Theoretical Computer Science 403 (2008) 382–405

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Verification of well-formed communicating recursive state machines☆,☆☆

Laura Bozzelli[a,*], Salvatore La Torre[b], Adriano Peron[c]

[a] Università dell'Insubria, Via Valleggio 11, 22100 - Como, Italy

[b] Università degli Studi di Salerno, Via Ponte Don Melillo - 84084 Fisciano, Italy

[c] Università di Napoli "Federico II", Via Cintia, 80126 - Napoli, Italy

ARTICLE INFO

Article history:
Received 23 November 2006
Received in revised form 20 February 2008
Accepted 10 June 2008
Communicated by J. Esparza

Keywords:
Infinite-state systems
Pushdown systems
Formal models of concurrency and recursion
Model checking
Linear-time logics
Context-free specifications
Tree automata

ABSTRACT

In this paper we introduce a new (non-Turing equivalent) formal model of recursive concurrent programs called well-formed communicating recursive state machines (*CRSM*). *CRSM* extend recursive state machines (*RSM*) by allowing a restricted form of concurrency: a state of a module can be refined into a finite collection of modules (working in parallel) in a potentially recursive manner. Communication is only possible between the activations of modules invoked on the same fork. We study the model-checking problem of *CRSM* with respect to specifications expressed in a temporal logic that extends CaRet with a parallel operator (ConCaRet). We propose a decision algorithm that runs in time exponential in both the size of the formula and the maximum number of modules that can be invoked simultaneously. This matches the known lower bound for deciding CaRet model checking of *RSM*, and therefore, we prove that model checking *CRSM* with respect to ConCaRet specifications is Exptime-complete.

1. Introduction

Analysis of recursive concurrent programs. Computer programs often involve the concurrent execution of multiple threads interacting with each other. Each thread can require recursive procedure calls and thus make use of a local stack. In general, combining recursion and task synchronization leads to Turing-equivalent models; thus, simple verification problems such as reachability analysis are undecidable [32]. Therefore, there have been essentially two approaches in the literature that address the problem of analyzing recursive concurrent programs: (i) abstraction (approximate) techniques on 'unrestricted models' and (ii) precise techniques for 'weaker models' (with decidable reachability), obtained by imposing a restriction on the amount of parallelism and synchronization. Concerning the first approach, meaningful results are given in [8,10], where it is shown how to compute overapproximations of the set of reachable states, and in [31], where a non-trivial technique to compute underapproximations by bounding the number of context-switches in the analysis of concurrent systems is introduced. Context-bounded analysis of concurrent systems is also dealt with in [26], where the computations of systems communicating via FIFO queues are checked up to a bounded number of process switches, and in [25], where the computations of multistack pushdown systems are considered up to a bounded number of phases (in each phase, only one stack can be popped).

In the second approach, many *non*-Turing-equivalent formalisms, suitable to model the control flow of recursive concurrent programs, have been proposed. One of the most powerful formalisms is represented by *Process Rewrite Systems*

☆ A preliminary version of this paper appears in the Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'06, Lecture Notes of Computer Science, vol. 3855, Springer Verlag, 2006, pp. 412–426.

☆☆ This research was partially supported by the MIUR grant ex-60% 2003–2004 Università degli Studi di Salerno.

* Corresponding author. Tel.: +39 081 662316; fax: +39 081 7145110.
E-mail addresses: laura.bozzelli@dma.unina.it (L. Bozzelli), latorre@dia.unisa.it (S. La Torre), peron@na.infn.it (A. Peron).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.06.012

(*PRS*, for short) [27,28], based on term rewriting which combines prefix and multiset rewrite rules, and subsumes many common infinite-states models such as Pushdown Processes, Petri Nets, and processes of Process Algebra (PA processes) [5]. *PRS* can be adopted as a formal model for programs with dynamic creation and (a restricted form of) synchronization of concurrent processes, and with recursive procedures (possibly with return values). A state or term in a *PRS* describes the activation hierarchy of threads (or processes in the terminology of *PRS*) together with their local call stacks. The number of active threads is unbounded and the communication is allowed only between threads that do not have ongoing procedure calls and belong to the same *local context*, where a local context represents a collection of threads sharing the same call stack. *PRS* can accommodate both *parallel-call* commands and a restricted form of *spawn* commands for the dynamic creation of threads: in a parallel-call command, the caller thread suspends its activation waiting for the termination of all called processes, while in a spawn command, the caller thread can pursue its execution concurrently to the new activated threads (in *PRS* the new activated threads and the caller thread in a spawn command belong to the same local context). Due to the presence of spawn commands, the number of threads forming a local context is unbounded. However, there is a price to pay for this expressive power: for the general framework of *PRS*, the only decidability results known in the literature concern the reachability problem between two given terms and the *reachable property* problem [27,28]. Model checking against standard propositional temporal logics is undecidable also for small fragments. Moreover, note that the best known upper bound for reachability of Petri nets (which represent a subclass of *PRS*) requires non-primitive recursive space [16].

In [9], symbolic reachability analysis of *PRS* is investigated (i.e., the constructibility problem of the potentially infinite set of terms that are reachable from a given possibly infinite set of terms). The algorithm given there can be applied only to a strict subclass of *PRS*, i.e. the *synchronization-free PRS* (the so-called PAD systems) which subsume Pushdown processes, *synchronization-free* Petri nets, and PA processes. In [17,18], symbolic reachability analysis of PA processes is used to allow the interprocedural data-flow analysis of programs represented by systems of *parallel flow graphs* (parallel *FGS*, for short), which extend classical recursive sequential flow graphs by parallel-call commands. More recently, in [11], Dynamic Pushdown Networks (*DPN*) are proposed for flow analysis of multithreaded programs without synchronization. Intuitively, a *DPN* is a network of pushdown processes that run independently in parallel. Each process can create new members of the network (spawn commands) as a side effect of a pushdown transition. An extension of this model that captures the modeling power of PAD is also considered, and it is shown that backward reachability preserves regularity (in particular, the set of configurations that can reach a given regular set of configurations can be computed in exponential time).

Well-formed communicating recursive state machines. We consider a new abstract model of concurrent programs with finite-domain variables and recursive procedures, the *well-formed communicating recursive state machines* (*CRSM*). A *CRSM* is an ordered collection of finite-state machines (called *modules*), where a state can represent a call, in a potentially recursive manner, to a finite collection of modules running in parallel. A parallel call to other modules models the activation of multiple threads in a concurrent program (*fork*). When a fork happens, the execution of the current module is stopped and the control moves into the modules activated by the fork. On termination of such modules, the control returns to the calling module and its execution is resumed (*join*). *CRSM* allow the communication only between module instances (threads) that belong to the same local context (i.e., threads that are activated on the same fork) and do not have ongoing procedure calls. We allow multiple entries and exits for each module, which can be used to handle finite-domain local variables and return values from procedure calls (see [1]).

The restriction on the form of communication in *CRSM* is similar to that of *PRS*, in the sense that synchronization is allowed only between threads belonging to the same local context. However, unlike *PRS*, *CRSM* cannot model spawn commands. As a consequence while the number of active threads is unbounded, the number of threads forming a local context is instead bounded by a fixed constant. Intuitively, *CRSM* correspond to the subclass of *PRS* obtained by disallowing rewrite rules which model spawn commands, i.e. rewrite rules whose right-hand sides are of the form $t_1 \| t_2$, where t_1 and t_2 are terms and $\|$ denotes parallel composition (see [28] for syntax and semantics of *PRS*).

CRSM extend parallel *FGS* since they also allow (a restricted form of) synchronization and return values from (parallel) procedure calls. With respect to *DPN* [11], *CRSM* allow synchronization. Moreover, *CRSM* extend both the recursive state machines (*RSM*) [1] by allowing parallelism and the well-structured communicating (finite-state) hierarchical state machines [4] by allowing recursion.

CRSM are strictly more expressive than *RSM*. In fact, *RSM* correspond to pushdown systems [1,2]. Moreover, it is easy to prove that *synchronization-free CRSM* correspond to a complete normal form of *Ground Tree Rewriting systems* (*GTR*) [12], and thus the related class of languages is located in the Chomsky hierarchy strictly between the context-free and the context-sensitive languages [24]. Establishing whether unrestricted *CRSM* are strictly more expressive than *GTR* (w.r.t. standard notions of equivalence such as bisimulation) is open. We recall that many problems for *GTR* are decidable, among them reachability [12,13], fair termination [34], and confluence [14]. In particular, forward reachability of *GTR* preserves regularity and the set of terms which are reachable from a given term can be computed in polynomial time. In [15] it is shown that the first-order theory of ground tree rewriting systems is decidable. More recently, Löding in [24] proved the decidability of the *existential* recurrence problem (i.e., the problem whether there exists an infinite path from the initial term that visits infinitely often a given regular set of terms) and consequently gave a symbolic model-checking algorithm for a fragment of CTL. The *universal* recurrence problem is instead undecidable. As a consequence checking state-based fairness properties of *GTR* is undecidable too. To the best of our knowledge the model-checking problem of *GTR* against standard propositional linear-time temporal logics has not been investigated so far.

The advantage of considering *CRSM* instead of formalisms based on term rewriting like *GTR* and *PRS* is that *CRSM* are more appropriate for state-based visual modeling. In particular, *CRSM* can be viewed as a variant of visual notations for hierarchical state machines, such as Statecharts [21] and UML [6], where both recursion and a restricted form of concurrency are allowed. Furthermore, *CRSM* provide an explicit representation of modularity, thus making easier the formalization of concepts related to the local behavior of threads in recursive concurrent programs.

Our contribution. In this paper, we address the model-checking problem of *CRSM* against propositional linear-time specifications. As for *PRS*, the model-checking problem of *CRSM* against standard propositional *LTL*, in which the semantics of the next operator is given in terms of the global successor along a computation, is undecidable. The undecidability mostly resides in the ability of *LTL* to enforce unrestricted synchronization among the parallel threads of a computation. In fact, we prove that undecidability already holds for synchronization-free *CRSM* and for the fragment of standard *LTL* using only the temporal operators *next* and *infinitely often* (which is strictly less expressive than the one with *next* and *eventually*).

Due to these negative results, we define a new temporal logic which allow us to express *LTL*-like properties with respect to the local computations of threads and to their local call stacks, and synchronization only among different threads of the same local context. We call such logic ConCaRet. ConCaRet is essentially a version for concurrent recursive programs (extended with a parallel operator) of the logic CaRet [3], which has been recently introduced to specify correctness requirements of recursive sequential programs. Recall that CaRet extends standard *LTL* allowing the specification of non-regular context-free properties that are useful to express correctness of procedures with respect to pre- and post-conditions. Model checking of *RSM* against CaRet specifications is known to be Exptime-complete [3].

The semantics of ConCaRet is given with respect to (infinite) computations of *CRSM*. In our model, threads can fork, thus a global state (i.e., the stack content and the current local state of each active thread) is represented as a ranked tree. Hence, a computation corresponds to a sequence of these ranked trees. As in CaRet, we consider three different notions of local successor, for any local state along a computation, and the corresponding counterparts of the usual temporal operators of *LTL* logic: the *abstract successor* captures the local computation within a module removing computation fragments corresponding to nested calls within the module; the *caller* denotes the local state (if any) corresponding to the "innermost call" that has activated the current local state; a (local) *linear successor* of a local state is the usual notion of successor within the corresponding thread. In case the current local state corresponds to a fork, its linear successors give the starting local states of the activated threads. With respect to the paths generated by the linear successors, we allow the standard *LTL* modalities coupled with existential and universal quantification. In ConCaRet we also allow a parallel operator that can express properties about communicating modules such as "every time a resource p is available for a process I, then it will be eventually available for all the processes running in parallel with I" (in formulas, $\Box(p \rightarrow \| \Diamond^a p)$). Note that the linear successor corresponds to the global successor when interpreted on computations of *RSM*. Thus, for *RSM*, the logic ConCaRet corresponds exactly to CaRet.

Even if ConCaRet has no temporal modalities for the global successor, we prove that the modalities associated with the linear successor allow us to capture a meaningful fragment of *LTL*, which subsumes the fragment with only the infinitely often temporal operator. Within this fragment important classes of (global) propositional regular properties like invariants, as well as strong and weak fairness constraints, can be expressed. Note that by the undecidability result about model checking of *CRSM* against the fragment of global *LTL* using only the temporal operators next and infinitely often, it follows that any non-trivial global-*LTL* extension of ConCaRet including the global next operator leads to undecidability of *CRSM* model checking.

We show that model checking *CRSM* against ConCaRet is decidable. Our approach is based on automata-theoretic techniques: given a *CRSM* $\mathcal{S}$ and a formula φ, we construct a *Büchi CRSM* $\mathcal{S}_{\neg\varphi}$ (i.e., a *CRSM* equipped with generalized Büchi acceptance conditions) such that model checking $\mathcal{S}$ against φ is reduced to check the emptiness of the Büchi *CRSM* $\mathcal{S}_{\neg\varphi}$. We solve this last problem by a non-trivial reduction to the emptiness problem for a straightforward variant of classical Büchi Tree Automata. Our construction of $\mathcal{S}_{\neg\varphi}$ extends the construction given in [3] for an *RSM* and a CaRet formula. Overall, our model-checking algorithm runs in time exponential in both the maximal number ρ of modules that can be invoked simultaneously and the size of the formula, and thus matches the known lower bound for deciding CaRet model checking. Therefore, we prove that the model-checking problem of *CRSM* with respect to ConCaRet specifications is Exptime-complete. The main difference with respect to *RSM* is the time complexity in the size of the model that for *RSM* is polynomial, while for *CRSM*, it is exponential in ρ.

Related work. Besides the already mentioned research, other interesting work which is related to our results concerns the analysis of *Interacting Pushdown Systems* (Interacting *PDS*, for short) synchronizing via the standard primitives – locks, nested locks, pairwise rendezvous, and broadcasts – with respect to fragments of propositional *LTL* [22,23]. In particular, in [22], it is shown that model checking Interacting *PDS* by nested locks against *single-index* propositional *LTL* formulas which do not use next modalities is decidable in time singly-exponential in the number of locks. Like ConCaRet, single-index *LTL* formulas refer to the local computations of the threads (*PDS*) composing the system. More recently, [22], the authors delineate precisely the decidability boundary for model-checking dual-*PDS* (i.e., systems consisting of two *PDS*), interacting via the standard synchronization primitives, against standard propositional *global LTL*. For example, for dual-*PDS* interacting via pairwise rendezvous, the model-checking problem against the fragment of *LTL* formulas in positive normal form containing only occurrences of the next and always modalities is decidable. On the other hand, for the fragment of *LTL* using only the *infinitely often* modality (which is crucial for specifying liveness requirements), the problem is instead

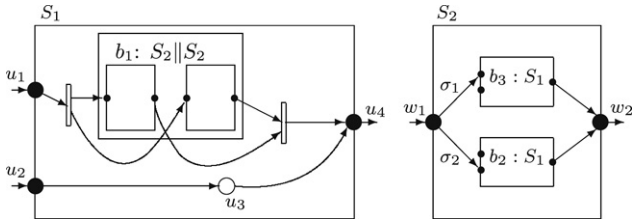

Fig. 1. A sample *CRSM*.

undecidable. Note that the communication by pairwise rendezvous is similar to the form of communication between threads in *CRSM*. However, *CRSM* and Interacting *PDS* are uncomparable since interacting *PDS* do not allow dynamic creation of threads and the synchronization in *CRSM* is only allowed between threads belonging to the same local context (in particular, the communicating threads share the same call stack).

Plan of the paper. In Section 2, we introduce the framework of *CRSM*. In Section 3, we recall *LTL* and define *LTL* model checking over *CRSM*. Then, we show that checking *CRSM* against *LTL* formulas using only the temporal operators *next* and *infinitely often* is already undecidable. In Section 4, we introduce the logic ConCaRet and in Section 5, we prove that model checking *CRSM* against ConCaRet is decidable in Exptime. Finally, we give a few conclusions. Some technical proofs are reported in the Appendix A.

2. Well-formed communicating recursive state machines

In this section we define the syntax and semantics of *well-formed Communicating Recursive State Machines* (*CRSM*, for short).

Syntax. A *CRSM* is an ordered collection of finite-state machines (*FSM*) augmented with the ability of refining a state with a collection of *FSM* (working in parallel) in a potentially recursive manner.

Definition 1. A *CRSM* $\mathcal{S}$ over a set of propositions *AP* is a tuple $\langle (S_1, \ldots, S_k), start \rangle$, where for $1 \leq i \leq k$, $S_i = \langle \Sigma_i, \Sigma_i^s, N_i, B_i, Y_i, En_i, Ex_i, \delta_i, \eta_i \rangle$ is a *module* and $start \subseteq \bigcup_{i=1}^{k} N_i$ is a set of *start nodes*. Each module S_i is defined as follows:

- Σ_i is a finite alphabet and $\Sigma_i^s \subseteq \Sigma_i$ is the set of *synchronization symbols*;
- N_i is a finite set of *nodes* and B_i is a finite set of *boxes* (with $N_i \cap B_i = \emptyset$);
- $Y_i : B_i \to \{1, \ldots, k\}^+$ is the *refinement function* which associates with every box a finite sequence of module indexes;
- $En_i \subseteq N_i$ (resp., $Ex_i \subseteq N_i$) is a set of *entry nodes* (resp., *exit nodes*);
- $\delta_i : (N_i \cup Retns_i) \times \Sigma_i \to 2^{N_i \cup Calls_i}$ is the *transition function*, where $Calls_i$ is the set $\{(b, e_1, \ldots, e_m) \mid b \in B_i, e_j \in En_{h_j}$ *for each* $1 \leq j \leq m$, *and* $Y_i(b) = h_1 \ldots h_m\}$ which denotes the set of *calls* of S_i, and $Retns_i = \{(b, x_1, \ldots, x_m) \mid b \in B_i, x_j \in Ex_{h_j}$ *for each* $1 \leq j \leq m$, *and* $Y_i(b) = h_1 \ldots h_m\}$ denotes the set of *returns* of S_i; we assume without loss of generality that exits have no outgoing transitions and $En_i \cap Ex_i = \emptyset$;
- $\eta_i : V_i \to 2^{AP}$ is the *labeling function*, with $V_i = N_i \cup Calls_i \cup Retns_i$ (V_i is the set of *vertices*).

We assume that $(V_i \cup B_i) \cap (V_j \cup B_j) = \emptyset$ for $i \neq j$. Also, we let $\Sigma = \bigcup_{i=1}^{i=k} \Sigma_i$, $\Sigma^s = \bigcup_{i=1}^{i=k} \Sigma_i^s$, $V = \bigcup_{i=1}^{i=k} V_i$, $Calls = \bigcup_{i=1}^{i=k} Calls_i$, $Retns = \bigcup_{i=1}^{i=k} Retns_i$, $N = \bigcup_{i=1}^{i=k} N_i$, $B = \bigcup_{i=1}^{i=k} B_i$, $En = \bigcup_{i=1}^{i=k} En_i$, and $Ex = \bigcup_{i=1}^{i=k} Ex_i$. Functions $\eta : V \to 2^{AP}$, $Y : B \to \{1, \ldots, k\}^+$, and $\delta : (N \cup Retns) \times \Sigma \to 2^{N \cup Calls}$ are defined as the natural extensions of functions η_i, Y_i and δ_i (with $1 \leq i \leq k$). The set of states of a module is partitioned into a set of nodes and a set of boxes. Performing a transition to a box b can be interpreted as a (parallel) procedure call (*fork*) which simultaneously activates a collection of modules (the list of modules given by the refinement function Y applied to b). Since Y gives a list of module indexes, a fork can activate different instances of the same module. Note that a transition leading to a box specifies the entry node (initial state) of each activated module. All the module instances, which are simultaneously activated in a call, run in parallel, whereas the calling module instance suspends its activity waiting for the return of the (parallel) procedure call. The return of a parallel procedure call to a box b is represented by a transition from b that specifies an exit node (exiting state) for each module copy activated by the procedural call (a synchronous return or *join* from all the activated modules).

In Fig. 1, we show a simple *CRSM* consisting of two modules S_1 and S_2. Module S_1 has two entry nodes u_1 and u_2, an exit node u_4, an internal node u_3 and one box b_1 that is mapped to the parallel composition of two copies of S_2. The module S_2 has an entry node w_1, an exit node w_2, and two boxes b_2 and b_3 both mapped to S_1. The transition from node u_1 to box b_1 in S_1 is a *fork* transition from u_1 (source) to the entry nodes of the two copies of S_2 (targets). Similarly, the transition from box b_1 to node u_4 is a *join* transition from the exit nodes of the two copies of S_2 (sources) to u_4 (target).

In our model, the communication is allowed only between module instances that are activated on the same fork and are not busy in a (parallel) procedure call. As for the communicating (finite-state) hierarchical state machines [4], the form of communication we allow is synchronous and maximal in the sense that if a component (module) takes a transition labeled by a synchronization symbol σ, then each other parallel component which has σ in its synchronization alphabet must be

able to take a transition labeled by σ. For instance, assuming that the symbols σ_1 and σ_2 in Fig. 1 are synchronization symbols, the two copies of S_2 which refine box b_1 either both take the transition labeled by σ_1 activating a copy of S_1 with start node u_1 or both take the transition labeled by σ_2 activating a copy of S_1 with start node u_2. Transitions labeled by symbols in $\Sigma \setminus \Sigma^s$ are instead performed independently without any synchronization requirement.

A *synchronization-free CRSM* is a *CRSM* in which $\Sigma^s = \emptyset$.

The *rank* of $\mathcal{S}$, $rank(\mathcal{S})$, is the maximum of $\{|Y(b)| \mid b \in B\}$. Note that if $rank(\mathcal{S}) = 1$, then $\mathcal{S}$ is a *Recursive State Machine* (*RSM*) as defined in [1].

Semantics. In order to define the semantics of *CRSM* we need some notation. A *tree* t is a prefix closed subset of $\mathbb{N}^*$ such that if $y \cdot i \in t$, then $y \cdot j \in t$ for all $0 \leq j < i$. The empty word ε is the *root* of t. The set of *leaves* of t is $leaves(t) = \{y \in t \mid y \cdot 0 \notin t\}$. For $y \in t$, the set of *children* of y (in t) is $children(t, y) = \{y \cdot i \in t\}$ and the set of *siblings* of y (in t) is $siblings(t, y) = \{y\} \cup \{y' \cdot i \in t \mid y = y' \cdot j \text{ for some } j \in \mathbb{N}\}$. For $y, y' \in \mathbb{N}^*$, we write $y \prec y'$ to mean that y is a proper prefix of y'.

The semantics of a *CRSM* $\mathcal{S}$ is defined in terms of a labeled transition system $K_{\mathcal{S}} = \langle Q, R\rangle$. Q is the set of (global) states which correspond to activation hierarchies of instances of modules, and are represented by finite trees whose locations are labeled with vertices and boxes of the *CRSM*. Leaves correspond to active modules and a path in the tree leading to a leaf y (excluding y) corresponds to the local call stack of the module instance associated with y.

Formally, a state is a pair (t, D), where t is a (finite) tree and $D : t \to B \cup V$ satisfies the following:

- if $y \in leaves(t)$, then $D(y) \in V$ (i.e. a vertex of $\mathcal{S}$);
- if $y \in t \setminus leaves(t)$ and $children(t, y) = \{y \cdot 0, \ldots, y \cdot m\}$, then $D(y) = b \in B$, $Y(b) = h_0 \ldots h_m$, and $D(y \cdot j) \in B_{h_j} \cup V_{h_j}$ for $j = 0, \ldots, m$.

The *global transition relation* $R \subseteq Q \times (2^{\mathbb{N}^*} \times 2^{\mathbb{N}^*}) \times Q$ is a set of tuples of the form $\langle (t, D), (\ell, \ell'), (t', D')\rangle$, also written as $(t, D) \xrightarrow{(\ell,\ell')} (t', D')$, where (t, D) (resp., (t', D')) is the source (resp., target) of the transition, ℓ keeps track of the elements of t corresponding to the (instances of) modules of $\mathcal{S}$ performing the transition, and:

- for an *internal move* $\ell' = \ell$,
- for a *call*, ℓ' points to the modules activated by the (parallel) procedure call, and
- for a *return from a call*, ℓ' points to the reactivated caller module.

Formally, $\langle (t, D), (\ell, \ell'), (t', D')\rangle \in R$ iff one of the following holds:

Single internal move: $t = t'$, there is $y \in leaves(t)$ and $\sigma \in \Sigma \setminus \Sigma^s$ such that $\ell = \ell' = \{y\}$, $D'(y) \in \delta(D(y), \sigma)$, and $D'(z) = D(z)$ for each $z \in t \setminus \{y\}$.

Synchronous internal move: $t' = t$, there are $y \in t$ with $siblings(t, y) = \{y_1, \ldots, y_m\}$, $\sigma \in \Sigma^s$, and indexes $k_1, \ldots, k_p \in \{1, \ldots, m\}$ such that the following holds: $\ell = \ell' = \{y_{k_1}, \ldots, y_{k_p}\} \subseteq leaves(t)$, $D'(z) = D(z)$ for each $z \in t \setminus \{y_{k_1}, \ldots, y_{k_p}\}$, $D'(y_{k_j}) \in \delta(D(y_{k_j}), \sigma)$ for each $1 \leq j \leq p$, and for each $j \in \{1, \ldots, m\} \setminus \{k_1, \ldots, k_p\}$, σ is *not* a synchronization symbol of the module associated with $D(y_j)$.

Module call: there is $y \in leaves(t)$ such that $D(y) = (b, e_0, \ldots, e_m) \in Call$, $t' = t \cup \{y \cdot 0, \ldots, y \cdot m\}$, $\ell = \{y\}$, $\ell' = \{y \cdot 0, \ldots, y \cdot m\}$, $D'(z) = D(z)$ for each $z \in t \setminus \{y\}$, $D'(y) = b$, and $D'(y \cdot j) = e_j$ for each $0 \leq j \leq m$.

Return from a call: there is $y \in t \setminus leaves(t)$ such that $\ell = children(t, y) = \{y \cdot 0, \ldots, y \cdot m\} \subseteq leaves(t)$, $\ell' = \{y\}$, $(D(y), D(y \cdot 0), \ldots, D(y \cdot m)) \in Retns$, $t' = t \setminus \{y \cdot 0, \ldots, y \cdot m\}$, $D'(z) = D(z)$ for each $z \in t' \setminus \{y\}$, and $D'(y) = (D(y), D(y \cdot 0), \ldots, D(y \cdot m))$.

For $v \in V$, we denote with $\langle v\rangle$ the global state $(\{\varepsilon\}, D)$ where $D(\varepsilon) = v$.

A *run* of $\mathcal{S}$ is an infinite path in $K_{\mathcal{S}}$ starting from a state of the form $\langle v\rangle$.

In Fig. 2, we illustrate a prefix of a sample run of the *CRSM* given in Fig. 1, where the synchronization symbols σ_1 and σ_2 used in component S_2 are equal. Moreover, assume that S_1 does not use synchronization symbols, i.e. $\Sigma_1^s = \emptyset$. Then, for instance, the transition from 0 to 1 and the transition from 6 to 7 are single internal moves, while the transition 2–3 is a synchronous internal move. Module calls are illustrated by the transitions 1–2, 3–4, and 5–6. Finally, the transition 8–9 corresponds to a return from a call.

We conclude this section by describing a small example, which shows as the flow of control of a standard algorithm for visiting a blank and white image encoded by a quadtree can be naturally modeled by exploiting the ability of *CRSM* of combining parallel calls and recursion.

In a quadtree, the image is statically split into four sub-images corresponding to the four quadrants Top-Right, Top-Left, Bottom-Left and Bottom-Right. Each quadrant that is not a uniform (totally black or white) area of the image, is in its turn split into four quadrants. The structure of the image representation therefore allows a natural decomposition of a computational task into subtasks and to exploit a parallel computational environment (grid computing, for instance).

The *CRSM* of the example consists of only one module which calls recursively and in parallel instances of itself. The module is depicted in Fig. 3. A module starts its activity in an entry node which keeps track of the quadrant where the module is supposed to work (i.e., the entry nodes *TR*, *TL*, *BL* and *BR*). The module checks whether the assigned region is uniformly black ($Check_B$), uniformly white ($Check_W$), or it is not uniform ($Check_G$). In the first two cases, the activity of the

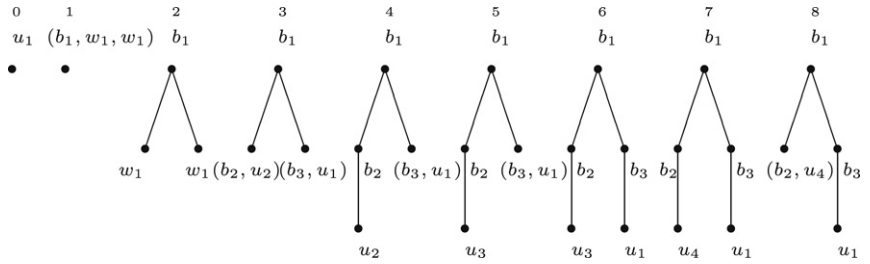

Fig. 2. A sample run of the *CRSM* of Fig. 1.

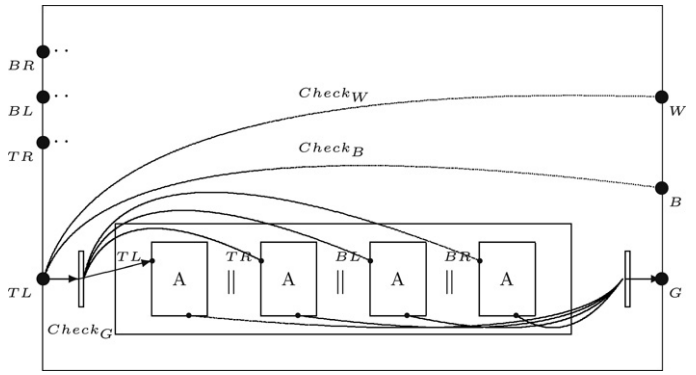

Fig. 3. The *CRSM* for the Quadtree inspection.

module terminates, leading to the exit nodes *B* and *W*, respectively. If instead the region is not uniform, then an instance of the module is recursively called in parallel for each of the four quadrants of the region. With reference to the figure, such a parallel call corresponds with entering the box, which is refined in four copies of the module itself, each of which initialized with a different entry node (i.e., a different quadrant). The module has a third exit node *G*, which is taken when the region is not uniformly black or white.

As previously said, the *CRSM* describes the control flow of the recursive and parallel call of modules and abstracts away from the details of the data inspection (the result of the uniformity check over quadrants is represented by symbols $Check_B$, $Check_W$, and $Check_G$). Notice that each module instance assigned to a different quadrant acts independently (with respect to the module instances associated with the other quadrants), and so the set of synchronization symbols is the empty set.

The return from a recursive parallel call composes the values reported by the modules activated on the four quadrants. The reached exit node is assumed to be the node *G*, when either the exit node of each module in the parallel call is *G* or the modules return a different value. For the sake of the presentation, only the transitions from the entry node *TL* are explicitly depicted. Analogous transitions are to be placed from each other of the entry nodes to the box and the exit nodes *B* and *W*. Also, the return transitions from the box are not completely depicted.

3. *LTL* model-checking

In this section we show that the model-checking problem of *CRSM* against standard propositional *LTL* is undecidable. In fact, we prove that undecidability already holds for synchronization-free *CRSM* and for the fragment of *LTL* using only the temporal operators *next* and *infinitely often* (which is strictly less expressive than the one with *next* and *eventually*). The undecidability proof has analogies with that described in [27] (see also [7]) to show undecidability of *LTL* model checking for PA-processes. However, the *LTL* formula exploited in [27] uses the *next* and *eventually* modalities. Thus, our undecidability result does not follow from [27,7].

Syntax and semantics of *LTL*. The syntax of *LTL* [30] over a finite set *AP* of atomic propositions is defined as follows

$$\varphi ::= tt \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc \varphi \mid \varphi \mathsf{U} \varphi,$$

where *tt* denotes "true", $p \in AP$, U is the *until* operator, and $\bigcirc$ is the *next* operator. We also will use some standard shortcuts: $\varphi \vee \psi$ stands for $\neg(\neg\varphi \wedge \neg\psi)$, $\varphi \rightarrow \psi$ stands for $\neg\varphi \vee \psi$, $\Diamond \varphi$ stands for $tt \mathsf{U} \varphi$ (the *eventually* operator), $\Box\varphi$ stands for $\neg\Diamond \neg\varphi$ (the *always* operator), $\overset{\infty}{\Diamond} \varphi$ stands for $\Box\Diamond \varphi$ (the *infinitely often* operator), and $\overset{\infty}{\Box}\varphi$ stands for $\neg\overset{\infty}{\Diamond} \neg\varphi$ (the *almost always* operator).

The logic *LTL* is interpreted on infinite words over the alphabet 2^{AP}. Given such a word $w = w(0)w(1)w(2)\ldots$, by w^i we denote the ith suffix of w, i.e. $w^i = w(i)w(i+1)\ldots$. The semantics of *LTL* formulas is defined as follows (we omit the rules for boolean connectives, which are standard):

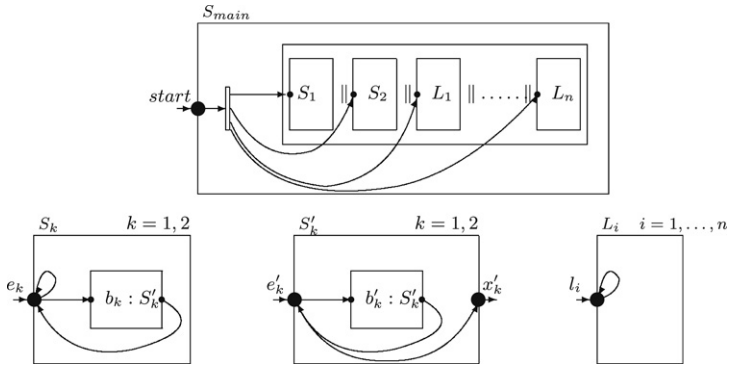

Fig. 4. The *CRSM* used in the undecidability proof.

$w \models p$ iff $p \in w(0)$
$w \models \bigcirc\varphi$ iff $w^1 \models \varphi$
$w \models \varphi_1 \mathsf{U} \varphi_2$ iff $\exists i \geq 0\,.\,(\, w^i \models \varphi_2$ and $\forall\, 0 \leq j < i\,.\, w^j \models \varphi_1)$.

We say that an infinite word w *satisfies* φ whenever $w \models \varphi$.

For a set $\{O_1, \ldots, O_n\}$ of temporal operators, let $LTL(O_1, \ldots, O_n)$ be the *LTL* fragment consisting of all formulas with temporal operators $O_1, \ldots, O_n$ only.

***LTL* model checking of *CRSM*.** Let $\mathcal{S} = \langle (S_1, \ldots, S_k), start \rangle$ be a *CRSM* and $\varsigma = (t, D) \xrightarrow{(\ell,\ell')} (t', D')$ be a (global) transition of $K_{\mathcal{S}}$. The 2^{AP}-*label* of ς is the set of atomic propositions which label the current vertices of the module instances that represent the target ℓ' of the transition, i.e. the set $\bigcup_{y \in \ell'} \eta(D'(y))$.[1] Given a run $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$ of $\mathcal{S}$ and an infinite word $w = w(0)w(1)\ldots$ over 2^{AP}, we say that π is a *run over* w if for every $i \geq 0$, $w(i)$ is the 2^{AP}-label of the global transition $q_i \xrightarrow{(\ell_i,\ell'_i)} q_{i+1}$. The *language generated by* $\mathcal{S}$, denoted by $L(\mathcal{S})$, is the set of infinite words w such that there is a run of $\mathcal{S}$ over w starting from a state $\langle v \rangle$ with $v \in start$.

Given a *CRSM* $\mathcal{S}$ and an *LTL* formula φ, we say that $\mathcal{S}$ *satisfies* φ if for each word $w \in L(\mathcal{S})$, w satisfies φ. The *LTL model-checking problem of CRSM* is to decide for a given *CRSM* $\mathcal{S}$ and *LTL* formula φ, whether $\mathcal{S}$ satisfies φ.

Undecidability of model checking *CRSM* against *LTL*$(\overset{\infty}{\Diamond}, \bigcirc)$. We show that the model-checking problem of *CRSM* against $LTL(\overset{\infty}{\Diamond}, \bigcirc)$ is undecidable by a reduction from the non-halting problem for Minsky 2-counter machines, which is known to be undecidable [29].

First we recall the definition of Minsky 2-counter machines. A *Minsky 2-counter machine*, or a *machine* for short, is a finite sequence $M = l_1 : i_1,\ l_2 : i_2,\ \ldots,\ l_{n-1} : i_{n-1},\ l_n : \texttt{halt}$, where $n \geq 1$, $l_1, l_2, \ldots, l_n$ are *labels*, and each i_j is an instruction for

- *increment*: $\texttt{c}_k\texttt{:=c}_k\texttt{+1; goto } l_r$, or
- *test-and-decrement*: $\texttt{if c}_k\texttt{>0 then c}_k\texttt{:=c}_k\texttt{-1; goto } l_r \texttt{ else goto } l_s$

where $k \in \{1, 2\}$ and $1 \leq r, s \leq n$.

The machine M induces a transition relation $\longrightarrow$ over *configurations* of the form (l_j, v_1, v_2), where l_j is a label of an instruction to be executed and $v_1, v_2 \geq 0$ represent current values of counters $\texttt{c}_1$ and $\texttt{c}_2$, respectively.
We say that the machine M *halts* if there are numbers $v_1, v_2 \geq 0$ such that $(l_1, 0, 0) \longrightarrow^* (l_n, v_1, v_2)$. The *non-halting problem* is to decide whether a given machine M does not halt. This problem is known to be undecidable [29].

Theorem 1. *The model-checking problem of synchronization-free* CRSM *against the logic* LTL$(\overset{\infty}{\Diamond}, \bigcirc)$ *is undecidable.*

Proof. The proof is given by a reduction from the non-halting problem for Minsky 2-counter machines. Given a machine M, we construct a synchronization-free *CRSM* $\mathcal{S}$ and an *LTL* formula φ using only the temporal modalities $\overset{\infty}{\Diamond}$ and $\bigcirc$ such that M does not halt if and only if $\mathcal{S}$ satisfies $\neg\varphi$.

The synchronization-free *CRSM* $\mathcal{S}$ is defined as follows. The alphabet Σ is a singleton and the unique symbol of Σ is not a synchronization symbol. $\mathcal{S}$ consists of $n + 5$ modules, which are depicted in Fig. 4: the *main* module S_{main}, the modules $L_1, \ldots, L_n$ associated with the labels of the instructions of M, and for $k = 1, 2$, the modules S_k and S'_k (which are used to encode the instructions associated with the counter $\texttt{c}_k$). Each of these modules has one entry. The entry node of the main

[1] The 2^{AP}-label could be defined in terms of the module instances performing the transition. However, concerning decidability issues there is no difference with respect to the other definition.

module S_{main} is the unique *start* node of $\mathcal{S}$ and there is a transition from such a node to a box that is refined in the parallel composition of exactly one copy of module S_1, one copy of module S_2, and one copy of module L_i for $i = 1, \ldots, n$. Module L_i has only one vertex l_i, which is an entry node, and there is a transition from l_i to itself. For each $k = 1, 2$, modules S_k and S'_k are defined as follows. Module S_k has one entry node e_k and a box b_k that is mapped to a copy of module S'_k. Moreover, it has a self-loop on the entry node e_k, a transition from e_k to the call vertex (b_k, e'_k), and a transition from the return vertex (b_k, x'_k) to e_k. Module S'_k has one entry node e'_k, one exit node x'_k, and a box b'_k which is mapped to a copy of S'_k itself. Also, there is a transition from e'_k to the call vertex (b'_k, e'_k), a transition from the return vertex (b'_k, x'_k) to e'_k, and a transition from the entry node to the exit node. Intuitively, module S_k (resp., S'_k) is active when the value of the counter c_k is null (resp., is not null). The module calls associated with the call vertices (b_k, e'_k) and (b'_k, e'_k) correspond to the increment of value of counter c_k, while the returns from these calls (by the exit node x'_k) encode a decrement operation on c_k. Finally, the labeling η of the vertices of $\mathcal{S}$ is defined as follows ($k = 1, 2$):

- the set of propositions labeling the vertices of S_{main} is empty;
- $\eta(l_i) = \{l_i\}$ for $i = 1, \ldots, n$;
- $\eta((b_k, e'_k)) = \eta((b'_k, e'_k)) = \{inc_k\}$;
- $\eta(x'_k) = \{dec_k\}$;
- $\eta(e_k) = \{in_k, zero_k\}$;
- $\eta(e'_k) = \{in_k\}$ and $\eta((b_k, x'_k)) = \eta((b'_k, x'_k)) = \{ret_k\}$.

Intuitively, the proposition inc_k (resp., dec_k) denotes the increment (resp., the decrement) of counter c_k, while the proposition $zero_k$ means that the value of c_k is null. Note that at any instant there are exactly $n + 2$ active module instances: for each $1 \leq i \leq n$, there is one copy of L_i (whose local stack height is always one) and for each $k = 1, 2$, there is one copy I_k of either module S_k or S'_k whose local stack height represents the current value of counter c_k (note that if I_k is a copy of S_k, then its local stack height is 1, encoding the null value of c_k). While the semantics of *CRSM* cannot enforce a synchronization between these module instances, we can achieve this by a suitable *LTL* formula φ. Formula φ is defined in such a way that it allows us to select only the runs of $\mathcal{S}$ which are faithful to the evolution of machine M.

In order to define such a φ, first we define a formula ψ describing a correct step of the constructed *CRSM* when simulating machine M.

$$\begin{aligned}\psi \;=\; & \bigwedge_{l_i:\mathtt{c}_k:=\mathtt{c}_k+1;\ \mathtt{goto}\ l_r} \Big(l_i \rightarrow (\bigcirc inc_k \wedge \bigcirc^2 in_k \wedge \bigcirc^3 l_r)\Big) \wedge \\ & \bigwedge_{l_i:\mathtt{if}\ \mathtt{c}_k>0\ \mathtt{then}\ \mathtt{c}_k:=\mathtt{c}_k-1;\ \mathtt{goto}\ l_r\ \mathtt{else}\ \mathtt{goto}\ l_s} \Big(l_i \rightarrow \big((\bigcirc zero_k \wedge \bigcirc^2 l_s) \vee \\ & \qquad\qquad (\bigcirc dec_k \wedge \bigcirc^2 ret_k \wedge \bigcirc^3 in_k \wedge \bigcirc^4 l_r)\big)\Big).\end{aligned}$$

Moreover, we define a formula ρ describing a correct step of resetting counters and restarting the simulation.

$$\begin{aligned}\rho = l_n \rightarrow \Big(& (\bigcirc zero_1 \wedge \bigcirc^2 zero_2 \wedge \bigcirc^3 l_1) \vee (\bigcirc dec_1 \wedge \bigcirc^2 ret_1 \wedge \bigcirc^3 in_1 \wedge \\ & \bigcirc^4 l_n) \vee (\bigcirc zero_1 \wedge \bigcirc^2 dec_2 \wedge \bigcirc^3 ret_2 \wedge \bigcirc^4 in_2 \wedge \bigcirc^5 l_n)\Big).\end{aligned}$$

The formula $\varphi = \overset{\infty}{\Box}(\psi \wedge \rho) \wedge \overset{\infty}{\Diamond} l_n$ requires that at some point the l_n action (associated with the *halt* instruction) occurs, both counters are reset, a correct simulation is started, and whenever the simulation ends (with l_n action), this sequence of events is performed again. Moreover, note that φ is satisfied only if the action l_n appears infinitely many times. Hence, there is a run of $\mathcal{S}$ from $\langle start \rangle$ over a word w satisfying φ if and only if M halts. That is, the machine M does not halt if and only if $\mathcal{S}$ satisfies $\neg\varphi$. □

4. The temporal logic ConCaRet

In this section, we first introduce different notions of local successor along *CRSM* runs. Then, we define the logic ConCaRet and give some examples that illustrate its expressiveness. We conclude by showing that an interesting fragment of (global) *LTL* can be captured by ConCaRet formulas.

4.1. Local successors in structured computations of CRSM

We are interested in defining a logic to express properties concerning the local behaviors of module instances. Therefore, we need to introduce different notions of local successor of a module instance along a given run.

We fix a *CRSM* $\mathcal{S}$ and a run $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$ of $\mathcal{S}$, where $q_i = (t_i, D_i)$ for any i. We denote by Q_π the set $\{(i, y) \mid y \in leaves(t_i)\}$. An element (i, y) of Q_π, called a *local state* of π, corresponds to an instance of a module that at state

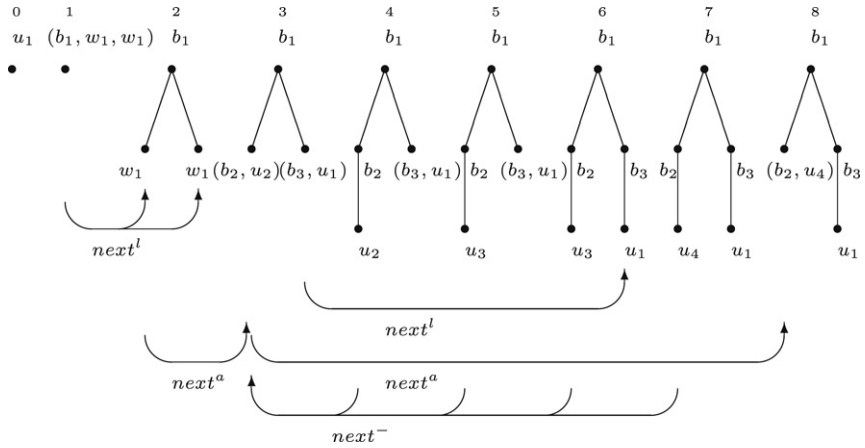

Fig. 5. Linear successors, abstract successors and callers over the sample run of Fig. 2.

q_i is active[2] and is currently in the vertex $D_i(y)$. Note that for a given local state (i, y), the set $\{(i, y') \mid y' \prec y\}$ gives the corresponding local call stack.

Now, we define two notions of *next* local state in a computation of a *CRSM*. For $(i, y) \in Q_\pi$, $next^\ell_\pi(i, y)$ gives the set of active module instances, called *linear successors* of (i, y), that are obtained by the first transition of the run (after state q_i) affecting the module instance corresponding to (i, y). Note that such a transition may occur at $j \geq i$ or may not occur at all. In the former case, if $D_i(y) \notin Calls \cup Ex$ (i.e. for an internal move[3]), then $next^\ell_\pi(i, y) = \{(j, y)\}$ is a singleton and points to the vertex (node or call) which is the target of the local transition performed by the module instance associated with (i, y). If instead $D_i(y) \in Ex$ (i.e., for a return from a call), then the unique linear successor points to the reactivated caller module. Finally, if $D_i(y) \in Calls$, then the linear successors correspond to the entry nodes of the called modules. Formally, we define $next^\ell_\pi(i, y) = \emptyset$, if $\{m \geq i \mid y \in \ell_m\} = \emptyset$. Otherwise, $next^\ell_\pi(i, y)$ given by

$$\{(h + 1, y') \mid h = min\{m \geq i \mid y \in \ell_m\}, y' \in \ell'_h, \text{ and either } y' \preceq y \text{ or } y \preceq y'\}.$$

Let us consider the following Fig. 5, which illustrates the same sample run π in Fig. 2 of the *CRSM* given in Fig. 1. For instance, we have $next^\ell_\pi(3, 1) = \{(6, 1 \cdot 1)\}$ and $next^\ell_\pi(1, \varepsilon) = \{(2, 0), (2, 1)\}$.

Note that if $rank(\mathcal{S}) = 1$ (i.e. $\mathcal{S}$ is an *RSM*), then there is exactly one local state at any instant and the *linear* successor corresponds to the (standard) *global* successor.

For each $(i, y) \in Q_\pi$, we also give a notion of *abstract successor*, denoted $next^a_\pi(i, y)$. Intuitively, the abstract successor captures the local computations within a module A skipping over invocations of other modules called from A. Formally, if (i, y) corresponds to a call that returns, i.e., $D_i(y) \in Calls$, $next^\ell_\pi(i, y) \neq \emptyset$, and there is $j > i$ such that $y \in leaves(t_j)$, then $next^a_\pi(i, y) = (h, y)$ where h is the smallest of such j (the local state (h, y) corresponds to the matching return). For the internal moves, the abstract successor coincides with the (unique) linear successor, i.e., if $next^\ell_\pi(i, y) = \{(j, y)\}$, then $next^a_\pi(i, y) = (j, y)$ (note that in this case $D_i(y) \in Retns \cup N \setminus Ex$). In all the other cases, the abstract successor is not defined and we denote this with $next^a_\pi(i, y) = \perp$. For instance, in Fig. 5, we have $next^a_\pi(2, 0) = (3, 0)$ and $next^a_\pi(3, 0) = (8, 0)$.

Linear and abstract successors are forward modalities. We define also a backward modality that assigns to a local state (i, y) the 'innermost call' that has activated (i, y). Notice that only local states of the form (i, ε) have no callers. Formally, the *caller* of (i, y) (if any), written $next^-_\pi(i, y)$, is defined as follows: if $y = y' \cdot m$ for some $m \in \mathbb{N}$, then $next^-_\pi(i, y) = (j, y')$, where $j = max\{h < i \mid y' \in leaves(t_h)\}$; otherwise, $next^-_\pi(i, y)$ is undefined, written $next^-_\pi(i, y) = \perp$. For instance, in Fig. 5, we have $next^-_\pi(7, 0 \cdot 0) = (3, 0)$.

The above defined notions of forward and backward successor allow us to define sequences of local moves (i.e. moves affecting local states) in a run. For $(i, y) \in Q_\pi$, the set of *linear paths* of π *starting from* (i, y) is the set of (finite or infinite) maximal sequences of local states $r = (j_0, y_0)(j_1, y_1) \ldots$ such that $(j_0, y_0) = (i, y)$, $(j_{h+1}, y_{h+1}) \in next^\ell_\pi(j_h, y_h)$ for any h, and either r is infinite or leads to a local state (j_p, y_p) such that $next^\ell_\pi(j_p, y_p) = \emptyset$. Analogously, the notion of *abstract path* (resp., *caller path*) of π *starting from* (i, y) can be defined by using in the above definition the abstract successor (resp., caller) instead of the linear successor. Note that a caller path is always finite and uniquely determines the content of the call stack locally to the instance of the module active at (i, y).

For module instances involved in a call, i.e., corresponding to pairs (i, y) such that $y \in t_i \setminus leaves(t_i)$, we denote the local state (if any) at which the call pending at (i, y) will return by $return_\pi(i, y)$. Formally, if $y \in leaves(t_j)$ for some $j > i$, then $return_\pi(i, y) = (h, y)$ where h is the smallest of such j, otherwise $return_\pi(i, y) = \perp$. Finally, we denote by $call_\pi(i, y)$,

[2] This is in opposition to the status of pairs $(i, z) \in t_j \setminus leaves(t_j)$ that correspond to instances of modules that are waiting for a return from a call and thus are currently inactive.

[3] We recall that exit nodes have no outgoing transitions.

the local state corresponding to the call associated with the module instance at (i, y). Formally, $call_\pi(i, y) = (h, y)$ where $h = max\{j < i \mid y \in leaves(t_j)\}$.

4.2. ConCaRet: Syntax and semantics

Let AP be a finite set of *atomic propositions*. The logic ConCaRet over AP is the set of formulas inductively defined as follows:

$$\varphi ::= tt \mid p \mid call \mid ret \mid int \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc^b \varphi \mid \varphi \mathsf{U}^b \varphi \mid \|\varphi$$

where $b \in \{\exists, \forall, a, -\}$ and $p \in AP$.

A formula is interpreted over runs of a *CRSM* $\mathcal{S} = \langle (S_1, \ldots, S_k), start \rangle$. Let $\pi = q_0 \xrightarrow{(\ell_0, \ell'_0)} q_1 \xrightarrow{(\ell_1, \ell'_1)} \ldots$ be a run of $\mathcal{S}$ with $q_i = (t_i, D_i)$ for $i \geq 0$. For a linear (resp., abstract, caller) path $r = (j_0, y_0)(j_1, y_1) \ldots$ of π, we denote by $r(i)$ the ith local state along r (i.e. the local state (j_i, y_i)). The truth value of a formula with respect to a local state (i, y) of π is inductively defined as follows (we omit the rules for boolean connectives which are standard):

$(i, y) \models_\pi p$ iff $p \in \eta(D_i(y))$ (where $p \in AP$)
$(i, y) \models_\pi call$ iff $D_i(y) \in Calls$
$(i, y) \models_\pi ret$ iff $D_i(y) \in Retns$
$(i, y) \models_\pi int$ iff $D_i(y) \in N$
$(i, y) \models_\pi \bigcirc^a \varphi$ iff $next^a_\pi(i, y) \neq \bot$ and $next^a_\pi(i, y) \models_\pi \varphi$
$(i, y) \models_\pi \bigcirc^- \varphi$ iff $next^-_\pi(i, y) \neq \bot$ and $next^-_\pi(i, y) \models_\pi \varphi$
$(i, y) \models_\pi \bigcirc^\exists \varphi$ iff there is $(j, z) \in next^\ell_\pi(i, y)$ such that $(j, z) \models_\pi \varphi$
$(i, y) \models_\pi \bigcirc^\forall \varphi$ iff for all $(j, z) \in next^\ell_\pi(i, y)$, $(j, z) \models_\pi \varphi$
$(i, y) \models_\pi \varphi_1 \mathsf{U}^b \varphi_2$ iff $\exists h \geq 0 \,.\, (r(h) \models \varphi_2$ and $\forall\, 0 \leq j < h \,.\, r(j) \models \varphi_1)$ where
if $b = a$, then r is the abstract path from (i, y),
if $b = -$, then r is the caller path from (i, y).
if $b = \exists$, then r is *some* linear path from (i, y),
if $b = \forall$, then r range over *all* linear paths from (i, y)
(in this case, index h depends on the specific path r)
$(i, y) \models_\pi \|\varphi$ iff for $h = min\{j \leq i \mid (j, y) \in Q_\pi$ and $next^\ell_\pi(j, y) = next^\ell_\pi(i, y)\}$ and for all $z \in siblings(t_h, y) \setminus \{y\}$:
– if $D_h(z) \in V$, then $(h, z) \models_\pi \varphi$;
– if $D_h(z) \in B$, then $call_\pi(h, z) \models_\pi \varphi$.

For each type of local successor (forward or backward), the logic provides the corresponding versions of the usual (global) next operator $\bigcirc$ and until operator U. For instance, formula $\bigcirc^- \varphi$ demands that the caller of the current local state satisfies φ, while $\varphi_1 \mathsf{U}^- \varphi_2$ demands that the caller path (that is always finite) from the current local state satisfies $\varphi_1 \mathsf{U} \varphi_2$. Moreover, the forward modalities defined over the linear paths are branching since they quantify over the possible linear successors of the current local state. Thus, we have both existential and universal versions of the standard modalities $\bigcirc$ and U. Finally, the operator $\|$ is a new modality introduced to express properties of parallel modules. The formula $\|\varphi$ holds at a local state (i, y) of a module instance I *iff*, being $h \leq i$ the time when vertex $D_i(y)$ was first entered and such that I has been idle from h to i, every module instance (different from I) in parallel with I and not busy in a module call (at time h) satisfies φ at time h, and every module instance in parallel with I and busy in a module call (at time h) satisfies φ at the call time.

Note that the semantics of the parallel operator ensures the following desirable property for the logic ConCaRet: for each pair of local states (i, y) and (j, y) such that $i < j$ and $next^\ell_\pi(j, y) = next^\ell_\pi(i, y)$ (i.e., associated with a module instance which remains idle from i to j), the set of formulas that hold at (i, y) coincides with the set of formulas that hold at (j, y).

Given a run π of $\mathcal{S}$ and a formula φ, we say that the run π *satisfies* φ, written $\pi \models \varphi$, if $(0, \varepsilon) \models_\pi \varphi$. Moreover, we say that $\mathcal{S}$ satisfies φ, written $\mathcal{S} \models \varphi$, iff for every $v \in start$ and for every run π of $\mathcal{S}$ starting from $\langle v \rangle$, it holds that $\pi \models \varphi$.
Now, we can define the model-checking problem we are interested in:

Model-checking problem.

Given a CRSM $\mathcal{S}$ *and a* ConCaRet *formula* φ, *does* $\mathcal{S} \models \varphi$? In the following, as we have done for *LTL*, we use $\varphi \vee \psi$ as an abbreviation for $\neg(\neg\varphi \wedge \neg\psi)$, and $\varphi \rightarrow \psi$ as an abbreviation for $\neg\varphi \vee \psi$. Analogously, we use $\Diamond^b \varphi$ as an abbreviation for $tt \mathsf{U}^b \varphi$, for $b \in \{\exists, \forall, a, -\}$. Also, for $b \in \{a, -\}$, let $\Box^b \varphi$ stand for $\neg\Diamond^b \neg\varphi$, $\Box^\forall \varphi$ stand for $\neg\Diamond^\exists \neg\varphi$, and $\Box^\exists \varphi$ stand for $\neg\Diamond^\forall \neg\varphi$.

4.3. Specifying requirements with ConCaRet

In this section, we illustrate some interesting properties which can be expressed in ConCaRet.

Abstract modalities. As in CaRet [3], the abstract modalities can be used to specify a variety of non-regular pushdown properties, which require matching of calls and returns.

Parallel Pre- and Post-conditions:

We can encode partial and total correctness requirements of procedures (modules) with respect to pre- and post-conditions. Since *CRSM* model parallel calls also, pre- and post-conditions can be extended to multiple threads activated in parallel. A *parallel partial correctness requirement* for two modules A and B asserts that whenever modules A and B are both activated in parallel and pre-condition p holds, then if both A and B terminate, post-condition q holds upon the synchronous return. *Total correctness* additionally requires both A and B to terminate. Assuming that parallel calls to the modules A and B are characterized by the proposition $p_{A,B}$, then the total correctness is expressed by the formula:

$$\varphi_{total} = \Box^{\forall}\big[(call \wedge p \wedge p_{A,B}) \rightarrow \bigcirc^{a} q\big]$$

while the partial correctness is expressed by the formula

$$\varphi_{partial} = \Box^{\forall}\big[(call \wedge p \wedge p_{A,B}) \rightarrow \neg\bigcirc^{a}\neg q\big].$$

Local properties:

Local properties refer to the local computations (abstract paths) of a module A that skip over invocations of other modules called from A. The abstract versions of the *LTL* temporal modalities can be used to specify regular properties of such local paths. For example, if the proposition p_A denotes that the control is within a module A, then the formula $\Box^{\forall}\big[(p \wedge p_A) \rightarrow \Diamond^{a} q\big]$ specifies the local version of the response property "every request p is followed by a response q".

Caller modalities. With caller modalities we can express as in CaRet properties that require inspection of the (local) call stacks of modules. As shown in [20,19], stack inspection can specify a variety of non-regular security properties in modern programming languages such as Java.

For instance, a stack inspection property such as "if module A is called, then module B must not be on the local call stack of A" (i.e., module A cannot be invoked within the context of module B") can be expressed by the formula

$$\Box^{\forall}\big[(p_A \wedge \bigcirc^{-} tt) \rightarrow \neg\Diamond^{-} p_B\big]$$

where p_A (resp., p_B) denotes that the control is within module A (resp., B). More in general, one can require that when a critical thread (module) is activated, then all modules in the local call stack have the necessary privileges. (See [3] for more examples using these modalities).

Parallel modality. The parallel modality gives the ability to express alignment properties among parallel threads. For instance, when a parallel call occurs, we can require that there will be an instant in the future such that the same property φ holds in all the parallel threads activated by the parallel call. Such a property can be written as

$$\Box^{\forall}\big[call \rightarrow \bigcirc^{\exists}(\Diamond^{a}(\varphi \wedge \| \varphi))\big].$$

Thus, for example, if the atomic proposition p_σ is associated only with vertices (nodes or returns) where a transition labeled by the synchronization symbol σ can be taken, and p_σ stands for φ in the above formula, the property expressed is that there will be a point in the future where all the threads activated by the call are ready for a maximal synchronization by the common synchronization symbol σ. In general, we will be able to express reactivity properties of modules, namely the ability of a module to continuously interact with its parallel components.

We can also express properties specifying constraints on the use of shared resources by modules operating in parallel. Assume that the proposition p is associated with a shared resource. Modules activated in parallel use the resource in a mutually exclusive way iff the following formula holds: $\Box^{\forall}(p \rightarrow \| \neg p)$.

As another example, let us consider the requirement "every time the resource p is available for a module I, then it will be eventually available for all the modules in parallel with I". This requirement can be expressed by the formula

$$\Box^{\forall}(p \rightarrow \| \Diamond^{a} p).$$

Linear modalities. Linear modalities refer to the branching structure of *CRSM* computations. They can be used, for instance, to express invariance properties like "every time a call occurs, then each activated module has to satisfy property φ" or "modules B and C will be never activated simultaneously in the local context of A" (where for local context of A, we mean the whole computation within A including the possible nested calls to other modules). In the first case, the corresponding formula is

$$\Box^{\forall}(call \rightarrow \bigcirc^{\forall}\varphi)$$

while in the second case we have

$$\Box^{\forall}\big[(call \wedge \bigcirc^{\exists} p_B \wedge \bigcirc^{\exists} p_C) \rightarrow \neg\Diamond^{-} p_A\big]$$

where p_A (resp., p_B, p_C) denotes that the control is within module A (resp., B, C). We can also express simple global eventually properties of the kind "every time the computation starts from module A, then module B eventually will be activated", expressed by the formula $p_A \rightarrow \Diamond^{\exists} p_B$. We can also capture more interesting global properties as shown in the next section.

4.4. Checking global LTL properties with ConCaRet

In this section we show that ConCaRet captures a meaningful fragment of the logic *LTL*. This fragment is defined as follows:

$$\phi ::= \Box\psi \mid \overset{\infty}{\Box}\psi \mid \neg\phi \mid \phi \wedge \phi$$

where ψ denotes a propositional formula over AP (i.e. a boolean combination of atomic propositions in AP). We denote this *LTL* fragment by Υ. Within this fragment important classes of (global) regular properties like invariants, as well as strong and weak fairness constraints, can be expressed. Also, note that Υ subsumes $LTL(\overset{\infty}{\Diamond})$ since $\overset{\infty}{\Box}$ is the dual of $\overset{\infty}{\Diamond}$ and $LTL(\overset{\infty}{\Diamond})$ is equivalent to the fragment $\phi = \overset{\infty}{\Diamond}\psi \mid \neg\phi \mid \phi \wedge \phi$, where ψ is a propositional formula. Furthermore, note that if we add the global next operator to the fragment Υ, then by Theorem 1 the model-checking problem of (synchronization-free) *CRSM* is undecidable. Now, we show that the model-checking problem of *CRSM* against Υ formulas can be reduced to the model-checking problem for ConCaRet.

Theorem 2. *Given a* CRSM $\mathcal{S}$ *and a formula* φ *of* Υ*, one can construct a new* CRSM $\mathcal{S}'$ *and a* ConCaRet *formula* φ'*, whose sizes are polynomial in the sizes of* $\mathcal{S}$ *and* φ*, such that* $\mathcal{S}$ *satisfies* φ *if and only if* $\mathcal{S}'$ *satisfies* φ'*.*

Proof. First, we give a translation for the class of synchronization-free *CRSM*. Then, we show how to modify the proposed construction for general *CRSM*. Note that in synchronization-free *CRSM* the only global transitions $q \xrightarrow{(\ell,\ell')} q'$ such that ℓ' is not a singleton correspond to a module call.

Now, we define a function f which maps every formula ϕ in Υ to a ConCaRet formula $f(\phi)$, such that for every synchronization-free *CRSM* $\mathcal{S}$, $\mathcal{S}$ satisfies ϕ if and only if $\mathcal{S}$ satisfies $f(\phi)$. Since the operators $\Box$ and $\overset{\infty}{\Box}$ are distributive with respect to $\wedge$, we can assume that the propositional formulas ψ have the form $\bigvee_{i\in I} p_i \vee \bigvee_{j\in J} \neg q_j$. The mapping f is defined as follows:

- f is homomorphic with respect to the boolean connectives;
- for $\psi = \bigvee_{i\in I} p_i \vee \bigvee_{j\in J} \neg q_j$, $f(\Box\psi) = \Box^{\forall}\widetilde{\psi}$ and $f(\overset{\infty}{\Box}\psi) = \Diamond^{\forall}\Box^{\forall}\widetilde{\psi}$, where $\widetilde{\psi} = \bigcirc^{\exists} tt \rightarrow \big((\bigvee_{i\in I} \bigcirc^{\exists} p_i) \vee (\bigvee_{j\in J} \bigcirc^{\forall} \neg q_j)\big)$.

Correctness of the translation directly follows from the following claim:

Claim 1. *Let* $\mathcal{S}$ *be a synchronization-free* CRSM *and* $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$ *be a run over the word* w*. Then, for every* Υ*-formula* ϕ*,* $\pi \models f(\phi)$ *if and only if* $w \models \phi$*.*

Proof of the claim. The proof is by structural induction on ϕ. The only non-trivial cases are when ϕ is either of the form $\overset{\infty}{\Box}\psi$ or of the form $\Box\psi$. We analyze the first case (the second being simpler). Let $\phi = \overset{\infty}{\Box}\psi$ with $\psi = \bigvee_{i\in I} p_i \vee \bigvee_{j\in J} \neg q_j$. Then, $f(\phi) = \Diamond^{\forall}\Box^{\forall}\widetilde{\psi}$. By definition of $\widetilde{\psi}$, it easily follows that:

(*) for every local state (i, y) of π with $y \in \ell_i$, $(i, y) \models_\pi \widetilde{\psi}$ iff $w^i \models \psi$.

We first prove that "$w \models \overset{\infty}{\Box}\psi \implies \pi \models \Diamond^{\forall}\Box^{\forall}\widetilde{\psi}$". Assume that $w \models \overset{\infty}{\Box}\psi$. This means that there is $N \geq 0$ such that for every $i \geq N$, $w^i \models \psi$. We prove that $\pi \models \Diamond^{\forall}\Box^{\forall}\widetilde{\psi}$. We need to show that for every linear path r of π starting from $(0, \varepsilon)$, there is a local state (i, y) along r such that for every local state (j, z) reachable from (i, y) by a linear path, $(j, z) \models_\pi \widetilde{\psi}$ holds. If r is a finite linear path, then by definition of $\widetilde{\psi}$, the above property trivially holds. Thus, let r be infinite. Then, there is a local state (i, y) along r with $i \geq N$. Let (j, z) be a local state which is reachable from (i, y) by a linear path. If $next^{\ell}_{\pi}(j, z) = \emptyset$, then $(j, z) \models_\pi \widetilde{\psi}$. Otherwise, there is a local state (h, z) such that $next^{\ell}_{\pi}(h, z) = next^{\ell}_{\pi}(j, z)$, $h \geq N$ (note that $j \geq N$), and $z \in \ell_h$. Since $w^h \models \psi$, by Property (*), it follows that $(h, z) \models_\pi \widetilde{\psi}$. Hence $(j, z) \models_\pi \widetilde{\psi}$, and therefore, $\pi \models \Diamond^{\forall}\Box^{\forall}\widetilde{\psi}$.

Consider, now, the converse assertion "$\pi \models \Diamond^{\forall}\Box^{\forall}\widetilde{\psi} \implies w \models \overset{\infty}{\Box}\psi$". Assume that $\pi \models \Diamond^{\forall}\Box^{\forall}\widetilde{\psi}$. Let (T, D) with $D : T \to Q_\pi$ be the infinite Q_π-labeled tree inductively defined as follows: the root ε of T is labeled by the local state $(0, \varepsilon)$ and for every location $y \in T$, the children of y correspond to the linear successors of $D(y)$ on π. Note that every maximal path in the tree T starting from the root corresponds to a linear path of π starting from $(0, \varepsilon)$ and vice versa. Let $T' \subseteq T$ be the subset of T obtained by pruning all the locations x of T such that $D(x) \models_\pi \Box^{\forall}\widetilde{\psi}$. Clearly, T' is a tree. We show by contradiction that T' is finite. Suppose that T' is infinite. Since the set of children of every location is finite, by König's Lemma, we deduce that there is an infinite linear path r starting from $(0, \varepsilon)$ such that for every $i \geq 0$, $r(i) \not\models \Box^{\forall}\widetilde{\psi}$. But this contradicts the assumption $\pi \models \Diamond^{\forall}\Box^{\forall}\widetilde{\psi}$. Therefore, T' is finite. Therefore, there is $N \geq 0$ such that for all $i \geq N$ and $y \in \ell_i$, $(i, y) \models \Box^{\forall}\widetilde{\psi}$, hence $(i, y) \models \widetilde{\psi}$. By Property (*) it follows that for every $i \geq N$, $w^i \models \psi$. Thus, we obtain that $w \models \overset{\infty}{\Box}\psi$. This concludes the proof of Claim 1. □

We have shown that the theorem holds for the class of synchronization-free *CRSM*. Now, let us consider an arbitrary *CRSM* $\mathcal{S}$ with transition function δ. Without loss of generality we can assume that for all vertices v and v' and symbols σ and σ', if $v' \in \delta(v, \sigma)$ and $v' \in \delta(v, \sigma')$, then $\sigma = \sigma'$. Let $\mathcal{S}'$ be the *CRSM* obtained from $\mathcal{S}$ by extending the labeling of the vertices as follows. For every vertex v of $\mathcal{S}$, we add to the label $\eta(v)$ of v all the synchronization symbols σ of the module of v, a fresh proposition p_v, and for every vertex v' such that $v \in \delta(v', \sigma)$ and σ is a synchronization symbol, a fresh proposition $p_{(v',\sigma)}$. Now, let us consider the mapping $f_{\mathcal{S}}$ defined as follows:

- $f_{\mathcal{S}}$ is homomorphic with respect to the boolean connectives;
- for $\psi = \bigvee_{i\in I} p_i \vee \bigvee_{j\in J} \neg q_j$, $f_{\mathcal{S}}(\Box\psi) = \Box^{\forall}\widetilde{\psi}$ and $f_{\mathcal{S}}(\overset{\infty}{\Box}\psi) = \Diamond^{\forall}\Box^{\forall}\widetilde{\psi}$, where
  $\widetilde{\psi} = \bigcirc^{\exists} tt \rightarrow \Big(\big[(call \vee \bigcirc^{\exists} ret) \rightarrow \big((\bigvee_{i\in I} \bigcirc^{\exists} p_i) \vee (\bigvee_{j\in J} \bigcirc^{\forall} \neg q_j)\big)\big] \wedge$
  $\big[\neg(call \vee \bigcirc^{\exists} ret) \rightarrow \bigvee_{v\in V}\big(p_v \wedge (\bigcirc^{\exists}(\psi \wedge \bigwedge_{\sigma} \neg p_{(v,\sigma)}) \vee \rho)\big)\big]\Big)$
  and ρ is the formula:
  $\bigvee_{\sigma} \bigcirc^{\exists}\Big(p_{(v,\sigma)} \wedge \big(\bigvee_{i\in I}((\neg \| \neg(\sigma \wedge p_i)) \vee p_i) \vee \bigvee_{j\in J}(\|(\sigma \rightarrow \neg q_j) \wedge \neg q_j\big)\Big)$.

Proceeding as in the proof of Claim 1, we deduce that for every formula ϕ of Υ, $\mathcal{S}$ satisfies ϕ iff $\mathcal{S}'$ satisfies $f_{\mathcal{S}}(\phi)$. This concludes the proof of Theorem 2. □

5. Model checking *CRSM* against ConCaRet

In this section we show that the model-checking problem of *CRSM* against ConCaRet is decidable. Our approach is a non-trivial generalization of the automata-theoretic approach proposed in [3] to solve model checking of *RSM* against CaRet specifications.

First, we equip *CRSM* with acceptance conditions over the runs. An acceptance condition is composed of three parts. The first one corresponds to a standard generalized Büchi condition on the set of vertices visited infinitely often by the infinite linear paths of an accepting run. The second part forces the accepting runs to reach some given vertices on terminal local states (i.e., the local states whose set of linear successors is empty). The third one expresses constraints over the vertices which are visited by module instances running in parallel. We call the obtained model *Büchi CRSM*.

Then, we reduce the model-checking problem we are interested in to the emptiness problem of Büchi *CRSM*. More precisely, given a *CRSM* $\mathcal{S}$ and a ConCaRet formula φ, we construct a Büchi *CRSM* $\mathcal{S}_{\neg\varphi}$ such that $\mathcal{S}$ satisfies φ if and only if $\mathcal{S}_{\neg\varphi}$ has *no* accepting run starting from a start node.

The rest of this section is organized as follows. In Section 5.1 we introduce the notion of Büchi *CRSM* and solve the (non)emptiness problem for this class of infinite-state machines. Then, in Section 5.2 we reduce the model-checking problem to the emptiness of Büchi *CRSM*.

5.1. *Büchi* CRSM

In this section, we extend *CRSM* with acceptance conditions and address the nonemptiness problem for the resulting class of machines, i.e., the problem about the existence of an *accepting* run from a *start* node. Besides acceptance conditions on the finite linear paths of a run π, we require a synchronized acceptance condition on modules running in parallel, and a generalized Büchi acceptance condition on the infinite linear paths of π. We call this model a Büchi *CRSM* (*B-CRSM*, for short). Formally, a *B-CRSM* $\mathcal{S} = \langle (S_1, \ldots, S_k), start, \mathcal{F}, \mathcal{P}_f, \mathcal{P}_{sync}\rangle$ consists of a *CRSM* $\langle (S_1, \ldots, S_k), start\rangle$ together with the following acceptance conditions:

- $\mathcal{F} = \{F_1, \ldots, F_n\}$ is a family of accepting sets of vertices of $\mathcal{S}$;
- $\mathcal{P}_F$ is the set of *terminal* vertices;
- $\mathcal{P}_{sync}$ is a predicate defined over pairs (v, H) such that v is a vertex and H is a set of vertices such that $|H| \leq rank(\mathcal{S})$.

Let $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$ be a run of $\mathcal{S}$ with $q_i = (t_i, D_i)$ for $i \geq 0$. For each $i \geq 0$ and $y \in t_i$, we denote by $v(i, y)$ the vertex of $\mathcal{S}$ defined as follows: if $(i, y) \in Q_\pi$ (i.e., (i, y) is a local state), then $v(i, y) = D_i(y)$; otherwise, $v(i, y) = D_h(y)$ where $(h, y) = call_\pi(i, y)$. We say that the run π is *accepting* iff the following three conditions are satisfied:

1. for every infinite linear path $r = (i_0, y_0)(i_1, y_1) \ldots$ of π and $F \in \mathcal{F}$, there are infinitely many $h \in \mathbb{N}$ such that $D_{i_h}(y_h) \in F$ (*generalized Büchi acceptance condition*);
2. for every *terminal* local state $(i, y) \in Q_\pi$, i.e. such that $next^{\ell}_{\pi}(i, y) = \emptyset$, condition $D_i(y) \in \mathcal{P}_F$ holds (*terminal acceptance condition*);
3. for all $i \geq 0$ and $y \in \ell'_i$, $\mathcal{P}_{sync}(v(i+1, y), \{v(i+1, y_1), \ldots, v(i+1, y_{m_i})\})$ holds, where $\{y_1, \ldots, y_{m_i}\}$ is $siblings(t_{i+1}, y) \setminus \{y\}$ (*synchronized acceptance condition*).

We say that the run π is *monotone* iff for all $i \geq 0$, q_{i+1} is obtained from q_i either by a module call or by an internal move. Note that in a monotone path either the tree t_{i+1} is equal to t_i (for an internal move) or it is obtained from t_i by adding some children to a leaf (for a module call).

As we will see in Section 5.2, for a *CRSM* $\mathcal{S}$ and a ConCaRet formula φ, it is possible to construct a *B-CRSM* $\mathcal{S}_\varphi$ such that $\mathcal{S}_\varphi$ has an accepting run iff $\mathcal{S}$ has a run that satisfies φ. More precisely, an accepting run of $\mathcal{S}_\varphi$ corresponds to a run π of $\mathcal{S}$, where each local state is annotated with the information concerning the set of subformulas of φ that hold at it along π. Intuitively, as in standard *LTL*, the generalized Büchi acceptance condition is used to guarantee the fulfillment of liveness requirements in until subformulas of φ. The terminal acceptance condition is instead used to ensure that forward next formulas cannot be asserted at terminal local states. Finally, the synchronized acceptance condition is used to check the fulfillment of parallel subformulas $\|\psi$ of φ.

In the rest of this section, we give an algorithm to decide the *nonemptiness problem* of *B-CRSM*, i.e. to decide given a *B-CRSM* $\mathcal{S}$ whether $\mathcal{S}$ has an accepting run starting from a node in *start*. We solve this problem in two main steps:

1. We give an algorithm to decide the problem about the existence of accepting *monotone* runs starting from a given vertex.
2. We reduce nonemptiness of *B-CRSM* to the problem addressed in Step 1.

5.1.1. Deciding the existence of accepting monotone runs

We decide the existence of accepting *monotone* runs in *B-CRSM* by a reduction to the emptiness problem for *Invariant Büchi tree automata*. These differ from the standard formalism of Büchi tree automata [33] for a partitioning of states into *invariant* and *non-invariant* states, with the constraint that transitions from non-invariant to invariant states are forbidden. Also, the standard Büchi acceptance condition is strengthened by requiring in addition that an accepting run must have a path consisting of invariant states only.

Formally, an (alphabet free) *invariant Büchi tree automaton* is a tuple $\mathbb{U} = \langle \mathcal{D}, P, P_0, M, Inv, Acc \rangle$, where $\mathcal{D} \subset \mathbb{N} \setminus \{0\}$ is a finite set of branching degrees, P is the finite set of states, $P_0 \subseteq P$ is the set of initial states, $M : P \times \mathcal{D} \to 2^{P^*}$ is the transition function with $M(s,d) \in 2^{P^d}$ for all $(s,d) \in P \times \mathcal{D}$, $Inv \subseteq P$ is the invariance condition, and $Acc \subseteq P$ is the Büchi condition. Also, for every $s \in P \setminus Inv$ and $d \in \mathcal{D}$, we require that if s' occurs in $M(s,d)$, then $s' \in P \setminus Inv$. A *complete* $\mathcal{D}$-tree is an infinite tree $t \subseteq \mathbb{N}^*$ such that for every $y \in t$, the cardinality of $children(t,y)$ belongs to $\mathcal{D}$. A *path* of t is a maximal subset of t linearly ordered by $\prec$. A run of $\mathbb{U}$ is a pair (t,r) where t is a complete $\mathcal{D}$-tree and $r : t \to P$ is a P-labeling of t such that $r(\varepsilon) \in P_0$ and for all $y \in t$, $(r(y \cdot 0), r(y \cdot 1), \ldots, r(y \cdot d)) \in M(r(y), d+1)$, where $d+1 = |children(t,y)|$. The run (t,r) is *accepting* iff: (1) there is a path ν of t such that for every $y \in \nu$, $r(y) \in Inv$, and (2) for every path ν of t, the set $\{y \in \nu \mid r(y) \in Acc\}$ is infinite. The algorithm in [35] for checking emptiness in Büchi tree automata can be easily generalized to handle also the invariance condition, thus we obtain the following.

Proposition 1. *The emptiness problem for invariant Büchi tree automata is logspace-complete for* Ptime *and can be decided in quadratic time.*

In the following, we fix a *B-CRSM* $\mathcal{S} = \langle (S_1, \ldots, S_k), start, \mathcal{F}, \mathcal{P}_F, \mathcal{P}_{sync} \rangle$.

Remark 1. Apart from a preliminary step which is computable in linear time (in the size of $\mathcal{S}$), we can restrict ourselves to consider only accepting monotone runs π of S starting from call vertices. In fact, if π starts at a non-call vertex v of a module S_h, then either π stays within S_h forever, or π enters a call v' of S_h that never returns. In the first case, one has to check the existence of an accepting run in the *generalized* Büchi (word) automaton given by $A_h = \langle V_h, \delta_h, \mathcal{F}_h \rangle$, where $\mathcal{F}_h$ is the restriction of $\mathcal{F}$ to the set V_h. This can be done in linear time [33]. In the second case, one has to check that there is a call v' reachable from v in A_h, and then that there is an accepting monotone run in $\mathcal{S}$ from v'.

Now, we construct an invariant Büchi tree automaton $\mathbb{U}$ capturing the monotone accepting runs of $\mathcal{S}$ starting from calls. The idea is to model a monotone run π of $\mathcal{S}$ as an infinite tree corresponding to a run of $\mathbb{U}$. There are some technical issues to be handled.

First, there can be finite linear paths. We use symbol $\top$ to capture terminal local states of π. Therefore, the subtree rooted at the node corresponding to a terminal local state is completely labeled by $\top$. Also, since we are interested in runs of $\mathcal{S}$, we need to check that there is at least one infinite linear path in π. We do this using as invariant set the set of all $\mathbb{U}$ states except for state $\top$.

Second, when a module call associated with a box b occurs, multiple module instances $I_1, \ldots, I_m$ are activated and start running. We encode these local runs (corresponding to linear paths of π) on the same path of the run of $\mathbb{U}$ by using states of the form $(b, v_1, \ldots, v_m, i_1, \ldots, i_m, j)$, where $v_1, \ldots, v_m$ are the current nodes or calls of each module, and $i_1, \ldots, i_m, j$ are finite counters used to check the fulfillment of the Büchi condition (see below). Since in monotone runs there are no returns from calls, when a module I_j (with $1 \leq j \leq m$) moves to a call vertex v (by an internal move), we can separate the linear paths starting at v from the linear paths associated with all modules $I_i \neq I_j$, $i = 1, \ldots, m$. Therefore, to simulate an internal move (in the context of modules $I_1, \ldots, I_m$), $\mathbb{U}$ nondeterministically splits in $d+1$ copies for some $0 \leq d \leq m$ such that d copies correspond to those modules (among $I_1, \ldots, I_m$) which move to call vertices, and the $(d+1)$th copy goes to a state s of the form $(b, v'_1, \ldots, v'_m, i'_1, \ldots, i'_m, j')$ which describes the new status of modules $I_1, \ldots, I_m$. Note that in s, we maintain the information of these modules which are busy in a parallel call. This is necessary for locally checking the fulfillment of the synchronized acceptance condition $\mathcal{P}_{sync}$.

Last, we need to check the fulfillment of the acceptance conditions of $\mathcal{S}$. The generalized Büchi condition $\mathcal{F}$ of $\mathcal{S}$ is captured with the Büchi condition of $\mathbb{U}$ along with the use of finite counters implemented in the states. For the ease of presentation, we assume that $\mathcal{F}$ consists of a single accepting set F. For modeling a call v, we use states of the form (v, i) where the counter i is used to check that linear paths (in the simulated monotone run of $\mathcal{S}$) containing infinite occurrences of calls satisfy the Büchi condition. In particular, such counter has default value 0 and is set to 1 if either $v \in F$ or a vertex in F is visited in the portion of the linear path from the last call before entering v. In the second case, the needed information is kept in the counters i_h of the states of the form $(b, v_1, \ldots, v_m, i_1, \ldots, i_m, j)$. Counter i_h has default value 0 and is set to 1 if a node in F is entered in the local computation of the corresponding module. Counter $j \in \{0, \ldots, m\}$ is instead used to check that the Büchi condition F of $\mathcal{S}$ is satisfied by the linear paths starting from those vertices among $v_1, \ldots, v_m$ whose module instances never enter a call. Essentially, the counter j is incremented (reset to 0 if its value is m) unless $j < m$ and the $(j+1)$th module refining box b moves by the current internal move of $\mathcal{S}$ from vertex v_{j+1} (which is a node) to a vertex which is *not* in F. Thus, the counter j reaches the value m infinitely often if and only if the linear paths in the simulated run of $\mathcal{S}$, associated with the module instances refining box b that never enter a call, are accepting. Moreover, in order to check that a node v_h corresponds to a terminal node (i.e., a node without linear successors in the simulated monotone run of $\mathcal{S}$), $\mathbb{U}$ can choose nondeterministically to set the corresponding counter i_h to -1. Consistently, $\mathbb{U}$ will simulate only the internal moves from vertices $v_1, \ldots, v_m$ in which the module instance associated with v_h does not evolve. Formally, the invariant Büchi tree automaton $\mathbb{U} = \langle \mathcal{D}, P, P_0, M, Inv, Acc \rangle$ is defined as follows:

- $\mathcal{D} = \{1, \ldots, rank(\mathcal{S}) + 1\}$;
- $P = \{\top\} \cup Calls \times \{0, 1\} \cup \bigcup_{b \in B} P_b$, where for $b \in B$ with $Y(b) = i_1 \ldots i_m$, $P_b = \{(b, v_1, \ldots, v_m, h_1, \ldots, h_m, h) \mid h \in \{0, 1, \ldots, m\}$ and for all $1 \leq j \leq m$, either $v_j \in N_{i_j}$ and $h_j \in \{0, 1, -1\}$, or $v_j \in Calls_{i_j}$ and $h_j = 1\}$;
- $P_0 = Calls \times \{0\}$; $\quad Inv = P \setminus \{\top\}$;
- $Acc = \{\top\} \cup \{(v, 1) \mid v \in Calls\} \cup \bigcup_{b \in B} \{(b, v_1, \ldots, v_m, j_1, \ldots, j_m, m) \in P_b\}$;

It remains to define the transition function M: $(s_1, \ldots, s_d) \in M(s, d)$ iff one of the following holds.

Case $s = \top$**:** $d = 1$ and $s_1 = \top$.

Case $s = ((b, e_1, \ldots, e_m), i) \in Calls \times \{0, 1\}$**:** $d = 1$, and either (i) $s_1 = \top$ and $(b, e_1, \ldots, e_m) \in \mathcal{P}_f$, or (ii) $s_1 = (b, e_1, \ldots, e_m, h_1, \ldots, h_m, 0)$, and for all $1 \leq j \leq m$, *or* $h_j = -1$ and $e_j \in \mathcal{P}_f$, *or* $h_j = 1$ and $e_j \in F$, *or* $h_j = 0$ and $e_j \notin F$; moreover, $\mathcal{P}_{sync}(e_j, \{e_1, \ldots, e_m\} \setminus \{e_j\})$ holds.

Case $s = (b, v_1, \ldots, v_m, i_1, \ldots, i_m, h) \in P_b$**:** let $C = \{j \in \{1, \ldots, m\} \mid$ either $v_j \in Calls$ or $i_j = -1\}$. Then, one of the following holds:

- **Deadlock situation:** $C = \{1, \ldots, m\}$ and $\{v_1, \ldots, v_m\} \not\subseteq Ex$. In this case, $d = 1$ and $s_1 = \top$.
- **Internal move and activation of parallel calls:** $C \subset \{1, \ldots, m\}$. In this case there are indexes $k_1, \ldots, k_p \in \{1, \ldots, m\} \setminus C$ (with $k_1 < \cdots < k_p$), $\sigma \in \Sigma$, and vertices $v'_1, \ldots, v'_m$ such that $p = 1$ if $\sigma \notin \Sigma^s$, and for each $1 \leq j \leq m$, the following holds: if $j \notin \{k_1, \ldots, k_p\}$, then $v'_j = v_j$ and σ is not a synchronization symbol of the module associated with v_j; if $j \in \{k_1, \ldots, k_p\}$, then $v'_j \in \delta(v_j, \sigma)$ and $P_{sync}(v'_j, \{v'_1, \ldots, v'_m\} \setminus \{v'_j\})$. Moreover, denoted by $h_1, \ldots, h_t$ (with $h_1 < \cdots < h_t$) the subsequence (possibly empty) of $k_1, \ldots, k_p$ constituted from all and only the indexes k_j such that $v'_{k_j} \in Calls$, we have that $d = t + 1$ and the following holds:
  - **activation of new parallel calls:** for all $1 \leq j \leq t$, $s_j = (v'_{h_j}, j')$, where $j' = 1$ if either $v'_{h_j} \in F$ or $i_{h_j} = 1$, and $j' = 0$ otherwise.
  - **internal move:** $s_{t+1} = (b, v'_1, \ldots, v'_m, i'_1, \ldots, i'_m, h')$ such that for each $j \in \{1, \ldots, m\} \setminus \{k_1, \ldots, k_p\}$, $i_j = i'_j$, and for each $1 \leq j \leq p$, the following holds: if $i'_{k_j} = -1$, then $v'_{k_j} \in \mathcal{P}_f$; if $i'_{k_j} \neq -1$, then $i'_{k_j} = 1$ if or $v'_{k_j} \in Calls$ or $v'_{k_j} \in F$ or $i_{k_j} = 1$, and $i'_{k_j} = 0$, otherwise. Moreover, $h' = h$ if $h < m$, $i'_{h+1} \neq -1$, $v'_{h+1} \notin Calls$, and $v'_{h+1} \notin F$ if $h+1 \in \{k_1 \ldots, k_p\}$; otherwise, $h' = (h + 1) \bmod (m + 1)$.

By construction we obtain the following result.

Lemma 1. *For a call v, there is an accepting monotone run of $\mathcal{S}$ from $\langle v \rangle$ iff there is an accepting run in $\mathbb{U}$ starting from $(v, 0)$.*

When the Büchi condition consists of $n > 1$ accepting sets, the only changes in the above construction concern the counters: we need to check that each set is met and thus the counters which count up to 1 become counters up to n and the remaining counter (which is up to m) becomes a counter up to $m \cdot n$. Therefore, denoting $\rho = rank(\mathcal{S})$, n_V the number of vertices of $\mathcal{S}$ and n_δ the number of transitions of $\mathcal{S}$, we have that the number of $\mathbb{U}$ states is $O(\rho \cdot n^{\rho+1} \cdot n_V^{\rho+1})$ and the number of $\mathbb{U}$ transitions is $O(\rho^2 \cdot n^{2\rho+2} \cdot n_V \cdot (n_V + n_\delta)^\rho)$. Thus, by Proposition 1, Remark 1, Lemma 1, and the above observation we obtain the following result.

Lemma 2. *The problem of checking the existence of accepting monotone runs in a* B-CRSM *$\mathcal{S}$ can be decided in time $O(\rho^4 \cdot n^{4\rho+4} \cdot (n_V + n_\delta)^{2\rho+2})$.*

5.1.2. The nonemptiness problem for Büchi CRSM

In this part of Section 5.1, we show that the nonemptiness problem for Büchi *CRSM* can be reduced to check the existence of accepting monotone runs. Fix a *B-CRSM* $\mathcal{S} = \langle (S_1, \ldots, S_k), start, \mathcal{F}, \mathcal{P}_F, \mathcal{P}_{sync} \rangle$. First, we informally describe the reduction, and then we give the technical details. Our approach is a generalization of that used in [1,3] to reduce nonemptiness of generalized Büchi *RSM* to nonemptiness of ordinary generalized Büchi finite-state automata.

As in [1], the reduction algorithm proceeds in two phases. In the first phase, for each module S_i of $\mathcal{S}$, we compute for every call $v = (b, e_1, \ldots, e_n)$ and matching return $v' = (b, x_1, \ldots, x_n)$, whether there is a path in the global labeled transition system $K_{\mathcal{S}}$ starting from $\langle v \rangle$ and leading to $\langle v' \rangle$, and if so whether for each Büchi component $F_i \in \mathcal{F}$, there is an F_i-*accepting path* from $\langle v \rangle$ to $\langle v' \rangle$, i.e. a path from $\langle v \rangle$ to $\langle v' \rangle$ such that every linear path starting from the unique local state associated with $\langle v \rangle$ visits some vertex in F_i (*generalized reachability problem*). This involves to solve reachability in AND-OR graphs and takes time singly-exponential in $rank(\mathcal{S})$. Then, in the second phase, we keep track of the information computed in the first phase by augmenting the *B-CRSM* $\mathcal{S}$ with "summary edges" to indicate reachability from calls v to matching returns v', and if it is possible to get from $\langle v \rangle$ to $\langle v' \rangle$ by an F_i-accepting path for any component $F_i \in \mathcal{F}$. Thus, we obtain a *B-CRSM* $\mathcal{S}'$ such that for each node u of $\mathcal{S}$, $\mathcal{S}$ has an accepting run starting from $\langle u \rangle$ iff $\mathcal{S}'$ has an accepting *monotone* run starting from $\langle u \rangle$. Now, we describe in detail the reduction algorithm.

Generalized reachability problem. For $F \subseteq V$, a *finite* path of $K_{\mathcal{S}}$, $\pi = q_0 \xrightarrow{(\ell_0, \ell'_0)} q_1 \xrightarrow{(\ell_0, \ell'_0)} \ldots q_n$ (with $q_i = (t_i, D_i)$ for all $0 \leq i \leq n$, and $t_0 = \{\varepsilon\}$) is F-*accepting* iff π satisfies the synchronized acceptance condition $\mathcal{P}_{sync}$ and all the linear paths of π starting from the local state $(0, \varepsilon)$ contain occurrences of local states (i, z) such that $D_i(z) \in F$. For a box $b \in B$, we say that π is a b-*path* if $D_i(\varepsilon) = b$ for all $1 \leq i \leq n-1$.

- The *generalized reachability problem* can be formulated as follows: given a call $v = (b, e_1, \ldots, e_m)$ and a matching return $v' = (b, x_1, \ldots, x_m)$, is there an F-accepting b-path starting from $\langle v \rangle$ and leading to $\langle v' \rangle$?

For a box b of $\mathcal{S}$ with $Y(b) = i_1 \ldots i_m$, the synchronous product of the sequence of machines that refine b is $A_b = \langle V_b, V_b^0, \Delta_b \rangle$ where $V_b = V_{i_1} \times \cdots \times V_{i_m}$, $V_b^0 = \{(e_1, \ldots, e_m) \in En_{i_1} \times \cdots \times En_{i_m} \mid \text{for all } 1 \leq j \leq m, \mathcal{P}_{sync}(e_j, \{e_1, \ldots, e_m\} \setminus \{e_j\})$ holds$\}$, and $\Delta_b \subseteq V_b \times V_b$ is such that $((u_1, \ldots, u_m), (v_1, \ldots, v_m)) \in \Delta_b$ *iff* there are $\sigma \in \Sigma$ and distinct indexes $k_1, \ldots, k_p \in \{1, \ldots, m\}$ such that $p = 1$ if $\sigma \notin \Sigma^s$ and for each $1 \leq j \leq m$, the following holds: if $j \notin \{k_1, \ldots, k_p\}$, then $v_j = u_j$ and σ is not a synchronization symbol of the module associated with u_j; if $j \in \{k_1, \ldots, k_p\}$, then $v_j \in \delta(u_j, \sigma)$ and $P_{sync}(v_j, \{v_1, \ldots, v_m\} \setminus \{v_j\})$ holds.

For $v = (v_0, \ldots, v_m) \in V_b$, we write $\langle v \rangle_b$ for the (global) state of $\mathcal{S}$ given by (t, D) where $t = \{\varepsilon, 0, \ldots, m\}$, $D(\varepsilon) = b$, and $D(j) = v_j$ for all $0 \leq j \leq m$.

To solve the generalized reachability problem for a given set $F \subseteq V$, we inductively define for each box $b \in B$, a set $R_b^F \subseteq V_b^0 \times V_b \times \{0, 1\}^m$ where $m = |Y(b)|$. The intended meaning is: for $u = (e_1, \ldots, e_m) \in V_b^0$ and $v = (v_1, \ldots, v_m) \in V_b$, $(u, v, acc_1, \ldots, acc_m) \in R_b^F$ iff there is a b-path π from $\langle u \rangle_b$ to $\langle v \rangle_b$ such that π satisfies the synchronized acceptance condition $\mathcal{P}_{sync}$ and for each $1 \leq j \leq m$ with $acc_j = 1$, all the linear paths of π starting from the local state $(0, j-1)$ (corresponding to the entry e_j) visit some vertex in F (note that each of these linear paths terminates in a local state labeled by v_j).

The set R_b^F is defined by the following rules:

1. $(x, x, acc_1, \ldots, acc_m) \in R_b^F$ *if*
   $x = (e_1, \ldots, e_m) \in V_b^0$ and for $1 \leq j \leq m$, $acc_j = 1$ iff $e_j \in F$;
2. $(x, u, acc'_1, \ldots, acc'_m) \in R_b^F$ *if*
   $(x, w, acc_1, \ldots, acc_m) \in R_b^F$, $u = (u_1, \ldots, u_m) \in V_b$, $w \in V_b$, $(w, u) \in \Delta_b$, and for $1 \leq j \leq m$: $acc'_j = 1$ if either $acc_j = 1$ or $u_j \in F$, and $acc'_j = 0$ otherwise.
3. $(x, (u_1, \ldots, u_m), acc'_1, \ldots, acc'_m) \in R_b^F$ *if*
   $(x, (w_1, \ldots, w_m), acc_1, \ldots, acc_m) \in R_b^F$ and there is $1 \leq j \leq m$ such that $w_j = (b_j, e_1^j, \ldots, e_{m_j}^j) \in Calls$, $u_j = (b_j, x_1^j, \ldots, x_{m_j}^j) \in Retns$, for each $h \neq j$, $w_h = u_h$ and $acc'_h = acc_h$, and $P_{sync}(u_j, \{u_1, \ldots, u_m\} \setminus \{u_j\})$, $((e_1^j, \ldots, e_{m_j}^j), (x_1^j, \ldots, x_{m_j}^j), acc_1^j, \ldots, acc_{m_j}^j) \in R_{b_j}^F$, and

$$acc'_j = \begin{cases} 1 & \text{if } \textit{either } acc_j = 1 \textit{ or } u_j \in F \textit{ or } acc_h^j = 1 \text{ for all } 1 \leq h \leq m_j \\ 0 & \text{otherwise.} \end{cases}$$

For $x = (e_1, \ldots, e_m) \in V_b^0$, Rule 1 says that $\langle x \rangle_b$ can reach itself by a null path π and clearly, for all $1 \leq j \leq m$, all linear paths of π from the local state $(0, j-1)$ of π visit F iff $e_j \in F$. Rule 2 manages internal moves. If there is a b-path π from $\langle x \rangle_b$ to $\langle w \rangle_b$ satisfying $\mathcal{P}_{sync}$ and w can reach $u = (u_1, \ldots, u_m) \in V_b$ by an internal move (which is consistent with $\mathcal{P}_{sync}$), then there is a b-path π' from $\langle x \rangle_b$ to $\langle u \rangle_b$ satisfying $\mathcal{P}_{sync}$ and, for all $1 \leq j \leq m$, all linear paths of π' starting from the local state $(0, j-1)$ visit F iff either the same holds for π or $u_j \in F$. Finally, Rule 3 handles module calls. If there is a b-path π from $\langle x \rangle_b$ to $\langle (w_1, \ldots, w_m) \rangle_b$ satisfying $\mathcal{P}_{sync}$ and there is $1 \leq j \leq m$ such that $w_j = (b_j, e_1^j, \ldots, e_{m_j}^j) \in Calls$, then there is a b-path π' from $\langle x \rangle_b$ to $\langle (u_1, \ldots, u_m) \rangle_b$ satisfying $\mathcal{P}_{sync}$ provided that $u_h = w_h$ for $h \neq j$, $u_j = (b_j, x_1^j, \ldots, x_{m_j}^j)$ is a return

that matches the call w_j, the synchronized acceptance condition $\mathcal{P}_{sync}(u_j, \{u_1, \ldots, u_m\} \setminus \{u_j\})$ holds, and there is a b_j-path π_j from $\langle (e^j_1, \ldots, e^j_{m_j}) \rangle_{b_j}$ to $\langle (x^j_1, \ldots, x^j_{m_j}) \rangle_{b_j}$ satisfying $\mathcal{P}_{sync}$. Clearly, all linear paths of π' starting from $(0, j-1)$ visit F iff either the same holds for π or $u_j \in F$ or π_j is an F-accepting path. Thus, by construction we obtain the following result for a given call $v = (b, e_1, \ldots, e_m)$ and matching return $v' = (b, x_1, \ldots, x_m)$.

Lemma 3. *There is a b-path π from $\langle v \rangle$ to $\langle v' \rangle$ satisfying $\mathcal{P}_{sync}$ iff $(e_1, \ldots, e_m) \in V_b^0$ and $((e_1, \ldots, e_m), (x_1, \ldots, x_m), acc_1, \ldots, acc_m) \in R_b^F$ for some $acc_1, \ldots, acc_m$. Also, π is F-accepting iff either $\{v, v'\} \cap F \neq \emptyset$ or $acc_j = 1$ for all $1 \leq j \leq m$.*

Each set R_b^F can be constructed effectively (by a standard iterative least fixed point construction) using the above rules. The construction can be optimized computing such sets via reachability analysis over an AND-OR graph (as in [1]). Denoting $\rho = rank(\mathcal{S})$, with n_V the number of vertices and with n_δ the number of transitions of the *B-CRSM* $\mathcal{S}$, such a graph has $O(2^\rho \cdot n_V^{\rho+2})$ vertices and $O(n_V \cdot 2^\rho \cdot (n_\delta^\rho + \rho \cdot 2^\rho \cdot n_V^{\rho+1}))$ edges. Thus, we obtain the following result.

Lemma 4 (*Generalized Reachability Problem*). *Given $F \subseteq V$, the set of pairs (v, v') such that $v = (b, e_1, \ldots, e_m)$ is a call, $v' = (b, x_1, \ldots, x_m)$ is a matching return, and there is an F-accepting b-path from $\langle v \rangle$ to $\langle v' \rangle$, is* computable *in time $O(n_V^2 \cdot 4^\rho \cdot (n_V + n_\delta)^\rho)$.*

Now, we show how to solve the nonemptiness problem for *B-CRSM* using the results stated by Lemmata 2 and 4. Starting from the *B-CRSM* $\mathcal{S}$ with $\mathcal{F} = \{F_1, \ldots, F_n\}$, we construct a new *B-CRSM* $\mathcal{S}'$ such that nonemptiness for $\mathcal{S}$ is reduced to check the existence of accepting *monotone* runs in $\mathcal{S}'$.
$\mathcal{S}' = \langle (S'_1, \ldots, S'_k), start, \mathcal{F}', \mathcal{P}'_f, \mathcal{P}'_{sync} \rangle$, with $\mathcal{F}' = \{F'_1, \ldots, F'_n\}$, is defined as follows. For $1 \leq i \leq k$, S'_i is obtained extending the set of nodes and the transition function of S_i as follows. For any call $v = (b, e_1, \ldots, e_m)$ of S_i and matching return $v' = (b, x_1, \ldots, x_m)$ such that there is a V-accepting b-path in $\mathcal{S}$ from $\langle v \rangle$ to $\langle v' \rangle$, we add two new nodes u^c_{new} and u^r_{new}, and the edge $(u^c_{new}, \perp, u^r_{new})$, where $\perp$ is a fresh non-synchronization symbol. We say that u^c_{new} (resp., u^r_{new}) is *associated* with the call v (resp., return v'). Moreover, for every edge in S_i of the form (u, σ, v) (resp., of the form (v', σ, u)) we add in S'_i the edge (u, σ, u^c_{new}) (resp., the edge (u^r_{new}, σ, u)). Also, for $1 \leq i \leq n$, if there is an F_i-accepting b-path from $\langle v \rangle$ to $\langle v' \rangle$, then we add u^r_{new} to F'_i (F'_i also contains all elements of F_i). Still, if $v' \in \mathcal{P}_f$, then we add u^r_{new} to $\mathcal{P}'_f$ ($\mathcal{P}'_f$ also contains all elements of $\mathcal{P}_f$). Note that $u^c_{new} \notin \mathcal{P}_f$. In fact, if an accepting run of $\mathcal{S}'$ has a local state labeled by u^c_{new}, then the linear successor of this local state is defined and is labeled by u^r_{new}. Finally, $\mathcal{P}'_{sync}(v'_0, \{v'_1, \ldots, v'_m\})$ (with $m \leq rank(\mathcal{S})$) holds iff there are $v_0, \ldots, v_m \in V$ such that $\mathcal{P}_{sync}(v_0, \{v_1, \ldots, v_m\})$ holds and for all $0 \leq j \leq m$, either $v'_j = v_j$, or v_j is a return (resp., a call) and v'_j is a "new" node associated with it. Thus, we obtain the following result.

Lemma 5. *For every node u of $\mathcal{S}$, there is an accepting run in $\mathcal{S}$ from $\langle u \rangle$ iff there is an accepting* monotone *run in $\mathcal{S}'$ from $\langle u \rangle$.*

Note that the number of new nodes is bounded by $2n_V^2$, the number of new edges is bounded by $n_V \cdot n_\delta + n_V^2$, and by Lemma 4, $\mathcal{S}'$ can be constructed in time $O(|\mathcal{F}| \cdot n_V^2 \cdot 4^\rho \cdot (n_V + n_\delta)^\rho)$. Thus, by Lemmata 2 and 5 we obtain the main result of this section.

Theorem 3. *Given a* B-CRSM *$\mathcal{S}$ and $u \in start$, checking the existence of an accepting run of $\mathcal{S}$ from $\langle u \rangle$ can be decided in time $O\big((|\mathcal{F}| \cdot (n_V + n_\delta))^{O(\rho)}\big)$.*

5.2. Reduction of model checking to emptiness of Büchi CRSM

In this section, we decide the model-checking problem of *CRSM* against ConCaRet using an automata-theoretic approach: for a *CRSM* $\mathcal{S}$ and a ConCaRet formula φ, we construct a *B-CRSM* $\mathcal{S}_\varphi$ such that $\mathcal{S}_\varphi$ has an accepting run iff $\mathcal{S}$ has a run that satisfies φ. More precisely, an accepting run of $\mathcal{S}_\varphi$ corresponds to a run π of $\mathcal{S}$, where each local state q is equipped with the information concerning the set of subformulas of φ that hold at q along π. The construction proposed here follows and extends that given in [3] for CaRet.

First, we describe informally the main aspects of the construction.

The next modalities are handled similarly as in [3]. In particular, for the abstract modalities, we need to ensure that if a call $(b, e_1, \ldots, e_n)$ returns and its matching return is $(b, x_1, \ldots, x_n)$, then denoting with A the set of φ subformulas which hold at call $(b, e_1, \ldots, e_n)$, the abstract-next requirements belonging to A get satisfied at return $(b, x_1, \ldots, x_n)$. In order to satisfy this condition, at the call we keep track of A pushing it onto the local stack along with the box b. For the caller modalities, if we are at a call $(b, e_1, \ldots, e_n)$, then a formula $\bigcirc^- \psi$ is true in the ith module being called (for $i = 1, \ldots, n$) if and only if ψ is true at the call. The caller formulas are hence passed down from the caller to the called modules. For the branching modalities, we have to ensure that the existential (resp., universal) next requirements are met in some (in each) linear successor of the current local state. This is captured locally in the transitions of $\mathcal{S}_\varphi$. Moreover, we use the terminal acceptance condition in order to guarantee that forward next formulas cannot be asserted at local states whose set of linear successors is empty.

For the until modalities, the main technical issue is to guarantee the fulfillment of liveness requirements ψ_2 in until subformulas of φ of the form $\psi_1 \mathsf{U}^b \psi_2$ where $b \in \{a, \exists, \forall\}$ (caller-until formulas do not require such condition since a caller path is always finite). This is done using a generalized Büchi condition having an accepting component for each of such until subformulas of φ. The abstract and universal until formulas are managed as in [3]. In particular, if an abstract until

formula $\psi_1 \mathsf{U}^a \psi_2$ is asserted at a local state (i, y) of the given run of $\mathcal{S}$, its liveness requirement must be met in the abstract path r starting from (i, y). If r is infinite, then a local state reachable from (i, y) by a linear path is visited by r if and only if the associated module instance does not return. Thus, we keep track in a vertex of $\mathcal{S}_\varphi$ whether the corresponding local state of $\mathcal{S}$ belongs to the invocation of a module that will eventually return or not. Then, for an abstract until formula, the associated Büchi accepting component has only vertices corresponding to local states whose associated module instances do not return. For the existential until formulas ψ, when ψ is asserted at a local state (i, y), we have to ensure that ψ is satisfied in at least one of the linear paths from (i, y). In order to achieve this and ensure the acceptance of all infinite linear paths from (i, y), we use a fresh atomic proposition τ_ψ.

Finally, parallel formulas are handled by the synchronization predicate.

Now, we give the details of the construction. Fix a *CRSM* $\mathcal{S} = \langle (S_1, \ldots, S_k), start \rangle$, where $S_i = \langle \Sigma_i, \Sigma_i^s, N_i, B_i, Y_i, En_i, Ex_i, \delta_i, \eta_i \rangle$ (for $1 \leq i \leq k$), and a formula φ over AP. First, we need some notation.

The *closure* of φ, denoted by $Cl(\varphi)$, is the smallest set containing $call, ret, int, tt, \bigcirc^\exists tt$, propositions τ_ψ for each existential until subformula ψ of φ, all subformulas of φ, $\bigcirc^b(\psi_1 \mathsf{U}^b \psi_2)$ for any subformula $\psi_1 \mathsf{U}^b \psi_2$ of φ with $b \in \{a, -, \exists, \forall\}$, and the negations of all these formulas (we identify $\neg\neg\psi$ with ψ).

Note that the size of $Cl(\varphi)$ is linear in the size of φ. An *atom* of φ is a set $A \subseteq Cl(\varphi)$ such that $tt \in A$ and the following properties are satisfied:

- A contains exactly one of the elements in the set $\{call, ret, int\}$;
- for each $\psi \in Cl(\varphi)$, $\psi \in A$ iff $\neg\psi \notin A$;
- for each $\psi_1 \wedge \psi_2 \in Cl(\varphi)$, $\psi_1 \wedge \psi_2 \in A$ iff $\psi_1 \in A$ and $\psi_2 \in A$;
- for each $\psi_1 \mathsf{U}^b \psi_2 \in Cl(\varphi)$, where $b \in \{a, -, \exists\}$, $\psi_1 \mathsf{U}^b \psi_2 \in A$ iff *either* $\psi_2 \in A$, *or* $\psi_1 \in A$ and $\bigcirc^b(\psi_1 \mathsf{U}^b \psi_2) \in A$;
- for each $\psi_1 \mathsf{U}^\forall \psi_2 \in Cl(\varphi)$, $\psi_1 \mathsf{U}^\forall \psi_2 \in A$ iff *either* $\psi_2 \in A$, *or* $\psi_1 \in A$, $\bigcirc^\forall(\psi_1 \mathsf{U}^\forall \psi_2) \in A$, and $\bigcirc^\exists tt \in A$.

Let $Atoms(\varphi)$ be the set of atoms of φ. Note that $|Atoms(\varphi)|$ is $2^{O(|\varphi|)}$. Intuitively, an atom of φ represents a maximal set of formulas in $Cl(\varphi)$ that can consistently hold at a local state of a run of $\mathcal{S}$. We say that an atom A is *consistent* with a vertex v of $\mathcal{S}$ if $\eta(v) \cap Cl(\varphi) = A \cap AP$ and, further: if v is a node then $int \in A$, if v is a call then $call \in A$, and if v is a return then $ret \in A$.

For an atom A, we denote by $CallerForm(A)$ the set $\{\bigcirc^- \psi \mid \bigcirc^- \psi \in A\}$. Moreover, for atoms A and A', we define a predicate $AbsNextReq(A, A')$ that holds iff for each $\bigcirc^a \psi \in Cl(\varphi)$, $\bigcirc^a \psi \in A$ iff $\psi \in A'$ (i.e. the abstract-next requirements in A are exactly the ones that hold in A'). Similarly, we define predicates $\forall NextReq(A, \{A_1, \ldots, A_n\})$ and $\exists NextReq(A, \{A_1, \ldots, A_n\})$, where A is an atom and $\{A_1, \ldots, A_n\}$ is a set of atoms. Predicate $\forall NextReq(A, \{A_1, \ldots, A_n\})$ holds iff: for each $\bigcirc^\forall \psi \in Cl(\varphi)$, $\bigcirc^\forall \psi \in A$ iff $\psi \in \bigcap_{i=1}^{i=n} A_i$. Predicate $\exists NextReq(A, \{A_1, \ldots, A_n\})$ holds iff: (i) for each $\bigcirc^\exists \psi \in Cl(\varphi)$, $\bigcirc^\exists \psi \in A$ iff $\psi \in \bigcup_{i=1}^{i=n} A_i$, and (ii) for each $\psi \in A$ of the form $\psi_1 \mathsf{U}^\exists \psi_2$, if $\psi_2 \notin A$ then there is $1 \leq h \leq n$ such that $\psi \in A_h$ and $\tau_\psi \in A_h \setminus \bigcup_{i \neq h} A_i$ (*nondeterministic path selection*).

The *B-CRSM* $\mathcal{S}_\varphi$ is given by $\langle (S'_1, \ldots, S'_k), start', \mathcal{F}, \mathcal{P}_f, \mathcal{P}_{sync} \rangle$, where for each $1 \leq i \leq k$, $S'_i = \langle \Sigma_i, \Sigma_i^s, N'_i, B'_i, Y'_i, En'_i, Ex'_i, \delta'_i, \eta'_i \rangle$, and is defined in detail in the following.

For each node u in S_i, S'_i contains nodes of the form (u, A, tag), where A is an atom giving the set of formulas that hold at u and tag is a flag set to *fin* iff the local run in the current activation I of S_i leads to an exit node of I whose linear successor is defined (return from I), and set to *inf* otherwise. Similarly, for every box b in S_i, S'_i contains boxes of the form (b, A, R, tag), where A and R are atoms, and $tag \in \{inf, fin\}$. Intuitively, A contains formulas that hold at calls $(b, e_1, \ldots, e_n)$ and R contains formulas that hold at matching returns $(b, x_1, \ldots, x_n)$. Formally:

- $N'_i = \{(u, A, tag) \in N_i \times Atoms(\varphi) \times \{fin, inf\} \mid A \text{ is consistent with } u\}$;
- $B'_i = B_i \times Atoms(\varphi) \times Atoms(\varphi) \times \{fin, inf\}$; $Y'_i((b, A, R, tag)) = Y_i(b)$;
- $En'_i = \{(e, A, tag) \in N'_i \mid e \in En_i\}$; $Ex'_i = \{(x, A, tag) \in N'_i \mid x \in Ex_i\}$;
- $\eta'_i((u, A, tag)) = \eta_i(u)$;
  $\eta'_i(((b, A, R, tag), (u_1, A_1, tag_1), \ldots, (u_n, A_n, tag_n))) = \eta_i((b, u_1, \ldots, u_n))$.

To define the transition function δ'_i we introduce some notation. For a vertex/box v of $\mathcal{S}_\varphi$, $Pr(v)$ denotes the corresponding vertex/box in $\mathcal{S}$, while $Tag(v)$ denotes the tag-component of the associated box, if v is a call/return, and the tag-component of v otherwise. For a node $v = (u, A, tag)$, $Atom(v)$ denotes A, and for a call (resp. return) v with box (b, A, R, tag), $Atom(v)$ denotes A (resp., R). A call $v = ((b, A, R, tag), (e_1, A_1, tag_1), \ldots, (e_n, A_n, tag_n))$ is *well defined* iff:

- A is consistent with $Pr(v)$, $CallerForm(A_j) = \{\bigcirc^- \psi \in Cl(\varphi) \mid \psi \in A\}$ for $1 \leq j \leq n$, and $\forall NextReq(A, \{A_1, \ldots, A_n\})$ and $\exists NextReq(A, \{A_1, \ldots, A_n\})$ hold.
- If $tag_j = inf$ for some $1 \leq j \leq n$, then $tag = inf$ and there is no formula of the form $\bigcirc^a \psi$ in A; otherwise, $AbsNextReq(A, R)$ holds and $CallerForm(A) = CallerForm(R)$.

Note that the rules given above conform to the interpretation that A corresponds to the set of formulas true at call v and $A_1, \ldots, A_n$ are the sets of formulas true respectively at the next vertices (the linear successors) which correspond to the entries $e_1, \ldots, e_n$. Consistently, the call returns (i.e. its abstract successor is defined) iff $tag_j = fin$ for all $1 \leq j \leq n$. In this case, R is the set of formulas true at the return, and we require that the abstract-next requirements in A are met in R and the caller formulas of A and R coincide.

We say that a return $v = ((b, A, R, tag), (x_1, A_1, tag_1), \ldots, (x_n, A_n, tag_n))$ is *well defined* if R is consistent with $Pr(v)$ and for each $1 \leq j \leq n$, $tag_j = fin$, A_j does not contain formulas of the form $\bigcirc^a \psi$, and $\forall NextReq(A_j, \{R\})$ and $\exists NextReq(A_j, \{R\})$ hold.

Now, we define the transition function δ'_i: we have that $v' \in \delta'_i(v, \sigma)$ iff the following holds (we write A and A' for $Atom(v)$ and $Atom(v')$, respectively):

- $Pr(v') \in \delta_i(Pr(v), \sigma)$ and $Tag(v) = Tag(v')$;
- $AbsNextReq(A, A')$, $\forall NextReq(A, \{A'\})$, and $\exists NextReq(A, \{A'\})$ hold, and $CallerForm(A) = CallerForm(A')$;
- if v (resp., v') is a return (resp., a call), then v (resp., v') is well defined.

The rest of the construction of $\mathcal{A}_\varphi$ is as follows. The set of initial nodes $start'$ is the set of nodes (u, A, tag) such that $u \in start$, $tag = inf$, $\varphi \in A$, $CallerForm(A) = \emptyset$, and $A \supseteq \{\|\psi \in Cl(\varphi)\}$.
The generalized Büchi condition $\mathcal{F}$ is given by the following sets:

- the set of all vertices v such that $Tag(v) = inf$;
- for each formula $\psi_1 \mathsf{U}^\forall \psi_2 \in Cl(\varphi)$, the set of all vertices v such that either $\psi_2 \in Atom(v)$ or $\psi_1 \mathsf{U}^\forall \varphi_2 \notin Atom(v)$;
- for each formula $\psi \in Cl(\varphi)$ of the form $\psi_1 \mathsf{U}^\exists \psi_2$, the set of all vertices v such that or $\psi_2 \in Atom(v)$ or $\psi_1 \mathsf{U}^\exists \psi_2 \notin Atom(v)$ or $\tau_\psi \notin Atom(v)$;
- for each formula $\psi_1 \mathsf{U}^a \psi_2 \in Cl(\varphi)$, the set of all vertices v such that $Tag(v) = inf$, and either $\psi_2 \in Atom(v)$ or $\psi_1 \mathsf{U}^a \psi_2 \notin Atom(v)$.

The first set ensures that the tags are guessed correctly. The second class of sets ensures that when a formula $\psi_1 \mathsf{U}^\forall \psi_2$ is asserted at a vertex u, then for every linear path from u, ψ_2 is eventually satisfied. The third class of sets ensures that when a formula ψ of the form $\psi_1 \mathsf{U}^\exists \psi_2$ is asserted at a vertex u, the linear path from u marked by τ_ψ eventually satisfies ψ_2. The last class of sets ensures that the liveness requirements in abstract-until formulas, asserted at vertices where the abstract path is infinite, get eventually satisfied.

The terminal acceptance set $\mathcal{P}_F$ is the set of vertices v of $\mathcal{A}_\varphi$ such that $tag(v) = inf$, $Atom(v)$ does *not* contain formulas of the form $\bigcirc^a \varphi'$, $\bigcirc^\exists \varphi'$, $\neg\bigcirc^\forall \varphi'$, and if v is a return (resp., a call), then it is well defined.

Finally, the synchronized acceptance condition $\mathcal{P}_{sync}$ is used to handle parallel formulas. If a module instance I is one of the targets of the current move of $\mathcal{A}$ and A is the set of formulas currently asserted at I, then, consistently with the semantics of the parallel modality, we require that for each parallel formula $\|\psi \in Cl(\varphi)$, $\|\psi \in A$ iff for each module instance J in parallel with I (i.e., sibling of I) the following holds: if J is not busy in a module call, then J has to satisfy ψ from the current vertex; otherwise, J has to satisfy ψ from the call vertex associated with the current module call of J.

Therefore, $\mathcal{P}_{sync}$ is defined as follows. For each vertex v and each set of vertices H of $\mathcal{A}_\varphi$ with $|H| \leq rank(\mathcal{A})$, $\mathcal{P}_{sync}(v, H)$ holds iff for each $\|\psi \in Cl(\varphi)$: $\|\psi \in Atom(v)$ *iff* for each $v' \in H$, $\psi \in Atom(v')$. Note that H can be empty.

Correctness of the construction. We extend the function Pr to the set of global states $q = (t, D)$ of $\mathcal{A}_\varphi$ as follows: $Pr(q) = (t, D')$ where for each $y \in t$, $D'(y) = Pr(D(y))$ (note that $Pr(q)$ is a global state of $\mathcal{A}$). Let $\widetilde{Cl}(\varphi) = Cl(\varphi) \setminus \{\tau_\psi, \neg\tau_\psi \mid \psi = \psi_1 \mathsf{U}^\exists \psi_2 \in Cl(\varphi)\}$ (i.e., $\widetilde{Cl}(\varphi)$ is obtained from $Cl(\varphi)$ by removing all the fresh propositions associated with the existential until formulas). Correctness of the construction is stated by the following two lemmata whose proofs are reported in the Appendix A.

Lemma 6. *Let $\pi = q_0 \xrightarrow{(\ell_0, \ell'_0)} q_1 \xrightarrow{(\ell_1, \ell'_1)} q_2 \ldots$ be an accepting run of $\mathcal{A}_\varphi$ with $q_i = (t_i, D_i)$ for every $i \geq 0$ and $D_0(\varepsilon) \in start'$. Then, $\pi' = Pr(q_0) \xrightarrow{(\ell_0, \ell'_0)} Pr(q_1) \xrightarrow{(\ell_1, \ell'_1)} Pr(q_2) \ldots$ is a run of $\mathcal{A}$, and for every local state $(i, y) \in Q_{\pi'}$ and $\psi \in Atom(D_i(y)) \cap \widetilde{Cl}(\varphi)$, it holds that $(i, y) \models_{\pi'} \psi$.*

Lemma 7. *Let π be a run of $\mathcal{A}$ starting from $\langle v \rangle$ with $v \in start$ and satisfying φ. Then, there is an accepting run of $\mathcal{A}_\varphi$ starting from a state $\langle v' \rangle$ with $v' \in start'$ and $Pr(v') = v$.*

By Lemmata 6 and 7 it follows that for every $v \in start$, there is a run of $\mathcal{A}$ starting from $\langle v \rangle$ and satisfying φ if and only if there is an accepting run of $\mathcal{A}_\varphi$ starting from a state $\langle v' \rangle$ with $v' \in start'$ and $Pr(v') = v$. Thus, the model-checking problem of $\mathcal{A}$ against φ is reduced to check emptiness for the Büchi *CRSM* $\mathcal{A}_{\neg\varphi}$ (associated with the negation of φ). In the proposed construction, for every vertex/edge in $\mathcal{A}$, we have $2^{O(|\varphi| \cdot rank(\mathcal{A}))}$ vertices/edges in $\mathcal{A}_\varphi$. Also, the number of accepting sets in the generalized Büchi condition is at most $O(|\varphi|)$ and $rank(\mathcal{A}_\varphi) = rank(\mathcal{A})$. For $rank(\mathcal{A}) = 1$, the considered problem coincides with the model-checking problem of *RSM* against CaRet that is Exptime-complete (even for a fixed *RSM*). Therefore, by Theorem 3, we obtain the following result.

Theorem 4. *For a* CRSM *$\mathcal{A}$ and a formula φ of* ConCaRet*, the model-checking problem for $\mathcal{A}$ against φ can be decided in time exponential in $|\varphi| \cdot (rank(\mathcal{A}))^2$. The problem is* Exptime*-complete (even when the* CRSM *is fixed).*

Let us consider the fragment Υ of global *LTL* defined in Section 4.4. As a consequence of Theorems 2 and 4, we obtain the following.

Corollary 1. *The model-checking problem of* CRSM *against the fragment Υ of global* LTL *is decidable in* Exptime.

6. Conclusion

In this paper, we have introduced a model for concurrent programs with recursive procedural calls and a temporal logic which allows us to express properties on threads running in parallel. To achieve decidability we have placed a restriction in the model on the capability of threads to synchronize (only threads activated on the same fork and which are not waiting for a return from a parallel procedure call can synchronize) and removed the global operators from our logic. Nevertheless, we have argued that the obtained formalisms are still interesting by showing that we can check interesting properties locally to each parallel thread and globally on the whole computation.

We observe that the notions of local successor, which we have defined to deal with local properties of runs, can be also given for term rewriting systems such as ground tree rewriting systems and process rewrite systems in which the global states (terms) represent finite trees. However, we argue that *CRSM* allow a more intuitive formalization of these notions since they provide an explicit representation of modularity.

Finally, we remark that our results can be applied to infinite-state formalisms such as ground tree rewriting systems and parallel flow graphs which can be effectively translated into *CRSM*.

Appendix A

A.1. Proof of Lemma 6

In order to prove Lemma 6, we need the following preliminary result.

Proposition 2. *Let* $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$ *be an accepting run of* $\mathcal{S}_\varphi$ *with* $q_i = (t_i, D_i)$ *for every* $i \geq 0$ *and* $D_0(\varepsilon) \in start'$. *Then, for every* $(i, y) \in Q_\pi$:

1. *if* $next^a_\pi(i, y) = \bot$, *then* $Atom(D_i(y))$ *does not contain formulas having the form* $\bigcirc^a \psi$;
2. *if* $next^a_\pi(i, y) = (j, y)$, *then* $AbsNextReq(Atom(D_i(y)), Atom(D_j(y)))$;
3. *if the abstract path* $(i_0, y)(i_1, y) \ldots$ *of* π *starting from* (i, y) *is infinite, then for every* $\psi_1 \mathrm{U}^a \psi_2 \in Cl(\varphi)$, *the set* $\{h \in \mathbb{N} \mid \{\neg(\psi_1 \mathrm{U}^a \varphi_2), \psi_2\} \cap Atom(D_{i_h}(y)) \neq \emptyset\}$ *is infinite.*

Proof. In order to prove the proposition, first we prove two claims. The first claim directly follows from the definition of the transition function of $\mathcal{S}_\varphi$.

Claim 1. *If* $(i, y) \in Q_\pi$ *and* $next^a_\pi(i, y) = (j, y)$, *then* $Tag(D_i(y)) = Tag(D_j(y))$.

Claim 2. *If* $(i, y) \in Q_\pi$ *and* $D_i(y) = ((b, A, R, tag), (e_1, A_1, tag_1), \ldots, (e_n, A_n, tag_n))$ *is a call vertex, then* $next^a_\pi(i, y) = \bot$ *iff* $tag_j = inf$ *for some* $1 \leq j \leq n$.

Proof of the claim. First, assume that $next^a_\pi(i, y) = \bot$. Let $r = (i_0, y_0)(i_1, y_1) \ldots$ be a linear path of π such that $(i_0, y_0) = (i, y)$. Evidently, $D_{i_1}(y_1) = (e_j, A_j, tag_j)$ for some $1 \leq j \leq n$. Since $next^a_\pi(i, y) = \bot$, it holds that $y_h \succ y_0$ for all $1 \leq h < |r|$. Since π is accepting, we deduce that there exists $1 \leq m < |r|$ such that $Tag(D_{i_m}(y_m)) = inf$. We prove that $tag_j = inf$ by induction on m. If $m = 1$, then the claim holds. Now, assume that $m > 1$. If $D_{j_{m-1}}(y_{m-1})$ is a call vertex, then since such a call vertex is well defined, we deduce that $Tag(D_{j_{m-1}}(y_{m-1})) = inf$. Therefore, in this case the claim follows from the induction hypothesis. If instead $D_{j_{m-1}}(y_{m-1})$ is *not* a call vertex, then since $y_m \succ y_0$, it easily follows that there is $0 < h < m$ such that $next^a_\pi(i_h, y_h) = (i_m, y_m)$. Then, in this case the claim follows from Claim 1 and the induction hypothesis. Thus, the right implication in Claim 2 holds.

Now, assume that $tag_j = inf$ for some $1 \leq j \leq n$. Evidently, there is a linear path of π, $r = (i_0, y_0)(i_1, y_1) \ldots$ such that $(i, y) = (i_0, y_0)$, $D_{i_1}(y_1) = (e_j, A_j, tag_j)$, and $y_1 \succ y_0$. We claim that $next^a_\pi(i, y) = \bot$. Assume on the contrary that $next^a_\pi(i, y) \neq \bot$ and derive a contradiction. Then, there exists $1 < m < |r|$ such that $(i_m, y_m) = next^a_\pi(i, y)$. Note that $y_m = y$ and $D_{i_m}(y_m)$ corresponds to the matching return of $D_i(y)$. Moreover, $D_{i_{m-1}}(y_{m-1})$ is an exit node. Since $D_{i_m}(y_m)$ is well defined, it follows that $Tag(D_{i_{m-1}}(y_{m-1})) = fin$ and the abstract path of π starting from (i_1, y_1) is finite and leads to (i_{m-1}, y_{m-1}). Therefore, by Claim 1 it follows that $tag_j = Tag(D_{i_1}(y_1)) = fin$, which is a contradiction. This concludes the proof of Claim 2. □

Now, we prove the proposition.

1. Let $next^a_\pi(i, y) = \bot$. If $next^\ell_\pi(i, y) = \emptyset$, then, since π satisfies the terminal acceptance condition $\mathcal{P}_F$, by def. of $\mathcal{P}_F$, Property 1 holds. Now, assume that $next^\ell_\pi(i, y) \neq \emptyset$. Then, either $D_i(y)$ is an exit node or $D_i(y)$ is a call vertex. In the first case, $next^\ell_\pi(i, y) \neq \emptyset$ corresponds to a return vertex, which is well defined. Thus, in this case Property 1 holds. In the second case, $D_i(y)$ is a call of the form $((b, A, R, tag), (e_1, A_1, tag_1), \ldots (e_n, A_n, tag_n))$. By Claim 2 there is $1 \leq j \leq n$ such that $tag_j = inf$. Since $D_i(y)$ is well defined, Property 1 holds also in this case.
2. Let $next^a_\pi(i, y) = (j, y)$. If $D_i(y)$ is not a call, then, Property 2 directly follows from the definition of the transition function of $\mathcal{S}_\varphi$. Now, assume that $D_i(y)$ is a call. Since $next^a_\pi(i, y) = (j, y)$, it follows that $D_j(y)$ is a return vertex and the box associated with $D_j(y)$ is the same as that associated with $D_i(y)$. Therefore, $D_i(y)$ has the form $((b, A, R, tag), (e_1, A_1, tag_1), \ldots, (e_n, A_n, tag_n))$, $Atom(D_i(y)) = A$, and $Atom(D_j(y)) = R$. By Claim 2, $tag_j = fin$ for each $1 \leq j \leq n$. Thus, since $D_i(y)$ is well defined, Property 2 holds.

3. Let $\psi_1 \mathsf{U}^a \psi_2 \in Cl(\varphi)$ and $r = (i_0, y)(i_1, y)\ldots$ be an infinite abstract path of π. Let us consider a linear path $r' = (j_0, z_0)(j_1, z_1)\ldots$ of π starting from (i_0, y) (i.e., $(j_0, z_0) = (i_0, y)$). Evidently, r is a subsequence of r', i.e. there is a monotone sequence of integers $(k_n)_{n\in\mathbb{N}}$ such that $k_0 = 0$, and, for every $n \in \mathbb{N}$, $(j_{k_n}, z_{k_n}) = (i_n, y)$. Let $n, h \in \mathbb{N}$ such that $k_n < h < k_{n+1}$. Evidently, the abstract path of π starting from (j_h, z_h) is finite and leads to an exit node whose linear successor is defined and is a return vertex, which is well defined. Therefore, by Claim 1 it follows that $Tag(D_{j_h}(z_h)) = fin$. Then, Property 3 directly follows from the definition of the generalized Büchi acceptance condition $\mathcal{F}$ and the fact that π is accepting. □

Now, we can prove Lemma 6.

Lemma 6. *Let $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$ be an accepting run of $\mathcal{A}_\varphi$ with $q_i = (t_i, D_i)$ for every $i \geq 0$ and $D_0(\varepsilon) \in start'$. Then, $\pi' = Pr(q_0) \xrightarrow{(\ell_0,\ell'_0)} Pr(q_1) \xrightarrow{(\ell_1,\ell'_1)} Pr(q_2)\ldots$ is a run of $\mathcal{A}$, and for every local state $(i, y) \in Q_{\pi'}$ and $\psi \in Atom(D_i(y)) \cap \widetilde{Cl}(\varphi)$, it holds that $(i, y) \models_{\pi'} \psi$.*

Proof. Let $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$ be an accepting run of $\mathcal{A}_\varphi$ with $q_i = (t_i, D_i)$ for every $i \geq 0$. Let us consider the infinite sequence $\pi' = Pr(q_0) \xrightarrow{(\ell_0,\ell'_0)} Pr(q_1) \xrightarrow{(\ell_1,\ell'_1)} Pr(q_2)\ldots$. By definition of the transition function of $\mathcal{A}_\varphi$, it easily follows that π' is a run of $\mathcal{A}$. Now, let $(i, y) \in Q_{\pi'}$ be any local state of π'. In order to complete the proof it suffices to prove the following claim.

Claim. for each $\psi \in \widetilde{Cl}(\varphi)$, $(i, y) \models_{\pi'} \psi$ if and only if $\psi \in Atom(D_i(y))$.

Proof of the claim. The proof is by structural induction on ψ. By construction we have that $Atom(D_i(y))$ is consistent with $Pr(D_i(y))$. Thus, for the case in which ψ is an atomic proposition, the claim holds. The cases concerning boolean connectives easily follow from the induction hypothesis. It remains to analyze the cases related to the temporal operators. Here, we consider the most relevant ones (the other ones are similar or easier).

- $\psi = \psi_1 \mathsf{U}^a \psi_2$. Let $r = (i_0, y)(i_1, y)\ldots$ be the abstract path of π' starting from (i, y). Note that r is also the abstract path of π starting from (i, y).
  $(i, y) \models_{\pi'} \psi_1 \mathsf{U}^a \psi_2 \Rightarrow \psi_1 \mathsf{U}^a \psi_2 \in Atom(D_i(y))$. Assume that $(i, y) \models_{\pi'} \psi_1 \mathsf{U}^a \psi_2$. Then, there is $h \geq 0$ such that $(i_h, y) \models_{\pi'} \psi_2$ and for all $0 \leq j < h$, $(i_j, y) \models_{\pi'} \psi_1$. By the induction hypothesis, it follows that $\psi_2 \in Atom(D_{i_h}(y))$ and for all $0 \leq j < h$, $\psi_1 \in Atom(D_{i_j}(y))$. By definition of atom and Proposition 2(2), it follows that $\psi_1 \mathsf{U}^a \psi_2 \in Atom(D_{i_j}(y))$ for all $0 \leq j \leq h$. Since $i = i_0$, the implication above holds.
  $\psi_1 \mathsf{U}^a \psi_2 \in Atom(D_i(y)) \Rightarrow (i, y) \models_{\pi'} \psi_1 \mathsf{U}^a \psi_2$. Let $\psi_1 \mathsf{U}^a \psi_2 \in Atom(D_i(y))$. Assume that the abstract path r is infinite (the other case being easier). By definition of atom and Proposition 2(2), it follows that *either* (1) there is $h \geq 0$ such that $\psi_2 \in Atom(D_{i_h}(y))$ and for all $0 \leq j < h$, $\psi_1 \in Atom(D_{i_j}(y))$, *or* (2) for all $h \geq 0$, $\{\neg\psi_2, \psi_1 \mathsf{U}^a \psi_2\} \subseteq Atom(D_{i_h}(y))$. By Proposition 2(3), condition (2) cannot hold. Then, by the induction hypothesis, it follows that $(i_h, y) \models_{\pi'} \psi_2$ and for all $0 \leq j < h$, $(i_j, y) \models_{\pi'} \psi_1$. Hence, $(i, y) \models_{\pi'} \psi_1 \mathsf{U}^a \psi_2$. Thus, the implication above holds.
- $\psi = \psi_1 \mathsf{U}^\exists \psi_2$.
  $(i, y) \models_{\pi'} \psi_1 \mathsf{U}^\exists \psi_2 \Rightarrow \psi_1 \mathsf{U}^\exists \psi_2 \in Atom(D_i(y))$. Let $(i, y) \models_{\pi'} \psi_1 \mathsf{U}^\exists \psi_2$. Then, there is a linear path r of π' of the form $r = (i_0, y_0)(i_1, y_1)\ldots$ such that $(i_0, y_0) = (i, y)$ and for some $h \geq 0$, $(i_h, y_h) \models_{\pi'} \psi_2$ and $(i_j, y_j) \models_{\pi'} \psi_1$ for all $0 \leq j < h$. By the induction hypothesis, it follows that $\psi_2 \in Atom(D_{i_h}(y_h))$ and for all $0 \leq j < h$, $\psi_1 \in Atom(D_{i_j}(y_j))$. Since r is also a linear path of π and every call or return vertex along π is well defined, by the definition of the transition function of $\mathcal{A}_\psi$, the definition of atom, and the definition of the predicate $\exists NextReq$, it follows that $\psi_1 \mathsf{U}^\exists \psi_2 \in Atom(D_{i_j}(y_j))$ for all $0 \leq j \leq h$. Since $(i, y) = (i_0, y_0)$, the implication above holds.
  $\psi_1 \mathsf{U}^\exists \psi_2 \in Atom(D_i(y)) \Rightarrow (i, y) \models_{\pi'} \psi_1 \mathsf{U}^\exists \psi_2$. Let $\psi = \psi_1 \mathsf{U}^\exists \psi_2 \in Atom(D_i(y))$. Since every call or return vertex along π is well defined, by the definition of the transition function of $\mathcal{A}_\psi$, the terminal acceptance $\mathcal{P}_F$, the definition of atom, and the definition of the predicate $\exists NextReq$, it follows that there is a linear path r of π of the form $r = (i_0, y_0), (i_1, y_1)\ldots$ such that $(i, y) = (i_0, y_0)$ and *either* (1) there is $h \geq 0$ such that $\psi_2 \in Atom(D_{i_h}(y_h))$ and for all $0 \leq j < h$, $\psi_1 \in Atom(D_{i_j}(y_j))$, *or* (2) r is infinite and for all $h \geq 0$, $\{\neg\psi_2, \psi_1 \mathsf{U}^\exists \psi_2, \tau_\psi\} \subseteq Atom(D_{i_h}(y_h))$. Since π satisfies the generalized Büchi condition $\mathcal{F}$, condition (2) cannot hold. Then, by the induction hypothesis, it follows that $(i_h, y_h) \models_{\pi'} \psi_2$ and for all $0 \leq j < h$, $(i_j, y_j) \models_{\pi'} \psi_1$. Since r is also a linear path of π' starting from (i, y), it follows that $(i, y) \models_{\pi'} \psi_1 \mathsf{U}^\exists \psi_2$. Thus, the implication above holds.
- $\psi = \|\psi'$. Let $h = min\{j \leq i \mid next^\ell_{\pi'}(j, y) = next^\ell_{\pi'}(i, y)\}$. Assume that $h \neq 0$ (the other case being trivial). Then, it holds that $D_h(y) = D_i(y)$ and $y \in \ell'_{h-1}$. By the semantics of ConCaRet, $(i, y) \models_{\pi'} \|\psi'$ if and only if for all $y' \in siblings(t_h, y) \setminus \{y\}$, $(h, y') \models_{\pi'} \psi'$ if (h, y') is a local state, and $call_{\pi'}(h, y') \models_{\pi'} \psi'$ otherwise. By the induction hypothesis, it follows that $(i, y) \models_{\pi'} \|\psi'$ if and only if for all $y' \in siblings(t_h, y) \setminus \{y\}$, $\psi' \in Atom(v(h, y'))$, where $v(h, y')$ is defined as follows: if (h, y') is a local state, then $v(h, y') = D_h(y')$; otherwise, $v(h, y') = D_m(y')$ where $(m, y') = call_{\pi'}(h, y')$. Since $y \in \ell'_{h-1}$ and π satisfies the synchronized acceptance condition $\mathcal{P}_{sync}$, by def. $\mathcal{P}_{sync}$ it follows that $(i, y) \models_{\pi'} \|\psi'$ if and only if $\|\psi' \in Atom(D_h(y)) = Atom(D_i(y))$. Thus, also in this case the claim holds.

This concludes the proof of the claim. □

A.2. Proof of Lemma 7

In order to prove Lemma 7, we need additional definitions and simple results stated by the following Propositions 3–5.

In the following we fix a run of $\mathcal{S}$, $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$, where $q_i = (t_i, D_i)$ for each $i \geq 0$. For each existential until formula $\psi = \psi_1 \mathsf{U}^{\exists} \psi_2 \in Cl(\varphi)$, we define a mapping $I_\psi : Q_\pi \to \mathbb{N}$ associating an index to every local state (i, y) of π as follows:

- if $(i, y) \models_\pi \neg\psi$, then $I_\psi(i, y) = 0$,
- otherwise, $I_\psi(i, y)$ is the *smallest* h such that there is a linear path of π from (i, y) of the form $(i_0, y_0) \ldots (i_h, y_h) \ldots$ such that $(i_h, y_h) \models \psi_2$.

Furthermore, we denote by $\mathcal{A}_\pi : Q_\pi \to Atoms(\varphi)$ a mapping satisfying the following properties for all local states $(i, y) \in Q_\pi$:

1. $\mathcal{A}_\pi(i, y) \cap \widetilde{Cl}(\varphi) = \{\psi \in \widetilde{Cl}(\varphi) \mid (i, y) \models_\pi \psi\}$,
2. for every $\psi = \psi_1 \mathsf{U}^{\exists} \psi_2 \in Cl(\varphi)$, if $(i, y) \models_\pi \psi \wedge \neg\psi_2$ and $next^\ell_\pi(i, y) = \{(h, z_1), \ldots, (h, z_p))\} \neq \emptyset$, then there is $1 \leq m \leq p$ such that $(h, z_m) \models \psi$, $I_\psi(h, z_m) = I_\psi(i, y) - 1$ and $\tau_\psi \in \mathcal{A}_\pi(h, z_m) \setminus \bigcup_{j \neq m} \mathcal{A}_\pi(h, z_j)$ (*minimal path selection*),
3. for each $(j, y) \in Q_\pi$ such that $next^\ell_\pi(j, y) = next^\ell_\pi(i, y)$, $\mathcal{A}_\pi(j, y) = \mathcal{A}_\pi(i, y)$.

By the semantics of ConCaRet, for each $\psi \in \widetilde{Cl}(\varphi)$ and pair of local states (i, y), (j, y) such that $next^\ell_\pi(j, y) = next^\ell_\pi(i, y)$, it holds that $(j, y) \models_\pi \psi$ iff $(i, y) \models_\pi \psi$. Moreover, if ψ is an existential until formula, then $I_\psi(j, y) = I_\psi(i, y)$. Therefore, a mapping satisfying Conditions 1, 2, and 3 exists. Intuitively, $\mathcal{A}_\pi(i, y)$ contains all and only the formulas in $\widetilde{Cl}(\varphi)$ which are fulfilled at (i, y) along π. In addition, for each local state (i, y) satisfying an until existential formula ψ, we keep track by proposition τ_ψ of a "minimal" linear path starting from (i, y) satisfying ψ.

Finally, let $\tau_\pi : Q_\pi \to \{inf, fin\}$ be the mapping defined as: for each local state (i, y), $\tau_\pi(i, y) = inf$ iff the abstract path of π from (i, y) either is infinite, or leads to a local state (j, y) such that either $D_j(y) \in Calls$ or $next^\ell_\pi(j, y) = \emptyset$.

Proposition 3. *Let $r = (i_0, y_0)(i_1, y_1) \ldots$ be an infinite linear path of π and $\psi = \psi_1 \mathsf{U}^{\exists} \psi_2 \in Cl(\varphi)$. Then, the set $\{h \in \mathbb{N} \mid \{\neg\psi, \psi_2, \neg\tau_\psi\} \cap \mathcal{A}_\pi(i_h, y_h) \neq \emptyset\}$ is infinite.*

Proof. Assume on the contrary that the set $\{h \in \mathbb{N} \mid \{\neg\psi, \psi_2, \neg\tau_\psi\} \cap \mathcal{A}_\pi(i_h, y_h) \neq \emptyset\}$ is finite and derive a contradiction. Then there is $h \geq 0$ such that for all $j \geq h$, $\{\psi, \neg\psi_2, \tau_\psi\} \subseteq \mathcal{A}_\pi(i_j, y_j)$ or, equivalently, $(i_j, y_j) \models_\pi \psi \wedge \neg\psi_2$ and $\tau_\psi \in \mathcal{A}_\pi(i_j, y_j)$. By Condition 2 in the definition of $\mathcal{A}_\pi$, it follows that for each $j > h$, $I_\psi(i_{j+1}, y_{j+1}) = I_\psi(i_j, y_j) - 1$. Therefore, there is $j > h$ such that $I_\psi(i_j, y_j) = 0$, hence $(i_j, y_j) \models_\pi \psi_2 \vee \neg\psi$, which is a contradiction. □

Proposition 4. *Let $r = (j_0, y_0)(j_1, y_1) \ldots$ be a linear path of π. Then,*

1. *if $y_h \succeq y_0$ for all $0 \leq h < |r|$, then $\tau_\pi(j_0, y_0) = inf$,*
2. *if r is infinite, then the set $\{h \in \mathbb{N} \mid \tau_\pi(j_h, y_h) = inf\}$ is infinite.*

Proof. First, we prove Property 1. Assume that Property 1 does not hold. By definition of τ_π, the abstract path r_a from (j_0, y_0) leads a local state (i, y_0) such that $D_i(y_0) \in Ex$ and $next^\ell_\pi(i, y_0) \neq \emptyset$. Since there are not transitions outgoing from exit nodes, we deduce that $next^\ell_\pi(i, y_0)$ corresponds to a return. Since r_a is a subsequence of r, there is $0 \leq h < |r|$ such that $j_h = i$ and $y_h = y_0$. Moreover, $next^\ell_\pi(i, y_0) = \{(j_{h+1}, y_{h+1})\}$. Hence, $y_0 \succ y_{h+1}$, which is a contradiction. Therefore, Property 1 holds. Now, let us consider Property 2. Assume on the contrary that the set $\{h \in \mathbb{N} \mid \tau_\pi(j_h, y_h) = inf\}$ is finite. Then, there is $m \geq 0$ such that for all $k \geq m$, $\tau_\pi(j_k, y_k) = fin$. Let $p \geq m$ such that $y_i \succeq y_p$ for all $i \geq p$ (note that such y_p exists). We have that $\tau_\pi(i_p, y_p) = fin$. On the other hand, by Property 1, $\tau_\pi(i_p, y_p) = inf$, which is a contradiction. □

Proposition 5. *Let $(i, y) \in Q_\pi$ such that $D_i(y) \in Calls$ and $next^\ell_\pi(i, y) = \{(j, y \cdot 0), \ldots, (j, y \cdot m)\}$. Then, $next^a_\pi(i, y) = \perp$ iff there is $0 \leq h \leq m$ such that $\tau_\pi(j, y \cdot h) = inf$.*

Proof. Let $\tau_\pi(j, y \cdot h) = inf$ for some $0 \leq h \leq m$. We show that $next^a_\pi(i, y) = \perp$. Assume on the contrary that $next^a_\pi(i, y) \neq \perp$. By hypothesis there is a linear path of π of the form $r = (k_0, z_0)(k_1, z_1) \ldots$ such that $(k_0, z_0) = (i, y)$ and $(k_1, z_1) = (j, y \cdot h)$. Since $next^a_\pi(i, y) \neq \perp$, there exists $p > 1$ such that $(k_p, z_p) = next^a_\pi(k_0, z_0)$. Since $z_1 = y \cdot h \succ y = z_0 = z_p$, it easily follows that there is $1 < q < p$ such that $D_{k_q}(z_q) \in Ex$ and the abstract path of π starting from (k_1, z_1) leads to (k_q, z_q). Since $next^\ell_\pi(k_q, z_q) \neq \emptyset$, by definition of τ_π, it holds that $\tau_\pi(k_1, z_1) = \tau_\pi(k_q, z_q) = fin$, which is a contradiction.

Now, assume that $next^a_\pi(i, y) = \perp$. Let $r = (k_0, z_0)(k_1, z_1) \ldots$ be a linear path of π starting from (i, y) (i.e., $(k_0, z_0) = (i, y)$). Evidently, $(k_1, z_1) = (j, y \cdot h)$ for some $0 \leq h \leq m$. Since $next^a_\pi(i, y) = \perp$, it easily follows that $z_p \succeq z_1$ for all $1 \leq p < |r|$. Therefore, by Proposition 4(1) we obtain that $\tau_\pi(j, y \cdot h) = \tau_\pi(k_1, z_1) = inf$. This concludes the proof. □

Now, we can prove Lemma 7.

Lemma 7. *Let π be a run of $\mathcal{S}$ starting from $\langle v \rangle$ with $v \in start$ and satisfying φ. Then, there is an accepting run of $\mathcal{S}_\varphi$ starting from a state $\langle v' \rangle$ with $v' \in start'$ and $Pr(v') = v$.*

Proof. Let $\pi = q_0 \xrightarrow{(\ell_0,\ell'_0)} q_1 \xrightarrow{(\ell_1,\ell'_1)} q_2 \ldots$, where $q_i = (t_i, D_i)$ for each $i \geq 0$. First, we prove the following claim.
Claim. For each $i \geq 0$, there is a global state $q'_i = (t_i, D'_i)$ of $\mathcal{S}_\varphi$ such that

1. $Pr(q'_i) = q_i$ (in particular, $q'_0 = \langle v' \rangle$ with $v' \in start'$),
2. for each $y \in leaves(t_i)$, $Atom(D'_i(y)) = \mathcal{A}_\pi(i, y)$ and $Tag(D'_i(y)) = \tau_\pi(i, y)$,
3. for each $y \in leaves(t_i)$ such that $D'_i(y)$ is a call or a return vertex, $D'_i(y)$ is well defined,
4. if $i > 0$, then $q'_{i-1} \xrightarrow{(\ell_{i-1},\ell'_{i-1})} q'_i$ is an edge of the labeled transition system induced by $\mathcal{S}_\varphi$.

Proof of the claim. The proof is by induction on i. We define $q'_0 = (\{\varepsilon\}, D'_0)$ (note that $t_0 = \{\varepsilon\}$ and $D_0(\varepsilon) \in start$), where $D'_0(\varepsilon) = (D_0(\varepsilon), \mathcal{A}_\pi(0, \varepsilon), \tau_\pi(0, \varepsilon))$. By Proposition 4(1) $\tau_\pi(0, \varepsilon) = inf$. Since $siblings(t_0, \varepsilon) = \{\varepsilon\}$ and $next^-_\pi(0, \varepsilon) = \perp$, by definition of the mapping $\mathcal{A}_\pi$, it follows that $D'_0(\varepsilon) \in start'$. Now, assume that $i > 0$. We define $q'_i = (t_i, D'_i)$ as follows. There are three cases:

- **Internal move**: $t_i = t_{i-1}$, $l_{i-1} = l'_{i-1} = \{y_1, \ldots, y_m\} \subseteq leaves(t_i)$, for each $y \in t_i \setminus \ell_{i-1}$, $D_i(y) = D_{i-1}(y)$, and for each $1 \leq j \leq m$, $D_i(y_j) \in \delta(D_{i-1}(y_j), \sigma)$ for some $\sigma \in \Sigma$. Note that $D_i(y_j) \in Calls \cup N$ and $D_{i-1}(y_j) \in N \cup Retns$. By the induction hypothesis, $Pr(q'_{i-1}) = q_{i-1}$. Moreover, if $D_{i-1}(y_j) \in Retns$, then $D'_{i-1}(y_j)$ is well defined. Then, D'_i is defined as follows: for each $y \in t_i \setminus \ell_{i-1}$, $D'_i(y) = D'_{i-1}(y)$, and for each $1 \leq j \leq m$,
  - if $D_i(y_j) \in N$, then $D'_i(y_j) = (D_i(y_j), \mathcal{A}_\pi(i, y_j), \tau_\pi(i, y_j))$
  - if $D_i(y_j) = (b, e_1, \ldots, e_p) \in Calls$, then we proceed as follows. If $next^\ell_\pi(i, y_j) = \emptyset$, then $D'_i(y_j)$ is some call vertex v such that v is well defined, $Pr(v) = D_i(y_j)$, $Atom(v) = \mathcal{A}_\pi(i, y_j)$, and $Tag(v) = \tau_\pi(i, y_j)$ (note that such a vertex exists). Otherwise, we have that $next^\ell_\pi(i, y_j) = \{(h, z_1), \ldots, (h, z_p)\}$ for some $h > i$ such that $\ell'_{h-1} = \{z_1, \ldots, z_p\}$ and $D_h(z_l) = e_l$ for each $1 \leq l \leq p$. In this case define $D'_i(y_j) = ((b, \mathcal{A}_\pi(i, y_j), R, \tau_\pi(i, y_j)), (e_1, A_1, tag_1), \ldots, (e_p, A_p, tag_p))$, where for each $1 \leq l \leq p$, $A_l = \mathcal{A}_\pi(h, z_l)$ and $tag_l = \tau_\pi(h, z_l)$. Moreover, $R = \mathcal{A}_\pi(k, y_j)$ if $next^a_\pi(i, y_j) = (k, y_j)$, and R is some arbitrary atom of φ otherwise. Note that by Proposition 5 and the induction hypothesis $D'_i(y_j)$ is well defined.

  Evidently, Properties 1–4 directly follows from the induction hypothesis and the definitions of the mappings $\mathcal{A}_\pi$ and τ_π.
- **Module call**: $\ell_{i-1} = \{z\}$, $\ell'_{i-1} = \{z \cdot 0, \ldots, z \cdot n\} \subseteq leaves(t_i)$, $t_i = t_{i-1} \cup l'_{i-1}$, for all $y \in t_{i-1} \setminus \{z\}$, $D_i(y) = D_{i-1}(y)$, $D_{i-1}(z) = (b, e_0, \ldots, e_n) \in Calls$, $D_i(z) = b$, and, for each $0 \leq j \leq n$, $D_i(z \cdot j) = e_j$. By the induction hypothesis $q_{i-1} = Pr(q'_{i-1})$. Therefore, $D'_{i-1}(z) = ((b, A, R, tag), (e_0, A_0, tag_0), \ldots, (e_n, A_n, tag_n))$. Moreover, by construction (see the "internal move" case),[4] for each $0 \leq j \leq n$, $A_j = \mathcal{A}_\pi(i, z \cdot j)$ and $tag_j = \tau_\pi(i, z \cdot j)$. D'_i is defined as follows: for all $y \in t_{i-1} \setminus \{z\}$, $D'_i(y) = D'_{i-1}(y)$, $D'_i(z) = (b, A, R, tag)$, and, for each $0 \leq j \leq n$, $D'_i(z \cdot j) = (e_j, A_j, tag_j)$. Evidently, Properties 1–4 hold.
- **Return from a call**: $\ell'_{i-1} = \{z\}$, $\ell_{i-1} = \{z \cdot 0, \ldots, z \cdot n\} \subseteq leaves(t_{i-1})$, $t_i = t_{i-1} \setminus \ell_{i-1}$, for all $y \in t_i \setminus \{z\}$, $D_i(y) = D_{i-1}(y)$, $D_i(z) = (b, x_0, \ldots, x_n) \in Retns$, $D_{i-1}(z) = b$, and for each $0 \leq j \leq n$, $D_{i-1}(z \cdot j) = x_j$. By the induction hypothesis, $D'_{i-1}(z) = (b, A, R, tag)$, and for each $0 \leq j \leq n$, $D'_{i-1}(z \cdot j) = (x_j, A_j, tag_j)$ with $A_j = \mathcal{A}_\pi(i - 1, z \cdot j)$ and $tag_j = \tau_\pi(i - 1, z \cdot j)$. In particular $tag_j = fin$. Let h be the greatest index $m < i - 1$ such that $D_m(z)$ is a call vertex (i.e., $D_h(z)$ is the call matching the return $D_i(z)$). By construction (see the "internal move" case and the "module call" case), $A = \mathcal{A}_\pi(h, z)$, $tag = \tau_\pi(h, z) (= \tau_\pi(i, z))$, and $R = \mathcal{A}_\pi(i, z)$. Therefore, D'_i is defined as follows: $D'_i(z) = ((b, A, R, tag), (x_0, A_0, tag_0), \ldots, (x_n, A_n, tag_n))$, and for all $y \in t_i \setminus \{z\}$, $D'_i(y) = D'_{i-1}(y)$. By the above observations, it follows that Properties 1–4 hold.

This concludes the proof of the claim. □

By the claim above it follows that $\pi' = q'_0 \xrightarrow{(\ell_0,\ell'_0)} q'_1 \xrightarrow{(\ell_1,\ell'_1)} q'_2 \ldots$ is a run of $\mathcal{S}_\varphi$ such that $q'_0 = \langle v' \rangle$ with $v' \in start'$ and for all $i \geq 0$, $q'_i = (t_i, D'_i)$. Thus, it remains to prove that π' is accepting. We focus on the generalized Büchi acceptance condition $\mathcal{F}$ of $\mathcal{S}_\varphi$ (the fulfillment of the terminal acceptance condition $\mathcal{P}_F$ and the synchronized acceptance condition $\mathcal{P}_{sync}$ easily follows from the definitions of $\mathcal{P}_F$ and $\mathcal{P}_{sync}$). By definition of $\mathcal{F}$, we have to show that given an infinite linear path $r = (j_0, y_0)(j_1, y_1) \ldots$ of π', the following holds:

a. The set $\{h \in \mathbb{N} \mid Tag(D'_{j_h}(y_h)) = inf\}$ is infinite.
b. For each formula $\psi = \psi_1 \mathsf{U}^\exists \psi_2 \in Cl(\varphi)$, the following set is infinite
$$\{h \in \mathbb{N} \mid \{\neg(\psi_1 \mathsf{U}^\exists \psi_2), \psi_2, \neg\tau_\psi\} \cap Atom(D'_{j_h}(y_h)) \neq \emptyset\}.$$
c. For each formula $\psi_1 \mathsf{U}^\forall \psi_2 \in Cl(\varphi)$, the following set is infinite
$$\{h \in \mathbb{N} \mid \{\neg(\psi_1 \mathsf{U}^\forall \psi_2), \psi_2\} \cap Atom(D'_{j_h}(y_h)) \neq \emptyset\}.$$
d. For each formula $\psi_1 \mathsf{U}^a \psi_2 \in Cl(\varphi)$, the following set is infinite
$$\{h \in \mathbb{N} \mid \{\neg(\psi_1 \mathsf{U}^a \psi_2), \psi_2\} \cap Atom(D'_{j_h}(y_h)) \neq \emptyset \quad \text{and} Tag(D'_{j_h}(y_h)) = inf\}.$$

[4] Note that $D_{i-1}(z)$ belongs to the target of an internal move since $D_0(\varepsilon)$ is a node.

Note that r is an infinite linear path of π starting from (j_0, y_0). Property **a** directly follows from Proposition 4(2) and Property 2 of the claim above, while Property **b** directly follows from Proposition 3 and Property 2 of the claim above. Now, we prove Property **d** (the proof of Property **c** is easier).

Proof of Property ***d***. Assume that the set $\{h \in \mathbb{N} \mid \neg(\psi_1 \mathsf{U}^a \psi_2) \in Atom(D'_{j_h}(y_h)) \text{ and } Tag(D'_{j_h}(y_h)) = inf\}$ is finite (if this set is infinite, then Property **d** trivially holds). Then, since $Atom(D'_{j_h}(y_h)) = \mathcal{A}_\pi(j_h, y_h)$ and $Tag(D'_{j_h}(y_h)) = \tau_\pi(j_h, y_h)$ for all $h \geq 0$, it suffices to show that for each $k \geq 0$, there is $p \geq k$ such that $\psi_2 \in \mathcal{A}_\pi(j_p, y_p)$ and $\tau_\pi(j_p, y_p) = inf$. By Property **a** and the definition of atom it follows that the set $\{h \in \mathbb{N} \mid \psi_1 \mathsf{U}^a \psi_2 \in Atom(D'_{j_h}(y_h)) \text{ and } Tag(D'_{j_h}(y_h)) = inf\}$ is infinite. Let $k \geq 0$. Then, there is $m \geq k$ such that $\psi_1 \mathsf{U}^a \psi_2 \in Atom(D'_{j_m}(y_m)) = \mathcal{A}_\pi(j_m, y_m)$ and $Tag(D'_{j_m}(y_m)) = \tau_\pi(j_m, y_m) = inf$. Let $r^a = (l_0, y_m)(l_1, y_m)\ldots$ be the abstract path of π starting from (j_m, y_m) (where $l_0 = j_m$). Since $\tau_\pi(l_0, y_m) = inf$, by the definition of τ_π we deduce that $\tau(l_i, y_m) = inf$ for all $i < |r^a|$. Since $\psi_1 \mathsf{U}^a \psi_2 \in \mathcal{A}_\pi(l_0, y_m)$, it follows that $(l_0, y_m) \models_\pi \psi_1 \mathsf{U}^a \psi_2$. Therefore, there is $s \geq 0$ such that $(l_s, y_m) \models_\pi \psi_2$. In particular, $\psi_2 \in \mathcal{A}_\pi(l_s, y_m)$ and $\tau_\pi(l_s, y_m) = inf$. Since r^a is a subsequence of the suffix of r given by $(j_m, y_m)\,(j_{m+1}, y_{m+1})\ldots$, there is $p \geq m \geq k$ such that $j_p = l_s$ and $y_p = y_m$. Thus, Property **d** holds.

This concludes the proof of Lemma 7. □

References

[1] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, M. Yannakakis, Analysis of recursive state machines, ACM Transactions on Programming Languages and Systems 27 (4) (2005) 786–818.
[2] R. Alur, S. Chaudhuri, K. Etessami, P. Madhusudan, On-the-fly reachability and cycle detection for recursive state machines, in: Proc. 11th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems, TACAS'05, in: LNCS, vol. 3440, Springer, 2005, pp. 61–76.
[3] R. Alur, K. Etessami, P. Madhusudan, A temporal logic of nested calls and returns, in: Proc. 10th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems, TACAS'04, in: LNCS, vol. 2988, Springer, 2004, pp. 467–481.
[4] R. Alur, S. Kannan, M. Yannakakis, Communicating hierarchical state machines, in: Proc. of 26th International Colloquium on Automata, Languages and Programming, ICALP'99, in: LNCS, vol. 1644, Springer, 1999, pp. 169–178.
[5] J.C.M. Baeten, W.P. Weijland, Process Algebra, in: Cambridge Tracts in Theoretical Computer Science, vol. 18, Cambridge Univ. Press, 1990.
[6] G. Booch, L. Jacobson, J. Rumbaugh, Unifying Modeling Language User Guide, Addison-Wesley, Boston, 1997.
[7] A. Bouajjani, P. Habermehl, Constrained properties, semilinear systems, and Petri nets, in: Proc. 7th International Conference on Concurrency Theory, CONCUR'96, in: LNCS, vol. 1119, Springer, 1996, pp. 481–497.
[8] A. Bouajjani, J. Esparza, T. Touili, A generic approach to the static analysis of concurrent programs with procedures, in: Proc. 30rd ACM Symp. on Principles of Programming Languages, POPL'03, 2003, pp. 62–73.
[9] A. Bouajjani, T. Touili, Reachability analysis of process rewrite systems, in: Proc. 23th Conf. Foundations of Software Technology and Theoretical Computer Science, FSTTCS'03, in: LNCS, vol. 2914, Springer, 2003, pp. 74–87.
[10] A. Bouajjani, J. Esparza, T. Touili, Reachability analysis of synchronized PA-systems, in: Proc. of Infinity'04, 2004.
[11] A. Bouajjani, M. Müller-Olm, T. Touili, Regular symbolic analysis of dynamic networks of pushdown systems, in: Proc. 16th Int. Conf. Concurrency Theory, CONCUR'05, in: LNCS, vol. 3653, Springer, 2005, pp. 473–487.
[12] W. Brainerd, Tree generating regular systems, Information and Control 14 (1969) 217–231.
[13] J. Coquide, R. Gilleron, Proofs and reachability problem for ground rewrite systems, in: Proc. of 6th International Meeting of Young Computer Scientists, IMYCS'90, in: LNCS, vol. 464, Springer, 1990, pp. 120–129.
[14] M. Dauchet, T. Heuillard, P. Lescanne, S. Tison, Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems, Information and Computation 88 (2) (1990) 187–201.
[15] M. Dauchet, S. Tison, The theory of ground rewrite systems is decidable, in: Proc. 5th Ann. Symp. Logic in Computer Science, LICS'90, IEEE Computer Society Press, 1990, pp. 242–248.
[16] J. Esparza, M. Nielsen, Decidability issues for Petri nets, Journal Information Processing and Cybernetics 30 (3) (1994) 143–160.
[17] J. Esparza, J. Knoop, An automata-theoretic approach to interprocedural parallel flow graphs, in: Proc. 2nd Int. Conf. Foundations of Software Science and Computation Structures, FoSSaCS'99, in: LNCS, vol. 1578, Springer, 1999.
[18] J. Esparza, A. Podelski, Efficient algorithms for pre* and post* on interprocedural parallel flow graphs, in: Proc. 27th ACM Symp. on Principles of Programming Languages, POPL'00, 2000, pp. 1–11.
[19] J. Esparza, A. Kucera, S.S. Schwoon, Model-checking LTL with regular valuations for pushdown systems, Information and Computation 186 (2) (2003) 355–376.
[20] T. Jensen, D. Le Metayer, T. Thorn, Verification of control flow based security properties, in: Proc. of the IEEE Symposium on Security and Privacy, 1999. pp. 89–103.
[21] D. Harel, Statecharts: A visual formalism for complex systems, Science of Computer Programming 8 (1987) 231–274.
[22] V. Kahlon, A. Gupta, An Automata theoretic approach for model checking threads for LTL properties, in: Proc. 21th IEEE Symposium on Logic in Computer Science, LICS'06, 2006, pp. 101–110.
[23] V. Kahlon, A. Gupta, On the analysis of interacting pushdown systems, in: Proc. 34th Symposium on Principles of Programming Languages, POPL'07, 2007, pp. 303–314.
[24] C. Löding, Infinite graphs generated by tree rewriting, Doctoral Thesis, RWTH Aachen, 2003.
[25] S. La Torre, P. Madhusudan, G. Parlato, A robust class of context-sensitive languages, in: Proc. 21th IEEE Symposium on Logic in Computer Science, LICS'07, 2007, pp. 161–170.
[26] S. La Torre, P. Madhusudan, G. Parlato, Context-bounded analysis of concurrent queue systems, in: Proc. of the 14th International Conference on Tools and Algorithms for the Construction and the Analysis of Systems, TACAS'08, in: LNCS, vol. 4963, Springer, 2008, pp. 299–314.
[27] R. Mayr, Decidability and complexity of model checking problems for infinite-state systems, PhD. Thesis, Techn. Univ. of Munich, 1998.
[28] R. Mayr, Process rewrite systems, Information and Computation 156 (2000) 264–286.
[29] Marvin L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, 1967.
[30] A. Pnueli, The temporal logic of programs, in: Proc. 18th IEEE Symposium on the Foundations of Computer Science, 1977, pp. 46–57.
[31] S. Qadeer, J. Rehof, Context-bounded model checking of concurrent software, in: Proc. 11th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems, TACAS'05, in: LNCS, vol. 3440, Springer, 2005, pp. 93–107.
[32] G. Ramalingam, Context sensitive synchronization-sensitive analysis is undecidable, ACM Transactions on Programming Languages and Systems 22 (2000) 416–430.
[33] W. Thomas, Automata on infinite objects, in: J. Van Leeuwen (Ed.), in: Handbook of Theoretical Computer Science, vol. B, 1990, pp. 133–191.
[34] S. Tison, Fair termination is decidable for ground systems, in: Proc. of the 3rd International Conference on Rewriting Techniques and Applications, RTA'89, in: LNCS, vol. 355, Springer, 1989, pp. 462–476.
[35] M.Y. Vardi, P. Wolper, Automata-theoretic techniques for modal logics of programs, Journal of Computer and System Science 32 (2) (1986) 183–221.