Bridging the Gap Between Fair Simulation and Trace Inclusion ⋆

Yonit Kesten [a] Nir Piterman [b,∗,1] Amir Pnueli [b]

[a] *Department of Communication Systems Engineering, Ben Gurion University, Beer-Sheva, Israel. Email: ykesten@bgumail.bgu.ac.il*

[b] *Department of Applied Mathematics and Computer Science, Weizmann Institute of Science,Rehovot 76100, Israel. Email: firstname.lastname@weizmann.ac.il*

Abstract

The paper considers the problem of checking abstraction between two finite-state *fair discrete systems* (FDS). In automata-theoretic terms this is trace inclusion between two nondeterministic Streett automata. We propose to reduce this problem to an algorithm for checking fair simulation between two generalized Büchi automata. For solving this question we present a new triply nested μ-calculus formula which can be implemented by symbolic methods.

We then show that every trace inclusion of this type can be solved by fair simulation, provided we augment the concrete system (the contained automaton) by an appropriate 'non-constraining' automaton. This establishes that fair simulation offers a complete method for checking trace inclusion for finite-state systems. We illustrate the feasibility of the approach by algorithmically checking abstraction between finite state systems whose abstraction could only be verified by deductive methods up to now.

Key words: Streett automata, trace inclusion, fair simulation

1 Introduction

A frequently occurring problem in verification of reactive systems is the problem of *abstraction* (symmetrically *refinement*) in which we are given a concrete reactive

⋆ This research was supported in part by the Israel Science Foundation (grant no. 106/02-1) and the John von-Neumann Minerva center for Verification of Reactive Systems.

∗ To whom correspondence should be addressed.

[1] Current address: School of Computer and Communication Sciences, Ecole Polytechnique Federal de Lausanne, Switzerland. *Email:* firstname.lastname@epfl.ch

system C and an abstract reactive system A and are asked to check whether A *abstracts* C, denoted $C \sqsubseteq A$. In the linear-semantics framework this question calls for checking whether any observation of C is also an observation of A. For the case that both C and A are finite-state systems with weak and strong fairness this problem can be reduced to the problem of language inclusion between two Streett automata (e.g., [38]).

In theory, this problem has an exponential-time algorithmic solution based on the complementation of the automaton representing the abstract system [33]. However, the complexity of this algorithm makes its application prohibitively expensive. For example, our own interest in the finite-state abstraction problem stems from applications of the verification method of *network invariants* [17,20,40]. In a typical application of this method, we are asked to verify the abstraction $P_1 \parallel P_2 \parallel P_3 \parallel P_4 \sqsubseteq P_5 \parallel P_6 \parallel P_7$, claiming that 3 parallel copies of the dining philosophers process abstract 4 parallel copies of the same process. The system on the right has about 1800 states. Obviously, to complement a Streett automaton of 1800 states is hopelessly expensive.

A partial but more effective solution to the problem of checking abstraction between systems (trace inclusion between automata) is provided by the notion of *simulation*. Introduced first by Milner [30], we say that system A simulates system C, denoted $C \preceq A$, if there exists a *simulation relation* R between the states of C and the states of A. It is required that if $\langle s, t\rangle \in R$ and system C can move from state s to state s', then system A can move from t to some t' such that $\langle s', t'\rangle \in R$. Additional requirements on R are that if $\langle s, t\rangle \in R$ then s and t agree on the values of their observables, and for every s initial in C there exists t initial in A such that $\langle s, t\rangle \in R$. It is obvious that $C \preceq A$ is a sufficient condition for $C \sqsubseteq A$. For finite-state systems, we can check $C \preceq A$ in time proportional to $(|\Sigma_C| \cdot |\Sigma_A|)^2$ where Σ_C and Σ_A are the sets of states of A and C respectively [3,12].

While being a sufficient condition, simulation is definitely not a necessary condition for abstraction. This is illustrated by the two systems presented in Fig. 1

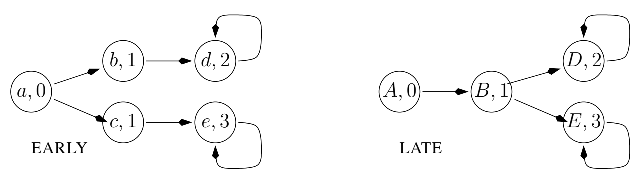


Fig. 1. Systems EARLY and LATE

The labels in these two systems consist of a local state name (a–e, A–E) and an observable value. Clearly these two systems are (observation)-equivalent because they each have the two possible observations $012^\omega + 013^\omega$. Thus, each of them abstracts the other. However, when we examine their simulation relation, we find that EARLY $\preceq$ LATE but LATE $\not\preceq$ EARLY. This example illustrates that, in some cases we can use simulation in order to establish abstraction (trace inclusion) but this method is not complete.

The above discussion only covered the case that C and A did not have any fairness constraints. There were many suggestions about how to enhance the notion of simulation in order to account for fairness [25,10,13,14]. The one we found most useful for our purposes is the definition of fair simulation from [13]. Henzinger et al. proposed a game-based view of simulation. As in the unfair case, the definition assumes an underlying simulation relation R which implies equality of the observables. However, in the presence of fairness, it is not sufficient to guarantee that every step of the concrete system can be matched by an abstract step with corresponding observables. Here we require that the abstract system has a *strategy* such that any joint run of the two systems, where the abstract player follows this strategy either satisfies the fairness requirements of the abstract system or fails to satisfy the fairness requirements of the concrete system. This guarantees that every concrete (fair) observation has a corresponding abstract observation with matching values of the observables.

In order to determine whether one system fairly simulates another (*solve fair simulation*) we have to solve games [13]. When the two systems in question are reactive systems with strong fairness (Streett), the winning condition of the resulting game is an implication between two Streett conditions (*Streett-Streett-games*). In [13] the solution of Streett-Streett-games is reduced to the solution of Streett games (i.e., a game where the winning condition is a Streett condition). In [23] an algorithm for solving Streett games is presented. The time complexity of this approach is $O((|\Sigma_A| \cdot |\Sigma_C| \cdot (3^m + n))^{2m+n} \cdot (2m+n)!)$ where n and m denote the number of Streett pairs of C and A respectively. Clearly, this complexity is too high. It is also not clear whether this algorithm can be implemented symbolically.

In [6], a solution for games with winning condition expressed as a general LTL formula is presented. The algorithm in [6] constructs a deterministic parity word automaton for the winning condition. The automaton is then converted into a μ-calculus formula that evaluates the set of winning states for the relevant player.

In [9], Emerson and Lei show that a μ-calculus formula is in fact a recipe for symbolic model checking [2]. The main factor in the complexity of μ-calculus model checking is the *alternation depth* of the formula. The symbolic algorithm for model checking a μ-calculus formula of alternation depth k takes time proportional to $(mn)^k$ where m is the size of the formula and n is the size of the model [9].

In Streett-Streett-games the winning condition is an implication between two Streett conditions. A deterministic Streett automaton for this winning condition has $3^m \cdot n$

[2] There are more efficient algorithms for μ-calculus model checking [26,34,15]. The first two require space exponential in the alternation depth of the formula. Jurdzinski's algorithm, which requires linear space, cannot be implemented symbolically.

states and $2m + n$ pairs (where n and m denote the number of Streett pairs of C and A respectively). A deterministic parity automaton for the same condition has $3^m \cdot n \cdot (2m + n)!$ states and index $4m + 2n$. The μ-calculus formula constructed by [6] is of alternation depth $4m + 2n$ and proportional in size to $3^m \cdot n \cdot (2m + n)!$. Hence, in this case, there is no advantage in using [6]. Recently, it was shown that deciding Streett-Street games is PSPACE-complete [2], so we cannot hope for much lower complexity.

In the context of fair simulation, Streett systems cannot be reduced to simpler systems [22]. That is, in order to solve the question of fair simulation between Streett systems we have to solve Streett-Streett-games in their full generality. However, we are only interested in fair simulation as a precondition for trace inclusion. In the context of trace inclusion we can reduce the problem of two reactive systems with strong fairness to an equivalent problem with weak fairness. Formally, for the reactive systems C and A with Streett fairness requirements, we construct C^B and A^B with generalized Büchi requirements, such that $C \sqsubseteq A$ iff $C^B \sqsubseteq A^B$. Solving fair simulation between C^B and A^B is simpler. The winning condition of the resulting game is an implication between two generalized Büchi conditions (denoted *generalized Streett[1]*).

In the case of generalized Streett[1] games, a deterministic parity automaton for the winning condition has $|J_C| \cdot |J_A|$ states and index 3, where $|J_C|$ and $|J_A|$ denote the number of Büchi sets in the fairness of C^B and A^B respectively. The μ-calculus formula of [6] is proportional to $3|J_C| \cdot |J_A|$ and has alternation depth 3.

We give an alternative μ-calculus formula that solves generalized Streett[1] games. Our formula is also of alternation depth 3 but its length is proportional to $2|J_C| \cdot |J_A|$ and it is simpler than that of [6]. Obviously, our algorithm is tailored for the case of generalized-Streett[1] games while [6] give a generic solution for any LTL game [3]. The time complexity of solving fair simulation between two reactive systems after converting them to systems with generalized Büchi fairness requirements is $O((|\Sigma_A| \cdot |\Sigma_C| \cdot 2^{m+n} \cdot (|J_A| + |J_C| + m + n))^3)$ where n and m denote the number of Streett pairs of C and A respectively.

Even if we succeed to present a complexity-acceptable algorithm for checking fair simulation between generalized-Büchi systems, there is still a major drawback to this approach which is its incompleteness. As shown by the example of Fig. 1, there are (trivially simple) systems C and A such that $C \sqsubseteq A$ but this abstraction cannot be proven using fair simulation. Fortunately, we are not the first to be concerned by the incompleteness of simulation as a method for proving abstraction. In the context of infinite-state system verification, Abadi and Lamport studied the method of

[3] One may ask why not take one step further and convert the original reactive systems to Büchi systems. In this case, the induced game is a parity[3] game and there is a simple algorithm for solving it. Although both algorithms work in cubic time, the latter performed much worse than the one described above. We cannot explain this phenomenon.

simulation using an *abstraction mapping* [1]. It is not difficult to see that this notion of simulation implies fair simulation as defined in [13]. However, [1] did not stop there but proceeded to show that if we are allowed to add to the concrete system auxiliary *history* and *prophecy* variables, then the simulation method becomes *complete*. That is, with appropriate augmentation by auxiliary variables, every abstraction relation can be proven using fair simulation. Intuitively, the prophecy and history variables help reduce the nondeterminism of the concrete system, enabling to establish simulation between the two systems. For finite state fair discrete systems (FDS) we show that there always exists a non-constraining FDS such that the synchronous composition of this FDS with the concrete system is fairly-simulated by the abstract system. It is well known that for every LTL formula [31] one can construct a non-constraining FDS such that a state of the FDS contains information regarding the current validity of every one of the subformulas of the formula [37]. We call such an FDS a *temporal tester*. We use such a temporal tester to augment LATE in order to establish LATE $\sqsubseteq$ EARLY.

The application of Abadi-Lamport, being deductive in nature, requires the users to decide on the appropriate history and prophecy variables, and then design their abstraction mapping which makes use of these auxiliary variables. Implementing these ideas in the finite-state (and therefore algorithmic) world, we expect the strategy (corresponding to the abstraction mapping) to be computed fully automatically. Thus, in our implementation, the user is still expected to find the non-constraining FDS, but following that, the rest of the process is automatic. For example, wishing to apply our algorithm to check the abstraction LATE $\sqsubseteq$ EARLY, the user has to specify the augmentation of the concrete system by a temporal tester for the LTL formula $\diamondsuit(x = 2)$, i.e. a non-constraining FDS that anticipates whether a state marked by 2 is eventually reached or not. Using this augmentation, the algorithm manages to prove that the augmented system (LATE +tester) is fairly simulated (hence abstracted) by EARLY.

Our interest in abstraction stems from its application in the method of *network invariants* [17,20,40]. Given a parameterized system $S(n)$: $P_1 \parallel \cdots \parallel P_n$ and a property p, *uniform verification* attempts to verify that $S(n) \models p$ for every value of the parameter n. The main idea of the *network invariants* method is to abstract $n-1$ of the processes, say the composition $P_2 \parallel \cdots \parallel P_n$, into a single finite-state process $\mathcal{I}$, independent of n. We refer to $\mathcal{I}$ as the *network invariant*. If possible, this reduces the uniform verification problem into the fixed size verification problem $(P_1 \parallel \mathcal{I}) \models p$. In order to show that $\mathcal{I}$ is a correct abstraction of any number of processes (assuming that $P_2, \ldots, P_n$ are all identical), it is sufficient to apply an inductive argument, using $P \sqsubseteq \mathcal{I}$ as the induction base and $(P \parallel \mathcal{I}) \sqsubseteq \mathcal{I}$ as the inductive step.

As mentioned, the problem of abstraction is computationally intractable. Consequently, in the past we have proved refinement by establishing a step-by-step simulation relation between the concrete computation and an abstract one, following the *abstraction mapping* method of Abadi and Lamport [1]. Using abstraction mapping, we have to supply the network invariant itself (abstract system), and a mapping from the concrete system to the abstract one (not to mention the possible need of augmenting the concrete system). In many cases, we can form a trivial network invariant by combining a small number of the processes themselves. In these cases, providing the abstraction mapping can be extremely complicated. On the other hand, usually, there exists a network invariant that requires only a simple abstraction mapping. However, the divination of such a network invariant can be extremely complicated. Either way, one of the stages (if not both) of finding and proving a network invariant deductively, can be very complicated. It is our hope, that replacing the abstraction mapping technique by the automatic proof of fair-simulation, will allow us to use the trivial network invariants and the laborious work of finding the abstraction mapping will not be required.

In summary, the contributions of this paper are:

(1) Suggesting the usage of fair simulation as a precondition for abstraction between two reactive systems (Streett automata).
(2) Observing that in the context of fair simulation for checking abstraction we can simplify the game acceptance condition from implication between two Streett conditions to implication between two generalized Büchi conditions.
(3) Providing a more efficient μ-calculus formula and its implementation by symbolic model-checking tools for solving the fair simulation between two generalized Büchi systems.
(4) Proving the completeness of the fair-simulation method to establish abstraction between two systems, at the price of augmenting the concrete system by a non-constraining automaton.

A preliminary version of this paper appeared in [16].

2 The Computational Model

As a computational model, we take the model of *fair discrete system* (FDS) [18]. An FDS $\mathcal{D} : \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$ consists of the following components.

- $V = \{u_1, ..., u_n\}$: A finite set of typed *state variables* over finite domains. We define a *state* s to be a type-consistent interpretation of V, assigning to each variable $u \in V$ a value $s[u]$ in its domain. We denote by Σ the set of all states. In this paper we assume that Σ is finite. An *assertion* over V is a Boolean combination of comparisons $u = a$ or $u = v$ where $u, v \in V$ range over the same domain and a is a value in the domain of u. A state s satisfies an assertion φ, denoted $s \models \varphi$, if φ evaluates to $\mathbf{true}$ by assigning $s[u]$ to every one of the variables appearing in φ. We say that s is a φ-state if $s \models \varphi$.

- $\mathcal{O} \subseteq V$: A subset of *observable variables*. These are the variables which can be externally observed.
- Θ : The *initial condition*. This is an assertion characterizing all the initial states of the FDS. A state is called *initial* if it satisfies Θ.
- ρ : A *transition relation*. This is an assertion $\rho(V, V')$, relating a state $s \in \Sigma$ to its $\mathcal{D}$-successor $s' \in \Sigma$ by referring to both unprimed and primed versions of the state variables. The transition relation $\rho(V, V')$ identifies state s' as a $\mathcal{D}$-*successor* of state s if $(s, s') \models \rho(V, V')$, where (s, s') is the joint interpretation which interprets $x \in V$ as $s[x]$, and x' as $s'[x]$.
- $\mathcal{J} = \{J_1, \ldots, J_k\}$: A set of assertions expressing the *justice* requirements (*weak fairness*). Intentionally, the justice requirement $J \in \mathcal{J}$ stipulates that every computation contains infinitely many J-states (states satisfying J).
- $\mathcal{C} = \{\langle p_1, q_1 \rangle, \ldots \langle p_n, q_n \rangle\}$: A set of assertions expressing the *compassion* requirements (*strong fairness*). Intentionally, the compassion requirement $\langle p, q \rangle \in \mathcal{C}$ stipulates that every computation containing infinitely many p-states also contains infinitely many q-states.

Note that an FDS can be viewed as a Streett automaton [35] where the labels are on vertices instead of on edges. A Streett pair $\langle p, q \rangle$ is included in the set of justice requirements if $p = \mathrm{T}$ and in the set of compassion requirements if $p \neq \mathrm{T}$.

Let $\sigma : s_0, s_1, ...,$ be a sequence of states, φ be an assertion, and $j \geq 0$ be a natural number. We say that j is a φ-position of σ if s_j is a φ-state. Let $\mathcal{D}$ be an FDS for which the above components have been identified. We define a *run* of $\mathcal{D}$ to be a maximal sequence of states $\sigma : s_0, s_1, ...,$ satisfying the requirements of

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For every $j \geq 0$, the state s_{j+1} is a $\mathcal{D}$-successor of the state s_j.

The sequence σ being maximal means that either σ is infinite, or $\sigma = s_0, \ldots, s_k$ and s_k has no $\mathcal{D}$-successor.

We denote by *runs*$(\mathcal{D})$ the set of runs of $\mathcal{D}$. A state is called *reachable* if it participates in some run. An infinite run of $\mathcal{D}$ is called a *computation* if it satisfies the following:

- *Justice:* For each $J \in \mathcal{J}$, σ contains infinitely many J-positions.
- *Compassion:* For each $\langle p, q \rangle \in \mathcal{C}$, if σ contains infinitely many p-positions, it must also contain infinitely many q-positions.

We denote by $Comp(\mathcal{D})$ the set of all computations of $\mathcal{D}$. An FDS $\mathcal{D}$ is called *deadlock-free* if every reachable state has a $\mathcal{D}$-successor. Note that all runs of a deadlock-free FDS are infinite. We say that a state s of $\mathcal{D}$ is *feasible* if it participates in some computation of $\mathcal{D}$. An FDS $\mathcal{D}$ is called *viable* if all reachable states in $\mathcal{D}$ are feasible. It is not difficult to see that every viable FDS is deadlock-free.

Systems $\mathcal{D}_1 : \langle V_1, \mathcal{O}_1, \Theta_1, \rho_1, \mathcal{J}_1, \mathcal{C}_1 \rangle$ and $\mathcal{D}_2 : \langle V_2, \mathcal{O}_2, \Theta_2, \rho_2, \mathcal{J}_2, \mathcal{C}_2 \rangle$ are *composable* if the intersection of their variables is observable in both systems, i.e. $V_1 \cap V_2 \subseteq \mathcal{O}_1 \cap \mathcal{O}_2$. For composable systems $\mathcal{D}_1$ and $\mathcal{D}_2$, we define their *asynchronous parallel composition*, denoted by $\mathcal{D}_1 \parallel \mathcal{D}_2$, as the FDS whose sets of variables, observable variables, justice, and compassion sets are the unions of the corresponding sets in the two systems, whose initial condition is the conjunction of the initial conditions, and whose transition relation is the following disjunction.

$$\rho = \rho_1 \wedge pres(V_2 \setminus V_1) \vee \rho_2 \wedge pres(V_1 \setminus V_2)$$

Here $pres$ denotes the function that preserves the values of all variables, namely

$$pres(V) = \bigwedge_{v \in V} v = v'$$

Thus, the execution of the combined system is the interleaved execution of $\mathcal{D}_1$ and $\mathcal{D}_2$.

For composable systems $\mathcal{D}_1$ and $\mathcal{D}_2$, we define their *synchronous parallel composition*, denoted by $\mathcal{D}_1 \;|||\; \mathcal{D}_2$, as the FDS whose sets of variables and initial condition are defined similarly to the asynchronous composition, and whose transition relation is the conjunction of the two transition relations. Thus, a step in an execution of the combined system is a joint step of systems $\mathcal{D}_1$ and $\mathcal{D}_2$. The primary use of synchronous composition is for augmenting an FDS with a non-constraining system. For more details, we refer the reader to [17].

The *projection* of a state s on a set $W \subseteq V$, denoted $s \Downarrow_W$, is the interpretation of the variables in W according to their values in s. Projection is generalized to sequences of states and to sets of sequences of states in the natural way. The *observations* of $\mathcal{D}$ are the projections of $\mathcal{D}$-computations onto $\mathcal{O}$. We denote by *Obs*$(\mathcal{D})$ the set of all observations of $\mathcal{D}$. Systems $\mathcal{D}_C : \langle V_C, \mathcal{O}_C, \Theta_C, \rho_C, \mathcal{J}_C, \mathcal{C}_C \rangle$ and $\mathcal{D}_A : \langle V_A, \mathcal{O}_A, \Theta_A, \rho_A, \mathcal{J}_A, \mathcal{C}_A \rangle$ are said to be *comparable* if there is a one to one correspondence between their observable variables, i.e. a bijection $b : \mathcal{O}_C \rightarrow \mathcal{O}_A$ such that for every $v \in \mathcal{O}_C$, v and $b(v)$ are of the same type and range over the same domain. We assume that $V_C \cap V_A = \emptyset$. We write $s \Downarrow_{\mathcal{O}_C} = t \Downarrow_{\mathcal{O}_A}$ to denote that for every $v \in \mathcal{O}_C$ the assignment $s[v] = t[b(v)]$. This notion is generalized in the natural way to observations and sets of observations. System $\mathcal{D}_A$ is said to be an *abstraction* of the comparable system $\mathcal{D}_C$, denoted $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$, if *Obs*$(\mathcal{D}_C) \subseteq$ *Obs*$(\mathcal{D}_A)$. The abstraction relation is reflexive and transitive. It is also *property restricting*. That is, if $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$ then $\mathcal{D}_A \models p$ implies that $\mathcal{D}_C \models p$ for an LTL property p. We say that two comparable FDS's $\mathcal{D}_1$ and $\mathcal{D}_2$ are *equivalent*, denoted $\mathcal{D}_1 \sim \mathcal{D}_2$ if *Obs*$(\mathcal{D}_1) =$ *Obs*$(\mathcal{D}_2)$.[4]

[4] The definitions of comparable and composable are exactly the conditions needed in order to handle these operations in symbolic state manipulation environments. In order to compose or compare two systems we would like the variables of both systems to co-exists in

Consider an FDS $\mathcal{D}$: $\langle V, \mathcal{O}, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$. Let $\Pi_{\mathcal{O}}$ denote the set of possible assignments to the variables in $\mathcal{O}$. We say that $\mathcal{D}$ is *non-constraining with respect to* $\mathcal{D}_C$: $\langle V_C, \mathcal{O}_C, \Theta_C, \rho_C, \mathcal{J}_C, \mathcal{C}_C \rangle$ if $\mathcal{D}$ and $\mathcal{D}_C$ are composable and we have $Obs(\mathcal{D}_C) \subseteq Obs(\mathcal{D}_C \,|||\, \mathcal{D}) \Downarrow_{\mathcal{O}_C}$, i.e., the synchronous composition does not omit observable behaviors. We say that $\mathcal{D}$ is *non-constraining* if for every sequence $\pi \in (\Pi_{\mathcal{O}})^\omega$, we have that π is an observation of $\mathcal{D}$. In particular, if $\mathcal{D}$ is non-constraining and $\mathcal{D}$ and $\mathcal{D}_C$ are composable then $\mathcal{D}$ is non constraining with respect to $\mathcal{D}_C$. For a non-constraining system $\mathcal{D}$ such that $\mathcal{O} = \mathcal{O}_C$ it follows that $Obs(\mathcal{D}_C) = Obs(\mathcal{D}_C \,|||\, \mathcal{D})$.

All our concrete examples are given in SPL (Simple Programming Language), which is used to represent concurrent programs (e.g., [28,27]). Every SPL program can be compiled into an FDS in a straightforward manner. In particular, every statement in an SPL program contributes a disjunct to the transition relation. For example, the assignment statement "ℓ_0: $y := x + 1$; ℓ_1:" contributes to ρ the disjunct

$$\rho_{\ell_0}: \quad at\_\ell_0 \,\wedge\, at'\_\ell_1 \,\wedge\, y' = x + 1 \,\wedge\, x' = x.$$

The predicates at_ℓ_0 and at'_ℓ_1 stand, respectively, for the assertions $\pi_i = 0$ and $\pi_i' = 1$, where π_i is the control variable denoting the current location within the process to which the statement belongs. Every FDS that is generated by an SPL program is viable.

Every FDS can be converted to a viable FDS that has the same set of computations, by restricting the transition relation to viable states [19]. Without loss of generality, we assume that every FDS is viable. In particular, when considering the asynchronous or synchronous parallel composition of two FDS's we assume that the resulting FDS is viable. This is also the case with the conversion from FDS to JDS described in the following section.

From FDS *to* JDS

An FDS with no compassion requirements is called a *just discrete system* (JDS). Note that a JDS can be viewed as a generalized Büchi automaton.

Theorem 1 [5] *For every* FDS *with set of states* Σ *and set of compassion requirements* $\mathcal{C}$ *there exists a* JDS $\mathcal{D}^B$ *with* $|\Sigma| \cdot 2^{|\mathcal{C}|+1}$ *states such that* $Obs(\mathcal{D}) = Obs(\mathcal{D}^B)$.

the same environment. Accordingly, when comparing two systems we would like to be able to handle the states of each of the systems separately without affecting the other system, hence their variable sets should be disjoint. When composing two systems, we would like the composition to behave differently from each of the systems alone, hence their observable variables should intersect.

PROOF. Let $\mathcal{D}:\ \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$ be an FDS where $\mathcal{C} = \{\langle p_1, q_1\rangle, \ldots, \langle p_m, q_m\rangle\}$ and $m > 0$. We define a JDS $\mathcal{D}^{B}\ :\ \langle V^{B}, \mathcal{O}^{B}, \Theta^{B}, \rho^{B}, \mathcal{J}^{B}, \emptyset\rangle$ equivalent to $\mathcal{D}$, as follows:

- $V^{B} = V \cup \{n\_p_i : \textbf{boolean} \mid \langle p_i, q_i\rangle \in \mathcal{C}\} \cup \{x_c\}$.
  That is, for every compassion requirement $\langle p_i, q_i\rangle \in \mathcal{C}$, we add to V^{B} a boolean variable n_p_i. Variable n_p_i is a *prediction variable* intended to turn true at a point in a computation from which the assertion p_i remains false forever. Variable x_c, common to all compassion requirements, is intended to turn true at a point in a computation satisfying $\bigvee_{i=1}^{m}(p_i\ \wedge\ n\_p_i)$, which indicates an instance of *mis-prediction*.
- $\mathcal{O}^{B} = \mathcal{O}$
- $\Theta^{B} = \Theta\ \wedge\ x_c = 0\ \wedge \displaystyle\bigwedge_{\langle p_i, q_i\rangle \in \mathcal{C}} n\_p_i = 0$
  That is, initially all the newly introduced boolean variables are set to zero.
- $\rho^{B} = \rho\ \wedge\ \rho_{n\_p}\ \wedge\ \rho_c$, where

$$\rho_{n\_p} : \bigwedge_{\langle p_i, q_i\rangle \in \mathcal{C}} (n\_p_i \rightarrow n\_p_i')$$

$$\rho_c \quad : x_c' = \left(x_c\ \vee \bigvee_{\langle p_i, q_i\rangle \in \mathcal{C}} (p_i\ \wedge\ n\_p_i)\right)$$

  The augmented transition relation allows each of the n_p_i variables to change non-deterministically from 0 to 1. Variable x_c is set to 1 on the first occurrence of $p_i\ \wedge\ n\_p_i$, for some i, $1 \le i \le m$. Once set, it is never reset.
- $\mathcal{J}^{B} = \mathcal{J} \cup \{\neg x_c\} \cup \{n\_p_i\ \vee\ q_i \mid \langle p_i, q_i\rangle \in \mathcal{C}\}$
  The augmented justice set contains the additional justice requirement $n\_p_i\ \vee\ q_i$ for each $\langle p_i, q_i\rangle \in \mathcal{C}$. This requirement demands that either n_p_i turns true sometime, implying that p_i is continuously false from that time on, or q_i holds infinitely often.
  The justice requirement $\neg x_c$ ensures that a run with one of the variables n_p_i set prematurely, is not accepted as a computation.

The transformation of an FDS to a JDS follows the transformation of Streett automata to generalized Büchi automata (see [5] for finite state automata and [38] for infinite state automata). For completeness of presentation, we include in Appendix A the proof that $Obs(\mathcal{D}) = Obs(\mathcal{D}^{B})$.

3 The Open View of a System

Our main motivation for considering the problem of abstraction is the method of *verification by network-invariants* [17,20,40], aimed at the verification of *parame-*

terized systems.

A parameterized system has the general form $S(n) : P[1] \| \cdots \| P[n]$ and represents an infinite family of systems, one for each value of the parameter $n > 1$. We are interested in the *uniform verification* of this family, showing that the system $S(n)$ satisfies the property p for every value of $n > 1$. In order to ensure the soundness of the network-invariant method, it is necessary to have a *compositional abstraction*, i.e. a notion of abstraction such that $P \sqsubseteq Q$ implies $(P \| R) \sqsubseteq (Q \| R)$ [17]. To obtain this kind of abstraction, it is necessary to formulate a different notion of computation which can be applied to a component (process) in a system rather than to the entire system.

The standard definition of a computation of a program views the entire program as a *closed system*. When studying a process or a component of a system we need an *open-system* view. In order to enable an open view of processes within a bigger system, we identify for each process the variables *owned* by a process. These are the variables that only the process itself (never the environment) can modify.

Thus, we define a *fair discrete module* (FDM) to be given by $M = \langle V, W, \mathcal{O}, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ where the components $V, \mathcal{O}, \Theta, \mathcal{J}, \mathcal{C}$ are defined as for an FDS, and the added component is

- $W \subseteq V$ – A set of *owned variables*. These are variables which can only be modified by the module itself but not by its environment. By default, all the variables in $V - W$ can potentially be modified by the environment.

Open Computations and Observations

Let $M : \langle V, W, \mathcal{O}, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ be an FDM. An *open computation* of M is an infinite sequence of states

$$\sigma : s_0, s_1, s_2, ...,$$

satisfying the following requirements:

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For each $j = 0, 1, ...,$
  - $s_{2j+1}[W] = s_{2j}[W]$. That is, s_{2j+1} and s_{2j} agree on the interpretation of the owned variables W.
  - s_{2j+2} is a ρ-successor of s_{2j+1}.
- *Justice* and *Compassion:* As before.

Thus, an open computation of a module consists of alternating environment and module steps. An environment step, always applied to evenly indexed states, only guarantees to preserve the owned variables. A module step, always applied to oddly indexed states, must obey the transition relation ρ. In a *closed* system $W = V$, i.e., all variables are owned by the system and environment moves cannot be distinguished from idling moves.

We also provide a *restriction* operation, which moves a specified variable to the category of owned variables and makes it non-observable. We denote by [**local** x**;** $\mathcal{D}$] the system obtained by restricting variable x in system $\mathcal{D}$.

Two FDM's M_1 and M_2 are composable if $W_1 \cap W_2 = \emptyset$ and $V_1 \cap V_2 \subseteq O_1 \cap O_2$. The asynchronous parallel composition of two composable FDM's $M = M_1 \parallel M_2$ is defined similarly to the composition of two FDS's where, in addition, the owned variables of the newly formed module is obtained as the union of W_1 and W_2. The FDM M_2 is said to be an *abstraction* of a comparable FDM M_1, denoted $M_1 \sqsubseteq_M M_2$, if $\mathit{Obs}(M_1) \subseteq \mathit{Obs}(M_2)$.

Binary Processes

We define a *binary process* $Q(\vec{x}, \vec{y})$ to be a process with two ordered sequences of observable variables $\vec{x}$ and $\vec{y}$. When $\vec{x}$ and $\vec{y}$ consist of a single variable we use the notation $Q(x, y)$. Two binary processes Q and R can be composed to yield another binary process, using the *modular composition* operator $\circ$ defined by

$$(Q \circ R)(\vec{x}, \vec{z}) \quad = \quad [\textbf{local}\ \vec{y};\ Q(\vec{x}, \vec{y}) \parallel R(\vec{y}, \vec{z})]$$

Binary processes $P_1, \ldots, P_m$ can be composed into a closed ring structure (having no observables) defined by

$$(P_1 \circ \cdots \circ P_m \circ) = [\textbf{local}\ \vec{x}_1, \ldots, \vec{x}_m;\ \ P_1(\vec{x}_1, \vec{x}_2) \parallel \cdots \parallel P_m(\vec{x}_m, \vec{x}_1)]$$

The dangling $\circ$ denotes that process P_m is composed with P_1. We are interested in parameterized systems of the form $P(n) = [P_1 \circ \cdots \circ P_n \circ]$, where each P_i is a finite state binary process. Such a system represents in fact an infinite *family* of systems (one for each value of n). Our objective is to verify uniformly (i.e., for every value of $n > 1$) that a property p is valid. For simplicity of presentation, assume that the property p only refers to the observable variables of P_1 and that processes $P_2, \ldots, P_{n-1}$ are identical (up to renaming) and can be represented by the generic binary process Q. That is, $P_2(\vec{x}, \vec{y}) = \cdots = P_{n-1}(\vec{x}, \vec{y}) = Q(\vec{x}, \vec{y})$.

The *network invariants method* can be summarized as follows:

(1) Devise a *network invariant* $\mathcal{I} = \mathcal{I}(\vec{x}, \vec{y})$, which is an FDS intended to provide an abstraction for the (open) parallel composition $Q^n = \underbrace{Q \circ \cdots \circ Q}_{n}$ for any $n \geq 2$.
(2) Confirm that $\mathcal{I}$ is indeed a network invariant, by establishing that $Q \sqsubseteq_{M} \mathcal{I}$ and $(Q \circ \mathcal{I}) \sqsubseteq_{M} \mathcal{I}$.
(3) Model check $(Q \circ \mathcal{I} \circ P_n \circ) \models p$.

As presented here, the rule is adequate for proving properties of P_1. Another typical situation is when we wish to prove properties of a generic P_j for $j < N$. In this case, we model check in step 3 that $(\mathcal{I} \circ Q \circ \mathcal{I} \circ P_n \circ) \models p$.

4 Fair Simulation and Simulation Games

We already defined the notion of observations of an FDS and the notions of equivalence ($\sim$) and preorder ($\sqsubseteq$) with respect to observations. There are also other notions of equivalence / preorder for systems that consider the possible branching in every state. The main notion of preorder between two systems, considering branching, is *simulation* [30]. We say that state s of $\mathcal{D}_{A}$ simulates state t of $\mathcal{D}_{C}$ if they are observationally equivalent and for every transition of $\mathcal{D}_{C}$ to s' there exists a transition of $\mathcal{D}_{A}$ to t' such that t' simulates s'. Formally we have the following.

Let $\mathcal{D}_{C} : \langle V_{C}, \mathcal{O}_{C}, \Theta_{C}, \rho_{C}, \mathcal{J}_{C}, \mathcal{C}_{C} \rangle$ and $\mathcal{D}_{A} : \langle V_{A}, \mathcal{O}_{A}, \Theta_{A}, \rho_{A}, \mathcal{J}_{A}, \mathcal{C}_{A} \rangle$ be two comparable FDS's. Let Σ_{C} and Σ_{A} denote the set of states of $\mathcal{D}_{C}$ and $\mathcal{D}_{A}$ respectively. A relation $R \subseteq \Sigma_{C} \times \Sigma_{A}$ is a *simulation* relation between $\mathcal{D}_{C}$ and $\mathcal{D}_{A}$ if for every pair $\langle s, t \rangle$ the following hold.

(1) $s{\Downarrow}_{\mathcal{O}_{C}} = t{\Downarrow}_{\mathcal{O}_{A}}$.
(2) For every state s' such that $(s, s') \models \rho_{C}$ there exists a state t' such that $(t, t') \models \rho_{A}$ and $\langle s', t' \rangle \in R$.

We say that $\mathcal{D}_{A}$ *simulates* $\mathcal{D}_{C}$ if there exists a simulation relation R between $\mathcal{D}_{C}$ and $\mathcal{D}_{A}$ and for every initial state $s \in \Sigma_{C}$ there exists an initial state $t \in \Sigma_{A}$ such that $\langle s, t \rangle \in R$.

Note that the notation (s, s') is used for two states s and s' of the same structure where s is interpreted over the variables and s' over the primed copy of the variables. The notation $\langle s, t \rangle$ is used to bound together two states from (possibly) different structures for the purpose of simulation or (in what follows) to create a state of a game structure.

Simulation can be defined also by means of two player games. We define a game structure whose locations are pairs of states from $\mathcal{D}_C$ and $\mathcal{D}_A$. The game is played between two players A and C. Player A tries to show that $\mathcal{D}_A$ simulates $\mathcal{D}_C$, while player C tries to show that this is not the case. We establish that $\mathcal{D}_A$ simulates $\mathcal{D}_C$ by proving that player A wins the game. From a pair $\langle s,t\rangle$ the play proceeds by player C choosing a successor of s and then player A choosing a successor of t. The play ends if the play reaches a location where s and t do not agree on the values of the observable variables, in which case player C wins. Player A wins if the play goes ad-infinitum. We say that $\mathcal{D}_A$ simulates $\mathcal{D}_C$ if for every initial state $s \in \Sigma_C$ there exists an initial state $t \in \Sigma_A$ such that from $\langle s,t\rangle$ player A can win the game. One could verify that the game semantics of simulation is equivalent to the semantics given above [13].

The problem with simulation is that it does not account for fairness. There are many suggestions how to extend simulation to account for fairness [25,10,13,14]. We choose the definition of [13] and denote it as *fair-simulation*. In order to formally define fair-simulation we first give a definition of games.

Let $\mathcal{D}_C : \langle V_C, \mathcal{O}_C, \Theta_C, \rho_C, \mathcal{J}_C, \mathcal{C}_C\rangle$ and $\mathcal{D}_A : \langle V_A, \mathcal{O}_A, \Theta_A, \rho_A, \mathcal{J}_A, \mathcal{C}_A\rangle$ be two comparable FDS's, i.e. there is a bijection $b : \mathcal{O}_C \to \mathcal{O}_A$ and $V_C \cap V_A = \emptyset$. We denote by Σ_C and Σ_A the sets of states of $\mathcal{D}_C$ and $\mathcal{D}_A$ respectively. We define the *simulation game structure* (SGS) associated with $\mathcal{D}_C$ and $\mathcal{D}_A$ to be the tuple $G : \langle \mathcal{D}_C, \mathcal{D}_A\rangle$. A *state* of G is a type-consistent interpretation of the variables in $V_C \cup V_A$. We denote by Σ_G the set of states of G. We say that a state $s \in \Sigma_G$ is *correlated*, if $s{\Downarrow}_{\mathcal{O}_C} = s{\Downarrow}_{\mathcal{O}_A}$. We denote by $\Sigma_{cor} \subset \Sigma_G$ the subset of correlated states.

For two states s and s' of G, s' is an A-successor of s if $(s,s') \models \rho_A$ and $s{\Downarrow}_{V_C} = s'{\Downarrow}_{V_C}$. Similarly, s' is a C-successor of s if $(s,s') \models \rho_C$ and $s{\Downarrow}_{V_A} = s'{\Downarrow}_{V_A}$. A *play* σ of G is a maximal sequence of states $\sigma : s_0, s_1, \ldots$ satisfying the following:

- *Consecution:* For each $j \geq 0$,
  - s_{2j+1} is a C-successor of s_{2j}.
  - s_{2j+2} is an A-successor of s_{2j+1}.
- *Correlation:* For each $j \geq 0$,
  - $s_{2j} \in \Sigma_{cor}$.

Let G be an SGS and σ be a play of G. The play σ can be viewed as a play of a two player game. Player C, represented by $\mathcal{D}_C$, taking ρ_C transitions from even numbered states and player A, represented by $\mathcal{D}_A$, taking ρ_A transitions from odd numbered states. The observations of the two players are correlated on all even numbered states of σ.

A play σ is *winning for player A* if it is infinite and either $\sigma \Downarrow_{V_C}$ is not a computation of $\mathcal{D}_C$ or $\sigma \Downarrow_{V_A}$ is a computation of $\mathcal{D}_A$, i.e. if $\sigma \models \mathcal{F}_C \rightarrow \mathcal{F}_A$, where for $\eta \in \{A, C\}$,

$$\mathcal{F}_\eta : \bigwedge_{J \in \mathcal{J}_\eta} \Box \Diamond J \wedge \bigwedge_{\langle p,q \rangle \in \mathcal{C}_\eta} (\Box \Diamond p \rightarrow \Box \Diamond q).$$

Otherwise, σ is *winning for player C*.

Let D_A and D_C be some finite domains, intended to record facts about the past history of a computation (serve as a memory). A *strategy* for player A is a partial function $f : D_A \times \Sigma_G \mapsto D_A \times \Sigma_{cor}$ such that if $f(d, s) = (d', s')$ then s' is an A-successor of s. If $|D_A| = 1$, we say that f is memoryless and write $f : \Sigma_G \mapsto \Sigma_{cor}$. Let f be a strategy for player A, and $s_0 \in \Sigma_{cor}$. A play $s_0, s_1, \ldots$ is said to be *compliant* with strategy f if there exists a sequence of D_A-values $d_0, d_2, \ldots, d_{2j}, \ldots$ such that $(d_{2j+2}, s_{2j+2}) = f(d_{2j}, s_{2j+1})$ for every $j \geq 0$. Strategy f is *winning* for player A from state $s \in \Sigma_{cor}$ if all s-plays (plays departing from s) which are compliant with f are winning for A. We denote by W_A the set of states from which there exists a winning strategy for player A. A *strategy* for player C is a partial function $f : D_C \times \Sigma_{cor} \mapsto D_C \times \Sigma_G$ such that if $f(d, s) = (d', s')$ then s' is a C-successor of s. Memoryless strategy, play compliant with strategy, winning strategy, and winning set (W_C) are defined dually to the above.

An SGS G is called *determinate* if the sets W_A and W_C define a partition on Σ_{cor}. It is well known that every SGS is determinate [11].

We are now ready to define fair-simulation as in [13]. Just like simulation, fair-simulation is defined via a game where player A tries to establish fair-simulation while player C tries to falsify it. Given $\mathcal{D}_C$ and $\mathcal{D}_A$, we form the SGS $G : \langle \mathcal{D}_C, \mathcal{D}_A \rangle$. We say that $S \subseteq \Sigma_{cor}$ is a *fair-simulation* between $\mathcal{D}_A$ and $\mathcal{D}_C$ if there exists a strategy f for player A such that every f-compliant play σ from a state $s \in S$ is winning for player A and every even state in σ is in S. We say that $\mathcal{D}_A$ *fairly-simulates* $\mathcal{D}_C$, denoted $\mathcal{D}_C \preceq_f \mathcal{D}_A$, if there exists a fair-simulation S such that for every state $s \in \Sigma_C$ satisfying $s \models \Theta_C$ there exists a state $t \in S$ such that $t \Downarrow_{V_C} = s$ and $t \models \Theta_A$.

5 μ-Calculus over Game Structures

We define μ-calculus [21] over game structures. Consider two FDS's $\mathcal{D}_C : \langle V_C, \mathcal{O}_C, \Theta_C, \rho_C, \mathcal{J}_C, \mathcal{C}_C \rangle$, $\mathcal{D}_A : \langle V_A, \mathcal{O}_A, \Theta_A, \rho_A, \mathcal{J}_A, \mathcal{C}_A \rangle$ and the SGS $G : \langle \mathcal{D}_C, \mathcal{D}_A \rangle$. For every variable $v \in V_C \cup V_A$ the formulas $v = u$ and $v = i$ where u and v are type consistent and i is a constant that is type consistent with v are *atomic formulas* (denoted p below). Let $V = \{X, Y, \ldots\}$ be a set of *relational variables*. The μ-

calculus formulas are constructed as follows.

$$\varphi ::= p \mid \neg p \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Diamond \varphi \mid \Box \varphi \mid \mu X \varphi \mid \nu X \varphi$$

A formula φ is interpreted as the set of states in Σ_{cor} in which φ is true. We write such set of states as $[[\varphi]]^e_G$ where G is the SGS and $e : V \to 2^{\Sigma_{cor}}$ is an *environment*. The environment assigns to each relational variable a subset of Σ_{cor}. We denote by $e[X \leftarrow S]$ the environment such that $e[X \leftarrow S](X) = S$ and $e[X \leftarrow S](Y) = e(Y)$ for $Y \neq X$. The set $[[\varphi]]^e_G$ is defined inductively as follows [5].

- $[[p]]^e_G = \{s \in \Sigma_{cor} \mid s \models p\}$
- $[[\neg p]]^e_G = \{s \in \Sigma_{cor} \mid s \not\models p\}$
- $[[X]]^e_G = e(X)$
- $[[\varphi \vee \psi]]^e_G = [[\varphi]]^e_G \cup [[\psi]]^e_G$
- $[[\varphi \wedge \psi]]^e_G = [[\varphi]]^e_G \cap [[\psi]]^e_G$
- $[[\Diamond \varphi]]^e_G = \left\{ s \in \Sigma_{cor} \;\middle|\; \begin{array}{l} \forall s',\ (s, s') \models \rho_C \to \exists s'' \text{ such that } (s', s'') \models \rho_A \\ \text{and } s'' \in [[\varphi]]^e_G \end{array} \right\}$
  A state s is included in $[[\Diamond \varphi]]^e_G$ if player A can force the play to reach a state in $[[\varphi]]^e_G$. That is, regardless of how player C moves from s, player A can choose an appropriate move into $[[\varphi]]^e_G$.
- $[[\Box \varphi]]^e_G = \left\{ s \in \Sigma_{cor} \;\middle|\; \begin{array}{l} \exists s' \text{ such that } (s, s') \models \rho_C \text{ and} \\ \forall s'',\ (s', s'') \models \rho_A \to s'' \in [[\varphi]]^e_G \end{array} \right\}$
  A state s is included in $[[\Box \varphi]]^e_G$ if player C can force the play to reach a state in $[[\varphi]]^e_G$. As player C moves first, she chooses a C-successor of s all of whose A-successors are in $[[\varphi]]^e_G$.
- $[[\mu X \varphi]]^e_G = \cup_i S_i$ where $S_0 = \emptyset$ and $S_{i+1} = [[\varphi]]^{e[X \leftarrow S_i]}_G$
- $[[\nu X \varphi]]^e_G = \cap_i S_i$ where $S_0 = \Sigma_{cor}$ and $S_{i+1} = [[\varphi]]^{e[X \leftarrow S_i]}_G$

When all the variables in φ are bound by either μ or ν the initial environment is not important and we simply write $[[\varphi]]_G$. In case that G is clear from the context we simply write $[[\varphi]]$.

In our definition we allow applying negation only to atomic formulas (positive normal form). We can convert a μ-calculus formula with negations to positive normal form by using de-Morgan rules and replacing $\neg(\mu Y\ f(Y))$ by $\nu Y\ \neg f(\neg Y)$, replacing $\neg(\nu Y\ f(Y))$ by $\mu Y\ \neg f(\neg Y)$, replacing $\neg \Diamond f$ by $\Box \neg f$, and replacing $\neg \Box f$ by $\Diamond \neg f$.

Consider for example an SGS $G : \langle \mathcal{D}_C, \mathcal{D}_A \rangle$ and the formula $\varphi = \nu X(\Diamond X)$. A state $s \in \Sigma_{cor}$ is in $[[\nu X(\Diamond X)]]$ if $s \Downarrow_{V_A}$ simulates $s \Downarrow_{V_C}$. Indeed, player A can force the game to another state in $[[\nu X(\Diamond X)]]$ an so on ad-infinitum.

[5] Only for finite game structures.

The complement $\neg\varphi = \mu X(\circledcirc X)$ characterizes the set of states where simulation does not hold. Indeed, player C can force the game in a finite number of steps to the set $[[\circledcirc X]]^{e[X \leftarrow \emptyset]}$. A state s is in $[[\circledcirc X]]^{e[X \leftarrow \emptyset]}$ if it has some C-successor s' such that all the A-successors of s' are not correlated.

The *alternation depth* of a formula is the number of alternations in the nesting of least and greatest fixpoints. A μ-calculus formula defines a symbolic algorithm for computing $[[\varphi]]$ [9]. For a μ-calculus formula of alternation depth k, the run time of this algorithm is $O(|\Sigma_{cor}|^k)$. For a full exposition of μ-calculus we refer the reader to [7]. We often abuse notations and write a μ-calculus formula φ instead of the set $[[\varphi]]$.

In some cases, instead of using a very complex formula, it may be more readable to use *vector notation* as in Equation (1) below.

$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \begin{bmatrix} \mu Y(\diamondsuit Y \ \vee \ p \wedge \diamondsuit Z_2) \\ \mu Y(\diamondsuit Y \ \vee \ q \wedge \diamondsuit Z_1) \end{bmatrix} \tag{1}$$

Such a formula, may be viewed as the mutual fixpoint of the variables Z_1 and Z_2 or equivalently as an equal formula where a single variable Z replaces both Z_1 and Z_2 and ranges over pairs of states [24]. The formula above characterizes the set of states from which player A can force the game to visit p-states infinitely often and q-states infinitely often. We can characterize the same set of states by the following 'normal' formula [6].

$$\varphi = \nu Z\left([\mu Y(\diamondsuit Y \ \vee \ p \wedge \diamondsuit Z)] \ \wedge \ [\mu Y(\diamondsuit Y \ \vee \ q \wedge \diamondsuit Z)]\right)$$

6 Trace Inclusion and Fair Simulation

In the following, we summarize our solution to verifying abstraction between two FDS's, or equivalently, trace inclusion between two Streett automata.

Let $\mathcal{D}_C : \langle V_C, \mathcal{O}_C, \Theta_C, \rho_C, \mathcal{J}_C, \mathcal{C}_C \rangle$ and $\mathcal{D}_A : \langle V_A, \mathcal{O}_A, \Theta_A, \rho_A, \mathcal{J}_A, \mathcal{C}_A \rangle$ be two comparable FDS's. We want to verify that $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$. One solution to solve abstraction is by complementing the abstract system [33]. Let $\Pi_{\mathcal{O}_A}$ denote the set of possible assignments to the variables in $\mathcal{O}_A$. Then, we need to construct an FDS $\mathcal{D}_{\overline{A}}$ such that $Obs(\mathcal{D}_{\overline{A}}) = (\Pi_{\mathcal{O}_A})^\omega \setminus Obs(\mathcal{O}_A)$. It follows that $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$ iff $Obs(\mathcal{D}_C \,|||\, \mathcal{D}_{\overline{A}}) = \emptyset$. The problem with this approach is that the algorithm of [33] is exponential and

[6] This does not suggest a canonical translation from vector formulas to plain formulas. The same translation works for the formula in Equation (2) in Section 6. Note that the formula in Equation (1) and the formula in Equation (2) have a very similar structure.

hence impractical. We therefore advocate to verify fair simulation [13] as a precondition for abstraction.

Claim 2 [13] *If $\mathcal{D}_C \preceq_f \mathcal{D}_A$ then $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$. The reverse implication does not hold.*

It is shown in [13] that we can determine whether $\mathcal{D}_C \preceq_f \mathcal{D}_A$ by computing the set $W_A \subseteq \Sigma_{cor}$ of states which are winning for A in the SGS G: $\langle \mathcal{D}_C, \mathcal{D}_A \rangle$. If for every state $s_C \in \Sigma_c$ satisfying $s_C \models \Theta_C$ there exists some state $t \in W_A$ such that $t \Downarrow_{V_C} = s_C$ and $t \models \Theta_A$, then $\mathcal{D}_C \preceq_f \mathcal{D}_A$.

Let $n = |\mathcal{C}_C|$ (number of compassion requirements of $\mathcal{D}_C$), $m = |\mathcal{C}_A|$, $k = |\Sigma_C| \cdot |\Sigma_A| \cdot (3^m + n)$, and $h = 2m + n$.

Theorem 3 [13,23] *We can solve fair simulation for $\mathcal{D}_C$ and $\mathcal{D}_A$ in time $O(k^{2h+1} \cdot h!)$.*

As we are interested in fair simulation as a precondition for trace inclusion, we take a more economic approach. Given two FDS's, we first convert the two to JDS's using the construction in Section 4. We then solve the simulation game for the two JDS's.

Consider the FDS's $\mathcal{D}_C$ and $\mathcal{D}_A$. Let $\mathcal{D}_C^B : \langle V_C^B, \mathcal{O}_C^B, \Theta_C^B, \rho_C^B, \mathcal{J}_C^B, \emptyset \rangle$ and $\mathcal{D}_A^B : \langle V_A^B, \mathcal{O}_A^B, \Theta_A^B, \rho_A^B, \mathcal{J}_A^B, \emptyset \rangle$ be the JDS's equivalent to $\mathcal{D}_C$ and $\mathcal{D}_A$. Consider the game $G : \langle \mathcal{D}_C^B, \mathcal{D}_A^B \rangle$. The winning condition for this game is:

$$\bigwedge_{J_C \in \mathcal{J}_C^B} J_C \rightarrow \bigwedge_{J_A \in \mathcal{J}_A^B} J_A$$

We call such games *generalized Streett*[1] *games*. From here forward when we say game we mean generalized Streett[1] game. Let $\mathcal{D}_C$: $\langle V_C, O_C, \Theta_C, \rho_C, \mathcal{J}_C, \emptyset \rangle$ and $\mathcal{D}_A$: $\langle V_A, O_A, \Theta_A, \rho_A, \mathcal{J}_A, \emptyset \rangle$ be two JDSs where $\mathcal{J}_C = \{J_1^C, \ldots, J_m^C\}$ and $\mathcal{J}_A = \{J_1^A, \ldots, J_n^A\}$. Let G: $\langle \mathcal{D}_C, \mathcal{D}_A \rangle$ be the simulation game structure associated with $\mathcal{D}_C$ and $\mathcal{D}_A$. We claim that the formula in Equation (2) evaluates the set W_A of states winning for player A. Intuitively, for $i \in [1..n]$ and $j \in [1..m]$ the greatest fixpoint $\nu X(J_i^A \wedge \diamondsuit Z_{i \oplus 1} \ \vee \ \diamondsuit Y \ \vee \ \neg J_j^C \wedge \diamondsuit X)$ characterizes the set of states from which player A can force the play either to stay indefinitely in $\neg J_j^C$ states (thus violating the fairness of $\mathcal{D}_C$) or in a finite number of steps reach a state in the set $J_i^A \wedge \diamondsuit Z_{i \oplus 1} \vee \diamondsuit Y$. The two outer fixpoints make sure that player A wins from the set $J_i^A \wedge \diamondsuit Z_{i \oplus 1} \ \vee \ \diamondsuit Y$. The least fixpoint μY makes sure that the unconstrained phase of a play represented by the disjunct $\diamondsuit Y$ is finite and ends in a $J_i^A \wedge \diamondsuit Z_{i \oplus 1}$ state. Finally, the greatest fixpoint νZ_i is responsible to make sure that after visiting J_i^A we can loop and visit $J_{i \oplus 1}^A$ and so on. By the cyclic dependence of the outermost greatest fixpoint, either all the sets in $\mathcal{J}_A$ are visited or getting stuck in some inner greatest fixpoint, some set in $\mathcal{J}_C$ is visited finitely often.

$$\varphi = \nu \begin{bmatrix} Z_1 \\ \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{j=1}^{m} \nu X (J_1^A \wedge \diamondsuit Z_2 \vee \diamondsuit Y \vee \neg J_j^C \wedge \diamondsuit X) \right) \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X (J_2^A \wedge \diamondsuit Z_3 \vee \diamondsuit Y \vee \neg J_j^C \wedge \diamondsuit X) \right) \\ \vdots \\ \vdots \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X (J_n^A \wedge \diamondsuit Z_1 \vee \diamondsuit Y \vee \neg J_j^C \wedge \diamondsuit X) \right) \end{bmatrix} \tag{2}$$

Claim 4 $W_A = [[\varphi]]$

We show first that player A wins from every state in $[[\varphi]]$. We define N strategies for player A. The strategy f_i is defined on the states in Z_i. We show that the strategy f_i either forces the play to visit J_i^A and then proceed to $Z_{i\oplus 1}$, or eventually avoids some $J \in \mathcal{J}_C$. We show that by combining these strategies, either player A switches strategies infinitely many times and ensures that the play be winning according to the fairness of $\mathcal{D}_A$ or eventually uses a fixed strategy ensuring that the play does not satisfy the fairness of $\mathcal{D}_C$. In the other direction we show that in every stage of the computation, the value of Z_i (for all i) is an over approximation of W_A. Specifically, we show that when $Z_{i\oplus 1}$ is an over approximation of W_A, then even states winning for player A in a simpler game (i.e., winning in the simulation game implies winning in the simple game) are maintained in Z_i. The full proof of the claim is presented in Appendix B.

Using the algorithm in [9] the set $[[\varphi]]$ can be evaluated symbolically.

Theorem 5 *A generalized Streett[1] game G can be solved by a symbolic algorithm in time $O((|\Sigma_C^B| \cdot |\Sigma_A^B| \cdot |\mathcal{J}_C^B| \cdot |\mathcal{J}_A^B|)^3)$.*

PROOF. From Claim 4 it follows that the formula φ in Equation (2) computes the set of winning states in G. Using the symbolic algorithm of [9] we can compute the set of states that satisfy φ in time $O((|\Sigma_C^B| \cdot |\Sigma_A^B| \cdot |\mathcal{J}_C^B| \cdot |\mathcal{J}_A^B|)^3)$.

We note that using the algorithm in [15] the same set of states can be evaluated in time $O((|\Sigma_C^B| \cdot |\Sigma_A^B| \cdot |\mathcal{J}_C^B| \cdot |\mathcal{J}_A^B|)^2)$. However, Jurdzinski's algorithm cannot be implemented symbolically. Also the algorithms in [26,34] work in quadratic time rather than cubic time. Both can be implemented symbolically. Seidl's algorithm requires automatic modification of the μ-calculus formula which our tools do not support. The algorithm of Long et al. requires storing intermediate results of the fixpoint computation and using them in later stages of the computation. For a μ-calculus formula of alternation depth 3 the memory management is not complicated. We implemented the algorithm of [26]. On our examples, the algorithm of [26] shortens the run time in about 10% (vs. [9]). This is probably due to the fact that there are only a few iterations of the outer most fixpoint until convergence. To summarize, in order to use fair simulation as a precondition for trace inclusion we propose to convert every FDS into a JDS and use the formula in Equation (2) to evaluate symbolically the winning set for player A.

Corollary 6 *Given $\mathcal{D}_C$ and $\mathcal{D}_A$, we can determine using a symbolic algorithm whether $\mathcal{D}_C^B \preceq_f \mathcal{D}_A^B$ in time proportional to $O((|\Sigma_C| \cdot |\Sigma_A| \cdot 2^{n+m} \cdot (n + |\mathcal{J}_C| + m + |\mathcal{J}_A|))^3)$.*

7 Closing the Gap

As presented in Claim 2, fair simulation implies trace inclusion but not the other way around. In [1], a notion of fair simulation is considered in the context of infinite-state systems. It is easy to see that the definition of *fair simulation* given in [1], implies fair simulation according to the definition in [13]. As shown in [1], if we are allowed to add to the concrete system auxiliary *history* and *prophecy* variables, then the fair simulation method becomes complete for verifying trace inclusion. Similarly, for finite state systems, we prove that there exists a non-constraining FDS with respect to the concrete system that can be composed synchronously with the concrete system, making the method complete for checking refinement. The proof is based on using the abstract system. In practice, if we have to augment the concrete system, we find that in many realistic examples a simple FDS can be used, or even an LTL tester [18]. For example, the simple EARLY and LATE example requires the temporal tester for the LTL formula $\diamondsuit(x = 2)$. Dining-philosophers (see Section 9), on the other hand, does not require augmentation at all. We expect the user to devise this FDS.

Theorem 7 *Let $\mathcal{D}_C$ and $\mathcal{D}_A$ be two comparable FDS's such that $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$. Then there exists an FDS $\mathcal{D}_D$ that is non-constraining with respect to $\mathcal{D}_C$ such that $(\mathcal{D}_C \,|||\, \mathcal{D}_D) \preceq_f \mathcal{D}_A$.*

We first show that in order to establish Theorem 7 we must work with viable FDS. Consider the JDS's in Fig. 2. The double cycle represents a fair state. While both systems have the same set of traces, the system on the right cannot simulate the system on the left. In a way, the concrete system willingly enters a state that is unfair, however the abstract system cannot follow. This seems to be a 'technical difficulty' that stops us from proving fair-simulation. There are two ways in which we can solve this problem. We can either remove unfeasible states from both systems [7], or we can add an unfair sink component [8] to the abstract system and add an option to move to this sink component from every state of the abstract system. We choose the first option and assume that the first step of establishing fair-simulation is to remove the set of unfeasible states from both systems [19].

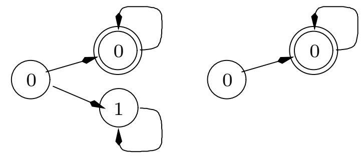


Fig. 2. Removal of unfeasible states.

PROOF. Let $\mathcal{D}_C: \langle V_C, \mathcal{O}_C, \Theta_C, \rho_C, \mathcal{J}_C, \mathcal{C}_C \rangle$ and $\mathcal{D}_A: \langle V_A, \mathcal{O}_A, \Theta_A, \rho_A, \mathcal{J}_A, \mathcal{C}_A \rangle$ be two comparable FDS (i.e., there is a bijection $b : \mathcal{O}_C \to \mathcal{O}_A$ and $V_C \cap V_A = \emptyset$).

Let $b : \mathcal{O}_C \to \mathcal{O}_A$ be the bijection between the observable variables of $\mathcal{D}_C$ and the observable variables of $\mathcal{D}_A$. Consider a copy $\mathcal{D}_D$ of $\mathcal{D}_A$ where the variables in V_A are renamed as follows. Every variable $v \in \mathcal{O}_A$ is renamed to $b(v) \in \mathcal{O}_C$ and every variable $v \in (V_A \setminus \mathcal{O}_A)$ is renamed $\dot{v}$. Accordingly, we adapt ρ_D, Θ_C, $\mathcal{J}_D$, and $\mathcal{C}_D$ according to this renaming scheme. Clearly, $\mathcal{D}_C$ and $\mathcal{D}_D$ have the same set of observable variables and hence are composable and can be synchronuously composed.

It is straight forward to see that $\mathcal{D}_D$ is non-constraining with respect to $\mathcal{D}_C$. Indeed, consider an observation $\pi \in Obs(\mathcal{D}_C)$. From the fact that $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$ it follows that there exists a computation σ of $\mathcal{D}_A$ such that $\sigma \Downarrow_{\mathcal{O}_A} = \pi$. We convert the computation σ to a computation $\dot{\sigma}$ of $\mathcal{D}_D$ according to the renaming of variables above. Obviously π is also an observation of $\mathcal{D}_D$.

Next, we show that $(\mathcal{D}_C \,|||\, \mathcal{D}_D) \preceq_f \mathcal{D}_A$. Let us consider the simulation game $G: \langle (\mathcal{D}_C \,|||\, \mathcal{D}_D), \mathcal{D}_A \rangle$.

Every state $p \in \Sigma_G$ is a pair $p = \langle s, t \rangle$ where s is a state of $\mathcal{D}_C \,|||\, \mathcal{D}_D$, and t is a state of $\mathcal{D}_A$. Let $S \subseteq \Sigma_{cor}$ be a simulation and $f : \Sigma_G \mapsto \Sigma_{cor}$ a memoryless strategy for player A defined as follows. For a state s of $\mathcal{D}_C \,|||\, \mathcal{D}_D$, let $s \Downarrow_{V_A}$ denote the state t of $\mathcal{D}_A$ such that for every $v \in \mathcal{O}_C$ we have $s[v] = t[b(v)]$ and for every

[7] Removing unfeasible states from the abstract system helps us by reducing its size. It does not help to establish fair simulation.
[8] Here, a component would be a set of states that form a clique, a state for every possible assignement to the observable variables. We have to add such a component and not a single state because we have to allow in the abstract system every possible sequence of observations.

$v \in V_A \setminus \mathcal{O}_A$ we have $s[\dot{v}] = t[v]$.

$$S = \{\langle s, t\rangle \mid t = s\Downarrow_{V_A}\} \qquad f(\langle s, t\rangle) = \begin{cases} s\Downarrow_{V_A} & \text{If } (t, s\Downarrow_{V_A}) \models \rho_A \\ \text{undefined} & \text{Otherwise} \end{cases}$$

We show that f is a winning strategy for player A from every state in S. Consider a play $\sigma : p_0, p_1, \ldots$ of G compliant with f. Let $p_i = \langle s^i, t^i\rangle$. We prove that if $p_0 \in S$ then for all $j > 0$ we have $p_{2j} \in S$ and that the strategy is well defined. Suppose that $p_{2j} \in S$. By definition of S, we have $s^{2j}\Downarrow_{V_A} = t^{2j}$. By definition of G, we have $(s^{2j}, s^{2j+1}) \models \rho_D$ and $t^{2j} = t^{2j+1}$. It follows that $(s^{2j}, s^{2j+1}) \models \rho_A$. In particular $(t^{2j+1}, f(p_{2j+1})) \models \rho_A$ and the strategy is well defined. Furthermore, since $f(p_{2j+1}) = s^{2j+1}\Downarrow_{V_A}$ it follows that $s^{2j+2}\Downarrow_{V_A} = t^{2j+2}$ and $p_{2j+2} \in S$.

Since for every j we have $s^{2j}\Downarrow_{V_A} = t^{2j}$ and $\mathcal{D}_D$ contains the justice and compassion requirements of $\mathcal{D}_A$, it follows that every play compliant with f is winning for player A. Finally, for every state $p \in \Sigma_G$ such that $s \models \Theta_C \wedge \Theta_D$ there exists a unique state $p' \in S$ such that $p'\Downarrow_{V_C \cup \dot{V}_D} = p\Downarrow_{V_C \cup \dot{V}_D}$ and $p' \models \Theta_A$.

To summarize, according to Theorem 7 for every two systems $\mathcal{D}_C$ and $\mathcal{D}_A$ such that $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$ we can find an FDS $\mathcal{D}_D$ non-constraining with respect to $\mathcal{D}_C$ such that by augmenting the concrete system with $\mathcal{D}_D$ we can establish fair-simulation. However, in order to prove that $\mathcal{D}_D$ can be used to augment $\mathcal{D}_C$ we have to show that $\mathcal{D}_D$ is non-constraining with respect to $\mathcal{D}_C$. The latter is exactly identical to the original problem we were facing: showing that $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$. In many cases, it makes more sense to use an FDS $\mathcal{D}_D$ that is composable with $\mathcal{D}_C$ and non-constraining (in the general sense). In particular, for an LTL formula φ [31] we can automatically construct a non-constraining FDS T_φ (called *temporal tester*) such that from the states of T_φ we can deduce whether the formula φ and every one of its subformulas is true for the future of the computation or not [37]. Similarly, for every FDS $\mathcal{D}$, we can construct a non-constraining FDS $\mathcal{D}'$ such that from the state of $\mathcal{D}'$ we can deduce whether the future of the computation is an observation of $\mathcal{D}$ or not [36]. In Fig. 5, we give a simple extension to EARLY and LATE that shows that LTL testers are not sufficient and sometimes we need the full power of automata.

8 Examples

The algorithm described in this paper was implemented within the TLV system [32]. TLV is a flexible verification tool implemented at the Weizmann Institute of Science. TLV provides a programming environment which uses OBDDs as its basic data type [4]. Deductive and algorithmic verification methods are implemented as procedures written within this environment. We extended TLV's functionality by

$$\text{EARLY} :: \left[\begin{array}{l} \ell_0 : x, z := \{1,2\}, 1 \\ \ell_1 : z := 2 \\ \ell_2 : y, z := x, 3 \end{array}\right] \qquad \text{LATE} :: \left[\begin{array}{l} \ell_0 : z := 1 \\ \ell_1 : x, z := \{1,2\}, 2 \\ \ell_2 : y, z := x, 3 \end{array}\right]$$

Fig. 3. Programs EARLY and LATE.

implementing the algorithms of [9,26] to evaluate the μ-calculus formula in Section 6. Our program gets two FDS's as input, constructs the appropriate simulation game structure, and evaluates the winning states for player A.

Example 1: Late and Early

Consider the programs EARLY and LATE in Fig. 3 (graphic representation in Fig. 1). The observable variables are y and z. Without loss of generality, assume that the initial values of all variables are 0. This is a well known example showing the difference between trace inclusion and simulation. Indeed, the two systems have the same set of traces. Either y assumes 1 or y assumes 2. On the other hand, EARLY does not simulate LATE. This is because we do not know whether state $\langle \ell_1,\ x{:}0,\ z{:}1\rangle$ of system LATE should be mapped to state $\langle \ell_1,\ x{:}1,\ z{:}1\rangle$ or state $\langle \ell_1,\ x{:}2,\ z{:}1\rangle$ of system EARLY. Our algorithm shows that indeed EARLY does not simulate LATE.

Since EARLY and LATE have the same set of traces, we can augment LATE with a non-constraining FDS that tells EARLY how to simulate it. In this case, we compose program LATE synchronously with a *tester* T_φ for the property $\varphi : \diamondsuit(y = 1)$. The tester introduces a new boolean variable b_φ which is true at a state s iff $s \models \varphi$. Whenever T_φ indicates that LATE will eventually choose $x = 1$, EARLY can safely choose $x = 1$ in the first step. Whenever T_φ indicates that LATE will never choose $x = 1$, EARLY can safely choose $x = 2$ in the first step. Denote by LATE$^+$ the synchronous composition of LATE with T_φ. Applying our algorithm to LATE$^+$ and EARLY, indicates that LATE$^+$ $\preceq_f$ EARLY implying *Obs*(LATE) $\subseteq$ *Obs*(EARLY).

Example 2: Late-count and Early-count

Consider the programs EARLY-COUNT and LATE-COUNT in Fig. 4 (graphic representation in Fig. 5). The difference between the two execution paths in both programs is the number of steps from the initial state to the state marked by 1 which is even in one branch and odd in the other. We present this example to illustrate that although in some cases augmentation by temporal testers is sufficient, in general we need the full power of ω-automata. Since LTL cannot count [39], it is quite obvious that no temporal tester can help EARLY-COUNT simulate LATE-COUNT.

$$
\begin{array}{c} \text{EARLY} \\ \text{count} \end{array} :: \left[\begin{array}{l}
\ell_0 : x := \{1,2\} \\
\ell_1 : \textbf{skip} \\
\ell_2 : \textbf{skip} \\
\ell_3 : \textbf{skip} \\
\ell_4 : \textbf{goto } \{\ell_2, \ell_5\} \\
\ell_5 : \textbf{if } (x = 2) \textbf{ goto } \ell_7 \\
\ell_6 : \textbf{skip} \\
\ell_7 : y := 1 \\
\ell_8 : y := 0
\end{array} \right]
\qquad
\begin{array}{c} \text{LATE} \\ \text{count} \end{array} :: \left[\begin{array}{l}
m_0 : \textbf{skip} \\
m_1 : x := \{1,2\} \\
m_2 : \textbf{skip} \\
m_3 : \textbf{skip} \\
m_4 : \textbf{goto } \{m_2, m_5\} \\
m_5 : \textbf{if } (x = 2) \textbf{ goto } m_7 \\
m_6 : \textbf{skip} \\
m_7 : y := 1 \\
m_8 : y := 0
\end{array} \right]
$$

$$\mathcal{J} : at\_\ell_8 \qquad\qquad \mathcal{J} : at\_m_8$$

Fig. 4. Programs EARLY-COUNT and LATE-COUNT.

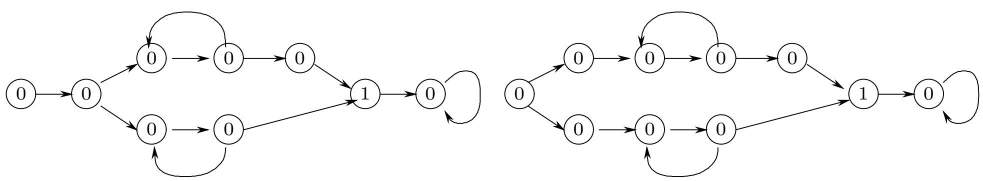
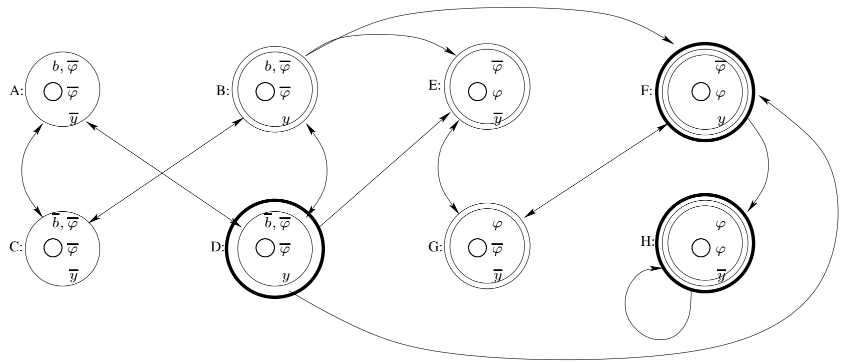


Fig. 5. Systems EARLY-COUNT and LATE-COUNT

Indeed, our algorithm shows that without augmenting LATE-COUNT, simulation does not hold. We can augment LATE-COUNT with the JDS EVEN-ODD presented in Fig. 6. EVEN-ODD tells EARLY-COUNT whether all states in even distance from the current location of (LATE-COUNT $|||$ EVEN-ODD) are 0 states. We obtained EVEN-ODD from the linear μ-calculus [8] formula $\varphi = \nu Z \,.\, ((y = 0 \wedge \bigcirc \bigcirc Z)$. The formula φ holds in state s, if every state t reachable from s in an even number of steps satisfies $y = 0$. The labels on the states of EVEN-ODD represent the Boolean values of φ, $\bigcirc \varphi$, and y, in addition to a Boolean variable b (see below). EVEN-ODD includes two justice requirements $\varphi \vee \bigcirc \varphi \vee (y \wedge b)$ and $(\varphi \wedge \bigcirc \varphi) \vee (y \wedge \overline{b})$. The states satisfying the first requirement are marked by an extra circle and the states satisfying the second requirement by an extra bold circle. The label b results from the translation of the μ-calculus formula φ into a non-constraining JDS [36] [9] . States not containing the value of b stand for $(b \vee \overline{b})$. Our algorithm shows that the synchronous composition of LATE-COUNT with the JDS EVEN-ODD is simulated by EARLY-COUNT as expected.

[9] Vardi's construction consists of translating the μ-calculus formula into a weak alternating automaton [29,23]. Vardi uses the weak alternating automaton and its complement together to create a non-constraining FDS.



$\mathcal{J}\ :\ \{\varphi \vee \bigcirc\varphi \vee (y \wedge b), (\varphi \wedge \bigcirc\varphi) \vee (y \wedge \overline{b})\}$

Fig. 6. JDS EVEN-ODD

(Q^n) where
$Q(\mathit{left}; \mathit{right})$ ::

loop forever do
- ℓ_0 : **NonCritical**
- ℓ_1 : **request** *left*
- ℓ_2 : **request** *right*
- ℓ_3 : **Critical**
- ℓ_4 : **release** *left*
- ℓ_5 : **release** *right*

$\mathcal{I}(\mathit{left}; \mathit{right})$ ::

loop forever do [ℓ_0 : **request** *left* ; ℓ_1 : **release** *left*] $\parallel$ **loop forever do** [m_0 : **request** *right* ; m_1 : **release** *right*]

$\mathcal{J} : \neg at\_m_1 \quad \mathcal{C} : (\mathit{right}, \neg at\_\ell_1)$

Fig. 7. (a) Program DINE. (b) the two halves abstraction.

Example 3: The Dining Philosophers

As a second example, we consider a solution to the dining philosophers problem. As originally described by Dijkstra, n philosophers are seated around a table, with a fork between each two neighbors. In order to eat, a philosopher needs to acquire the forks on both its sides. A solution to the problem consists of protocols to the philosophers (and, possibly, forks) that guarantee that no two adjacent philosophers eat at the same time and that every hungry philosopher eventually eats.

In Fig. 7a we present a chain of n deterministic philosophers, each represented by a binary process $Q(\mathit{left}; \mathit{right})$. This solution is studied in [20] as an example of parametric systems, for which we seek a *uniform* verification (i.e. a single verification valid for any n).

Here, we consider the same invariants, and verify all the necessary abstractions using our algorithm for fair simulation. As in both cases, no augmentation of the concrete system is needed, the algorithmic method is completely automatic.

The "Two-Halves" *Abstraction*

The first network invariant $\mathcal{I}(\mathit{left}; \mathit{right})$ is presented in Fig. 7b and can be viewed as the parallel composition of two "one-sided" philosophers. The compassion requirement reflects the fact that $\mathcal{I}$ can deadlock at location ℓ_1 only if, from some point on, the fork on the right (*right*) is continuously unavailable.

To establish that $\mathcal{I}$ is a network invariant, we verify the abstractions $(Q \circ Q) \sqsubseteq_M \mathcal{I}$ and $(Q \circ \mathcal{I}) \sqsubseteq_M \mathcal{I}$ using the fair simulation algorithm.

The "Four-by-Three" *Abstraction*

An alternative network invariant is obtained by taking $\mathcal{I} = Q^3$, i.e. a chain of 3 philosophers. To prove that this is an invariant, it is sufficient to establish the abstraction $Q^4 \sqsubseteq_M Q^3$, that is, to prove that 3 philosophers can faithfully emulate 4 philosophers.

Experimental Results

We include in our implementation the following optimization. Recall that fair simulation implies simulation [13]. Let $S \subseteq \Sigma_{cor}$ denote the maximal simulation relation. To optimize the algorithm we further restrict player A's moves to S instead of Σ_{cor}.

The following table summarizes the running time for some of the experiments we conducted.

$(Q \circ Q) \sqsubseteq_M \mathcal{I}$	44 secs.
$(Q \circ \mathcal{I}) \sqsubseteq_M \mathcal{I}$	6 secs.
$Q^4 \sqsubseteq_M Q^3$	178 secs.

9 Acknowledgments

We thank Orna Kupferman for suggesting the use of fair simulation in conjunction with network invariants and the referees for many helpful comments on how to improve the presentation of the paper. In particular, we thank one of the referees for suggesting the use of [26].

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[2] R. Alur, S. L. Torre, and P. Madhusudan. Playing games with boxes and diamonds. In *14th International Conference on Concurrency Theory (CONCUR03)*, Lecture Notes in Computer Science, Marseille, France, September 2003. Springer-Verlag. To appear.

[3] B. Bloom and R.Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24:189–220, 1996.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(12):1035–1044, 1986.

[5] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *J. Comp. Systems Sci.*, 8:117–141, 1974.

[6] L. de Alfaro, T. Henzinger, and R. Majumdar. From verification to control: dynamic programs for omega-regular objectives. In *Proc. 16th IEEE Symp. Logic in Comp. Sci.* IEEE Computer Society Press, 2001.

[7] E. Emerson. Model checking and the μ-calculus. In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, pages 185–214. American Mathematical Society, 1997.

[8] E. Emerson and E. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th Int. Colloq. Aut. Lang. Prog.*, volume 85 of *Lect. Notes in Comp. Sci.*, pages 169–181. Springer-Verlag, 1980.

[9] E. A. Emerson and C. L. Lei. Efficient model-checking in fragments of the propositional modal μ-calculus. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 267–278, 1986.

[10] O. Grumberg and D. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.

[11] Y. Gurevich and L. Harrington. Automata, trees and games. In *Proc. 14th ACM Symp. Theory of Comp.*, pages 60–65, 1982.

[12] M. Henzinger, T. Henzinger, and P. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th IEEE Symp. Found. of Comp. Sci.*, pages 453–462. IEEE Computer Society Press, 1995.

[13] T. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *8th International Conference on Concurrency Theory (CONCUR97)*, volume 1243 of *Lect. Notes in Comp. Sci.*, pages 273–287, Warsaw, July 1997. Springer-Verlag.

[14] T. Henzinger and S. Rajamani. Fair bisimulation. In *S. Graf and M. Schwartzbach, editors, Proc.* 6^{th} *Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00), volume 1785 of* Lect. Notes in Comp. Sci.*, Springer-Verlag*, pages 299–314, 2000.

[15] M. Jurdzinski. Small progress measures for solving parity games. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lect. Notes in Comp. Sci.*, pages 290–301. Springer-Verlag, 2000.

[16] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace inclusion. In *W. Hunt Jr and F. Somenzi, editors* Proc. 15^{th} Intl. Conference on Computer Aided Verification (CAV'03), pages 381–392, Boulder, CO, USA, August 2003.

[17] Y. Kesten and A. Pnueli. Control and data abstractions: The cornerstones of practical formal verification. *Software Tools for Technology Transfer*, 2(1):328–342, 2000.

[18] Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Inf. and Comp.*, 163:203–243, 2000.

[19] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proc. 25th Int. Colloq. Aut. Lang. Prog.*, volume 1443 of *Lect. Notes in Comp. Sci.*, pages 1–16. Springer-Verlag, 1998.

[20] Y. Kesten, A. Pnueli, E. Shahar, and L. Zuck. Network invariants in action. In *13th International Conference on Concurrency Theory (CONCUR02)*, volume 2421 of *Lect. Notes in Comp. Sci.*, pages 101–105. Springer-Verlag, 2002.

[21] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[22] O. Kupferman, N. Piterman, and M. Vardi. Fair equivalence relations. In S. Kapoor and S. Prasad, editors, *FST TCS 2000: Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lect. Notes in Comp. Sci.*, pages 151–163. Springer-Verlag, 2000.

[23] O. Kupferman and M. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symp. Theory of Comp.*, pages 224–233, Dallas, 1998.

[24] O. Lichtenstein. *Decidability, Completeness, and Extensions of Linear Time Temporal Logic*. PhD thesis, Weizmann Institute of Science, 1991.

[25] K. Lodaya and P. Thiagarajan. A modal logic for a subclass of events structures. In *Proc. 14th Int. Colloq. Aut. Lang. Prog.*, volume 267 of *Lect. Notes in Comp. Sci.*, pages 290–303. Springer-Verlag, 1987.

[26] D. Long, A. Brown, E. Clarke, S. Jha, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In *D. Dill, editor,* Proc. 6^{th} Intl. Conference on Computer Aided Verification (CAV'94)*, volume 818 of* Lect. Notes in Comp. Sci.*, Springer-Verlag*, pages 338–350, 1994.

[27] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. D. Alfaro, H. Devarajan, H. Sipma, and T. Uribe. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Dept. of Comp. Sci., Stanford University, Stanford, California, 1994.

[28] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[29] A. Melton, D. Schmidt, and D. Strecker. Galois connections and computer science applications. In D. Pitt, S. Abramsky, A. Poigne, and D. Rydeheard, editors, *Category Theory and Computer programming*, volume 240 of *Lect. Notes in Comp. Sci.*, pages 299–312. Springer-Verlag, 1986.

[30] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd International Joint Conference on Artificial Intelligence*, pages 481–489. British Computer Society, September 1971.

[31] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Found. of Comp. Sci.*, pages 46–57, 1977.

[32] A. Pnueli and E. Shahar. A platform for combining deductive with algorithmic verification. In R. Alur and T. Henzinger, editors, *R. Alur and T. Henzinger, editors,* Proc. 8^{th} Intl. Conference on Computer Aided Verification (CAV'96)*, volume 1102 of* Lect. Notes in Comp. Sci.*, Springer-Verlag*, pages 184–195, 1996.

[33] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symp. on Theory of Computing*, Victoria, May 1992.

[34] H. Seidl. Fast and simple nested fixpoints. *Info. Proc. Lett.*, 59(6):303–308, 1996.

[35] R. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Inf. and Cont.*, 54:121–141, 1982.

[36] M. Vardi. Personal communication. 2001.

[37] M. Vardi and P. Wolper. Reasoning about infinite computations. *Inf. and Cont.*, 115(1):1–37, 1994.

[38] M. Y. Vardi. Verification of concurrent programs – the automata-theoretic framework. *Annals of Pure and Applied Logic*, 51:79–98, 1991.

[39] P. Wolper. Temporal logic can be more expressive. *Inf. and Cont.*, 56:72–99, 1983.

[40] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lect. Notes in Comp. Sci.*, pages 68–80. Springer-Verlag, 1989.

A Proof of the Conversion of FDS to JDS

Claim 8 $Obs(\mathcal{D}) = Obs(\mathcal{D}^{B})$

PROOF. Consider a computation $\sigma : s_0, s_1, \ldots,$ of $\mathcal{D}$. It follows that there exists a set $I \subseteq [1..m]$ such that for every $i \in I$, σ contains infinitely many q_i-positions and for every $i \notin I$, σ contains finitely many p_i-positions. Let j be the maximal value such that j is a p_i-position for some $i \notin I$. Consider the computation $\sigma' : s'_0, s'_1, \ldots,$ of $\mathcal{D}^{B}$ where for every $k \geq 0$ and every $i \in I$ we have $s'_k \Downarrow_V = s_k$, $s'_k[x_c] = 0$, and $s'_k[n\_p_i] = 0$. For $i \notin I$ we have $s'_k[n\_p_i] = 0$ if $k \leq j$ and $s'_k[n\_p_i] = 1$ if $k > j$.

It is simple to see that σ' is a run of $\mathcal{D}^{B}$. The state s'_0 satisfies Θ^{B}. Every two adjacent states s'_k and s'_{k+1} satisfy the transition ρ^{B}:

- As $(s_k, s_{k+1}) \models \rho(V, V')$ it follows that the same holds for s'_k and s'_{k+1}.
- For all $i \notin I$, there are no p_i-positions after j and the transition ρ_c is satisfied.
- For every $i \in [1..m]$ we have $n\_p_i \rightarrow n\_p'_i$.

Similarly, σ' is also a computation. For every $J \in \mathcal{J}$ we know that there are infinitely many J-positions in σ and hence also in σ'. As x_c is constant 0, there are infinitely many $\neg x_c$-positions. Finally, for every $i \in I$ there are infinitely many q_i-positions and for every $i \notin I$ there are infinitely many n_p_i positions.

In the other direction, consider a computation $\sigma' : s'_0, s'_1, \ldots,$ of $\mathcal{D}^{B}$. Again there exists a set $I \subseteq [1..m]$ such that for every $i \in I$, σ' contains infinitely many q_i-positions and for every $i \notin I$, σ' contains infinitely many n_p_i-positions. Consider the run $\sigma : s_0, s_1, \ldots,$ of $\mathcal{D}$ where for every $k \geq 0$ we have $s_k = s' \Downarrow_V$. Obviously σ satisfies initiality and consecution of $\mathcal{D}$. As $\mathcal{J} \subseteq \mathcal{J}^{B}$, justice is also satisfied. Finally for every compassion requirement $\langle p_i, q_i \rangle$, if $i \in I$ we know that there are infinitely many q_i-positions and $\langle p_i, q_i \rangle$ is satisfied. If $i \notin I$ we know that n_p_i is set in σ' from some point onwards. As there are infinitely many $\neg x_c$-positions in σ', we conclude that there are finitely many p_i-positions in σ.

B Solving Generalized Streett[1] Games

Let $\mathcal{D}_C : \langle V_C, \mathcal{O}_C, \Theta_C, \rho_C, \mathcal{J}_C, \emptyset \rangle$ and $\mathcal{D}_A : \langle V_A, \mathcal{O}_A, \Theta_A, \rho_A, \mathcal{J}_A, \emptyset \rangle$ be two comparable JDS's where $\mathcal{J}_C = \{J_1^C, \ldots, J_m^C\}$ and $\mathcal{J}_A = \{J_1^A, \ldots, J_n^A\}$. Let $G : \langle \mathcal{D}_C, \mathcal{D}_A \rangle$ be an SGS. Let $M = [1..m]$, $N = [1..n]$, and $\mathbb{N}$ denote the set of natural numbers. We use the notation $i \oplus 1$ for $(i \bmod n) + 1$ (i.e. cyclic addition in N). To simplify notations we denote $\neg J_j^C$ by q_j and J_k^A by p_k. It follows that a play winning for player A must satisfy

$$\left(\bigwedge_{j \in M} \Box \Diamond \neg q_j \right) \rightarrow \left(\bigwedge_{k \in N} \Box \Diamond p_k \right) \equiv \left(\bigvee_{j \in M} \Diamond \Box q_j \right) \vee \left(\bigwedge_{k \in N} \Box \Diamond p_k \right)$$

The set $W_{_A} \subset \Sigma_G$ of winning states for player A is evaluated by the formula in Equation (B.1).

$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{j=1}^{m} \nu X (p_1 \wedge \diamondsuit Z_2 \ \vee \ \diamondsuit Y \ \vee \ q_j \wedge \diamondsuit X) \right) \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X (p_2 \wedge \diamondsuit Z_3 \ \vee \ \diamondsuit Y \ \vee \ q_j \wedge \diamondsuit X) \right) \\ \vdots \\ \vdots \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X (p_n \wedge \diamondsuit Z_1 \ \vee \ \diamondsuit Y \ \vee \ q_j \wedge \diamondsuit X) \right) \end{bmatrix} \tag{B.1}$$

Claim 4 $W_{_A} = [[\varphi]]$

PROOF. We claim that $W_{_A} = Z_1$ at the end of the fixpoint evaluation.[10]

Recall, that a computation of a fixpoint (such as above) starts by setting the initial values of greatest fixpoint (variables Z and X above) to Σ_{cor} and initial values of least fixpoint (variables Y above) to $\emptyset$. Then, the values are computed inductively, by using the previous value in order to get a better approximation of the fixpoint value. Once, two successive values are equivalent, we are ensured that the value of the fixpoint is reached. In particular, the value of the Z variables starts from Σ_{cor} and decreases until it reaches the fixpoint value for the first time. Then, the Y variables and X variables are initialized and the Zs are computed again to give the fixpoint value in the second (and last) time. In this last phase of the computation Y is initialized to $\emptyset$ and grows iteratively until it equals the value of the appropriate Z.

We start by establishing an auxiliary lemma characterizing the states computed by the minimal fixpoints in Equation (B.1). For simplicity of presentation we replace $p_i \wedge \diamondsuit Z_{i \oplus 1}$ by the atom P. The following fixpoint, is the fixpoint computing the value of Y in Equation (B.1).

$$\psi = \mu Y \left(\bigvee_{j=1}^{m} \nu X (P \quad \vee \quad \diamondsuit Y \quad \vee \quad q_j \wedge \diamondsuit X) \right) \tag{B.2}$$

We prove that the fixpoint in Equation (B.2) computes the set of states winning for player A in the game whose winning condition is $(\bigvee_{j=1}^{m} \Diamond \Box q) \vee \Diamond P$. Denote the winning set in this simpler game by W.

[10] Actually, all Z_i's evaluate the same set. This follows from the proof below. However, we do not use this fact in the proof.

Lemma 9 $[[\psi]] = W$

PROOF. We start by showing that $[[\psi]] \subseteq W$. We denote by Y^i the ith iteration of Y. Formally, let $Y^0 = \emptyset$ and, for every $i > 0$, $Y^i = \bigvee_{j=1}^{m} \nu X(P \quad \vee \quad \circledast Y^{i-1} \quad \vee \quad q_j \wedge \circledast X)$.

For every state $s \in Y^i$, there exists a $j \in M$ such that $s \in \nu X(P \quad \vee \quad \circledast Y^{i-1} \quad \vee \quad q_j \wedge \circledast X)$. It is quite simple to see that, from every such state s, player A can win the game whose winning condition is $\Box\, q_j \vee \Diamond(P \vee \circledast Y^{i-1})$. So player A either forces the game to visit P, forces the game to a lower rank Y, or remains in q_j-states forever. As $[[\psi]] = \bigcup_i Y^i$ and $Y_0 = \emptyset$ it follows that from every state in $[[\psi]]$ player A can win the game whose winning condition is $(\bigvee_{j=1}^{m} \Diamond \Box\, q_j) \vee \Diamond P$. That is, there exists a strategy that forces the play to a P-state, or the play eventually remains forever in q_j-states for some $j \in M$.

We prove now that $W \subseteq [[\psi]]$. In order to do that we complement ψ and show that every state in $[[\neg\psi]]$ wins for player C the game whose winning condition is $(\bigwedge_{j=1}^{m} \Box \Diamond \neg q_j) \wedge \Box \neg P$.

The following formula is the positive normal form and simplified complement of the formula in Equation (B.2). In the formula below we replace $\neg P$ by R and $\neg q_j$ by T_j. The 'translated' winning condition is $(\bigwedge_{j=1}^{m} \Box \Diamond T_j) \wedge \Box R$.

$$\neg\psi = \nu Y \left(\bigwedge_{j=1}^{m} \mu X(R \wedge T_j \wedge \circledcirc Y \quad \vee \quad R \wedge \circledcirc X) \right) \tag{B.3}$$

Let Y denote the value of $[[\neg\psi]]$ and X_j denote the value of $\mu X(R \wedge T_j \wedge Y \quad \vee \quad R \wedge \circledcirc X)$

It is quite simple to see that X_j is exactly the set of states from which player C has a strategy that forces the game to reach in a finite number of steps an $R \wedge T_j$-state from which player C can force the game to Y. Furthermore, all intermediate states are R-states. Associate this strategy with X_j.

Player C now combines these strategies in the following way. As $Y \subseteq X_1$, the play starts from a state in X_1. From a state in X_j player C uses her strategy to force the game to T_j and then to Y again. As $Y \subseteq X_{(j \bmod m)+1}$, player C switches to the strategy associated with $X_{(j \bmod m)+1}$. During all that time the play remains in R-states. It follows that player C wins the game whose winning condition is $(\bigwedge_{j=1}^{m} \Diamond \Box T_j) \wedge \Box R$.

We proceed now with the main part of Claim 4. We start by proving *soundness*, namely, showing that every state $s \in Z_1$ is winning for player A. Let $Z_1, \ldots, Z_n$

denote the values of the variables at the end of the fixpoint computation. We show that, for all i, every state in Z_i is winning for A in the simpler game whose winning condition is $(\bigvee_{j=1}^{m} \Diamond \Box q_j) \vee \Diamond(p_i \wedge \circledcirc Z_{i\oplus 1})$. That is, from a state in Z_i player A has a strategy so that every play either visits p_i and in the next round $Z_{i\oplus 1}$ or for some $j \in M$, the play eventually remains forever in q_j-states. These strategies are composed in the obvious way. The play starts from Z_1. From a state in Z_i player A uses the strategy associated with Z_i. If the play reaches a p_i-state and then $Z_{i\oplus 1}$, player A switches her strategy. Every time player A switches her strategy for some $i \in N$ a p_i-state is visited. It follows that if player A switches strategies infinitely often, then for every i, p_i-states are visited infinitely often. If from some stage onwards, player A uses the same strategy, then for some j, the game eventually remains in q_j-states. In both cases, player A wins. More formally, we have the following.

Given that $Z_{i\oplus 1}$ is the fixpoint value of the variable $Z_{i\oplus 1}$, the fixpoint

$$\mu Y \left(\bigvee_{j=1}^{m} \nu X (p_i \wedge \circledcirc Z_{i\oplus 1} \quad \vee \quad \circledcirc Y \quad \vee \quad q_j \wedge \circledcirc X) \right)$$

computes the value of Z_i.

According to Lemma 9, from every state in Z_i, player A has a strategy that either reaches a p_i-state followed by a $Z_{i\oplus 1}$-state in the next round (by replacing P in Equation (B.2) by $p_i \wedge \circledcirc Z_{i\oplus 1}$) or the play eventually remains in q_j states for some $j \in M$. Denote this strategy by f_i.

Player A combines the strategies $f_1, \ldots f_n$ as follows. She starts from Z_1 with f_1, if the game reaches a p_1-state followed by a Z_2-state, she switches to strategy f_2 and continues according to f_2. Whenever, player A switches strategy some p_i is visited. Consider an infinite play π. Either player A switches her strategy infinitely often along π or from some point onwards she plays according to f_i. In the first case, whenever she switches her strategy she visits p_i for some i and it follows that for all i, p_i is visited infinitely often. In the second case, playing indefinitely according to f_i means that the play eventually remains in q_j-states for some j and again A wins.

Next we prove *completeness* of Claim 4, namely, we show that for every state s winning for player A, we have $s \in Z_1$. We show that for all i, Z_i is an over approximation of W_A. Obviously, this is true for the beginning of the fixpoint evaluation when $Z_i = \Sigma_{cor}$. Given some value for $Z_{i\oplus 1}$ that is an over approximation of W_A, we show that computing the next iteration of Z_i cannot remove states that are winning for player A. If a state s is winning for A it is obviously winning also in the simpler game whose winning condition is $(\bigvee_{j=1}^{m} \Diamond \Box q_j) \vee \Diamond(p_i \wedge \circledcirc W_A)$. We show that even states winning in the game whose winning condition is $(\bigvee_{j=1}^{m} \Diamond \Box q_j) \vee \Diamond(p_i \wedge \circledcirc Z_{i\oplus 1})$ are maintained in the next approximation of Z_i. As $Z_{i\oplus 1}$ is an over approximation of $W_{_A}$ we conclude that winning states are never removed from Z_i and it remains an over approximation of $W_{_A}$. More formally, we have the following. For simplicity, we handle Z_1 and the generalization for $k \in N$ is obvious.

Recall that the computation of the fixpoint starts by setting all Z_i to Σ_{cor} and computing the inner subformulas in order to get better approximation of the fixpoint value. Let Z_2^l denote some intermediate value for Z_2 in the computation of the fixpoint. Assume by induction that it is an over approximation of $W_{_A}$. The following fixpoint computes the next approximation of Z_1:

$$\mu Y \left(\bigvee_{j=1}^{m} \nu X (p_1 \wedge \diamondsuit Z_2^l \quad \vee \quad \diamondsuit Y \quad \vee \quad q_j \wedge \diamondsuit X) \right) \tag{B.4}$$

Consider a state s winning for player A, i.e., $s \in W_{_A}$. In particular, player A can win from s the simpler game whose winning condition is $(\bigvee_{j=1}^{m} \Diamond \Box q_j) \vee \Diamond(p_1 \wedge \diamondsuit W_{_A})$. As $W_{_A} \subseteq Z_2^l$ it follows that player A can win from s also the game whose winning condition is $(\bigvee_{j=1}^{m} \Diamond \Box q_j) \vee \Diamond(p_1 \wedge \diamondsuit Z_2^l)$.

According to Lemma 9, the fixpoint in Equation (B.4) computes the states winning for player A in the game whose winning condition is $(\bigvee_{j=1}^{m} \Diamond \Box q_j) \vee \Diamond(p_1 \wedge \diamondsuit Z_2^l)$ (where we replace P by $p_1 \wedge \diamondsuit Z_2^l$). In particular, every state s winning for player A remains in the next approximation of Z_1.