Liveness of Randomised Parameterised Systems under Arbitrary Schedulers

Anthony W. Lin[1] and Philipp Rümmer[2]

[1] Yale-NUS College, Singapore
[2] Uppsala University, Sweden

Abstract. We consider the problem of verifying liveness for systems with a finite, but unbounded, number of processes, commonly known as *parameterised systems*. Typical examples of such systems include distributed protocols (e.g. for the dining philosopher problem). Unlike the case of verifying safety, proving liveness is still considered extremely challenging, especially in the presence of randomness in the system. In this paper we consider liveness under arbitrary (including unfair) schedulers, which is often considered a desirable property in the literature of self-stabilising systems. We introduce an automatic method of proving liveness for randomised parameterised systems under arbitrary schedulers. Viewing liveness as a two-player reachability game (between Scheduler and Process), our method is a CEGAR approach that synthesises a progress relation for Process that can be symbolically represented as a finite-state automaton. The method is incremental and exploits both Angluin-style L*-learning and SAT-solvers. Our experiments show that our algorithm is able to prove liveness automatically for well-known randomised distributed protocols, including Lehmann-Rabin Randomised Dining Philosopher Protocol and randomised self-stabilising protocols (such as the Israeli-Jalfon Protocol). To the best of our knowledge, this is the first fully-automatic method that can prove liveness for randomised protocols.

1 Introduction

Verification of parameterised systems is one of the most extensively studied problems in computer-aided verification. Parameterised systems are infinite families of finite-state systems that are described in some finite behavioral description language. Distributed protocols (e.g. for the dining philosopher problem) are typical examples of parameterised systems since they can represent any finite (but unbounded) number of processes. Verifying a parameterised system, then, amounts to verifying *every instance* of the infinite family. In the case of a dining philosopher protocol, this amounts to verifying the protocol with any number of philosophers. Although the problem was long known to be undecidable [11], a lot of progress has been made to tackle the problem resulting in such techniques as network invariants (including cutoff techniques), symbolic model checking (including regular model checking), and finite-range abstractions, to name a few. The reader is referred to the following excellent surveys [2, 7, 17, 77, 79] covering these different approaches to solving the problem.

Nowadays there are highly effective automatic methods that can successfully verify *safety* for many parameterised systems derived from real-world concurrent/distributed algorithms (e.g. see [2–7, 10, 13, 20–23, 36, 43, 44, 49, 54, 56, 63, 75, 77, 78]). In contrast, there has been much less progress in automatic techniques for proving *liveness* for parameterised systems. In fact, this difficulty has also been widely observed (e.g. see [8, 48, 67, 77]). Proving liveness amounts to proving that, under a class of adversarial schedulers (a.k.a. *adversaries* or just *schedulers*), something "good" will eventually happen. The problem is known to be reducible to finding an infinite path satisfying a Büchi condition (e.g. see [7, 24, 65, 67, 72–74, 77]). The latter problem (a.k.a. repeated reachability) in general requires reasoning about the transitive closure relations, which are generally observed to be rather difficult to compute automatically.

Randomised parameterised systems are infinite families of finite-state systems that allow both nondeterministic and probabilistic transitions (a.k.a. Markov Decision Processes [52]). This paper concerns the problem of verifying liveness for randomised parameterised systems, with an eye towards a fully-automatic verification algorithm for well-known *randomised distributed protocols* that commonly feature in finite-state probabilistic model checkers (e.g. PRISM [51]), but have so far resisted fully-automatic parameterised verification. Such protocols include Lehmann-Rabin's Randomised Dining Philosopher Protocol [55] and randomised self-stabilising protocols (e.g. Israeli-Jalfon's Protocol [47] and Herman's Protocol [46]), to name a few. Randomised protocols generalise deterministic protocols by allowing each process to make probabilistic transitions, i.e., not just a transition with probability 1. Randomisation is well-known to be useful in the design of distributed protocols, e.g., to break symmetry and simplifies distributed algorithms (e.g. see [39, 58]). Despite the benefits of randomisation in protocol design, the use of randomisation makes proving liveness substantially more challenging (e.g. see [58, 59, 69]). Proving liveness for probabilistic distributed protocols amounts to proving that, under a class of adversaries, something "good" will eventually happen *with probability 1* (e.g. see [12, 29, 35, 52, 53, 58, 76]). Unlike the case of deterministic protocols, proving liveness for probabilistic protocols requires reasoning about *games* between an adversary and a stochastic process player (a.k.a. $1\frac{1}{2}$-player game), which makes the problem computationally more difficult even in the finite-state case (e.g. see [53]). To the best of our knowledge, there is presently no fully-automatic technique which can prove liveness for such randomised distributed protocols as Lehmann-Rabin's Randomised Distributed Protocols [55], and self-stabilising randomised protocols including Israeli-Jalfon's Protocol [47] and Herman's Protocol [46].

Contribution: The main contribution of the paper is a fully-automatic method for proving liveness over randomised parameterised systems over various network topologies (e.g. lines, rings, stars, and cliques) under arbitrary (including unfair) schedulers. Liveness under arbitrary schedulers is a desirable property in the literature of self-stabilising algorithms since an unfair scheduler (a.k.a. daemon) enables a *worst-case* analysis of an algorithm and covers the situation when some process is "frozen" due to conditions that are external to the process (e.g. see [14, 33, 41, 50]). There are numerous examples of self-stabilising protocols that satisfy liveness even under unfair schedulers (e.g. see [14, 31, 39, 47, 50]). Similar examples are also available in the literature of mutual exclusion protocols (e.g. [34, 70]), and consensus/broadcast protocols (e.g. [25, 39]). Our algorithm can successfully verify liveness under arbitrary schedulers for a fragment of FireWire's symmetry breaking protocol [35, 60], Israeli-Jalfon's Protocol [47], Herman's Protocol [46] considered over a linear array, and Lehmann-Rabin Dining Philosopher Protocol [34, 55].

It is well-known that for proving liveness for a finite-state Markov Decision Process (MDP) only the topology of the system matters, not the actual probability values (e.g. see [29, 30, 45, 76]). Hence, the same is true for randomised parameterised systems since each instance is a finite MDP. In this paper, we follow this approach and view the problem of proving liveness under arbitrary schedulers as a 2-player reachability game between Scheduler (Player 1) and Process (Player 2) over non-stochastic parameterised systems, obtained by simply ignoring the actual probability values of transitions with non-zero probabilities (transitions with zero probability are removed). This simple reduction allows us to adopt any symbolic representation of non-stochastic parameterised systems. In this paper, we represent parameterised systems as finite-state letter-to-letter transducers, as is standard in *regular model checking* [2, 7, 23, 65, 77]. In this framework, configurations of parameterised systems are represented as words over a finite alphabet Σ (usually encoding a finite set of control states for each local process). Many distributed protocols that arise in practice can be naturally modelled as transducers.

To automatically verify liveness of parameterised systems in this representation, we develop a counterexample-guided method for synthesising Player 2 strategies. The core step of the approach is the computation of *well-founded relations* guiding Player 2 towards winning configurations (and the system towards “good” states). In the spirit of regular model checking, such well-founded relations are represented as letter-to-letter transducers; however, unlike most regular model checking algorithms, we use learning and SAT-based methods to compute the relations, in line with some of the recent research on the application of learning for program analysis (e.g. [40, 62–64]). This gives rise to a counterexample-guided algorithm for computing winning strategies for Player 2. We then introduce a number of refinements of the base method, which turn out to be essential for analysing challenging systems like the Lehmann-Rabin protocol: strategies for Player 2 can be constructed *incrementally,* reducing the size of automata that have to be considered in each inference step; symmetries of games (e.g., rotation symmetry in case of protocols with ring topology) can be exploited for acceleration; and inductive over-approximations of the set of reachable configurations can be precomputed with the help of learning. To the best of our knowledge, the last refinement also represents the first successful application of Angluin’s L*-algorithm [9] for learning DFAs representing inductive invariants in the regular model checking context.

We have implemented our method as a proof of concept. Besides the four aforementioned probabilistic protocols that we have successfully verified against liveness (under all schedulers), we also show that our tool is competitive with existing tools (e.g. [8, 65]) for proving liveness for deterministic parameterised systems (Szymanski’s mutual exclusion protocol [70], Left-Right Dining Philosopher Protocol [58], Lamport’s Bakery Algorithm [15, 39], and Resource-Allocator Protocol [32]). Finally, we report that our tool can also automatically solve classic examples from combinatorial game theory on infinite graphs (take-away game and Nim [38]). To the best of our knowledge, our tool is the first verification tool that can automatically solve these games.

Related Work: There are currently only a handful of fully-automatic techniques for proving liveness for randomised parameterised systems. We mention the works [27, 35, 61] on proving almost-sure termination of sequential probabilistic programs. Strictly speaking, these works are not directly comparable to our work since their tools/techniques handle only programs with variables over integer/real domains, and cannot naturally model the protocol examples over line/ring topology that we consider in this paper. Based on the work of Arons *et al.* [12], the approach of Esparza *et al.* [35] aims to guess a terminating pattern by constructing a nondeterministic program from a given probabilistic program and a terminating pattern candidate. This allows them to exploit model checkers and termination provers for nondeterministic programs. The approach is sound and complete for "weakly-finite" programs, which include parameterised programs, i.e., programs with parameters that can be initialised to arbitrary large values, but are finite-state for every valuation of the parameters. The approach of [27] is a constraint-based method to synthesise ranking functions for probabilistic programs based on martingales and may be able to prove almost sure termination for probabilistic programs that are not weakly finite. Monniaux [61] proposed a method for proving almost sure termination for probabilistic programs using abstract interpretation, though without tool support.

As previously mentioned, there is a lot of work on liveness for non-probabilistic parameterised systems (e.g. see [8, 24, 37, 65, 67, 68, 72–74]). We assess our technique in this context by using several typical benchmarking examples that satisfy liveness (more precisely, deadlock-freedom) under arbitrary schedulers including Szymanski's Protocol, Bakery Protocol, and Deterministic Dining Philosopher with Left-Right Strategy.

Two-player reachability games on automatic graphs (i.e. regular model checking with non-length preserving transducers) have been considered by Neider [62], who proposed an L*-based learning algorithm for constructing the set of winning regions enriched with "distance" information, which is a number that can be represented in binary or unary. [Embedding distance information in a reachability set was first done in regular model checking by Vardhan *et al.* [75]] Augmenting winning regions or reachability sets with distance information, however, often makes regular sets no longer regular [63]. In this paper, we do not consider non-length preserving transducers and our algorithm is based on constructing progress relations for Player 2. In particular, part of our algorithm employs an L*-based algorithm for synthesising an inductive invariant which, however, differs from [62, 75] since membership tests (i.e. reachability of a single configuration) are decidable. Recently Neider and Topcu [64] proposed a learning algorithm for solving safety games over rational graphs (an extension of automatic graphs), which are *dual* to reachability games.

2 Preliminaries

General notations: For any two given real numbers $i \leq j$, we use a standard notation (with an extra subscript) to denote real intervals, e.g., $[i, j]_{\mathbb{R}} = \{k \in \mathbb{R} : i \leq k \leq j\}$ and $(i, j]\{k \in \mathbb{R} : i < k \leq j\}$. We will denote intervals over integers by removing the subscript, e.g., $[i, j] := [i, j]_{\mathbb{R}} \cap \mathbb{Z}$. Given a set S, we use S^* to denote the set of all finite sequences of elements from S. The set S^* always includes the empty sequence which we denote by ϵ. Given two sets of words S_1, S_2, we use $S_1 \cdot S_2$ to denote the set $\{v \cdot w : v \in S_1, w \in S_2\}$ of words formed by concatenating words from S_1 with words from S_2. Given two relations $R_1, R_2 \subseteq S \times S$, we define their composition as $R_1 \circ R_2 = \{(s_1, s_3) : (\exists s_2)((s_1, s_2) \in R_1 \wedge (s_2, s_3) \in R_2)\}$.

Transition systems: Let ACT be a finite set of *action symbols*. A *transition system* over ACT is a tuple $\mathfrak{S} = \langle S; \{\to_a\}_{a \in \mathsf{ACT}}, \{U_b\}_{b \in \mathrm{AP}} \rangle$, where S is a set of *configurations*, $\to_a \subseteq S \times S$ is a binary relation over S, and $U_b \subseteq S$ is a unary relation on S. In the sequel, we will often consider transition systems where $\mathrm{AP} = \emptyset$ and $|\mathsf{ACT}| = 1$, in which case $\langle S; \{\to_a\}_{a \in \mathsf{ACT}}, \{U_b\}_{b \in \mathrm{AP}} \rangle$ will be denoted as $\langle S; \to \rangle$. If $|\mathsf{ACT}| > 1$, we use $\to$ to denote the relation $\left(\bigcup_{a \in \mathsf{ACT}} \to_a \right)$. The notation $\to^+$ (resp. $\to^*$) is used to denote the transitive (resp. transitive-reflexive) closure of $\to$. We say that a sequence $s_1 \to \cdots \to s_n$ is a *path* (or *run*) in $\mathfrak{S}$ (or in $\to$). Given two paths $\pi_1 : s_1 \to^* s_2$ and $\pi_2 : s_2 \to^* s_3$ in $\to$, we may concatenate them to obtain $\pi_1 \odot \pi_2$ (by gluing together s_2). We call π_1 a *prefix* of $\pi_1 \odot \pi_2$. For each $S' \subseteq S$, we use the notations $pre_{\to}(S')$ and $post_{\to}(S')$ to denote the pre/post image of S' under $\to$. That is, $pre_{\to}(S') := \{p \in S : \exists q \in S'(p \to q)\}$ and $post_{\to}(S') := \{q \in S : \exists p \in S'(p \to q)\}$.

Words and automata: We assume basic familiarity with word automata. Fix a finite alphabet Σ. For each finite word $w = w_1 \dots w_n \in \Sigma^*$, we write $w[i,j]$, where $1 \leq i \leq j \leq n$, to denote the segment $w_i \dots w_j$. Given an automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, a run of $\mathcal{A}$ on w is a function $\rho : \{0, \dots, n\} \to Q$ with $\rho(0) = q_0$ that obeys the transition relation δ. We may also denote the run ρ by the word $\rho(0) \cdots \rho(n)$ over the alphabet Q. The run ρ is said to be *accepting* if $\rho(n) \in F$, in which case we say that the word w is *accepted* by $\mathcal{A}$. The language $L(\mathcal{A})$ of $\mathcal{A}$ is the set of words in Σ^* accepted by $\mathcal{A}$.

Reachability games: We recall some basic concepts on 2-player reachability games (e.g. see [42, Chapter 2] on games with 1-accepting conditions). An *arena* is a transition system $\mathfrak{S} = \langle S; \to_1, \to_2 \rangle$, where S (i.e. the set of "game configurations") is partitioned into two disjoint sets V_1 and V_2 such that $pre_{\to_i}(S) \subseteq V_i$ for each $i = 1, 2$. The transition relation $\to_i$ denotes the actions of Player i. Similarly, for each $i = 1, 2$, the configurations V_i are controlled by Player i. In the sequel, Player 1 will also be called "Scheduler", and Player 2 "Process". Given a set $I_0 \subseteq S$ of initial states and a set $F \subseteq S$ of final (a.k.a. target) states, the goal of Player 2 is to reach F from I_0, while the goal of Player 1 is to avoid it. More formally, a *strategy* for Player i is a partial function $f : S^* V_i \to S$ such that, for each $v \in S^*$ and $p \in V_i$, if vp is a path in $\mathfrak{S}$ and that p is not a dead end (i.e. $p \to_i q$ for some q), then $f(vp)$ is defined in such a way that $p \to_i f(vp)$. Given a strategy f_i for Player $i = 1, 2$ and an initial state $s_0 \in S$, we can define a unique (finite or infinite) path in $\mathfrak{S}$ $\pi : s_0 \to_{j_1} s_1 \to_{j_2} \cdots$ such that $s_{j_{k+1}} = f_i(s_0 s_1 \dots s_{j_k})$ where $i \in \{1, 2\}$ is the (unique) number such that $s_{j_k} \in V_i$. Player 2 *wins* iff some state in F appears in π, or if the path is finite and the last configuration belongs to Player 1. Player 1 *wins* iff Player 2 does not win (i.e. *loses*). A strategy f for Player i is *winning* from I_0, for each strategy g for Player $i + 1 \pmod 2$, the unique path in $\mathfrak{S}$ from each $s_0 \in I_0$ witnesses a win for Player i. Such games (a.k.a. *reachability games*) are *determined* (e.g. see [42, Proposition 2.21]), i.e., either Player 1 has a winning strategy or Player 2 has a winning strategy.

Convention 1 *For simplicity's sake, we make the following assumptions on our reachability games. They suffice for the purpose of proving liveness for parameterised systems. The techniques can be easily adapted when these assumptions are lifted.*

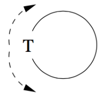
(A0) *Arenas are strictly alternating, i.e., a move made by a player does not take the game back to her configuration (i.e. $post_{\to_i}(S) \cap A_i = \emptyset$, for each $i \in \{1, 2\}$).*

(A1) *Initial and final configurations belong to Player 1, i.e.,* $I_0, F \subseteq V_1$.
(A2) *Non-final configurations are no dead ends, i.e.,* $\forall x \in S \backslash F, \exists y : x \rightarrow_1 y \vee x \rightarrow_2 y$.

3 The formal framework

Parameterised systems are an infinite family $\mathcal{F} = \{\mathfrak{S}_i\}_{i \in \mathbb{N}}$ of finite-state transition systems. Similarly, *randomised parameterised systems* are an infinite family $\mathcal{F} = \{\mathfrak{S}_i\}_{i \in \mathbb{N}}$ of *Markov Decision Processes* [52], which are finite-state transition systems $\mathfrak{S} = \langle S; \rightarrow_1, \rightarrow_2 \rangle$ that have both "nondeterministic" transitions $\rightarrow_1$ and "probabilistic" transitions $\rightarrow_2$.

We first informally illustrate the concept of randomised parameterised systems by means of Israeli-Jalfon Randomised Self-Stabilising Protocol [47] (also see [66]). The protocol has a ring topology and each process either holds a token (denoted by $\top$) or does not hold a token (denoted by $\perp$). At any given step, the Scheduler chooses a process P that holds a token. The process P can then pass the token to its left or right neighbour each with probability 0.5. In doing so, two tokens that are held by a process are merged into one token (held by the same process). It can be proven that under arbitrary schedulers, starting from any configuration with *at least* one token, the protocol will converge to a configuration with *exactly* one token with probability 1. This is an example of liveness under arbitrary schedulers.



It is well-known that the liveness problem for finite MDPs $\mathfrak{S}$ depends on the topology of the graph $\mathfrak{S}$, not on the actual probability values in $\mathfrak{S}$ (e.g. [29, 30, 45, 76]). In fact, this result easily transfers to randomised parameterised systems since *every* instance in the infinite family is a finite MDP. Following this approach, we may view the problem of proving (almost-sure) liveness for randomised parameterised systems under arbitrary schedulers as a 2-player reachability game between Scheduler (Player 1 with moves $\rightarrow_1$) and Process (Player 2 with moves $\rightarrow_2$) over the arena $\mathfrak{S} = \langle S; \rightarrow_1, \rightarrow_2 \rangle$ obtained by simply *ignoring* the actual probability values of transitions in $\rightarrow_2$ (with non-zero probabilities). This simple reduction allows us to view randomised parameterised systems as an *infinite family of finite arenas* and adopt standard symbolic representations of non-stochastic parameterised systems (many of which are known). Our formal framework uses the standard symbolic representation using letter-to-letter transducers. To simplify our presentation, *we will directly define liveness for randomised parameterised systems in terms of non-stochastic two player games and relegate this standard reduction in the full version for interested readers.*

3.1 Liveness as games

Given a randomised parameterised system $\mathcal{F} = \{\mathfrak{S}_i\}_{i \in \mathbb{N}}$, a set $I_0 \subseteq V_1$ of initial states, and a set $F \subseteq V_1$ of final states, we say that a randomised parameterised system *satisfies liveness under arbitrary schedulers with probability 1 (a.k.a. almost surely terminates)* if from *each* configuration $s_0 \in post_{\rightarrow^*}(I_0)$, Player 2 has a winning strategy reaching F in $\mathcal{F}$ (viewed as an arena). The justification of this definition is in the full version.

3.2 Representing infinite arenas

Our formal framework uses the standard symbolic representation of parameterised systems from regular model checking [7, 23, 65, 77], i.e., transducers. Many distributed protocols that arise in practice can be naturally modelled as transducers. *Transducers* are *letter-to-letter automata* that accept k-ary relations over words (cf. [19]). In this paper, we are only interested in binary *length-preserving relations* [7], i.e., a relation $R \subseteq \Sigma^* \times \Sigma^*$ such that each $(v, w) \in R$ implies that $|v| = |w|$. For this reason, we will only define length-preserving transducers and only for the binary case. Given two words $w = w_1 \ldots w_n$ and $w' = w'_1 \ldots w'_n$ over the alphabet Σ, we define a word $w \otimes w'$ over the alphabet $\Sigma \times \Sigma$ as $(w_1, w'_1) \cdots (w_n, w'_n)$. A letter-to-letter transducer is simply an automaton over $\Sigma \times \Sigma$, and a binary relation R over Σ^* is *regular* if the set $\{w \otimes w' : (w, w') \in R\}$ is accepted by a letter-to-letter automaton $\mathcal{R}$. Notice that the resulting relation R only relate words that are of the same length. In the sequel, to avoid notational clutter, we will use R to mean both a transducer and the binary relation that it recognises.

Definition 1 (Automatic systems). *A system $\mathfrak{S} = \langle S; \{\to_a\}_{a \in \mathsf{ACT}}, \{U_b\}_{b \in AP}\rangle$ is said to be automatic if S and U_b (for each $b \in AP$) are regular sets over some non-empty finite alphabet Σ, and each relation $\to_a$ (for each $a \in \mathsf{ACT}$) is given by a transducer over Σ.*

We warn the reader that the most general notion of automatic transition systems [19], which allow non-length preserving transducers, are not needed in this paper. When the meaning is understood, we shall confuse the notation $\to_a$ for the transition relation of $\mathfrak{S}$ and the transducer that recognises it.

Example 1. We shall now model Israeli-Jalfon Protocol as an automatic transition system $\mathfrak{S} = \langle S; \to_1, \to_2\rangle$, where Scheduler's actions are labeled by 1 and Process's actions are labeled by 2. In general, configurations of Israeli-Jalfon protocol are circular structures, but they can easily be turned into a word over a certain finite alphabet by linearising them. More precisely, the domain S of $\mathfrak{S}$ is the set of words over $\Sigma = \{\bot, \top, \hat{\top}\}$ of the form $(\bot + \top)^* \top (\bot + \top)^*$, or $(\bot + \top)^* \hat{\top} (\bot + \top)^*$.

For example, the configuration $\top\bot\top\bot$ denotes the configuration where the 1st and the 3rd (resp. 2nd and 4th) processes are (resp. are not) holding a token. The letter $\hat{\top}$ is used to denote that Scheduler chooses a specific process that holds a token. Note that the intersection of languages generated by these two regular expressions is empty. The transition relation $\to_1$ is given by the regular expression $I^*(\top, \hat{\top})I^*$ where $I := \{(\top, \top), (\bot, \bot)\}$. The transition relation $\top_2$ is given by a union of the following regular expressions:

- $I^*(\hat{\top}, \bot)\,((\bot, \top) + (\top, \top))\,I^*$
- $I^*\,((\bot, \top) + (\top, \top))\,(\hat{\top}, \bot)I^*$
- $((\bot, \top) + (\top, \top))\,I^*(\hat{\top}, \bot)$
- $(\hat{\top}, \bot)I^*\,((\bot, \top) + (\top, \top))$

Note that the right column represents transitions that handle the circular case. Also, note that if $I_0 = (\bot + \top)^* \top (\bot + \top)^*$ and $F = \bot^* \top \bot^*$, Player 2 can always win the game from any reachable configuration (note: $post_{\to^*}(I_0) = I_0$) by simply minimising the distance between the leftmost token and the rightmost token in the configuration.

□

3.3 Algorithm for liveness (an overview)

Our discussion thus far has led to a reformulation of liveness for probabilistic parameterised systems as the following decision problem: given an automatic arena $\mathfrak{G} = \langle S; \rightarrow_1, \rightarrow_2 \rangle$, a regular set $I_0 \subseteq S$ of initial configurations, and a regular set F of final configurations, decide if Player 2 can force the game to reach F in $\mathfrak{G}$ starting from each configuration in $post_{\rightarrow^*}(I_0)$. In the sequel, we will call $\langle \mathfrak{G}, I_0, F \rangle$ a *game instance*. Note that the aforementioned problem is undecidable even when $\rightarrow_2$ is restricted to identity relations, which amounts to the undecidable problem of safety [7]. We will show now that decidability can be retained if "advice bits" are provided in the input.

Advice bits are a pair $\langle A, \prec \rangle$, where $A \subseteq S$ is a set of game configurations and $\prec \subseteq S \times S$ is a binary relation over the game configurations. Intuitively, A is an inductive invariant, whereas $\prec$ is a well-founded relation that guides Player 2 to win. More precisely, the advice bits $\langle A, \prec \rangle$ are said to *conform* to the game instance $\langle \mathfrak{G}, I_0, F \rangle$ if:

(L1) $I_0 \subseteq A$,
(L2) A is $\rightarrow$-inductive, i.e., $\forall x, y : x \in A \wedge (x \rightarrow y) \Rightarrow y \in A$,
(L3) $\prec$ is a *strict preorder*[3] on S,
(L4) Player 2 can progress from A by following $\prec$:

$$\forall x \in A \setminus F, y \in S \setminus F : \ \big((x \rightarrow_1 y) \ \Rightarrow \ (\exists z \in A : \ (y \rightarrow_2 z) \wedge x \succ z)\big) .$$

Conditions **(L1)** and **(L2)** ensure that $post_{\rightarrow^*}(I_0) \subseteq A$, while conditions **(L3)–(L4)** ensure that Player 2 has a winning strategy from each configuration in $post_{\rightarrow^*}(I_0)$. Note that **(L3)** implies well-foundedness of $\prec$, provided that $\prec$ only relates words of the same length (which is always sufficient for advice bits, and will later follow from the use of length-preserving transducers to represent $\prec$).

Theorem 1. *Let $\mathfrak{G} = \langle S; \rightarrow_1, \rightarrow_2 \rangle$ be a $\rightarrow^*$-image-finite arena, i.e., $post_{\rightarrow^*}(s)$ is finite, for each $s \in S$. Given a set $I_0 \subseteq V_1$ of initial configurations, and a set $F \subseteq V_1$ of final configurations, the following are equivalent:*

1. *Player 2 has a winning strategy reaching F in $\mathfrak{G}$ starting from each configuration in $post_{\rightarrow^*}(I_0) \cap V_1$.*
2. *There exist advice bits $\langle A, \prec \rangle$ conforming to the input $\langle \mathfrak{G}, I_0, F \rangle$.*

Advice bits $\langle A, \prec \rangle$ are said to be *regular* if A (resp. $\prec$) is given as a regular set (resp. relation). With the help of regular advice bits, the problem of deciding a winning strategy for Player 2 becomes decidable:

Lemma 1. *Given an automatic arena $\mathfrak{G} = \langle S; \rightarrow_1, \rightarrow_2 \rangle$, a regular set $I_0 \subseteq S$ of initial configurations, a regular set F of final configurations, and regular advice bits $T = \langle A, \prec \rangle$, we can effectively decide whether T conforms to the game instance $\langle \mathfrak{G}, I_0, F \rangle$.*

[3] A binary relation $\prec$ on a set A is said to be a *strict preorder* if it is irreflexive (i.e. for each $s \in A$, $s \not\prec s$) and transitive (for each $s, s', s'' \in A$, $s \prec s'$ and $s' \prec s''$ implies that $s \prec s''$).

Lemma 1 follows from the fact that each of the conditions **(L1)**–**(L4)** is expressible in first-order logic interpreted over the given game instance extended with the advice bits, i.e., the transition systems $\langle S; \{\rightarrow_1, \rightarrow_2, \prec\}, \{I_0, F, A\}\rangle$. Decidability then follows since model checking first-order logic formulas over automatic transition systems is decidable (e.g. see [18, 19] and see [71] for a detailed complexity analysis), the proof of which is done by standard automata methods.

To decide whether Player 2 has a winning strategy for the reachability game, Lemma 1 tells us that one can systematically enumerate all possible regular advice bits and check whether they conform to the input game instance $\langle \mathfrak{S}, I_0, F\rangle$. A naive enumeration would simply go through each $k = 1, 2, \ldots$ and all advice bits $\langle A, \prec\rangle$ where each of the two automata have at most k states. This would be extremely slow.

4 Automatic liveness proofs

We now describe how regular advice bits $\langle A, \prec\rangle$ for (regular) game instances $\langle \mathfrak{S}, I_0, F\rangle$ can be computed automatically, thus proving that Player 2 can win from every reachable configuration, which (as we saw in the previous section) establishes liveness for randomised parameterised systems. We define a constraint-based method that derives $\langle A, \prec\rangle$ as the solution of a set of Boolean formulas representing the conditions **(L1)**–**(L4)** from Section 3.3. Since a full Boolean encoding of **(L1)**–**(L4)** would be exponential in the size of the automata representing the advice bits, our algorithm starts with a relaxed version of **(L1)**–**(L4)** and gradually refines the encoding with the help of counterexamples; in this sense, our approach is an instance of CEGAR [28], and has similarities with recent learning-based methods for computing inductive invariants [63].

Throughout the section we assume that an alphabet Σ and game instance $\langle \mathfrak{S}, I_0, F\rangle$ has been fixed. We will represent the well-founded relation $\prec$ using a transducer $\mathcal{T}_\prec = (\Sigma \times \Sigma, Q_\prec, \delta_\prec, q^0_\prec, F_\prec)$, and the set A as automaton $\mathcal{A}_A = (\Sigma, Q_A, \delta_A, q^0_A, F_A)$. Our overall approach for computing the automata makes use of two main components, which are invoked iteratively within a refinement loop:

SYNTHESISE Candidate automata $(\mathcal{A}_A, \mathcal{T}_\prec)$ with n_A and $n_\prec$ states, respectively, are computed simultaneously with the help of a SAT-solver, enforcing a relaxed set of conditions encoded as a Boolean constraint ψ. The transducer $\mathcal{T}_\prec$ is length-preserving and irreflexive by construction; this implies that the relation $\prec$ is a well-founded preorder iff it is transitive.

VERIFY It is checked whether the automata $(\mathcal{A}_A, \mathcal{T}_\prec)$ satisfy conditions **(L1)**–**(L4)** from Section 3.3. If this is not the case, ψ is strengthened to eliminate counterexamples, and SYNTHESISE is again invoked; otherwise, $(\mathcal{A}_A, \mathcal{T}_\prec)$ represent a winning strategy for Player 2 by Theorem 1.

This refinement loop is enclosed by an outer loop that increments the parameters n_A, and $n_\prec$ (initially set to some small number) when SYNTHESISE determines that no automata satisfying ψ exist anymore. Initially, the formula ψ approximates **(L1)**–**(L4)**, by capturing aspects that can be enforced by a Boolean formula of polynomial size. The next sections described SYNTHESISE and VERIFY in detail.

4.1 VERIFY: checking (L1)–(L4) precisely

Suppose that automata $(\mathcal{A}_A, \mathcal{T}_\prec)$ have been computed. In the VERIFY stage, it is determined whether the automata indeed satisfy the conditions **(L1)–(L4)**, which can effectively be done due to Lemma 1. The check will have one of the following outcomes:

1. $(\mathcal{A}_A, \mathcal{T}_\prec)$ represent correct advice bits.
2. **(L1)** is violated: some word $x \in I_0$ is not accepted by $\mathcal{A}_A$.
3. **(L2)** is violated: there are words $x \in A$ and y with $x \to y$, but $y \notin A$.
4. **(L3)** is violated: $\mathcal{T}_\prec$ does not represent a transitive relation (recall that $\mathcal{T}_\prec$ is length-preserving and irreflexive by construction).
5. **(L4)** is violated: there are words $x \in A \setminus F$ and $y \in S \setminus F$ such that $x \to_1 y$, but no word $z \in A$ exists with $y \to_2 z$ and $x \succ z$.

In cases 2–5, the computed words are counterexamples that are fed back to the SYNTHESISE stage; details for this are given in Sect. 4.3.

The required checks on $(\mathcal{A}_A, \mathcal{T}_\prec)$ can be encoded as validity of first-order formulas, and finally carried out using automata methods (e.g. see [71]). In **(L3)** and **(L4)**, it is in addition necessary to eliminate the quantifier $\exists z$ by means of projection. Note that all free variables in the formulas are implicitly universally quantified.

(L1) $I_0(x) \Rightarrow A(x)$

(L2) $A(x) \wedge (x \to_1 y \vee x \to_2 y) \Rightarrow A(y)$

(L3) $x \prec y \wedge y \prec z \Rightarrow x \prec z$

(L4) $A(x) \wedge \neg F(x) \wedge \neg F(y) \wedge (x \to_1 y) \Rightarrow \exists z.\ \big(A(z) \wedge (y \to_2 z) \wedge x \succ z\big)$

4.2 SYNTHESISE: computation of candidate automata

We now present the Boolean encoding used to search for (deterministic) automata $(\mathcal{A}_A, \mathcal{T}_\prec)$, and to this end make the simplifying assumption that the states of the transducer $\mathcal{T}_\prec$ are $Q_\prec = \{1, \ldots, n_\prec\}$, states of the automaton $\mathcal{A}_A$ are $Q_A = \{1, \ldots, n_A\}$, and that $q^0_\prec = q^0_A = 1$ are the initial states. The following Boolean variables are used to represent automata: a variable $x^\prec_t$ for each tuple $t = (q, a, b, q') \in Q_\prec \times \Sigma \times \Sigma \times Q_\prec$; a variable x^A_t for each tuple $t = (q, a, q') \in Q_A \times \Sigma \times Q_A$; and a variable z^M_q for each $q \in Q_M$ and $M \in \{\prec, A\}$. The assignment $x^M_t = 1$ is interpreted as the existence of the transition t in the automaton for M; likewise, we use $z^M_q = 1$ to represent that q is an accepting state (in DFAs it is in general necessary to have more than one accepting state).

The set of considered automata in step SYNTHESISE is restricted by imposing a number of conditions. Most importantly, only deterministic automata are considered, which is important for refinement: to eliminate counterexamples, it will be necessary to construct Boolean formulas that state *non-acceptance* of certain words, which can only be done succinctly in the case of languages represented by DFAs:

(C1) The automata $\mathcal{A}_A$ and $\mathcal{T}_\prec$ are deterministic.

The second condition encodes irreflexivity of the relation $\prec$:

(C2) Every accepting path in $\mathcal{T}_{\prec}$ contains a label (a, b) with $a \neq b$.

The third group of conditions captures minimality properties: automata that can (obviously) be represented with a smaller number of states are excluded:

(C3) Every state of the automata $\mathcal{A}_A$ and $\mathcal{T}_{\prec}$ is reachable from the initial state.
(C4) From every state in the automata $\mathcal{A}_A$ and $\mathcal{T}_{\prec}$ an accepting state can be reached.

Finally, we can observe that the states of the constructed automata can be reordered almost arbitrarily, which increases the search space that a SAT solver has to cover. The performance of SYNTHESISE can be improved by adding *symmetry breaking* constraints. Symmetries can be removed by asserting that automata states are sorted according to some structural properties extracted from the automaton; suitable properties include whether a state is accepting, or which self-transitions a state has:

(C5) The states $\{2, \ldots, n_M\}$ (for $M \in \{\prec, A\}$) are sorted according to the integer value of the bit-vector $\langle z_q^M, x^M_{(q,l_1,q)}, \ldots, x^M_{(q,l_k,q)} \rangle$ where $q \in \{2, \ldots, n_M\}$ and $l_1, \ldots, l_k$ is some fixed order of the transition labels in M.

Encoding as formulas The encoding of **(C1)** and **(C5)** as a Boolean constraint is straightforward. For **(C2)**, we assume additional Boolean variables r_q (for each $q \in Q_{\prec}$) to identify states that can be reached via paths with only (a, a) labels. **(C2)** is ensured by the following constraints, which are instantiated for each $q \in Q_{\prec}$:

$$(q \neq q_0^{\prec}) \vee r_q, \qquad \neg z_q^{\prec} \vee \neg r_q, \qquad \neg r_q \vee \bigwedge_{a \in \Sigma, q' \in Q_{\prec}} (\neg x^{\prec}_{(q,a,a,q')} \vee r_{q'}).$$

The first constraint ensures that r_q holds for the initial state, the second constraint excludes r_q for all final states. The third constraint expresses preservation of the r_q flags under (a, a) transitions.

We outline further how **(C3)** can be encoded for $\mathcal{A}_A$ (the other parts of **(C3)** and **(C4)** are similar). We assume additional variables y_q (for each $q \in Q_A$) ranging over the interval $[0, n_A - 1]$, to encode the distance of a state from the initial state; these integer variables can further be encoded in binary as a vector of Boolean variables. The following formulas, instantiated for each $q \in Q_A$, define the value of the variables, and imply that every state is only finitely many transitions away from the initial state:

$$y_1 = 0, \qquad (q = 1) \vee \bigvee_{a \in \Sigma, q' \in Q_A} \left(x^A_{(q',a,q)} \wedge y_q = y_{q'} + 1 \right).$$

4.3 Counterexample elimination

If the VERIFY step discovers that $(\mathcal{A}_A, \mathcal{T}_{\prec})$ violate some of the required conditions **(L1)–(L4)**, one of four possible kinds of counterexample will be derived, corresponding to outcomes #2–#5 described in Sect. 4.1. The counterexamples are mapped to constraints CE_i (for $i = 1, \ldots, 4$) to be added to ψ in SYNTHESISE as a conjunct:

- A configuration x from I_0 has to be included in A: $CE_1 = A(x)$

- A configuration y has to be included in A, under the assumption that x is included: $CE_2 \;=\; \neg A(x) \vee A(y)$
- Configurations x, z have to be related by $\prec$, under the assumption that x, y and y, z are related: $CE_3 \;=\; x \not\prec y \vee y \not\prec z \vee x \prec z$
- Player 2 has to be able to make a $\prec$-decreasing step from y, assuming $x \to_1 y$ and x is included in A: $CE_4 \;=\; \neg A(x) \vee \exists z.\ \big(A(z) \wedge (y \to_2 z) \wedge x \succ z\big)$

Each of the formulas can be directly translated to a Boolean constraint over the vocabulary introduced in Sect. 4.2, augmented with additional auxiliary variables; the most intricate case is CE_4, due to the quantifier $\exists z$. More details are given in the full version.

5 Optimisations and incremental liveness proofs

The monolithic approach introduced so far is quite fast when compact advice bits exist (as shown in Sect. 6), but tends to be limited in scalability for more complex systems, because the search space grows rapidly when increasing the size of the considered automata. To address this issue, we introduce a range of optimisations of the basic method, in particular an *incremental* algorithm for synthesising advice bits, computing the set A and the relation $\prec$ by repeatedly constructing small automata.

5.1 Incremental liveness proofs

We first introduce a disjunctive version of the advice bits used to witness liveness:

Definition 2. *Let $(J, <)$ be a non-empty well-ordered index set.*[4] *A* disjunctive advice bit *is a tuple $\langle A, (B_j, \prec_j)_{j \in J}\rangle$, where $A, B_j \subseteq S$ are sets of game configurations, and each $\prec_j \subseteq S \times S$ is a binary relation over the game configurations, such that:*

(D1) *$I_0 \subseteq A$;*
(D2) *A is $\to$-inductive, i.e., $\forall x, y : x \in A \setminus F \wedge (x \to y) \Rightarrow y \in A$;*
(D3) *A is covered by the B_j sets and F, i.e., $A \subseteq F \cup \bigcup_{j \in J} B_j$;*
(D4) *for each $j \in J$, the relation $\prec_j$ is a strict preorder on S;*
(D5) *for each $j \in J$, player 2 can progress from B_j by following $\prec_j$:*

$$\forall x \in A \cap B_j \setminus (F \cup \bigcup_{i<j} B_i), y \in S \setminus F \,:\, \begin{pmatrix} (x \to_1 y) \;\Rightarrow \\ \exists z \in B_j \,:\, (y \to_2 z) \wedge z \prec_j x \end{pmatrix} .$$

The difference to monolithic advice bits (as defined in Sect. 3.3) is that the global preorder $\prec$ is replaced by a set of preorders $\prec_j$. Player 2 progresses to sets B_i with smaller index $i < j$ by following $\prec_j$, and this way eventually reaches F. A monolithic order $\prec$ can be reconstructed by defining

$$x \prec y \quad\Leftrightarrow\quad \begin{cases} idx(x) < idx(y) & \text{if } idx(x) \neq idx(y) \\ x \prec_j y & \text{if } idx(x) = idx(y) = j \end{cases}$$

[4] This means, $<$ is a strict total well-founded order on J.

Algorithm 1: Incremental liveness checker

```
1 $A \leftarrow S$ ;                                  // Over-approximation of reachable configurations
2 $W \leftarrow F$ ;                                  // Under-approximation of winning configurations

3 while $A \not\subseteq W$ do
4     choose a word $u \in A \setminus W$;
5     if $u$ is reachable then
6         $W \leftarrow W \cup win(u, A, W)$ ;        // Widen set of winning configurations
7     else
8         $A \leftarrow A \cap invariant(u, A)$ ;     // Tighten set of reachable configurations

9 return "Player 2 can win from every reachable configuration!"
```

where $idx(x) = \min\{j \in J \mid x \in B_j\}$, and $idx(x) = \min J$ in case these is no $j \in J$ with $x \in B_j$. From this, it immediately follows that Theorem 1 also holds for disjunctive advice bits. We can further note that if J is finite and all sets in $(A, (B_j, <_j)_{j \in J})$ are regular, then the disjunctive advice bits correspond to regular monolithic advice bits; in general this is not the case for infinite J.

Alg. 1 outlines the incremental liveness checker, defined with the help of disjunctive advice bits. The algorithm repeatedly refines a set A over-approximating the reachable configurations, and a set F under-approximating the configurations from which player 2 can win, and terminates as soon as all reachable configurations are known to be winning. The algorithm makes use of two sub-routines: in line 8, $invariant(u, A)$ denotes a *relatively inductive* invariant I [26] excluding u, i.e., a set $I \subseteq S$ such that

(RI1) $u \notin I$;
(RI2) $I_0 \subseteq I$;
(RI3) A is $\rightarrow$-inductive relative to A, i.e., $\forall x, y : x \in (I \cap A \setminus F) \wedge (x \rightarrow y) \Rightarrow y \in I$.

If A satisfies conditions **(D1)** and **(D2)**, and I is inductive relative to A, then also $A \cap I$ is an inductive set in the sense of **(D1)** and **(D2)**. We can practically compute automata representing sets I using a SAT-based refinement loop similar to the one in Sect. 4.

The second function $win(u, A, W)$ (line 6) computes a further progress pair $(B, \prec)$ witnessing the ability of Player 2 to win from u, and returns the set B, subject to:

(PP1) $u \in B$;
(PP2) the relation $\prec$ is a strict preorder on S;
(PP3) Player 2 can progress from B by following $\prec$:

$$\forall x \in A \cap B \setminus W, y \in S \setminus F : \ \big((x \rightarrow_1 y) \ \Rightarrow \ \exists z \in B : \ (y \rightarrow_2 z) \wedge z \prec x\big) \ .$$

Again, a SAT-based refinement loop similar to the one in Sect. 4 can be used to find regular progress pairs $(B, \prec)$ satisfying the conditions. Comparing **(RI1)**–**(RI3)** and **(PP1)**–**(PP3)** with **(D1)**–**(D5)**, it is also clear that disjunctive advice bits can be extracted from every successful run of Alg. 1, which implies soundness. Alg. 1 is in addition complete in the following sense: if there exist (monolithic) regular advice bits conforming to a game $\langle \mathfrak{S}, I_0, F \rangle$, if the words u chosen in line 4 are always of minimum length, and if the functions $invariant$ and win always compute minimum-size automata (representing sets I and $(B, \prec)$) solving the conditions **(RI1)**–**(RI3)** and **(PP1)**–**(PP3)**, then Alg. 1 terminates. This minimality condition is satisfied for the learning-based algorithms derived in Sect. 4.

5.2 Pre-Computation of inductive invariants

Alg. 1 can be optimised in different regards. First of all, the assignment $A \leftarrow S$ (line 1) initialising the approximation A of reachable states can be replaced with more precise pre-computation of the reachable states, for instance with the help of abstract regular model checking [22]. In fact, any set A satisfying **(D1)** and **(D2)** can be chosen.

We propose an efficient method for initialising A by utilising Angluin's $L*$-learning algorithm [9], which is applicable due to the property of length-preserving arenas that reachability of a given configuration w (a word) from the initial configurations I_0 is *decidable.* Decidability follows from the fact that there are only finitely many configurations up to a certain length, and the words occurring on a derivation $w_0 \to w_1 \to \cdots \to w_n$ all have the same length, so that known (explicit-state or symbolic) model checking methods can be used to decide reachability.

Reachability of configurations enables us to construct an $L*$ teacher (a.k.a. oracle). Membership queries for individual words w are answered by checking reachability of w in the game. Once the learner produces an hypothesis automaton $\mathcal{H}$, the teacher verifies that:

1. $\mathcal{H}$ includes the language I_0, i.e., **(D1)** is satisfied. If this is not the case, the teacher informs the learner about some further word in I_0 that has to be accepted by $\mathcal{H}$.
2. $\mathcal{H}$ is inductive, i.e., satisfies condition **(D2)**, which can be checked by means of automata methods (as in Sect. 4). If **(D2)** is violated, the counterexample pair (x, y) is examined, and it is checked whether the configuration x is reachable. If x is not reachable, the teacher gives a negative answer and demands that x be removed from the language; otherwise, the teacher demands that y is added to the language.
3. $\mathcal{H}$ describes the precise set of reachable configurations, for configuration length up to some fixed n. In other words, whenever $\mathcal{H}$ accepts some word w with $|w| \leq n$, the configuration w has to be reachable; otherwise, the teacher demands that w is eliminated from the language.

If all three tests succeed, the teacher accepts the produced automaton $\mathcal{H}$, which indeed represents a set A satisfying **(D1)** and **(D2)**. Tests 1 and 2 ensure that $\mathcal{H}$ is an inductive invariant, while test 3 is necessary to prevent trivial solutions: without the test, the algorithm could always return an automaton $\mathcal{H}$ recognising the universal language Σ^*. The parameter n determines the precision of synthesised invariants: larger n lead to automata $\mathcal{H}$ that are tighter over-approximations of the precise language of reachable configurations.[5]

This algorithm is guaranteed to terminate if the set of reachable configurations in an arena is regular; but it might only produce some inductive over-approximation of the reachable configurations. In our experiments, the computed languages usually capture reachable configurations very precisely, and the learning process converges quickly.

[5] In our implementation we currently hard-code n to be 5.

5.3 Exploitation of game symmetries

As a second optimisation, the incremental procedure can be improved to take symmetries of game instances into account, thus reducing the number of iterations needed in the incremental procedure; algorithms to automatically find symmetries in parameterised systems have recently proposed in [57]. This corresponds to replacing line 6 of Alg. 1 with the assignment $W \leftarrow W \cup \sigma^*(win(u, A, W))$; where σ is an *automorphism* of the game instance $\langle \mathfrak{G}, I_0, F\rangle$, and $\sigma^*(L) = L \cup \sigma(L) \cup \sigma^2(L) \cup \cdots$ represents unbounded application of σ to a language $L \subseteq \Sigma^*$. An automorphism (or *symmetry pattern* [57]) is a length-preserving bijection $\sigma : \Sigma^* \to \Sigma^*$ such that 1. initial and winning configurations are σ-invariant, i.e., $\sigma(I_0) = I_0$ and $\sigma(F) = F$; and 2. σ is a homomorphism of the moves, i.e., $u \to_i v$ if and only if $\sigma(u) \to_i \sigma(v)$ for $i \in \{1, 2\}$.

A symmetry commonly present in systems with ring topology is *rotation,* defined by $\sigma_{\text{rot}}(u_1 u_2 \ldots u_n) = u_2 \ldots u_n u_1$; the Israeli-Jalfon protocol (Example 1) exhibits this symmetry, as do many other examples. In addition, the fixed-point $\sigma^*_{\text{rot}}(L)$ can effectively be constructed for any regular language $L \subseteq \Sigma^*$ using simple automata methods, which is of course important for implementing the optimised incremental algorithm.

In terms of disjunctive advice bits $\langle A, (B_j, \prec_j)_{j \in J}\rangle$, application of a symmetry σ corresponds to including a sequence $(B, \prec), (\sigma(B), \prec^\sigma), (\sigma^2(B), \prec^{\sigma^2}), \ldots$ of progress pairs, defining $(u \prec^\rho v) \Leftrightarrow (\rho^{-1}(u) \prec \rho^{-1}(v))$ for any bijection $\rho : \Sigma^* \to \Sigma^*$. The resulting monolithic progress relation will in general not be regular; in terms of ordinals, this means that a well-order $(J, <)$ greater than ω is chosen.

6 Experiments and Conclusion

All techniques introduced in this paper have been implemented in the liveness checker `SLRP` [1] for parameterised systems, using the SAT4J [16] solver for Boolean constraints. For evaluation, we consider a range of (randomised and deterministic) parameterised systems, as well as Take-away and Nim games, shown in Table 1. Two of the randomised protocols, Lehmann-Rabin and Israeli-Jalfon are symmetric under rotation. Since Herman's original protocol in a ring [46] only satisfies liveness under "fair" schedulers, we used the version of the protocol in a line topology, which does satisfy liveness under all schedulers. Firewire is an example taken from [35, 60] representing a fragment of Firewire symmetry breaking protocol. For handling combinatorial games, the monolithic method in Sect. 4 was adapted by removing condition **(L2)**; adaptation of the incremental algorithm from Sect. 5.1 to this setting has not been considered yet.

All models could be solved using at least one of the considered CEGAR modes. In most cases, the monolithic approach from Sect. 4 displays good performance, and in case of the deterministic systems is competitive with existing tools (e.g. [8, 65]). Monolithic reasoning outperforms the incremental methods (Sect. 5) in particular for Szymanski, which is because Alg. 1 spends a lot of time computing a good approximation A of reachable states, although liveness can even be shown using $A = \Sigma^*$.

In contrast, the most complex model, the Lehmann-Rabin protocol for Dining Philosophers, can only be solved using the incremental algorithm, and only when accelerating the procedure by exploiting the rotation symmetry of the game (Sect. 5.3). In configuration Incr+Inv+Symm, Alg. 1 computes an initial set A represented by a DFA

Table 1. Verification results for parameterised systems and games. **Mono** is the monolithic method from Sect. 4, **Incr** the incremental algorithm from Sect. 5.1, and **Inv** and **Symm** the optimisations introduced in Sect. 5.2 and 5.3, respectively. A dash — indicates that a model is not symmetric under rotation, or that the incremental algorithm is not applicable (in case of Take-away and Nim). The numbers in the table give runtime (wall-clock time) for the individual benchmarks and configurations; all experiments were done on an AMD Opteron 6282 32-core machine, Java heap memory limited to 20GB, timeout 2 hours.

	Mono	**Incr**	**Incr+Inv**	**Incr+Symm**	**Incr+Inv+Symm**
Randomised parameterised systems					
Lehmann-Rabin (DP) [34]	T/O	T/O	T/O	48min	10min
Israeli-Jalfon [47]	4.6s	22.7s	21.4s	9.9s	9.7s
Herman [46]	1.5s	1.6s	2.4s	—	—
Firewire [35, 60]	1.3s	1.3s	2.0s	—	—
Deterministic parameterised systems					
Szymanski [4, 65]	5.7s	27min	10min	—	—
DP, left-right strategy	1.9s	6.4s	3.4s	—	—
Bakery [4, 65]	1.6s	2.7s	1.9s	—	—
Resource allocator [32]	2.2s	2.2s	2.0s	—	—
Games on infinite graphs					
Take-away [38]	2.8s	—	—	—	—
Nim [38]	5.3s	—	—	—	—

with 23 states (Sect. 5.2), calls the function win 25 times to obtain further progress relations (Sect. 5.1), and overall needs 4324 iterations of the refinement procedure of Sect. 4. To the best of our knowledge, this is the first time that liveness under arbitrary schedulers for randomised parameterised systems like Lehmann-Rabin could be shown fully automatically.

Future Work We conclude with two concrete research questions among many others. The most immediate question is how to embed fairness in our framework of randomised parameterised systems. Another research direction concerns how to extend transducers to deal with data so as to model protocols where tokens may store arbitrary process IDs (examples of which include Dijkstra's Self-Stabilizing Protocol [31]).

Acknowledgment. We thank anonymous referees, Parosh Abdulla, Bengt Jonsson, Ondrej Lengal, Rupak Majumdar, and Ahmed Rezine for their helpful feedback. We thank Truong Khanh Nguyen for contributing with the development of the tool `parasymmetry` [57], on top of which our current tool (SLRP) builds.

References

1. SLRP website (referred in May 2016). `https://github.com/uuverifiers/autosat/tree/master/LivenessProver`.
2. P. A. Abdulla. Regular model checking. *STTT*, 14(2):109–118, 2012.
3. P. A. Abdulla, M. F. Atig, and J. Cederberg. Analysis of message passing programs using SMT-solvers. In *ATVA*, pages 272–286, 2013.

4. P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *TACAS*, pages 721–736, 2007.
5. P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *CAV*, pages 145–157, 2007.
6. P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 476–495, 2013.
7. P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. In *CONCUR*, pages 35–48, 2004.
8. P. A. Abdulla, B. Jonsson, A. Rezine, and M. Saksena. Proving liveness by backwards reachability. In *CONCUR*, pages 95–109, 2006.
9. D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, Nov. 1987.
10. A. Annichini, A. Bouajjani, and M. Sighireanu. Trex: A tool for reachability analysis of complex systems. In *CAV*, pages 368–372, 2001.
11. K. R. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.*, 22(6):307–309, 1986.
12. T. Arons, A. Pnueli, and L. D. Zuck. Parameterized verification by probabilistic abstraction. In *FoSSaCS*, pages 87–102, 2003.
13. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.
14. J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. *Distributed Computing*, 20(1):75–93, 2007.
15. M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Addison-Wesley, 2nd edition, 2006.
16. D. L. Berre and A. Parrain. The Sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.
17. R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
18. A. Blumensath. Automatic structures. Master's thesis, RWTH Aachen, 1999.
19. A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004.
20. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large (extended abstract). In *CAV*, pages 223–235, 2003.
21. A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract regular (tree) model checking. *STTT*, 14(2):167–191, 2012.
22. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV*, pages 372–386, 2004.
23. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *CAV*, pages 403–418, 2000.
24. A. Bouajjani, A. Legay, and P. Wolper. Handling liveness properties in (*omega*-)regular model checking. *Electr. Notes Theor. Comput. Sci.*, 138(3):101–115, 2005.
25. G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.
26. A. R. Bradley and Z. Manna. Property-directed incremental invariant generation. *Formal Aspects of Computing*, 20(4):379–405, 2008.
27. A. Chakarov and S. Sankaranarayanan. Probabilistic program analysis with martingales. In *CAV*, pages 511–526, 2013.

28. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, pages 154–169, 2000.
29. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.
30. L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *CONCUR*, pages 66–81, 1999.
31. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
32. A. F. Donaldson. *Automatic Techniques for Detecting and Exploiting Symmetry in Model Checking*. PhD thesis, University of Glasgow, 2007.
33. S. Dubois and S. Tixeuil. A taxonomy of daemons in self-stabilization. *CoRR*, abs/1110.0334, 2011.
34. M. Duflot, L. Fribourg, and C. Picaronny. Randomized dining philosophers without fairness assumption. *Distributed Computing*, 17(1):65–76, 2004.
35. J. Esparza, A. Gaiser, and S. Kiefer. Proving termination of probabilistic programs using patterns. In *CAV*, pages 123–138, 2012.
36. J. Esparza, P. Ganty, and T. Poch. Pattern-based verification for multithreaded programs. *ACM Trans. Program. Lang. Syst.*, 36(3):9:1–9:29, 2014.
37. Y. Fang, N. Piterman, A. Pnueli, and L. D. Zuck. Liveness with invisible ranking. *STTT*, 8(3):261–279, 2006.
38. T. S. Ferguson. *Game Theory*. Online Book, second edition, 2014.
39. W. Fokkink. *Distributed Algorithms*. MIT Press, 2013.
40. P. Garg, C. Löding, P. Madhusudan, and D. Neider. Learning universally quantified invariants of linear data structures. In *CAV*, pages 813–829, 2013.
41. W. Goddard and P. K. Srimani. Daemon conversions in distributed self-stabilizing algorithms. In *WALCOM*, pages 146–157, 2013.
42. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
43. P. Habermehl, L. Holík, A. Rogalewicz, J. Simácek, and T. Vojnar. Forest automata for verification of heap manipulation. *Formal Methods in System Design*, 41(1):83–106, 2012.
44. M. Hague and A. W. Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV*, pages 260–276, 2012.
45. S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent program. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, 1983.
46. T. Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.
47. A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC*, pages 119–131, 1990.
48. B. Jonsson and M. Saksena. Systematic acceleration in regular model checking. In *CAV*, pages 131–144, 2007.
49. A. Kaiser, D. Kroening, and T. Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *CAV*, pages 645–659, 2010.
50. H. Kakugawa and M. Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *IEEE Trans. Parallel Distrib. Syst.*, 8(2):154–163, 1997.
51. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.
52. M. Z. Kwiatkowska. Model checking for probability and time: from theory to practice. In *LICS*, page 351, 2003.
53. F. Laroussinie and J. Sproston. State explosion in almost-sure probabilistic reachability. *Inf. Process. Lett.*, 102(6):236–241, 2007.

54. A. Legay. T(O)RMC: A tool for (omega)-regular model checking. In *CAV*, pages 548–551, 2008.
55. D. Lehmann and M. Rabin. On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem (extended abstract). In *POPL*, pages 133–138, 1981.
56. A. W. Lin. Accelerating tree-automatic relations. In *FSTTCS*, pages 313–324, 2012.
57. A. W. Lin, T. K. Nguyen, P. Rümmer, and J. Sun. Regular symmetry patterns. In *VMCAI*, pages 455–475, 2016.
58. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
59. N. A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *PODC*, pages 314–323, 1994.
60. A. McIver, C. Morgan, and T. S. Hoang. Probabilistic termination in B. In *ZB*, pages 216–239, 2003.
61. D. Monniaux. An abstract analysis of the probabilistic termination of programs. In *SAS*, pages 111–126. Springer, 2001.
62. D. Neider. Reachability games on automatic graphs. In *CIAA*, pages 222–230, 2010.
63. D. Neider and N. Jansen. Regular model checking using solver technologies and automata learning. In *NFM*, pages 16–31, 2013.
64. D. Neider and U. Topcu. An automaton learning approach to solving safety games over infinite graphs. In *TACAS*, pages 204–221, 2016.
65. M. Nilsson. *Regular Model Checking*. PhD thesis, Uppsala Universitet, 2005.
66. G. Norman. Analysing randomized distributed algorithms. In *Validation of Stochastic Systems - A Guide to Current Research*, pages 384–418, 2004.
67. A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verification. In *CAV*, pages 328–343, 2000.
68. A. Pnueli, J. Xu, and L. D. Zuck. Liveness with $(0, 1, \infty)$-counter abstraction. In *CAV*, pages 107–122, 2002.
69. A. Pnueli and L. D. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.
70. B. K. Szymanski. A simple solution to Lamport's concurrent programming problem with linear wait. In *ICS*, pages 621–626, 1988.
71. A. W. To. *Model Checking Infinite-State Systems: Generic and Specific Approaches*. PhD thesis, LFCS, School of Informatics, University of Edinburgh, 2010.
72. A. W. To and L. Libkin. Recurrent reachability analysis in regular model checking. In *LPAR*, pages 198–213, 2008.
73. A. W. To and L. Libkin. Algorithmic metatheorems for decidable LTL model checking over infinite systems. In *FoSSaCS*, pages 221–236, 2010.
74. A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Using language inference to verify omega-regular properties. In *TACAS*, pages 45–60, 2005.
75. A. Vardhan and M. Viswanathan. LEVER: A tool for learning based verification. In *CAV*, pages 471–474, 2006.
76. M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338, 1985.
77. T. Vojnar. Cut-offs and automata in formal verification of infinite-state systems, 2007. Habilitation Thesis, Faculty of Information Technology, Brno University of Technology.
78. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV*, pages 88–97, 1998.
79. L. D. Zuck and A. Pnueli. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems & Structures*, 30(3-4):139–169, 2004.