf. Lemma 1); and $\mathit{InCom}(\mathsf{out}, a)$ is the set of subsets $\mathsf{in} \subseteq \mathcal{N}$ of component names which complement a given $\mathsf{out} \subseteq \mathcal{N}$ and $a \in \Sigma^{\bullet}$ to a team label in $\Lambda(\mathbf{st})$ (for potential communication). Observe that (1) $\mathit{rcpReq} \cap \Lambda(\mathbf{st}) = \varnothing$ since $0 \notin I$ for any $\mathbf{st}(a) = (O, I)$; (2) $[\Lambda(\mathbf{st})^*]$ ranges over all reachable states of the team $\mathbf{eta}(\mathcal{S}, \mathbf{st})$, since $\Lambda(\mathbf{st})$ is the finite set of team labels that denote the non-deterministic choice of these actions; and (3) the implication in rcpFrm is in $\mathit{Frm}(\Lambda)$ and not in $\mathit{Frm}(\Lambda(\mathbf{st}))$ since $\mathit{rcpReq} \cap \Lambda(\mathbf{st}) = \varnothing$ and $(\mathsf{out}, a, \varnothing) \in \mathit{rcpReq}$.

Similarly, a *"weak receptiveness formula"* $\mathit{wrcpFrm} \in \mathit{Frm}(\Lambda)$ is defined as:

$$\begin{aligned} \mathit{wrcpFrm} = [\Lambda(\mathbf{st})^*] &\textstyle\bigwedge_{(\mathsf{out}, a, \varnothing) \in \mathit{rcpReq}} \\ &\big(\langle(\mathsf{out}, a, \varnothing)\rangle\, \mathit{true} \rightarrow \textstyle\bigvee_{\mathsf{in} \in \mathit{InCom}(\mathsf{out}, a)} \big\langle (\Lambda(\mathbf{st})_{\setminus \mathsf{out}})^*; (\mathsf{out}, a, \mathsf{in}) \big\rangle\, \mathit{true} \big) \end{aligned}$$

Example 9. For Race, $\mathit{rcpReq} = \{(\mathsf{Ctrl}, \mathit{start}, \varnothing), (\mathsf{R1}, \mathit{finish}, \varnothing), (\mathsf{R2}, \mathit{finish}, \varnothing)\}$, $\mathit{InCom}(\mathsf{Ctrl}, \mathit{start}) = \{\{\mathsf{R1}, \mathsf{R2}\}\}$, $\mathit{InCom}(\mathsf{Ri}, \mathit{finish}) = \{\{\mathsf{Ctrl}\}\}$, for $i = 1, 2$, and

$$\begin{aligned} \mathit{rcpFrm} = [\Lambda(\mathbf{st}_{\mathsf{Race}})^*] \big(&\langle(\mathsf{Ctrl}, \mathit{start}, \varnothing)\rangle\, \mathit{true} \rightarrow \langle(\mathsf{Ctrl}, \mathit{start}, \{\mathsf{R1}, \mathsf{R2}\})\rangle\, \mathit{true} \\ \wedge\, &\langle(\mathsf{R1}, \mathit{finish}, \varnothing)\rangle\, \mathit{true} \rightarrow \langle(\mathsf{R1}, \mathit{finish}, \mathsf{Ctrl})\rangle\, \mathit{true} \\ \wedge\, &\langle(\mathsf{R2}, \mathit{finish}, \varnothing)\rangle\, \mathit{true} \rightarrow \langle(\mathsf{R2}, \mathit{finish}, \mathsf{Ctrl})\rangle\, \mathit{true} \big). \end{aligned}$$

This receptiveness formula is satisfied by the LTS of the Race system. For the check we use the tool described in Sect. 4. Together with Theorem 1 below this implies that the ETA $\mathbf{eta}(\mathsf{Race}, \mathbf{st}_{\mathsf{Race}})$ is receptive. ▷

The next lemma provides a characterisation of receptiveness requirements in terms of the set rcpReq and logical satisfaction (used for the proof of Thm. 1).

Lemma 1. *For all* $q \in \mathcal{R}(\mathbf{eta}(\mathcal{S}, \mathbf{st}))$ *it holds:* $\mathbf{rcp}(\mathsf{out}, a)@q$ *is a receptiveness requirement iff* $(\mathsf{out}, a, \varnothing) \in rcpReq$ *and* $\mathbf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}, a, \varnothing)\rangle\, true$.

Proof. "⇒": Let $q \in \mathcal{R}(\mathbf{eta}(\mathcal{S}, \mathbf{st}))$ and $\mathbf{rcp}(\mathsf{out}, a)@q$ be a receptiveness requirement. Then, for all $n \in \mathsf{out}$, we have $a \in \Sigma^!_n$, and, for $\mathbf{st}(a) = (O, I)$, it holds $|\mathsf{out}| \in O, 0 \notin I$. Hence, $(\mathsf{out}, a, \varnothing) \in \Lambda$ and, in particular, $(\mathsf{out}, a, \varnothing) \in rcpReq$. Moreover, we know that for all $n \in \mathsf{out}$, $q_n \xrightarrow{a}_{\mathcal{A}_n}$. Therefore $q \xrightarrow{(\mathsf{out},a,\varnothing)}_{\mathbf{lts}(\mathcal{S})}$ holds and thus $\mathbf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}, a, \varnothing)\rangle\, true$.

"⇐": Conversely, let $q \in \mathcal{R}(\mathbf{eta}(\mathcal{S}, \mathbf{st}))$ and $(\mathsf{out}, a, \varnothing) \in rcpReq$. Then we have $(\mathsf{out}, a, \varnothing) \in \Lambda$ and thus $a \in \Sigma^!_n$ for all $n \in \mathsf{out}$. Moreover, we know that $|\mathsf{out}| \in O, 0 \notin I$ for $\mathbf{st}(a) = (O, I)$. For $\mathbf{rcp}(\mathsf{out}, a)@q$ to be a receptiveness requirement it remains to show that $q_n \xrightarrow{a}_{\mathcal{A}_n}$ holds for all $n \in \mathsf{out}$. The latter is equivalent to $q \xrightarrow{(\mathsf{out},a,\varnothing)}_{\mathbf{lts}(\mathcal{S})}$ which is in turn equivalent to the assumption $\mathbf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}, a, \varnothing)\rangle\, true$. □

The proof of Theorem 1 uses also the facts stated in the following two lemmas.

Lemma 2. *For all* $\varphi \in Frm(\Lambda)$*:*

$$\{\mathbf{lts}(\mathcal{S}) \models [\Lambda(\mathbf{st})^*]\, \varphi\} \quad iff \quad \{\mathbf{lts}(\mathcal{S}), q \models \varphi \text{ for all } q \in \mathcal{R}(\mathbf{eta}(\mathcal{S}, \mathbf{st}))\}.$$

Lemma 3. *For all* $q \in \mathcal{R}(\mathbf{eta}(\mathcal{S}, \mathbf{st}))$ *and* $\alpha \in Act(\Lambda(\mathbf{st}))$: $q \xrightarrow{\alpha}_{\mathbf{lts}(\mathcal{S})}$ *iff* $q \xrightarrow{\alpha}_{\mathbf{eta}(\mathcal{S},\mathbf{st})}$.[5]

Theorem 1. *(1)* $\mathbf{eta}(\mathcal{S}, \mathbf{st})$ *is receptive iff* $\mathbf{lts}(\mathcal{S}) \models rcpFrm$ *and*
(2) $\mathbf{eta}(\mathcal{S}, \mathbf{st})$ *is weakly receptive iff* $\mathbf{lts}(\mathcal{S}) \models wrcpFrm$.

Proof. We only prove (1). The proof of (2) is a straightforward extension.

"⇐": Let $q \in \mathcal{R}(\mathbf{eta}(\mathcal{S}, \mathbf{st}))$. If there is no receptiveness requirement at q there is nothing to prove. Now let $\mathbf{rcp}(\mathsf{out}', a')@q$ be a receptiveness requirement established for q. By Lemma 1 ("⇒"), $(\mathsf{out}', a', \varnothing) \in rcpReq$ and $\mathbf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}', a', \varnothing)\rangle\, true$. By assumption $\mathbf{lts}(\mathcal{S}) \models rcpFrm$. Hence, by Lemma 2 ("⇒"),

$$\mathbf{lts}(\mathcal{S}), q \models \bigwedge_{(\mathsf{out},a,\varnothing)\in rcpReq} \Big(\langle(\mathsf{out}, a, \varnothing)\rangle\, true \rightarrow \bigvee_{\mathsf{in}\in InCom(\mathsf{out},a)} \langle(\mathsf{out}, a, \mathsf{in})\rangle\, true \Big).$$

Since $(\mathsf{out}', a', \varnothing) \in rcpReq$ this implies

$$\mathbf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}', a', \varnothing)\rangle\, true \rightarrow \bigvee_{\mathsf{in}\in InCom(\mathsf{out}',a')} \langle(\mathsf{out}', a', \mathsf{in})\rangle\, true.$$

As a consequence, since $\mathbf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}', a', \varnothing)\rangle\, true$, we get

$$\mathbf{lts}(\mathcal{S}), q \models \bigvee_{\mathsf{in}\in InCom(\mathsf{out}',a')} \langle(\mathsf{out}', a', \mathsf{in})\rangle\, true.$$

Hence, there exists $\mathsf{in} \in InCom(\mathsf{out}', a')$ such that $\mathbf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}', a', \mathsf{in})\rangle\, true$ and therefore, by definition of $InCom(\mathsf{out}', a')$, we have $(\mathsf{out}', a', \mathsf{in}) \in \mathbf{st}(\Lambda)$ and $q \xrightarrow{(\mathsf{out}',a',\mathsf{in})}_{\mathbf{lts}(\mathcal{S})}$. So, by Lemma 3, $\exists_{\mathsf{in}} \cdot q \xrightarrow{(\mathsf{out}',a',\mathsf{in})}_{\mathbf{eta}(\mathcal{S},\mathbf{st})}$. Hence, $\mathbf{eta}(\mathcal{S}, \mathbf{st})$

[5] This follows because for team labels system transitions and team transitions coincide.

is compliant with $\mathsf{rcp}(\mathsf{out}', a')@q$. Since $\mathsf{rcp}(\mathsf{out}', a')@q$ was chosen arbitrarily, $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ is receptive.

"⇒": The proof is by contradiction. Assume $\mathsf{lts}(\mathcal{S}) \not\models \mathit{rcpFrm}$. By definition, rcpFrm has the form $[\Lambda(\mathsf{st})^*]\,\varphi$ with $\varphi \in \mathit{Frm}(\Lambda)$. Hence, by Lemma 2 ("⇐"), there exists a state $q \in \mathcal{R}(\mathsf{eta}(\mathcal{S}, \mathsf{st}))$ such that $\mathsf{lts}(\mathcal{S}), q \not\models \varphi$. Hence, φ cannot be true and then, again by definition of rcpFrm, there exists $(\mathsf{out}, a, \varnothing) \in \mathit{rcpReq}$ such that $\mathsf{lts}(\mathcal{S}), q \not\models \langle(\mathsf{out}, a, \varnothing)\rangle \mathit{true} \to \bigvee_{\mathsf{in} \in \mathit{InCom}(\mathsf{out},a)} \langle(\mathsf{out}, a, \mathsf{in})\rangle \mathit{true}$. Thus, $\mathsf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}, a, \varnothing)\rangle\ \mathit{true}$ and $\mathsf{lts}(\mathcal{S}), q \not\models \bigvee_{\mathsf{in} \in \mathit{InCom}(\mathsf{out},a)} \langle(\mathsf{out}, a, \mathsf{in})\rangle\ \mathit{true}$. According to Lemma 1 ("⇐") the first part, together with $(\mathsf{out}, a, \varnothing) \in \mathit{rcpReq}$, implies that $\mathsf{rcp}(\mathsf{out}, a)@q$ is a receptiveness requirement. The second part implies that there is no $\mathsf{in} \in \mathit{InCom}(\mathsf{out}, a)$ such that $\mathsf{lts}(\mathcal{S}), q \models \langle(\mathsf{out}, a, \mathsf{in})\rangle\ \mathit{true}$, which would mean $q \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathsf{lts}(\mathcal{S})}$. Hence, by Lemma 3, there is no team label $(\mathsf{out}, a, \mathsf{in})$ such that $q \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathsf{eta}(\mathcal{S},\mathsf{st})}$. Therefore, $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ is not compliant with $\mathsf{rcp}(\mathsf{out}, a)@q$ and thus $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ is not receptive. □

Remark 1. Checks of $\mathsf{lts}(\mathcal{S}) \models \mathit{rcpFrm}$ ($\mathit{wrcpFrm}$, resp.) can be optimised if we use instead of the full LTS of $\mathcal{S}$ the usually much smaller sub-LTS $\mathsf{lts}(\mathcal{S})^{opt} \subseteq \mathsf{lts}(\mathcal{S})$ constructed as follows: the set of transitions of $\mathsf{lts}(\mathcal{S})^{opt}$ consists of the transitions of $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ to which we add all transitions $q \xrightarrow{(\mathsf{out},a,\varnothing)}_{\mathsf{lts}(\mathcal{S})} q'$ with $(\mathsf{out}, a, \varnothing) \in \mathit{rcpReq}$. These transitions, which do not belong to $\mathsf{eta}(\mathcal{S}, \mathsf{st})$, are needed to capture receptiveness requirements. ▷

3.3 Team Responsiveness

For input actions, one can formulate responsiveness requirements with the intuition that enabled inputs should be served by appropriate outputs. The expression $\mathsf{rsp}(\mathsf{in}, a)@q$ is a *responsiveness requirement* if $q \in \mathcal{R}(\mathsf{eta}(\mathcal{S}, \mathsf{st}))$, for all $n \in \mathsf{in}$ we have $a \in \Sigma^?_n$ and $q_n \xrightarrow{a}_{\mathcal{A}_n}$, and $|\mathsf{in}| \in I, 0 \notin O$ for $\mathsf{st}(a) = (O, I)$. The ETA $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ is *compliant* with $\mathsf{rsp}(\mathsf{in}, a)@q$ if $\exists_{\mathsf{out}} \cdot q \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathsf{eta}(\mathcal{S},\mathsf{st})}$. It is *weakly compliant* with $\mathsf{rsp}(\mathsf{in}, a)@q$ if $\exists_{\mathsf{out}} \cdot q \xrightarrow{(\Lambda(\mathsf{st})_{\setminus \mathsf{in}})^*\,;\,(\mathsf{out},a,\mathsf{in})}_{\mathsf{eta}(\mathcal{S},\mathsf{st})}$, where $\mathsf{st}(\Lambda)_{\setminus \mathsf{in}} = \{(\mathsf{out}, a, \mathsf{in}') \in \mathsf{st}(\Lambda) \mid (\mathsf{out} \cup \mathsf{in}') \cap \mathsf{in} = \varnothing\} \cup \{(n, a) \in \mathsf{st}(\Lambda) \mid n \notin \mathsf{in}\}$ denotes the set of team labels in which no component of in participates.

Unlike output actions, the selection of an input action of a component is not controlled by the component but by the environment, i.e. there is an external choice. If, for a choice of enabled inputs $\{a_1, \ldots, a_n\}$, *only one of them* can be supplied with a corresponding output of the environment this suffices to guarantee progress of components waiting for input.

Definition 2 ((weak) responsiveness). *The ETA* $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ *is* (weakly) responsive *if for all reachable states* $q \in \mathcal{R}(\mathsf{eta}(\mathcal{S}, \mathsf{st}))$*, either there is no responsiveness requirement at* q *or there is a responsiveness requirement* $\mathsf{rsp}(\mathsf{in}, a)@q$ *established for* q *such that the ETA* $\mathsf{eta}(\mathcal{S}, \mathsf{st})$ *is (weakly) compliant with it.*

Example 10. In the initial state $(0,0,0)$ of the race team, there is a responsiveness requirement of the two runners who want to be started, expressed by $\textbf{rsp}(\{\mathsf{R1},\mathsf{R2}\}, \mathit{start})@(0,0,0)$. The ETA $\textbf{eta}(\mathsf{Race}, \textbf{st}_{\mathsf{Race}})$ is compliant with this requirement. When the controller is in state 1, there are responsiveness requirements $\textbf{rsp}(\mathsf{Ctrl}, \mathit{finish})@(\text{q1},\text{q2},1)$ for any q1, q2 $\in \{1,2\}$. Only in state $(2,2,1)$ this requirement is immediately fulfilled; in all other cases, at least one *run* must happen before a *finish* is sent. Then $\textbf{eta}(\mathsf{Race}, \textbf{st}_{\mathsf{Race}})$ is weakly compliant. There are four more responsiveness requirements when the controller is in state 2. ▷

3.4 Logical Characterisations of Responsiveness

We now define a logical characterisation of responsiveness by the *"responsiveness formula" rspFrm* $\in \mathit{Frm}(\Lambda)$ below, for a given $\Lambda(\mathcal{S})$ and STS $\textbf{st}$ as above.

$$\begin{aligned}
\mathit{rspReq} &= \{(\varnothing, a, \mathsf{in}) \in \Lambda \mid |\mathsf{in}| \in I, 0 \notin O \text{ for } \textbf{st}(a) = (O, I)\}\\
\mathit{OutCom}(a, \mathsf{in}) &= \{\mathsf{out} \subseteq \mathcal{N} \mid (\mathsf{out}, a, \mathsf{in}) \in \Lambda(\textbf{st})\}\\
\mathit{rspFrm} &= [\Lambda(\textbf{st})^*]\Big(\big(\textstyle\bigvee_{(\varnothing,a,\mathsf{in})\in \mathit{rspReq}} \langle(\varnothing, a, \mathsf{in})\rangle\, \mathit{true}\big) \rightarrow\\
&\qquad \big(\textstyle\bigvee_{(\varnothing,a,\mathsf{in})\in \mathit{rspReq}} \bigvee_{\mathsf{out}\in \mathit{OutCom}(a,\mathsf{in})} \langle(\mathsf{out}, a, \mathsf{in})\rangle\, \mathit{true}\big)\Big)
\end{aligned}$$

where *rspReq* is the set of system labels which correspond to responsiveness requirements (when enabled in a reachable state of the ETA $\textbf{eta}(\mathcal{S}, \textbf{st})$); and $\mathit{OutCom}(a, \mathsf{in})$ is the set of subsets $\mathsf{out} \subseteq \mathcal{N}$ of component names which complement a given $\mathsf{in} \subseteq N$ and $a \in \Sigma^\bullet$ to a team label in $\Lambda(\textbf{st})$ (for potential communication). Note that the left side of the implication in *rspFrm* is true iff there is a responsiveness requirement for a, in at the current state q. Otherwise *rspFrm* holds anyway at q in accordance with the notion of responsiveness.

Similarly, a *"weak responsiveness formula" wrspFrm* $\in \mathit{Frm}(\Lambda)$ is defined as:

$$\begin{aligned}
\mathit{wrspFrm} = [\Lambda(\textbf{st})^*]\Big(&\big(\textstyle\bigvee_{(\varnothing,a,\mathsf{in})\in \mathit{rspReq}} \langle(\varnothing, a, \mathsf{in})\rangle\, \mathit{true}\big) \rightarrow\\
&\big(\textstyle\bigvee_{(\varnothing,a,\mathsf{in})\in \mathit{rspReq}} \bigvee_{\mathsf{out}\in \mathit{OutCom}(a,\mathsf{in})} \langle \textbf{st}(\Lambda)_{\backslash\mathsf{in}})^*; (\mathsf{out}, a, \mathsf{in})\rangle\, \mathit{true}\big)\Big)
\end{aligned}$$

Example 11. For Race, $\mathit{rspReq} = \{(\varnothing, \mathit{start}, \{\mathsf{R1},\mathsf{R2}\}), (\varnothing, \mathit{finish}, \mathsf{Ctrl})\}$, $\mathit{OutCom}(\mathit{start}, \{\mathsf{R1},\mathsf{R2}\}) = \{\{\mathsf{Ctrl}\}\}$, $\mathit{OutCom}(\mathit{finish}, \mathsf{Ctrl}) = \{\{\mathsf{R1}\},\{\mathsf{R2}\}\}$, and

$$\begin{aligned}
\mathit{wrspFrm} = [\Lambda(\textbf{st}_{\mathsf{Race}})^*]\big(\langle(\varnothing, \mathit{start}, \{\mathsf{R1},\mathsf{R2}\})\rangle\, \mathit{true} \;\vee\; \langle(\varnothing, \mathit{finish}, \mathsf{Ctrl})\rangle\, \mathit{true}\big) \rightarrow\\
\big(\langle(\mathsf{Ctrl}, \mathit{start}, \{\mathsf{R1},\mathsf{R2}\})\rangle\, \mathit{true} \vee\\
\langle((\mathsf{R1}, \mathit{run}) + (\mathsf{R2}, \mathit{run}))^*; (\mathsf{R1}, \mathit{finish}, \mathsf{Ctrl})\rangle\, \mathit{true} \vee\\
\langle((\mathsf{R1}, \mathit{run}) + (\mathsf{R2}, \mathit{run}))^*; (\mathsf{R2}, \mathit{finish}, \mathsf{Ctrl})\rangle\, \mathit{true}\;\big)
\end{aligned}$$

Note that $\Lambda(\textbf{st}_{\mathsf{Race}})_{\backslash\{\mathsf{R1},\mathsf{R2}\}} = \varnothing$ and $\Lambda(\textbf{st}_{\mathsf{Race}})_{\backslash\mathsf{Ctrl}} = \{(\mathsf{R1}, \mathit{run}), (\mathsf{R2}, \mathit{run})\}$.

The weak responsiveness formula is satisfied by the LTS of the Race system. For the check we use the tool described in Sect. 4. Together with Theorem 2, this implies that the $\textbf{eta}(\mathsf{Race}, \textbf{st}_{\mathsf{Race}})$ is weakly responsive. ▷

Lemma 4. *For all* $q \in \mathcal{R}(\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}}))$ *it holds:* $\textsf{\textbf{rsp}}(\textsf{in},a)@q$ *is a responsiveness requirement iff* $(\varnothing,a,\textsf{in}) \in \mathit{rspReq}$ *and* $\textsf{\textbf{lts}}(\mathcal{S}),q \models \langle(\varnothing,a,\textsf{in})\rangle\,\mathit{true}$.

Proof. The proof is analogous to the proof of Lemma 1. □

Theorem 2. *(1)* $\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}})$ *is responsive iff* $\textsf{\textbf{lts}}(\mathcal{S}) \models \mathit{rspFrm}$ *and*
(2) $\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}})$ *is weakly responsive iff* $\textsf{\textbf{lts}}(\mathcal{S}) \models \mathit{wrspFrm}$.

Proof. We only prove (1). The proof of (2) is a straightforward extension.

"⇐": Let $q \in \mathcal{R}(\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}}))$. If there is no responsiveness requirement at q there is nothing to prove. Otherwise, according to Lemma 4, there is at least one element $(\varnothing,a,\textsf{in}) \in \mathit{rspReq}$ such that $\textsf{\textbf{lts}}(\mathcal{S}),q \models \langle(\varnothing,a,\textsf{in})\rangle\,\mathit{true}$. Hence,

$$\textsf{\textbf{lts}}(\mathcal{S}),q \models \bigvee_{(\varnothing,a,\textsf{in})\in \mathit{rspReq}} \langle(\varnothing,a,\textsf{in})\rangle\,\mathit{true}.$$

By assumption $\textsf{\textbf{lts}}(\mathcal{S}) \models \mathit{rspFrm}$. Hence, by Lemma 2 ("⇒"),

$$\textsf{\textbf{lts}}(\mathcal{S}),q \models \bigvee_{(\varnothing,a,\textsf{in})\in \mathit{rspReq}} \bigvee_{\textsf{out}\in \mathit{OutCom}(a,\textsf{in})} \langle(\textsf{out},a,\textsf{in})\rangle\,\mathit{true}.$$

Therefore there exists $(\varnothing,a',\textsf{in}') \in \mathit{rspReq}$ and $\textsf{out}' \in \mathit{OutCom}(a',\textsf{in}')$ such that

$$\textsf{\textbf{lts}}(\mathcal{S}),q \models \big\langle(\textsf{out}',a',\textsf{in}')\big\rangle\,\mathit{true}.$$

Hence $q \xrightarrow{(\textsf{out}',a',\textsf{in}')}_{\textsf{\textbf{lts}}(\mathcal{S})}$ and, according to the defintion of $\mathit{OutCom}(a',\textsf{in}')$, we have that $(\textsf{out}',a',\textsf{in}') \in \Lambda(\textsf{\textbf{st}})$ is a team label. Therefore, by Lemma 3, $q \xrightarrow{(\textsf{out}',a',\textsf{in}')}_{\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}})}$. Consequently, also $q \xrightarrow{(\varnothing,a',\textsf{in}')}_{\textsf{\textbf{lts}}(\mathcal{S})}$ holds and therefore $\textsf{\textbf{lts}}(\mathcal{S}),q \models \big\langle(\varnothing,a',\textsf{in}')\big\rangle\,\mathit{true}$. Then, by Lemma 4, $\textsf{\textbf{rsp}}(\textsf{in}',a')@q$ is a responsiveness requirement at q and $\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}})$ is compliant with it. Since $q \in \mathcal{R}(\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}}))$ was chosen arbitrarily, $\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}})$ is responsive.

"⇒": The proof is by contradiction. Assume $\textsf{\textbf{lts}}(\mathcal{S}) \not\models \mathit{rspFrm}$. By definition, rspFrm has the form $[\textsf{\textbf{st}}(\Lambda)^*]\,\varphi$ with $\varphi \in \mathit{Frm}(\Lambda)$. So, by Lemma 2, there exists a state $q \in \mathcal{R}(\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}}))$ such that $\textsf{\textbf{lts}}(\mathcal{S}),q \not\models \varphi$. Then, by definition of rcpFrm,

$$\textsf{\textbf{lts}}(\mathcal{S}),q \models \bigvee_{(\varnothing,a,\textsf{in})\in \mathit{rspReq}} \langle(\varnothing,a,\textsf{in})\rangle\,\mathit{true}, \text{ and}$$
$$\textsf{\textbf{lts}}(\mathcal{S}),q \not\models \bigvee_{(\varnothing,a,\textsf{in})\in \mathit{rspReq}} \bigvee_{\textsf{out}\in \mathit{OutCom}(a,\textsf{in})} \langle(\textsf{out},a,\textsf{in})\rangle\,\mathit{true}.$$

The first part shows that $\mathit{rspReq} \neq \varnothing$ and that, by Lemma 4, there exists a responsiveness requirement established for q. To be responsive, $\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}})$ must be compliant with at least one of them. The second part shows that for all $(\varnothing,a,\textsf{in}) \in \mathit{rspReq}$ and all $\textsf{out} \in \mathit{OutCom}(a,\textsf{in})$, $\textsf{\textbf{lts}}(\mathcal{S}),q \not\models \langle(\textsf{out},a,\textsf{in})\rangle\,\mathit{true}$. So, for all responsiveness requirements $\textsf{\textbf{rsp}}(\textsf{in},a)$, there does *not* exist a team label $(\textsf{out},a,\textsf{in})$ with $q \xrightarrow{(\textsf{out},a,\mathit{In})}_{\textsf{\textbf{lts}}(\mathcal{S})}$ and so, by Lemma 3, there does *not* exist a team label $(\textsf{out},a,\textsf{in})$ with $q \xrightarrow{(\textsf{out},a,\textsf{in})}_{\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}})}$. Hence, $\textsf{\textbf{eta}}(\mathcal{S},\textsf{\textbf{st}})$ is not responsive. □

4 Model Checking Communication Properties

In this section we show, underpinned by our running example, how to transform CA, systems and ETA into mCRL2 processes as well as dynamic logic formulas, characterising communication properties, into μ-calculus formulas. We also justify briefly the correctness of these transformations and the soundness and completeness of our verification approach. Then we present the tool support that we developed (1) to perform the transformations and (2) to automatically check communication properties through the model-checking facilities offered by the mCRL2 toolset (https://www.mcrl2.org/) [21], similarly to how mCRL2 was used earlier to verify automata composed hierarchically [39].

An mCRL2 model is expressed in an elementary process language, where actions (and possibly data types) as well as processes are defined, and (for our purpose) the initial process is given in the following standard concurrent form:

```
allow( { a, a_1|...|a_n, ... }, proc_1 || ... || proc_n );
```

This is a parallel composition of sequential processes `proc_i`, with interleaving and multi-party synchronisation specified explicitly by `allow`. This restriction operator forbids some actions, to constrain interaction and prune the state space, by listing those allowed to occur in `allow`: so action a is interleaved and, similar to synchronisation of actions a and $\bar{a}$ yielding τ in CCS, actions `a_i` are synchronised, resulting in a multi-action `a_1|...|a_n`; all other actions are blocked.

To explain our transformation, along the lines of Fig. 2, we assume given a system $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_n)_{n \in \mathcal{N}})$ and a synchronisation type specification **st**.

Transformation of CA First, we transform each CA $\mathcal{A}_n$ into an mCRL2 process $\epsilon(\mathcal{A}_n)$, cf. Fig. 1(a). The transformation is defined and implemented in a straightforward way based on the idea that an LTS $\mathcal{L}$ can be represented by a process expression P, i.e. the LTS semantics of P is $\mathcal{L}$. In our context, the representation of the $\mathcal{A}_n$ is a bit more involved since we want to represent shared actions of different CA by different actions of their mCRL2 processes (later to be synchronised by multi-actions). Therefore we apply a renaming ρ which renames each action a of each $\mathcal{A}_n$ to the mCRL2 action `n_a` of $\epsilon(\mathcal{A}_n)$. Then the LTS semantics of mCRL2 processes (defined by SOS rules in [30, Def. 15.2.10]) applied to $\epsilon(\mathcal{A}_n)$ provides an LTS **lts**$(\epsilon(\mathcal{A}_n))$. (We ignore aspects of data and time included in mCRL2). Next we note that **lts**$(\epsilon(\mathcal{A}_n))$ is a reachable LTS which is, up to renaming w.r.t. ρ, isomorphic to the reachable part of $\mathcal{A}_n$, i.e. to the LTS obtained by restricting the state space of $\mathcal{A}_n$ to reachable states. For instance, the CA $\mathcal{A}_{\mathsf{R1}}$ from Fig. 1(a) is transformed into the mCRL2 process `proc R1(s:Int)` below. Its `act`ions are `R1_start`, `R1_run`, and `R1_finish`, a parameter `s` (an integer) holds the state, summation (`+`) represents non-deterministic choice, and `R1(0)` is its `init`ial state. The actions are renamed as explained above.

```
act R1_start, R1_run, R1_finish;
proc R1(s:Int) =
   ( s == 0 ) → ( R1_start . R1(1) ) +
   ( s == 1 ) → ( R1_run . R1(2) ) +
   ( s == 2 ) → ( R1_finish . R1(0) );
init R1(0);
```

Transformation of System $\mathcal{S}$ System $\mathcal{S}$ is transformed into an mCRL2 process $\epsilon(\mathcal{S})$ as follows. Any system label $(\mathsf{out}, a, \mathsf{in})$ is represented by the multi-action which synchronises all mCRL2 actions `o_a` with `o` $\in \mathsf{out}$ with all mCRL2 actions `i_a` with `i` $\in \mathsf{in}$. Any system label (n, a) for internal actions is represented by `n_a`. Then we construct the parallel composition of all mCRL2 processes $\epsilon(A_n)$ restricted to (multi-)actions that represent system labels. The restriction is realised by mCRL2's `allow` operator. By this construction the LTS semantics $\textbf{lts}(\epsilon(\mathcal{S}))$ is, up to the renaming of system labels, isomorphic to the reachable part of $\textbf{lts}(\mathcal{S})$. As non-reachable states are irrelevant for the satisfaction of formulas, this provides the basis for verifying our communication properties with mCRL2. For instance, the Race system is represented by this mCRL2 process:

```
act R1_start, R2_start, Ctrl_start, R1_run, R2_run, Ctrl_run, ...;
proc R1(s:Int) = ...;
     R2(s:Int) = ...;
     Ctrl(s:Int) = ...;
init allow ({R1_start, R1_finish, R1_run, R2_start, R2_finish, R2_run
     Ctrl_start, Ctrl_finish, Ctrl_start|R1_start, Ctrl_start|R2_start,
     R1_start|R2_start, Ctrl_start|R1_start|R2_start, ...},
     R1(0) || R2(0) || Ctrl(0)).
```

Thus we block multi-actions, like `R1_start|Ctrl_finish` and `R1_run|R2_run`, which do not correspond to system labels, by using the `allow` operator. In total there are 16 allowed multi-actions. The system's LTS can be computed by our tool.

We can also represent the ETA generated by the STS **st** over $\mathcal{S}$ if we further restrict the allowed actions to those whose corresponding system labels satisfy **st**. In our example, this would mean that we allow only the mCRL2 actions `Ctrl_start|R1_start|R2_start`, `R1_finish|Ctrl_finish`, `R2_finish|Ctrl_finish`, `R1_run`, and `R2_run`. Note that the representation of ETA is not used for verification of communication properties (see below). It is, however, useful for the graphical animation of ETA.

Transformation of Communication Formulas We characterised (weak) receptiveness and (weak) responsiveness in Sects. 3.2 and 3.4 by formulas $(w)\mathit{rcpFrm}$ and $(w)\mathit{rspFrm}$, resp. To automatically verify these formulas, we transform them into mCRL2's μ-calculus by the renaming of system labels explained above and by syntactic conversion of operators, e.g. $\wedge$ to `&&`, $\vee$ to `||`, and *some* to `true`. We write `<a+b+c>`ψ instead of `<a>`ψ`||<b>`ψ`||<c>`ψ for compactness. The receptiveness formula *rcpFrm* of our example is transformed into:

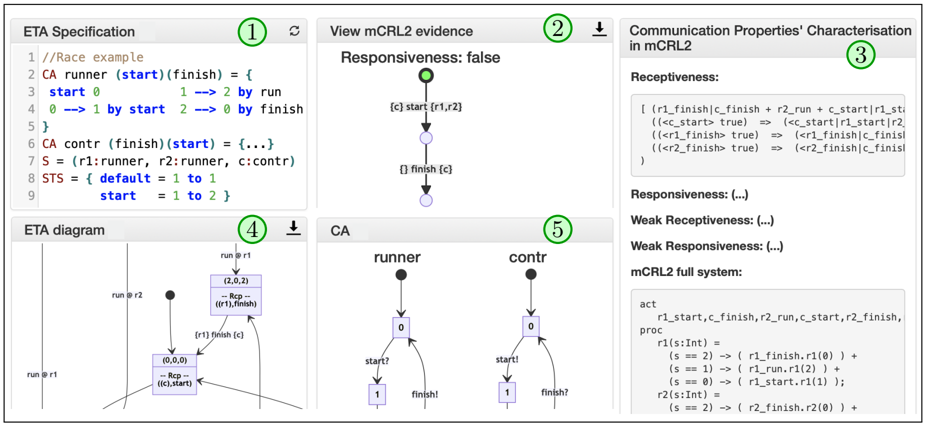


Fig. 3: Screenshot of some of the widgets in the ETA tools available online

```
[(Ctrl_start|R1_start|R2_start + R1_finish|Ctrl_finish +
  R2_finish|Ctrl_finish + R2_run + R1_run)*]
 (((<R1_finish> true) => (<R1_finish|Ctrl_finish> true)) &&
  ((<R2_finish> true) => (<R2_finish|Ctrl_finish> true)) &&
  ((<Ctrl_start> true) => (<Ctrl_start|R1_start|R2_start> true)))
```

Note that for the transformation of communication properties the given STS **st** is crucial. Indeed, the structured action used in the modal box operator refers exactly to those actions which correspond to the system labels satisfying the synchronisation type and hence to the team labels.

Verifying Communication Properties in mCRL2 As shown in Theorems 1 and 2 the validity of the logic formulas *cpFrm* characterising communication properties must be checked over the LTS of system $\mathcal{S}$. According to our semantics preserving transformation of system $\mathcal{S}$ into the process $\epsilon(\mathcal{S})$, checking validity of *cpFrm* in $\mathsf{lts}(\mathcal{S})$ is equivalent to checking the transformed version $\epsilon(cpFrm)$ over $\mathsf{lts}(\epsilon(\mathcal{S}))$. But the latter is exactly how satisfaction of formulas is defined for mCRL2 processes and therefore our verification approach is sound and complete.

Implementation An open-source prototype was implemented, which can be executed online at https://github.com/arcalab/team-a. It is written in Scala, compiled into JavaScript via *Scala.js*, and uses Scala and JavaScript libraries and external tools like the mCRL2 model checker. Most final code is in JavaScript running in an Internet browser (client-side), while the external tools are executed remotely (server-side). It is also possible to compile and run the server locally.

The screenshot in Fig. 3 depicts some of the available widgets, using our running Race example. More complete screenshots can be found in Appendix A. The input team automaton is specified in widget ①, where `S` defines the system composed of 2 runners and 1 controller, and `STS` specifies the synchronisation types. The remaining widgets provide analysis of the ETA: ③ outputs the encoded mCRL2 model and formulas being evaluated; ② outputs both the result of the formula and a counterexample or a witness — in this case stating that this

ETA is not responsive with a counterexample; and ④ and ⑤ depict the composed ETA and the individual component automata, resp. Note that widget ② also reports that Race is weakly responsive, as described in Sect. 3.4, producing a witness that matches the ETA diagram (cf. Fig. 5 in Appendix A).

A Note on Optimisation Our approach can be further optimised to reduce the model's size. For example, as mentioned in Remark 1, the mCRL2 process representing system $\mathcal{S}$ can be replaced by one that allows a smaller set of multi-actions corresponding to team labels from the ETA ($\textbf{eta}(\mathcal{S}, \textbf{st})$) only, but enriched with $(\mathsf{out}, a, \varnothing)$ labels (when proving (weak) receptiveness) or with $(\varnothing, a, \mathsf{in})$ (when proving (weak) responsiveness). Furthermore, all internal actions could be replaced by a single non-synchronising action (e.g. τ), which may, however, lead to less readable counterexamples. Using these optimisations, one could check for receptiveness or responsiveness of our Race example using a model that allows only 7 multi-actions instead of 16. In general, this reduction depends on (1) the number of shared actions, (2) the degree of flexibility of the synchronisation policies, and (3) the number of internal actions.

5 Conclusions and Future Work

We provide the first logical characterisation of communication properties of team automata in the form of (weak) receptiveness and (weak) responsiveness. I.e., we logically characterise whether all messages that can be sent can also be received, and that components waiting to receive some input message will get one. This provides the basis for an automated verification approach of communication properties of team automata. A prototype tool, available at https://github.com/arcalab/team-a, realises this automated verification, performed by mCRL2 [21].

Our results also apply to related automata-based models that interact through shared input and output actions, since many such models are subsumed by team automata, like I/O automata [14] but also a special type of Petri nets [15]. Moreover, we believe that our results can be adjusted to capture variants of compatibility like the "optimistic" approach proposed for interface automata [37].

Future work concerns generalising our logical characterisation and the tool to deal with variability and family-based compatibility checking for featured team automata [9], as well as a more comprehensive validation of our tool with larger case studies, to better identify limitations and optimisations of our approach. Furthermore, it could be interesting to adapt the framework from [4] to study the relation between a specification given as team automata and its implementation. Finally, an orthogonal approach is presented [18], where correct protocol composition is defined in terms of so-called 'assertions' akin to pre- and postconditions instead of synchronisation on common actions. Apparently not all resulting compositions are characterisable as team automata synchronisations (and vice versa), but the precise difference in synchronising behaviour between the two approaches remains to be studied.

Acknowledgments. Ter Beek received funding from the MIUR PRIN 2017FTXR7S project IT MaTTerS (Methods and Tools for Trustworthy Smart Systems) and PRIN 2020TL3X8X project T-LADIES (Typeful Language Adaptation for Dynamic, Interacting and Evolving Systems). Proença was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Unit (UIDP/UIDB/04234/2020) and the IBEX project (PTDC/CCI-COM/4280/2021); also by national funds through FCT and European funds through EU ECSEL JU, within project VALU3S (ECSEL/0016/2019 - JU grant nr. 876852) – The JU receives support from the EU's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey. Disclaimer: This document reflects only the authors' view and the Commission is not responsible for any use that may be made of the information it contains.

References

1. Adler, B.T., de Alfaro, L., da Silva, L.D., Faella, M., Legay, A., Raman, V., Roy, P.: Ticc: A Tool for Interface Compatibility and Composition. In: Ball, T., Jones, R.B. (eds.) CAV. LNCS, vol. 4144, pp. 59–62. Springer (2006). https://doi.org/10.1007/11817963_8
2. Bartoletti, M., Cimoli, T., Zunino, R.: Compliance in Behavioural Contracts: A Brief Survey. In: Bodei, C., Ferrari, G.L., Priami, C. (eds.) Programming Languages with Applications to Biology and Security. LNCS, vol. 9465, pp. 103–121. Springer (2015). https://doi.org/10.1007/978-3-319-25527-9_9
3. Basile, D., ter Beek, M.H.: Contract Automata Library. Sci. Comput. Program. **221** (2022). https://doi.org/10.1016/j.scico.2022.102841
4. Basile, D., ter Beek, M.H.: A Runtime Environment for Contract Automata. In: Chechik, M., Katoen, J.P., Leucker, M. (eds.) FM. LNCS, Springer (2023), in this volume
5. Basile, D., ter Beek, M.H., Degano, P., Legay, A., Ferrari, G.L., Gnesi, S., Di Giandomenico, F.: Controller synthesis of service contracts with variability. Sci. Comput. Program. **187** (2020). https://doi.org/10.1016/j.scico.2019.102344
6. Basile, D., Degano, P., Ferrari, G.L.: Automata for Specifying and Orchestrating Service Contracts. Log. Meth. Comp. Sci. **12**(4:6), 1–51 (2016). https://doi.org/10.2168/LMCS-12(4:6)2016
7. Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On Weak Modal Compatibility, Refinement, and the MIO Workbench. In: Esparza, J., Majumdar, R. (eds.) TACAS. LNCS, vol. 6015, pp. 175–189. Springer (2010). https://doi.org/10.1007/978-3-642-12002-2_15
8. ter Beek, M.H., Carmona, J., Hennicker, R., Kleijn, J.: Communication Requirements for Team Automata. In: Jacquet, J.M., Massink, M. (eds.) COORDINATION. LNCS, vol. 10319, pp. 256–277. Springer (2017). https://doi.org/10.1007/978-3-319-59746-1_14
9. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Featured Team Automata. In: Huisman, M., Pasareanu, C.S., Zhan, N. (eds.) FM. LNCS, vol. 13047, pp. 483–502. Springer (2021). https://doi.org/10.1007/978-3-030-90870-6_26
10. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Team Automata for Spatial Access Control. In: Prinz, W., Jarke, M., Rogers, Y., Schmidt, K., Wulf, V.

(eds.) Proceedings of the 7th European Conference on Computer Supported Cooperative Work (ECSCW'01). pp. 59–78. Kluwer (2001). https://doi.org/10.1007/0-306-48019-0_4
11. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Synchronizations in Team Automata for Groupware Systems. Comput. Sup. Coop. Work **12**(1), 21–69 (2003). https://doi.org/10.1023/A:1022407907596
12. ter Beek, M.H., Hennicker, R., Kleijn, J.: Compositionality of Safe Communication in Systems of Team Automata. In: Pun, V.K.I., Simão, A., Stolz, V. (eds.) ICTAC. LNCS, vol. 12545, pp. 200–220. Springer (2020). https://doi.org/10.1007/978-3-030-64276-1_11
13. ter Beek, M.H., Kleijn, J.: Team Automata Satisfying Compositionality. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME. LNCS, vol. 2805, pp. 381–400. Springer (2003). https://doi.org/10.1007/978-3-540-45236-2_22
14. ter Beek, M.H., Kleijn, J.: Modularity for teams of I/O automata. Inf. Process. Lett. **95**(5), 487–495 (2005). https://doi.org/10.1016/j.ipl.2005.05.012
15. ter Beek, M.H., Kleijn, J.: Vector Team Automata. Theor. Comput. Sci. **429**, 21–29 (2012). https://doi.org/10.1016/j.tcs.2011.12.020
16. ter Beek, M.H., Lenzini, G., Petrocchi, M.: Team Automata for Security: A Survey. Electron. Notes Theor. Comput. Sci. **128**(5), 105–119 (2005). https://doi.org/10.1016/j.entcs.2004.11.044
17. Beyer, D., Chakrabarti, A., Chatterjee, K., de Alfaro, L., Henzinger, T.A., Jurdzinski, M., Mang, F.Y.C., Song, C.: CHIC: Checking Interface Compatibility (December 2007), https://ptolemy.berkeley.edu/projects/embedded/research/chic
18. Bocchi, L., Orchard, D., Voinea, A.L.: A Theory of Composing Protocols. Art Sci. Eng. Program. **7**(2), 6:1–6:76 (2023). https://doi.org/10.22152/programming-journal.org/2023/7/6
19. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are Two Web Services Compatible? In: Shan, M.C., Dayal, U., Hsu, M. (eds.) TES. LNCS, vol. 3324, pp. 15–28. Springer (2005). https://doi.org/10.1007/978-3-540-31811-8_2
20. Brand, D., Zafiropulo, P.: On Communicating Finite-State Machines. J. ACM **30**(2), 323–342 (1983). https://doi.org/10.1145/322374.322380
21. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mCRL2 Toolset for Analysing Concurrent Systems. In: Vojnar, T., Zhang, L. (eds.) TACAS. LNCS, vol. 11428, pp. 21–39. Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_2
22. Carmona, J., Cortadella, J.: Input/Output Compatibility of Reactive Systems. In: Aagaard, M.D., O'Leary, J.W. (eds.) FMCAD. LNCS, vol. 2517, pp. 360–377. Springer (2002). https://doi.org/10.1007/3-540-36126-X_22
23. Carmona, J., Kleijn, J.: Compatibility in a multi-component environment. Theor. Comput. Sci. **484**, 1–15 (2013). https://doi.org/10.1016/j.tcs.2013.03.006
24. Carrez, C., Fantechi, A., Najm, E.: Behavioural Contracts for a Sound Assembly of Components. In: König, H., Heiner, M., Wolisz, A. (eds.) FORTE. LNCS, vol. 2767, pp. 111–126. Springer (2003). https://doi.org/10.1007/978-3-540-39979-7_8
25. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. ACM Trans. Program. Lang. Syst. **31**(5), 19:1–19:61 (2009). https://doi.org/10.1145/1538917.1538920
26. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Jurdzinski, M., Mang, F.Y.C.: Interface Compatibility Checking for Software Modules. In: Brinksma, E., Larsen, K.G. (eds.) CAV. LNCS, vol. 2404, pp. 428–441. Springer (2002). https://doi.org/10.1007/3-540-45657-0_35

27. de Alfaro, L., Henzinger, T.A.: Interface Automata. In: Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE). pp. 109–120. ACM (2001). https://doi.org/10.1145/503209.503226
28. Durán, F., Ouederni, M., Salaün, G.: A generic framework for n-protocol compatibility checking. Sci. Comput. Program. **77**(7-8), 870–886 (2012). https://doi.org/10.1016/j.scico.2011.03.009
29. Ellis, C.A.: Team Automata for Groupware Systems. In: Proceedings of the 1st International ACM SIGGROUP Conference on Supporting Group Work (GROUP). pp. 415–424. ACM (1997). https://doi.org/10.1145/266838.267363
30. Groote, J.F., Mousavi, M.R.: Modeling and Analysis of Communicating Systems. MIT Press (2014)
31. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. Foundations of Computing, MIT Press (2000). https://doi.org/10.7551/mitpress/2516.001.0001
32. Hennessy, M., Milner, R.: On Observing Nondeterminism and Concurrency. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP. LNCS, vol. 85, pp. 299–309. Springer (1980). https://doi.org/10.1007/3-540-10003-2_79
33. Hennicker, R., Bidoit, M.: Compatibility Properties of Synchronously and Asynchronously Communicating Components. Log. Meth. Comp. Sci. **14**(1), 1–31 (2018). https://doi.org/10.23638/LMCS-14(1:1)2018
34. Hennicker, R., Bidoit, M., Dang, T.: On Synchronous and Asynchronous Compatibility of Communicating Components. In: Lluch Lafuente, A., Proença, J. (eds.) COORDINATION. LNCS, vol. 9686, pp. 138–156. Springer (2016). https://doi.org/10.1007/978-3-319-39519-7_9
35. Hennicker, R., Knapp, A.: Moving from interface theories to assembly theories. Acta Inf. **52**(2-3), 235–268 (2015). https://doi.org/10.1007/s00236-015-0220-7
36. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP. LNCS, vol. 4421, pp. 64–79. Springer (2007). https://doi.org/10.1007/978-3-540-71316-6_6
37. Lüttgen, G., Vogler, W., Fendrich, S.: Richer interface automata with optimistic and pessimistic compatibility. Acta Inf. **52**(4-5), 305–336 (2015). https://doi.org/10.1007/s00236-014-0211-0
38. Lynch, N.A., Tuttle, M.R.: An Introduction to Input/Output Automata. CWI Q. **2**(3), 219–246 (1989), https://ir.cwi.nl/pub/18164
39. Proença, J., Madeira, A.: Taming Hierarchical Connectors. In: Hojjat, H., Massink, M. (eds.) FSEN. LNCS, vol. 11761, pp. 186–193. Springer (2019). https://doi.org/10.1007/978-3-030-31517-7_13

A Screenshots of the Tool

Our approach to synthesise characterising formulas for communication properties of team automata was implemented in Scala and JavaScript. In the body of the paper, we only included a short overview of the implementation in Sect. 4, and provide here more complete screenshots in Figs. 4 and 5. These are screenshots from the content of the Web browser, which interacts with a server that runs the mCRL2 model checker [21]. We briefly explain each widget below.

- **ETA Specification** is the input text box, where the user can specify the component automata (`CA`), the system (`S`), and the synchronisation type specifications (`STS`).
- **ETA Examples** is a collection of examples of ETA specifications — these screenshots illustrate the `Race` system, and clicking any of the other examples will replace the ETA specification by a different example.
- **ETA diagram** is the visual representation of the composed ETA, in this case produced by composing 2 runners and 1 controller, and restricting communication to the given synchronisation type.
- **System diagram** is the visual representation of the composed system, without imposing the synchronisation type.
- **Communication Properties' Characterisation in mCRL2** displays the transformation of the four communication properties into mCRL2 formulas and the transformation of the system into parallel mCRL2 processes. These transformations are internal representations that do not need to be exposed to the user to reason over communication properties, but can help to provide more insight and thus opens the possibility of verifying other communication properties in mCRL2.
- **Verification in mCRL2** outputs the result by mCRL2 from verifying the communication property formulas against the mCRL2 processes. In this case, only the responsiveness property failed.
- **View mCRL2 evidence** extends the verification outputs with evidences provided by mCRL2, after renaming the labels to match their original notation. In this case, it produced three equal witnesses for the satisfied formulas, which match the ETA, and a counterexample for the failed formula. This counterexample depicts a trace to a state where the controller should be able to receive a *finish* but cannot.
- **CA** is the visual representation of each of the individual components defined in the *ETA specification* widget.

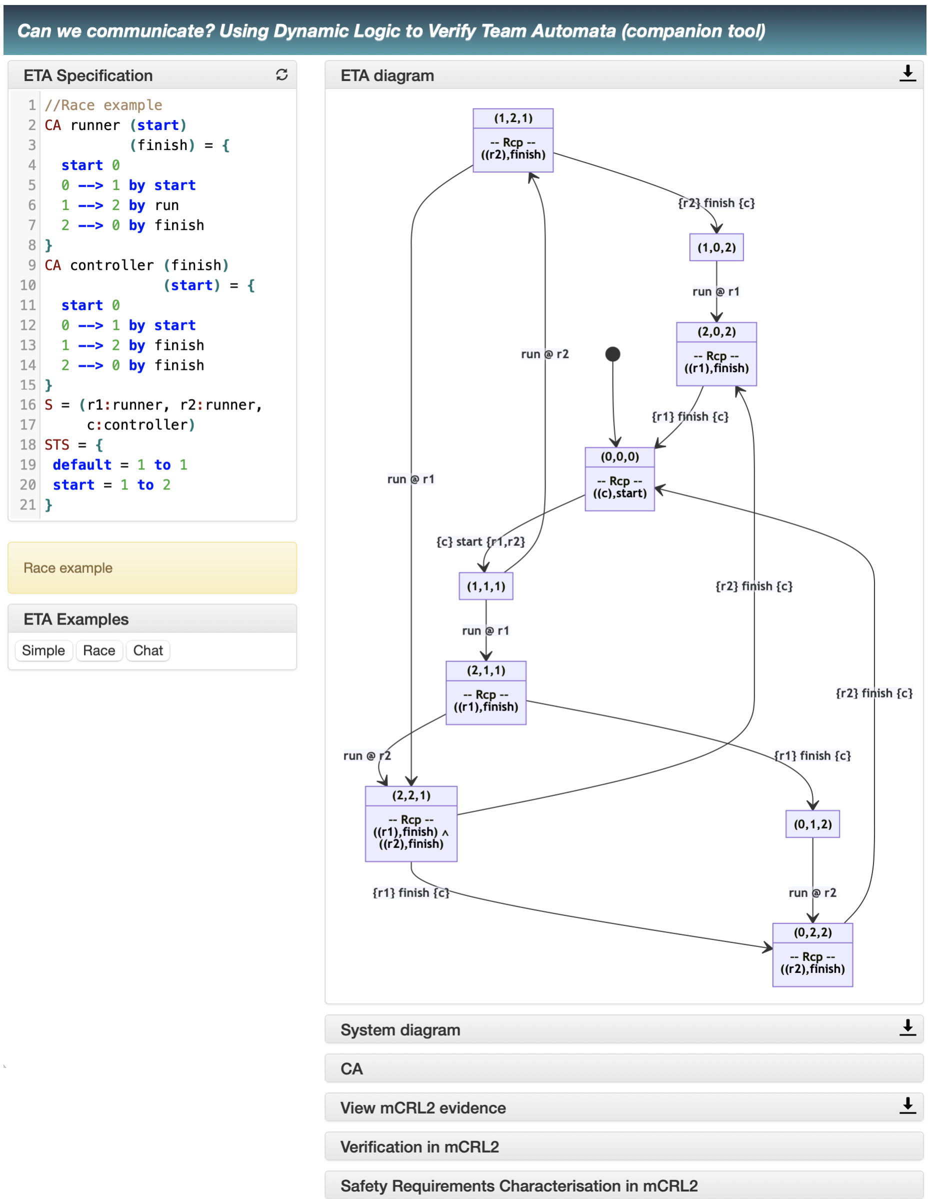


Fig. 4: Overall screenshot of the browser, with the 5 lower widgets minimised

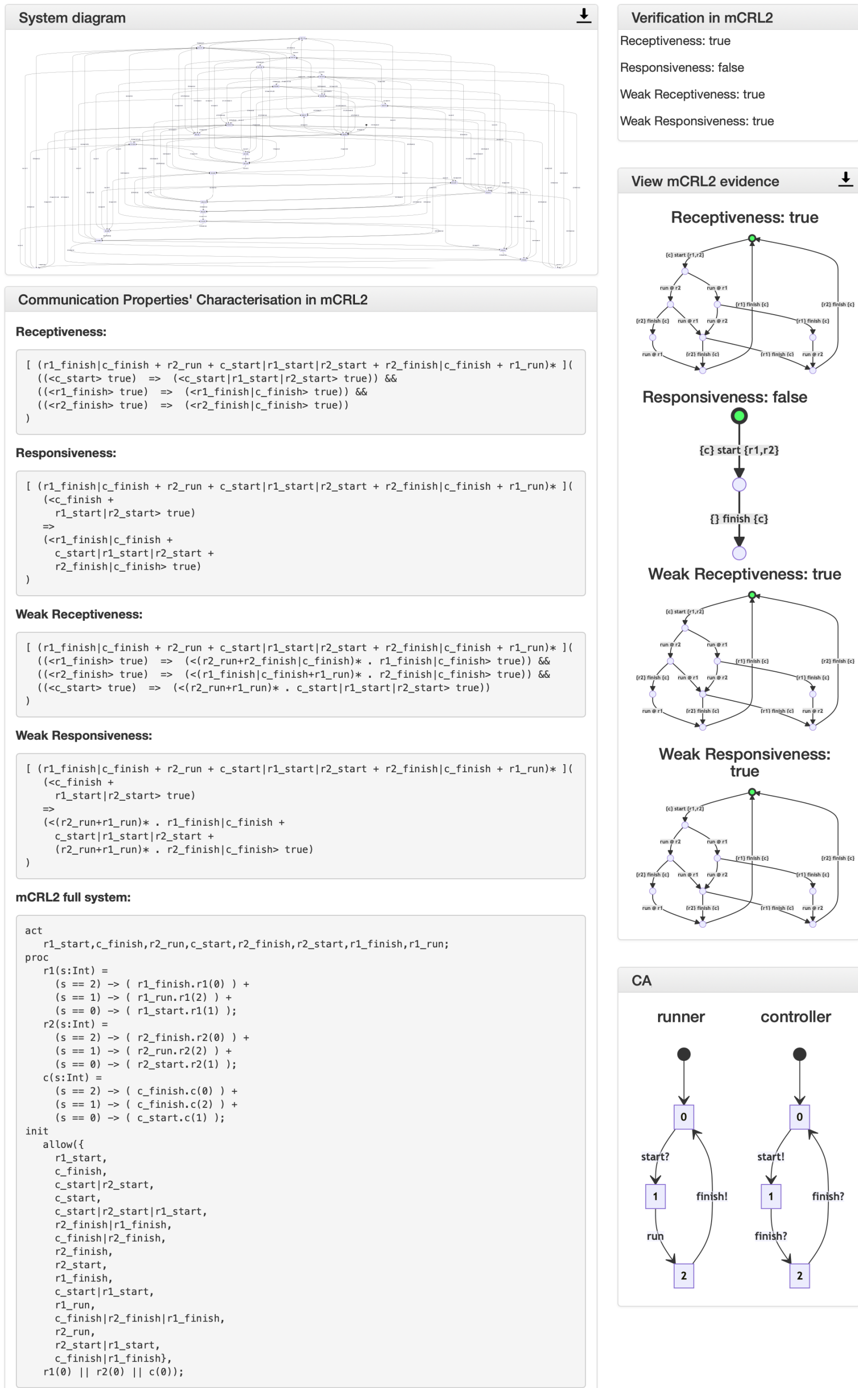


Fig. 5: Screenshot of the expanded widgets that were minimised in Fig. 4