Bridging the Gap Between Fair Simulation and Trace Inclusion

Yonit Kesten* Nir Piterman† Amir Pnueli†

March 24, 2003

Abstract

The paper considers the problem of checking abstraction between two finite-state *fair discrete systems*. In automata-theoretic terms this is trace inclusion between two 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 appropriate auxiliary variables. This establishes that fair simulation offers a complete method for checking trace inclusion.

We illustrate the feasibility of the approach by algorithmically checking abstraction between systems whose abstraction could only be verified by deductive methods up to now.

* Address: Ben Gurion University, Beer-Sheva, Israel. Email: ykesten@bgumail.bgu.ac.il

† Department of Computer Science and Applied Mathematics, Weizmann Institute, Rehovot 76100, Israel. Email: (nirp,amir)@wisdom.weizmann.ac.il

Submitted to CAV 2003 track A

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 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 which admit both weak and strong fairness this problem can be reduced to the problem of language inclusion between two Streett automata (e.g., [Var91]).

In theory, this problem has an exponential-time algorithmic solution based on the complementation of the automaton representing the abstract system A [Saf92]. 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* ([KPSZ02],[WL89]). 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 a system of 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 [Mil71], 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 $(s_C, s_A) \in R$ and system C can move from state s_C to state s'_C, then system A can move from s_A to some s'_A such that $(s'_C, s'_A) \in R$. Additional requirements on R are that if $(s_C, s_A) \in R$ then s_C and s_A agree on the values of their observables, and for every s_C initial in C there exists s_A initial in A such that $(s_C, s_A) \in R$. 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 [BR96, HHK95].

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

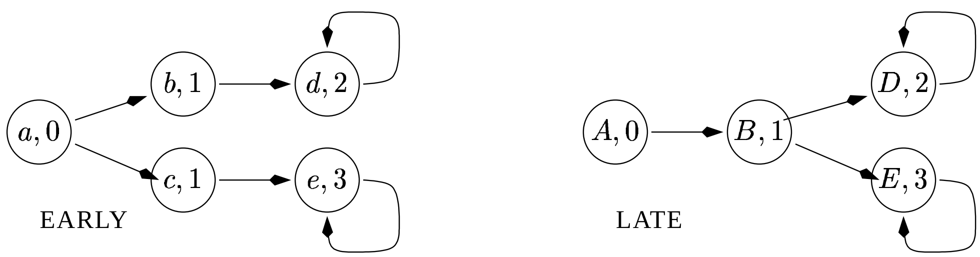

Figure 1: Systems EARLY and LATE

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 requirements associated with them. There were many suggestions about how to enhance the notion of simulation in order to account for fairness [GL94, LT87, HKR97, HR00]. The one we found most useful for our purposes is the definition of fair simulation from [HKR97]. 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 will either satisfy the fairness requirements of the abstract system or fail to satisfy one of the fairness requirements of the concrete system. This guarantees that every concrete observation has a corresponding abstract observation with matching values of the observables.

Algorithmic Considerations

In order to determine whether one system fairly simulates another (*solve fair simulation*) we have to solve games [HKR97]. 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 (*fsim-games*). In [HKR97] the solution of fsim-games is reduced to the solution of Streett games. In [KV98] an algorithm for solving Streett games is presented. The time complexity of this approach is $(|\Sigma_A|\cdot|\Sigma_C|\cdot(3^{k_A}+k_C))^{2k_A+k_C}\cdot(2k_A+k_C)!$ where k_C and k_A denote the number of Streett pairs of C and A respectively. Obviously, the complexity of this approach is too high. It is also not obvious whether this algorithm can be transformed into a symbolic one.

In the context of fair simulation, Streett systems cannot be reduced to simpler systems [KPV00]. That is, in order to solve the question of fair simulation between Streett systems we have to solve fsim-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 (generalized Streett[1]).

In [dAHM01], a solution for games with winning condition expressed as a general LTL formula is presented. The algorithm in [dAHM01] 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 [EL86], Emerson and Lei show that a μ-calculus formula is in fact a recipe for symbolic model checking [1]. 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 [EL86].

In fsim-games the winning condition is an implication between two Streett conditions. A deterministic Streett automaton for such a winning condition has $3^{k_A}\cdot k_C$ states and index $2k_A+k_C$. A deterministic parity automaton for the same condition has $3^{k_A}\cdot k_C\cdot(2k_A+k_C)!$ states and index $4k_A+2k_C$. The μ-calculus formula constructed by [dAHM01] is thus of alternation depth $4k_C+2k_C$ and its size is proportional to $3^{k_C}\cdot k_C\cdot(2k_C+k_C)!$. We can therefore conclude that, in the case of fsim-games, there is no advantage in using [dAHM01].

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 [dAHM01] is proportional to $3|J_C|\cdot|J_A|$ and has alternation depth 3.

We present an alternative μ-calculus formula for solving 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 [dAHM01].

[1] There are more efficient algorithms for μ-calculus model checking [Jur00]. However, Jurdzinski's algorithm cannot be implemented symbolically and we do not use it.

Obviously, our algorithm is tailored for the case of generalized-Streett[1] games while [dAHM01] give a generic solution for any LTL game [2]. The time complexity of solving fair simulation between two reactive systems after converting them to systems with generalized Büchi fairness requirements is $(|\Sigma_A| \cdot |\Sigma_C| \cdot 2^{k_A+k_C} \cdot (|J_A| + |J_C| + k_A + k_C))^3$.

Making the Method Complete

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 simulation using an *abstraction mapping* [AL91]. It is not difficult to see that this notion of simulation is the infinite-state counterpart of the fair simulation as defined in [HKR97] but restricted to the use of memory-less strategies. However, [AL91] 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. History variables remove the restriction to memory-less strategies, while prophecy variables allow to predict the future and use fair simulation to establish, for example, the abstraction 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 identify the necessary auxiliary history or prophecy variables, but following that, the rest of the process is automatic. For example, wishing to apply our algorithm in order 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 $\diamondsuit(x = 2)$. Using this augmentation, the algorithm manages to prove that the augmented system (LATE +tester) is fairly simulated (hence abstracted) by EARLY.

In summary, the contributions of this paper are:

1. Showing how to reduce the problem of checking fair simulation between two reactive systems (Streett automata) into a game with generalized-Street[1] acceptance condition.
2. Providing a new (and more efficient) μ-calculus formula and its implementation by symbolic model-checking tools for solving the fair simulation between two reactive systems.
3. Claiming and demonstrating the completeness of the fair-simulation method for proving abstraction between two systems, at the price of augmenting the concrete system by appropriately chosen “observers” and “testers”.

2 The Computational Model

As a computational model, we take the model of *fair discrete system* (FDS) [KP00]. An FDS $\mathcal{D} : \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$ consists of the following components.

[2] One may ask why not take one step further and convert the original reactive systems to Büchi systems (with one fairness set each). In this case, the induced game is a parity[3] game and there is a simple algorithm for solving it. We also tried to implement this approach. Although both algorithms work in cubic time, the latter performed much worse than the one described above.

- $V = \{u_1, ..., u_n\}$: A finite set of typed *state variables* over possibly infinite 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.

- $\mathcal{O} \subseteq V$: 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 $\langle s, s' \rangle \models \rho(V, V')$, where $\langle s, s' \rangle$ 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* (*weak fairness*) requirements. 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* (*strong fairness*) requirements . 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.

We require that every state $s \in \Sigma$ has at least one $\mathcal{D}$-successor. This is ensured by including in ρ the *idling* disjunct $V' = V$ (also called the *stuttering* step).

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 an infinite 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 each $j = 0, 1, ...,$ the state s_{j+1} is a $\mathcal{D}$-successor of the state s_j.

We denote by *runs*$(\mathcal{D})$ the set of runs of $\mathcal{D}$. A 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 $\mathcal{C}omp(\mathcal{D})$ the set of all computations of $\mathcal{D}$.

Systems $\mathcal{D}_1$ and $\mathcal{D}_2$ are *compatible* if the intersection of their variables is observable in both systems. For compatible 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 disjunction of the two transition relations. Thus, the execution of the combined system is the interleaved execution of $\mathcal{D}_1$ and $\mathcal{D}_2$.

For compatible 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 combining a system with a *tester* T_φ for an LTL formula φ.

The *observations* of $\mathcal{D}$ are the projection $\mathcal{D}\Downarrow_{\mathcal{O}}$ of $\mathcal{D}$-computations onto $\mathcal{O}$. We denote by *Obs*$(\mathcal{D})$ the set of all observations of $\mathcal{D}$. Systems $\mathcal{D}_C$ and $\mathcal{D}_A$ are said to be *comparable* if there is a one to one correspondence between their observable variables. 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 $Obs(\mathcal{D}_1) = Obs(\mathcal{D}_2)$. For compatibility with automata terminology, we refer to the observations of $\mathcal{D}$ also as the *traces* of $\mathcal{D}$.

All our concrete examples are given in SPL (Simple Programming Language), which is used to represent concurrent programs (e.g., [MP95, MAB+94]). 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.

From FDS to JDS

An FDS with no compassion requirements is called a *just discrete system* (JDS).

Let $\mathcal{D} : \langle V, \mathcal{O}, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$ be an FDS such that $\mathcal{C} = \{(p_1, q_1), \ldots, (p_m, q_m)\}$ 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:

- $V^B = V \cup \{n\_p_i : \textbf{boolean} \mid (p_i, q_i) \in \mathcal{C}\} \cup \{x_c\}$. That is, for every compassion requirement $(p_i, q_i) \in \mathcal{C}$, 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*.
- $\mathcal{O}^B = \mathcal{O}$.
- $\Theta^B = \Theta \ \wedge\ x_c = 0 \ \wedge \displaystyle\bigwedge_{(p_i,q_i)\in\mathcal{C}} n\_p_i = 0.$

  That is, initially all the newly introduced boolean variables are set to zero.
- $\rho^B = \rho \ \wedge\ \rho_{n\_p} \ \wedge\ \rho_c$, where

  $$\begin{aligned} \rho_{n\_p} &: \bigwedge_{(p_i,q_i)\in\mathcal{C}} (n\_p_i \rightarrow n\_p_i') \\ \rho_c &: x_c' = \left(x_c \ \vee \bigvee_{(p_i,q_i)\in\mathcal{C}} (p_i \ \wedge\ n\_p_i) \right) \end{aligned}$$

  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, $1 \leq i \leq m$. Once set, it is never reset.
- $\mathcal{J}^B = \mathcal{J} \cup \{\neg x_c\} \cup \{n\_p_i \ \vee\ q_i \mid (p_i, q_i) \in \mathcal{C}\}$.

  The augmented justice set contains the additional justice requirement $n\_p_i \ \vee\ q_i$ for each $(p_i, q_i) \in \mathcal{C}$. 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, will not be accepted as a computation.

The transformation of an FDS to a JDS follows the transformation of Streett automata to generalized Büchi Automata (see [Cho74] for finite state automata and [Var91] for infinite state automata).

3 Simulation 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. 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 a *correlated state*, if $s[\mathcal{O}_C] = s[\mathcal{O}_A]$. We denote by $\Sigma_{cor} \subset \Sigma_G$ the subset of correlated states of G .

For two states s and t we say that t is an A-successor of s if $(s,t) \models \rho_A$ and $s[V_C] = t[V_C]$. Similarly, we say that t is a C-successor of s if $(s,t) \models \rho_C$ and $s[V_A] = t[V_A]$. A *run* of G is a maximal sequence of states $\sigma :\ s_0, s_1, \ldots$ satisfying the following:

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

We say that a run is *initialized* if it satisfies

- *Initiality:* $s_0 \models \Theta_A \ \wedge \ \Theta_C$

Let G be an SGS and σ be a run of G. The run σ can be viewed as 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 a run.

A run 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 $\mathcal{D}_A$, namely if

$$\sigma \models \mathcal{F}_C \rightarrow \mathcal{F}_A$$

where for $\eta \in \{A, C\}$,

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

Otherwise, σ is *winning for player C*.

Let D be some finite domain, intended to record facts about the past history of a computation (serve as a memory). A *strategy* for player A is a partial function $f_A : D \times \Sigma_{cor} \times \Sigma_G \mapsto D \times \Sigma_{cor}$ such that if $f_A(d, s, s') = (d', t)$ then t is an A-successor of s'. A *strategy* for player C is a partial function $f_C : D \times \Sigma_{cor} \mapsto \Sigma_G$ such that if $f_C(d, s) = (d', s')$ then s' is a C-successor of s. Let f_A be a strategy for player A, and $s_0 \in \Sigma_{cor}$. A run $s_0, s_1, \ldots$ is said to be *compliant* with strategy f_A if there exists a sequence of D-values $d_0, d_2, \ldots, d_{2j}, \ldots$ such that $(d_{2j+2}, s_{2j+2}) = f_A(d_{2j}, s_{2j}, s_{2j+1})$ for every $j \geq 0$. Strategy f_A is *winning* for player A from state $s \in \Sigma_{cor}$ if all s-runs (runs departing from s) which are compliant with f_A are winning for A. A winning strategy for player C is defined similarly. We denote by W_A the set of states from which there exists a winning strategy for player A. The set W_C is defined similarly.

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 [GH82].

3.1 μ-calculus

We define μ-calculus [Koz83] 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 formula $v = i$ where i is a constant that is type consistent with v is an *atomic formula* (p). Let $V = \{X, Y, \ldots\}$ be a set of *relational variables*. Each relational variable can be assigned a subset of Σ_{cor}. The μ-calculus formulas are constructed as follows.

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

A formula f is interpreted as the set of states in which f is true. We write such set of states as $[[f]]^e_G$ where G is the SGS and $e : V \to 2^{\Sigma_{cor}}$ is an *environment*. 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 $[[f]]^e_G$ is defined inductively as follows[3].

- $[[p]]^e_G = \{s \in \Sigma_{cor} \mid s \models p\}$
- $[[\neg p]]^e_G = \{s \in \Sigma_{cor} \mid s \not\models p\}$
- $[[X]]^e_G = e(X)$
- $[[f \vee g]]^e_G = [[f]]^e_G \cup [[g]]^e_G$.
- $[[f \wedge g]]^e_G = [[f]]^e_G \cap [[g]]^e_G$.
- $[[\diamondsuit f]]^e_G = \{s \in \Sigma_{cor} \mid \forall t,\ (s,t) \models \rho_C \to \exists s',\ (t,s') \models \rho_A$ and $s' \in [[f]]^e_G\}$.
- $[[\Box f]]^e_G = \{s \in \Sigma_{cor} \mid \exists t,\ (s,t) \models \rho_C$ and $\forall s',\ (t,s') \models \rho_A \to s' \in [[f]]^e_G\}$.
- $[[\mu X f]]^e_G = \cup_i S_i$ where $S_0 = \emptyset$ and $S_{i+1} = [[f]]^{e[X \leftarrow S_i]}_G$.
- $[[\nu X f]]^e_G = \cap_i S_i$ where $S_0 = \Sigma_{cor}$ and $S_{i+1} = [[f]]^{e[X \leftarrow S_i]}_G$

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 $[[f]]$ [EL86]. For a μ-calculus formula of alternation depth k, the run time of this algorithm is $|\Sigma_{cor}|^k$. For a full exposition of μ-calculus we refer the reader to [Eme97]. We often abuse notations and write a μ-calculus formula f instead of the set $[[f]]$.

4 Trace Inclusion and Fair Simulation

In the following, we summarize our solution to the problem of checking abstraction between two finite-state fair discrete systems, 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 $\mathcal{D}_A$ abstracts $\mathcal{D}_C$ ($\mathcal{D}_C \sqsubseteq \mathcal{D}_A$). The best algorithm for solving abstraction is exponential [Saf92]. We therefore advocate to verify fair simulation [HKR97] as a precondition for abstraction. We adopt the definition of fair simulation presented in [HKR97]. 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_A such that every f_A-compliant run 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 $s_C \in \Sigma_C$ satisfying $s_C \models \Theta_C$ there exists a state $t \in S$ such that $t \Downarrow_{V_C} = s_C$ and $t \models \Theta_A$.

[3] Only for finite game structures.

Claim 1 [HKR97] *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 [HKR97] 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. If for every state $s_C \in \Sigma_c$ satisfying $s_C \models \Theta_C$ there exists some state $t \in \Sigma_{cor}$ such that $t \Downarrow_{V_C} = s_C$ and $t \models \Theta_A$, then $\mathcal{D}_C \preceq_f \mathcal{D}_A$. Let $k_C = |\mathcal{C}_C|$ (number of compassion requirements of $\mathcal{D}_C$), $k_A = |\mathcal{C}_A|$, $n = |\Sigma_C| \cdot |\Sigma_A| \cdot (3^{k_C} + k_A)$, and $f = 2k_C + k_A$.

Theorem 2 [HKR97, KV98] *We can solve fair simulation for $\mathcal{D}_C$ and $\mathcal{D}_A$ in time $O(n^{2f+1} \cdot f!)$.*

Since we are interested in fair simulation only as a precondition for trace inclusion, we can take a more economic approach. Given two FDS's, we first convert the two to JDS's using the construction in Section 2. We then solve the simulation game for the two JDS's.

Consider the FDS's $\mathcal{D}_C$ and $\mathcal{D}_A$. Let $\mathcal{D}^B_C : \langle V^B_C, \mathcal{O}^B_C, \Theta^B_C, \rho^B_C, \mathcal{J}^B_C, \emptyset\rangle$ and $\mathcal{D}^B_A : \langle V^B_A, \mathcal{O}^B_A, \Theta^B_A, \rho^B_A, \mathcal{J}^B_A, \emptyset\rangle$ be the JDS's equivalent to $\mathcal{D}_C$ and $\mathcal{D}_A$. Consider the game $G : \langle \mathcal{D}^B_C, \mathcal{D}^B_A\rangle$, the winning condition for this game is

$$\bigwedge_{J_C \in \mathcal{J}^B_C} J_C \rightarrow \bigwedge_{J_A \in \mathcal{J}^B_A} J_A$$

We call such games *generalized Streett*[1] *games*.

We claim that the formula in Fig. 2 evaluates the set W_A of states winning for player A. Intuitively, the greatest fixpoint νX evaluates the set of states from which player A can control the run to remain in $\neg J^C_k$ states. The least fixpoint μY then evaluates the states from which player A in a finite number of steps controls the run to avoid one of the justice conditions J^C_k. This represents the set H of all states from which player A wins as a result of the run of $\mathcal{D}^B_C$ violating justice. Finally, the outermost greatest fixpoint νZ_j adds to H the states from which player A can force the run to satisfy the fairness requirement of $\mathcal{D}^B_A$.

$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{k=1}^{m} \nu X (J^A_1 \wedge \diamondsuit Z_2 \quad \vee \quad \diamondsuit Y \quad \vee \quad \neg J^C_k \wedge \diamondsuit X) \right) \\ \mu Y \left(\bigvee_{k=1}^{m} \nu X (J^A_2 \wedge \diamondsuit Z_3 \quad \vee \quad \diamondsuit Y \quad \vee \quad \neg J^C_k \wedge \diamondsuit X) \right) \\ \vdots \\ \vdots \\ \mu Y \left(\bigvee_{k=1}^{m} \nu X (J^A_n \wedge \diamondsuit Z_1 \quad \vee \quad \diamondsuit Y \quad \vee \quad \neg J^C_k \wedge \diamondsuit X) \right) \end{bmatrix}$$

Figure 2: Algorithm for solving game simulation of two JDS's

Claim 3 $W_A = [[\varphi]]$

The proof of the claim is given in Appendix A.
Using the algorithm in [EL86] the set $[[\varphi]]$ can be evaluated symbolically.

Theorem 4 *The SGS G can be solved in time $O((|\Sigma^B_C| \cdot |\Sigma^B_A| \cdot |\mathcal{J}^B_C| \cdot |\mathcal{J}^B_A|)^3)$.*

To summarize, in order to use fair simulation as a precondition for trace inclusion we propose to convert the FDS's into JDS's and use the formula in Fig. 2 to evaluate the winning set for player A.

Corollary 5 *Given $\mathcal{D}_C$ and $\mathcal{D}_A$, we can determine whether $\mathcal{D}^B_C \preceq_f \mathcal{D}^B_A$ in time $O((|\Sigma_C| \cdot 2^{k_C} \cdot |\Sigma_A| \cdot 2^{k_A} \cdot (k_C + |\mathcal{J}_C| + k_A + |\mathcal{J}_A|))^3)$.*

5 Closing the Gap

As discussed in the introduction, fair simulation implies trace inclusion but not the other way around. In [AL91], fair simulation is considered in the context of infinite-state systems. It is easy to see that the definition of *fair simulation* given in [AL91], is the infinite-state counterpart of fair simulation as defined in [HKR97], but restricted to memory-less strategies. As shown in [AL91], 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.

Following [AL91], we allow the concrete system $\mathcal{D}_C$ to be augmented with a set V_H of *history variables* and a set V_P of *prophecy variables*. We assume that the three sets, V_C, V_H, and V_P, are pairwise disjoint. The result is an augmented concrete system $\mathcal{D}_C^* : \langle V_C^*, \Theta_C^*, \rho_C^*, \mathcal{J}_C, \mathcal{C}_C \rangle$, where

$$
\begin{array}{rcl}
V_C^* & = & V_C \cup V_H \cup V_P \\
\Theta_C^* & = & \Theta_C \wedge \bigwedge_{x \in V_H} (x = f_x(V_C, V_P)) \\
\rho_C^* & = & \rho_C \wedge \bigwedge_{x \in V_H} x' = g_x(V_C^*, V_C', V_P') \wedge \bigwedge_{y \in V_P} y = \varphi_y(V_C)
\end{array}
$$

In these definitions, each f_x and g_x are state functions, while each $\varphi_y(V_C)$ is a future temporal formula referring only to the variables in V_C. Thus, unlike [AL91], we use *transition relations* to define the values of history variables, and *future* LTL *formulas* to define the values of prophecy variables. The clause $y = \varphi_y(V_C)$ added to the transition relation implies that at any position $j \geq 0$, the value of the boolean variable y is 1 iff the formula $\varphi_y(V_C)$ holds at this position.

It is not difficult to see that the augmentation scheme proposed above is *non-constraining*. Namely, for every computation σ of the original concrete system $\mathcal{D}_C$ there exists a computation σ^* of $\mathcal{D}_C^*$ such that σ and σ^* agree on the values of the variables in V_C.

Handling of the prophecy variables definitions is performed by constructing an appropriate *temporal tester* [KP00] for each of the future temporal formulas appearing in the prophecy schemes, and composing it with the concrete system.

A similar augmentation of the concrete system has been used in [KPSZ02] in a deductive proof of abstraction, based on [AL94] *abstraction mapping*.

Although fair simulation is verified algorithmically, user intervention is still needed for choosing the appropriate temporal properties to be observed in order to ensure completeness with respect to trace inclusion.

6 Examples

Late and Early

As a first example we consider the two programs EARLY and LATE presented in Fig. 3 (a graphic representation for these two programs appeared in Fig. 1). The observable variables in these two programs 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, it is simple to see that 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 EARLY does not simulate LATE.

As mentioned EARLY and LATE have the same set of traces. Hence, we should be able to augment LATE with history and prophecy variables that tell EARLY how to simulate it. In this case, we add a *tester* T_φ for the property $\varphi : \diamondsuit(y = 1)$. The tester introduces a new boolean variable x_φ which is true at a state s iff $s \models \varphi$. Whenever the tester for $\diamondsuit(y = 1)$ indicates that LATE will eventually choose $x = 1$, EARLY can

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

Figure 3: Programs EARLY and LATE.

safely choose $x = 1$ in the first step. Whenever the tester for $\diamondsuit(y = 1)$ indicates that LATE will never choose $x = 1$, EARLY can safely choose $x = 2$ in the first step. Denote by LATE$^+$ the combination of LATE with the tester $\diamondsuit(y = 1)$. Applying our algorithm to LATE$^+$ and EARLY, indicates that LATE$^+$ $\preceq_f$ EARLY implying that *Obs*(LATE) $\subseteq$ *Obs*(EARLY).

Fair Discrete Modules and Open Computations

For the main application of our abstraction-checking technique, we need the notions of an *open system* and *open computations*.

We define a *fair discrete module* (FDM) to be a system $M : \langle V, \mathcal{O}, W, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$ consisting of the same components as an FDS plus the additional component:

- $W \subseteq V$: A subset of *owned* variables. These are variables which only the system itself can modify. All other variables can also be modified by steps of the environment.

An (*open*) *computation* of an FDM M is an infinite sequence $\sigma : s_0, s_1, \ldots$ of V-states which satisfies the requirements of initiality, justice, and compassion as any other FDS, and the requirement of consecution, reformulated as follows:

- *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}.

Thus, an (open) computation of an FDM consists of a strict interleaving of system with environment actions, where the system action has to satisfy the transition relation ρ, while the environment step is only required to preserve the values of the owned variables.

Two FDM's $\mathcal{D}_1$ and $\mathcal{D}_2$ are compatible if $W_1 \cap W_2 = \emptyset$ and $V_1 \cap V_2 = O_1 \cap O_2$. The asynchronous parallel composition of two compatible FDM's $M = M_1 \parallel M_2$ is defined similarly to the case of composition of two FDS's where, in addition, the owned variables of the newly formed module is obtained as the union of W_{M_1} and W_{M_2}. Module M_2 is said to be a *modular abstraction* of a comparable module M_1, denoted $M_1 \sqsubseteq_M M_2$, if *Obs*$(M_1) \subseteq$ *Obs*(M_2). A unique feature of the modular abstraction relation is that it is *compositional*. This means that $M_1 \sqsubseteq_M M_2$ implies $M_1 \parallel M \sqsubseteq_M M_2 \parallel M$. This compositionality allows us to replace a module M_1 in any context of parallel composition by another module M_2 which forms a modular abstraction of M_1 and obtain an abstraction of the complete system, which explains why we need modular abstraction for the application of the network invariants method.

It is straightforward to reduce the problem of checking modular abstraction between modules to checking abstraction between FDS's using the methods presented in this paper. This reduction is based on a transformation which, for a given FDM $M : \langle V, \mathcal{O}, W, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$, constructs an FDS $\mathcal{D}_M : \langle \widetilde{V}, \mathcal{O}, \widetilde{\Theta}, \widetilde{\rho}, \mathcal{J}, \mathcal{C}\rangle$, such that the set of observations of M is equal to the set of observations of $\mathcal{D}_M$. The new components of $\mathcal{D}_M$ are given by:

$$\begin{array}{lcl} \widetilde{V} & : & V \cup \{t : \textbf{boolean}\} \\ \widetilde{\Theta} & : & \Theta \,\wedge\, t \\ \widetilde{\rho} & : & \rho \,\wedge\, \neg t \,\wedge\, t' \quad \vee \quad \textit{pres}(W) \,\wedge\, t \,\wedge\, \neg t' \end{array}$$

$$
\begin{array}{c}
(Q^n) \quad \text{where} \\
Q(\mathit{left}; \mathit{right}) :: \\
\left[\begin{array}{l}
\textbf{loop forever do} \\
\quad \left[\begin{array}{ll}
\ell_0: & \textbf{NonCritical} \\
\ell_1: & \textbf{request } \mathit{left} \\
\ell_2: & \textbf{request } \mathit{right} \\
\ell_3: & \textbf{Critical} \\
\ell_4: & \textbf{release } \mathit{left} \\
\ell_5: & \textbf{release } \mathit{right}
\end{array}\right]
\end{array}\right]
\end{array}
$$

Figure 4: Program DINE: a chain of deterministic philosophers.

Thus, system $\mathcal{D}_M$ uses a fresh boolean variable t to encode the turn taking between system and environment transitions.

The Dinning Philosophers

As a second example, we consider a deterministic solution to the dinning philosophers problem (DDP). As originally described by Dijkstra, n philosophers are seated at a round table, with a fork placed in between each two neighbors. Each philosopher alternates between a thinking phase and a phase in which he becomes hungry and wishes to eat. 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 guarantees that no two adjacent philosophers eat at the same time (mutual exclusion) and that every hungry philosopher eventually gets to eat (individual accessibility).

A deterministic solution to the dinning philosophers is presented in [KPSZ02], in terms of *binary processes*. A binary process $Q(\vec{x}; \vec{y})$ is an FDM with two observable variables x and y. Two binary processes Q and R can be composed to yield another binary process, using the *modular composition* operator $\circ$ defined by

$$(Q \circ R)(x; z) \quad = \quad [\textbf{restrict } y \textbf{ in } Q(x; y) \parallel R(y; z)]$$

where **restrict y** is an operator that removes variable y from the set of observable variables and places it in the set of owned variables.

In Fig. 4 we present a chain of n deterministic philosophers, each represented by a binary process $Q(\mathit{left}; \mathit{right})$. This solution is studied in [KPSZ02] as an example of parametric systems, for which we seek a *uniform* verification (i.e. a single verification valid for any n). The uniform verification is presented using the network invariant method, which calls for the identification of a network invariant $\mathcal{I}$ which can safely replace the chain Q^n. The adequacy of the network invariant is verified using an inductive argument which calls for the verification of abstractions. In [KPSZ02] we present a deductive proof to the dinning philosophers, based on [AL94] abstraction mapping method, using two different network invariants.

In the current work, we consider the same invariants, and verify all the necessary abstractions using our algorithm for fair simulation. In both cases, no auxiliary (history and prophecy) variables are needed.

The "*Two-Halves*" Abstraction

The first network invariant $\mathcal{I}(\mathit{left}; \mathit{right})$ is presented in Fig. 5 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*) remains continuously unavailable.

$$\mathcal{I}(\mathit{left}; \mathit{right}) ::$$
$$\left[\begin{array}{l}\textbf{loop forever do}\\ \left[\begin{array}{ll}\ell_0: & \textbf{request}\ \mathit{left}\\ \ell_1: & \textbf{release}\ \mathit{left}\end{array}\right]\end{array}\right] \quad \| \quad \left[\begin{array}{l}\textbf{loop forever do}\\ \left[\begin{array}{ll}m_0: & \textbf{request}\ \mathit{right}\\ m_1: & \textbf{release}\ \mathit{right}\end{array}\right]\end{array}\right]$$
$$\mathcal{J}: \neg at\_m_1 \quad \mathcal{C}: (\mathit{right}, \neg at\_\ell_1)$$

Figure 5: The Two-Halves Network Invariant

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

In our first implementation of the algorithm, we could not establish simulation between very simple obviously correct examples. Player C could always win in a finite number of steps. The problem was with unfeasible states, namely states that do not participate in any computation. Player C would enter an unfeasible state and player A could not follow. To resolve this problem we remove all unfeasible states from both systems. Thus, the first step evaluates the set of feasible states for each of the players.

Since player A can only move to correlated states, we reduce the number of variables by using a single set of observable variables for both systems. Finally, we optimize by reducing the options of player A as follows. Recall that fair simulation implies simulation [HKR97]. Namely, simulation is a precondition for fair simulation. Let $S \subseteq \Sigma_{cor}$ denote the maximal simulation relation. Instead of restricting player A's steps to Σ_{cor} we restrict it further to S.

After all these optimizations, 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.

7 Acknowledgements

We thank Orna Kupferman for suggesting the use of fair simulation in conjunction with network invariants.

References

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[AL94] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Trans. Prog. Lang. Sys.*, 16(5):1543–1571, 1994.

[BR96] 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.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *J. Comp. Systems Sci.*, 8:117–141, 1974.

[dAHM01] L. de Alfaro, T.A. 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.

[EL86] 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.

[Eme97] E.A. Emerson. Model checking and the μ-calculus. In N. Immerman and Ph.G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, pages 185–214. American Mathematical Society, 1997.

[GH82] Y. Gurevich and L.A. Harrington. Automata, trees and games. In *Proc. 14th ACM Symp. Theory of Comp.*, pages 60–65, 1982.

[GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.

[HHK95] M.R. Henzinger, T.A. Henzinger, and P.W. 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.

[HKR97] T.A. 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.

[HR00] T. Henzinger and S. Rajamani. Fair bisimulation. In *Proc. 4th International Conference of Tools and Algorithms for Construction and Analysis of System*, volume 1785 of *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 2000.

[Jur00] 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.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KP00] Y. Kesten and A. Pnueli. Verification by finitary abstraction. *Information and Computation*, 163:203–243, 2000.

[KPSZ02] 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.

[KPV00] O. Kupferman, N. Piterman, and M.Y. 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.

[KV98] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 224–233, Dallas, 1998.

[LT87] K. Lodaya and P.S. 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.

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

[Mil71] 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.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[Saf92] S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symp. on Theory of Computing*, Victoria, May 1992.

[Var91] M. Y. Vardi. Verification of concurrent programs – the automata-theoretic framework. *Annals of Pure Applied Logic*, 51:79–98, 1991.

[Wal01] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.

[WL89] 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 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. We use the notation $i \oplus 1$ for $(i \bmod n) + 1$. Let $M = [1..m]$, $N = [1..n]$, and $\mathbb{N}$ denote the set of natural numbers.

The set $W_A \subset \Sigma_G$ of winning states for player A is evaluated by the formula in equation (1).

$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{k=1}^{m} \nu X (J_1^A \wedge \bigcirc\!\!\!\!\diamond\, Z_2 \quad \vee \quad \bigcirc\!\!\!\!\diamond\, Y \quad \vee \quad \neg J_k^C \wedge \bigcirc\!\!\!\!\diamond\, X) \right) \\ \mu Y \left(\bigvee_{k=1}^{m} \nu X (J_2^A \wedge \bigcirc\!\!\!\!\diamond\, Z_3 \quad \vee \quad \bigcirc\!\!\!\!\diamond\, Y \quad \vee \quad \neg J_k^C \wedge \bigcirc\!\!\!\!\diamond\, X) \right) \\ \vdots \\ \vdots \\ \mu Y \left(\bigvee_{k=1}^{m} \nu X (J_n^A \wedge \bigcirc\!\!\!\!\diamond\, Z_1 \quad \vee \quad \bigcirc\!\!\!\!\diamond\, Y \quad \vee \quad \neg J_k^C \wedge \bigcirc\!\!\!\!\diamond\, X) \right) \end{bmatrix} \tag{1}$$

We introduce some notations. Let σ be some run of G. The *game* g_σ is the restriction of σ to even locations. When σ is irrelevant or clear from the context we write g. Let f_C and f_A be strategies for players C and A respectively. Let $\sigma : s_0, s_1, \ldots$ be the run compliant with both strategies and let $d_0^C, d_2^C, \ldots$ and $d_0^A, d_2^A, \ldots$ be the matching sequences of memory values. Let the *outcome* o_{f_C,f_A} be the sequence $(d_0^C, d_0^A, s_0), (d_2^C, d_2^A, s_2), \ldots$ that consists of the memories of both strategies and the game g_σ. In case that one of the strategies is not important we remove its memory values from the sequence. We often abuse notations and confuse between g_σ and o_{f_C,f_A}, we also write g_{f_C,f_A} instead of g_σ

Claim 3 $W_A = [[\varphi]]$

Proof: We claim that $W_A = Z_1$ at the end of the fixpoint evaluation.[4]

We start by proving *soundness* of the claim, namely, showing that for every state $s \in Z_1$, s is a winning state for player A. We adopt Walukiewicz' analysis of the μ-calculus formula [Wal01]. From the evaluation of the fixpoint formula in (1), we derive a *ranking* for the states in Σ_{cor}. We then use the ranking to define a winning strategy for player A from states in Z_1.

Based on the computation of the last iteration of the outermost fixpoint, we define a set of ranking functions $\mathcal{R} : \{r_1, \ldots, r_n\}$. For every $j \in N$, let $r_j : \Sigma_{cor} \to D \cup \infty$ where $D = \mathbb{N} \times M$ with the usual lexicographic ordering. For simplicity we denote the minimal value in D by 1 instead of $(1, 1)$. Recall, that a run winning for player A satisfies

$$\left(\bigwedge_{k \in M} \Box \Diamond J_k^C \right) \to \left(\bigwedge_{j \in N} \Box \Diamond J_j^A \right)$$

That is, for a run to be winning, it is sufficient that for some $k \in M$, J_k^C be visited only finitely often or forall $j \in N$, J_j^A has to be visited infinitely often. Intuitively, when $r_j(s) = (l, k)$ we know that currently, player A is trying to force the run to stay within $\neg J_k^C$ states. If she cannot do that, she can try and decrease r_j. When r_j reaches 1 the state is a J_j^A state. Thus, the strategy of player A consists of decreasing r_j and once reaching 1 moving to $r_{j\oplus 1}$.

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

Let Z_j denote the fixpoint value of variable Z_j. We denote by Y_j^i the ith iteration of Y associated with Z_j. Formally, let $Y_j^0 = \emptyset$ and $Y_j^i = \bigvee_{k=1}^{m} X_{j,k}^i$, where

$$X_{j,k}^i = \nu X(J_j^A \wedge \circledcirc Z_{j\oplus 1} \quad \vee \quad \circledcirc Y_j^{i-1} \quad \vee \quad \neg J_k^C \wedge \circledcirc X).$$

We define the rankings as follows. For a state $s \in \Sigma_{cor}$ such that $s \notin Z_j$ set $r_j(s) = \infty$. For a state $s \in \Sigma_{cor}$ such that $s \in J_j^A \wedge \circledcirc Z_{j\oplus 1}$ set $r_j(s) = 1$. For a state $s \in \Sigma_{cor}$ such that $s \in X_{j,k}^i \setminus \bigcup_{k'<k} X_{j,k'}^i$ and $s \notin Z_j^{i-1}$ set $r_j(s) = (i,k)$ (that is, r_j is set to the least value (i,k) for which $s \in X_{j,k}^i$).

Based on $\mathcal{R}$, we define a strategy for player A. The memory used by the strategy is a value $j \in N$. Intuitively, when the memory of the strategy is j, player A tries to decrease r_j. When r_j reaches 1, player A updates her memory to $j \oplus 1$ and moves to a state for which $r_{j\oplus 1}$ is defined.

More formally, let $f : N \times \Sigma_{cor} \times \Sigma_G \mapsto N \times \Sigma_{cor}$ be the strategy for player A. If $r_j(s) = \infty$ or for a state t which is not an C-successor of s then $f(j,s,t)$ is undefined. Otherwise, if $r_j(s) = 1$ then $f(j,s,t) = (j\oplus 1, s')$ such that s' is some A-successor of t and $r_{j\oplus 1}(s') \neq \infty$. In Claim 6 we show that this is indeed possible. If $r_j(s) > 1$ then $f(j,s,t) = (j,s')$ such that s' is an A-successor of t and for every A-successor u of t we have $r_j(s') \leq r_j(u)$. In Claim 6 we show that such a successor state exists and that $r_j(s') \leq r_j(s)$.

Claim 6 *Let s be a state in Σ_{cor}. If $r_j(s) \neq \infty$ and $f(j,s,t) = (j',s')$ then $r_{j'}(s') \neq \infty$.*

Proof: For $j \in N$, Z_j is a fixpoint. Hence, there exists some value $p \geq 1$ such that $Z_j = Y_j^p = Y_j^{p+1}$. It follows that $Z_j = \bigvee_{k=1}^{m} X_{j,k}^p$ where $X_{j,k}^p = J_j^A \wedge \circledcirc Z_{j\oplus 1} \quad \vee \quad \circledcirc Z_j \quad \vee \quad \neg J_k^C \wedge X_{j,k}^p$. For every state $s \in Z_j$, either $s \in J_j^A \wedge \circledcirc Z_{j\oplus 1}$ or $s \in \circledcirc Z_j$. In the first case, $r_j(s) = 1$ and player A can control the game to reach a successor state s' such that $s' \in Z_{j\oplus 1}$ implying $r_{j\oplus 1}(s') \neq \infty$. In the second case, player A can control the game to reach a successor state s' such that $r_j(s') \neq \infty$. ∎

Claim 7 *Let s be a state in Σ_{cor}. If $r_j(s) \neq \infty$ and $f(j,s,t) = (j',s')$ then one of the following holds:*

1. *$r_j(s) = 1$.*
2. *$j = j'$ and $r_j(s') < r_j(s)$.*
3. *$j = j'$ and $r_j(s') = r_j(s)$, provided $r_j(s) = (l,k)$ and $s \models \neg J_k^C$.*

Proof: Recall that $Y_j^i = \bigvee_{k=1}^{m} X_{j,k}^i$ where $X_{j,k}^i = \nu X(J_j^A \wedge \circledcirc Z_{j\oplus 1} \quad \vee \quad \circledcirc Y_j^{i-1} \quad \vee \quad \neg J_k^C \wedge \circledcirc X)$. Suppose $s \in Y_j^i \setminus Y_j^{i-1}$. Then $s \in J_j^A \wedge \circledcirc Z_{j\oplus 1} \quad \vee \quad \circledcirc Y_j^{i-1} \quad \vee \quad \bigvee_{k=1}^{m} \neg J_k^C \wedge \circledcirc X_{j,k}^i$. If $s \in J_j^A \wedge \circledcirc Z_{j\oplus 1}$ then $r_j(s) = 1$ and we are done. If $s \in \circledcirc Y_j^{i-1}$ then player A can control the game to get to a successor state s' such that $r_j(s') < r_j(s)$. Suppose $s \in X_{j,k}^i \setminus (J_j^A \wedge \circledcirc Z_{j\oplus 1} \quad \vee \quad \circledcirc Y_j^{i-1})$ and $s \notin (\bigcup_{k'<k} X_{j,k'}^i)$. Then we know $r_j(s) = (i,k)$ and $s \in \neg J_k^C \wedge \circledcirc X_{j,k}^i$. In particular player A can control the game to get to a successor state s' such that $r_j(s') \leq r_j(s)$ and $s \models \neg J_k^C$ (the case of $<$ follows in the case that $s' \in X_{j,k'}^i$ for some $k' < k$). ∎

Let s be a state in Z_1.

Claim 8 *Every s-run compliant with f is winning for player A.*

Proof: Let $g_f : \langle n_0, s_0\rangle, \langle n_1, s_1\rangle, \ldots$ denote the outcome of f from state s (that is, all the states in g_f are the correlated states in the s-run compliant with f). From Claim 6 we know that for every $i \geq 0$ we have $r_{n_i}(s_i) \neq \infty$. From Claim 7 it follows that either there are infinitely many $i \geq 0$ such that $r_{n_i}(s_i) = 1$ or there exists a value $i \geq 0$ and $j \in N$ such that forall $i' > i$ we have $n_{i'} = j$. In the first case, whenever g_f visits a location (n_i, s_i) such that $r_{n_i}(s_i) = 1$ then $s_i \models J^A_{n_i}$ and $n_{i+1} = n_i \oplus 1$. Hence, $g_f \Downarrow_{V_A}$ is a computation of $\mathcal{D}_A$ and player A wins. In the second case, we know from Claim 7 that forall $l > i$ we have $r_j(s_l) \leq r_j(s_{l+1})$. Then, there exists some $p > i$ such that forall $p' > p$ we have $r_j(s_{p'}) = (a, k)$ for some $a \geq 0$ and $k \in M$. However, in this case forall $p' > p$ we have $s_{p'} \models \neg J^C_k$ and $g_f \Downarrow_{V_C}$ is not a computation of $\mathcal{D}_C$. Again player A wins. ∎

Next we prove *completeness* of claim 3, namely, showing that for every state $s \notin Z_1$, s is a winning state for player C. It is quite simple to see that the following fixpoint is the complement of Equation 1.

$$\neg\varphi = \mu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \left[\begin{array}{c} \nu Y \left(\bigwedge_{k=1}^{m} \mu X (\boxdot Z_2 \quad \vee \quad \neg J^A_1 \wedge J^C_k \wedge \boxdot Y \quad \vee \quad \neg J^A_1 \wedge \boxdot X) \right) \\ \nu Y \left(\bigwedge_{k=1}^{m} \mu X (\boxdot Z_3 \quad \vee \quad \neg J^A_2 \wedge J^C_k \wedge \boxdot Y \quad \vee \quad \neg J^A_2 \wedge \boxdot X) \right) \\ \vdots \\ \vdots \\ \nu Y \left(\bigwedge_{k=1}^{m} \mu X (\boxdot Z_1 \quad \vee \quad \neg J^A_n \wedge J^C_k \wedge \boxdot Y \quad \vee \quad \neg J^A_n \wedge \boxdot X) \right) \end{array} \right] \tag{2}$$

Note that we replace $\Diamond$ with $\boxdot$. Recall that both $\Diamond$ and $\boxdot$ allow player C to make the first choice. However $\Diamond$ demands that for every choice of player C there exist a choice of player A and $\boxdot$ demands that there exists a choice of player C that is good enough forall choices of player A. As before, from the evaluation of the fixpoint formula in 2 we derive a *ranking* for the states in Σ_{cor}. We then use the ranking to define a winning strategy for player C.

For each of the Z_j's, let Y_j denote the value of Y_j at the last iteration of νY. The value of Y_j is defined by a conjunction over m conjuncts. First we show that for every $k, k' \in [1..m]$ the k and k' conjuncts are equal regardless of the value of $Z_{j\oplus 1}$. Equation 3 is the fixpoint computing Y_j.

$$\nu Y \left[\bigwedge_{k=1}^{m} \mu X \left(\boxdot Z_{j\oplus 1} \quad \vee \quad \neg J^A_j \wedge J^C_k \wedge \boxdot Y \quad \vee \quad \neg J^A_j \wedge \boxdot X \right) \right] \tag{3}$$

For some value of $Z_{j\oplus 1}$ let Y_j denote the outcome of Equation 3. We denote by $x^i_{j,k}$ the ith iteration of the kth conjunct in νY_j. Formally, let $X^0_{j,k} = \emptyset$, and $X^i_{j,k} = \boxdot Z \quad \vee \quad \neg J^A_j \wedge J^C_k \wedge \boxdot Y_j \quad \vee \quad \neg J^A_j \wedge \boxdot X^{i-1}_{j,k}$.

We say that a state s is i-*far*$_{j,k}$ if there exists a strategy f_C such that every s-run $\sigma : s_0, s_1, \ldots$ compliant with f satisfies all the following.

- There exists some $l \leq i$ such that $s_{2l} \in J^C_k \wedge \boxdot Y_j \quad \vee \quad \boxdot Z_{j\oplus 1}$.
- Forall $l' < l$ we have $s_{2l'} \not\models J^A_j$.
- Either $s_{2l} \not\models J^A_j$ or $s_{2l} \in \boxdot Z_{j\oplus 1}$.

Claim 9 *$s \in X^i_{j,k}$ iff s is i-far$_{j,k}$.*

Proof: Suppose $s \in X^i_{j,k} \backslash X^{i-1}_{j,k}$. If $i = 1$ then it must be the case that $s \in \bigcirc Z_{j\oplus 1} \quad \vee \quad \neg J^A_j \wedge J^C_k \wedge \bigcirc Y_j$. Clearly, the claim follows. If $i > 1$ then it must be the case that $s \in \neg J^A_j \wedge \bigcirc X^{i-1}_{j,k}$, by induction the claim follows.

Suppose that s is i-far$_{j,k}$. We show by induction on i that $s \in X^i_{j,k}$. If $i = 1$ then clearly $s \in X^1_{j,k}$. Suppose $i > 1$ then player C can force the game to state t such that t is $(i-1)$-far$_{j,k}$. By induction $t \in X^{i-1}_{j,k}$ and $s \in X^i_{j,k}$. ∎

Denote $X_{j,k} = \bigcup_{i \geq 0} X^i_{j,k}$.

Corollary 10 *Forall $k, k' \in M$ we have $X_{j,k} = X_{j,k'}$.*

Proof: Suppose $s \in X_k$. Then s is i-far$_{j,k}$ for some i. Consider the strategy f_C and the run $\sigma : s_0, s_1, \ldots$ compliant with f_C. Suppose $s_{2l} \in J^C_k \wedge \bigcirc Y_j \quad \vee \quad \bigcirc Z_{j\oplus 1}$ for some $l \leq i$. If $s_{2l} \in J^C_k \wedge \bigcirc Y_j$. Then player C can force the game into some state $s' \in X_{j,k'}$. It follows that s' is i'-far$_{j,k'}$ with some strategy f'_C. By combining the strategies f_C and f'_C we show that $s \in X_{j,k'}$. ∎

Based on the computation of the fixpoint, we define a set of ranking functions $\mathcal{R} : \{r_1, \ldots, r_n\}$. For every $j \in N$, let $r_j : \Sigma_{cor} \rightarrow D \cup \{\infty\}$ where $D = \mathbb{N}^{m+1}$. For $d \in D$ we denote by $d[0]$ the first entry in d and for $k \in M$, $d[k]$ is the $k+1$ entry in d. Recall, that a run winning for player C satisfies

$$\left(\bigwedge_{k \in M} \Box \Diamond J^C_k \right) \wedge \neg \left(\bigwedge_{j \in N} \Box \Diamond J^A_j \right)$$

That is, for every $k \in M$, J^C_k is visited infinitely often and for some $j \in N$, J^A_j is visited only finitely often. Intuitively, when $r_j(s) = (l, l_1, \ldots, l_m)$ it means that currently, player C tries to avoid visiting J^A_j. She may still change her mind l times as to which $J \in \mathcal{J}^A$ she avoids. The number l_k denotes the distance to a J^C_k state in case that player C does not change her mind. Thus, player C reduces first l_1 until a visit to J^C_1 then reduces l_2 and so on. The strategy of player C consists of deciding which $J \in \mathcal{J}^A$ is visited finitely often and then for each $k \in M$ forcing a visit to J^C_k.

Let Z^i_j denote the ith iteration of μZ_j. Let $X^{i,l}_{j,k}$ denote the lth iteration of the kth conjunct in the computation of Z^i_j. Formally, we have the following. Let $Z^0_j = \emptyset$,

$$Z^i_j = \nu Y \left[\bigwedge_{k=1}^{m} \mu X \left(\bigcirc Z^{i-1}_{j\oplus 1} \quad \vee \quad \neg J^A_j \wedge J^C_k \wedge \bigcirc Y \quad \vee \quad \neg J^A_j \wedge \bigcirc X \right) \right]$$

For every state $s \in Z^i_j \setminus Z^{i-1}_j$ we set $r_j(s)[0] = i$. That is the first location in $r_j(s)$ stores the iteration of μZ where s is first included in Z^i_j. Notice that from Corollary 10 it follows that for every k,

$$Z^i_j = \bigcirc Z^{i-1}_{j\oplus 1} \quad \vee \quad \neg J^A_j \wedge J^C_k \wedge \bigcirc Z^i_j \quad \vee \quad \neg J^A_j \wedge \bigcirc Z^i_j \tag{4}$$

As above, let

$$X^{i,0}_{j,k} = \emptyset \text{ and } X^{i,l}_{j,k} = \bigcirc Z^{i-1}_{j\oplus 1} \quad \vee \quad \neg J^A_j \wedge J^C_k \wedge \bigcirc Z^i_j \quad \vee \quad \neg J^A_j \wedge \bigcirc X^{j,k}_{i,l-1}$$

Notice that for every $k \in M$ we have $Z^i_j = \bigcup_{l \geq 0} X^{i,l}_{j,k}$. For every state $s \in X^{i,l}_{j,k} \setminus (X^{i,l-1}_{j,k} \cup Z^{i-1}_j)$ we set $r_j(s)[k] = l$. That is, the kth entry in $r_j(s)$ stores the iteration of the kth conjunct in which s is first included in $X^{i,l}_{j,k}$. From Corollary 10 it follows that for every state s in Z^i_j we have either $r_j(s)[0] < i$ or $r_j(s)[0] = i$ and for every $k \in M$, $r_j(s)[k]$ is set in the ith stage.

Based on $\mathcal{R}$, we define a strategy for player C. The memory used by the strategy is a value $(j, k) \in N \times M$. That is, work with r_j and with entry k in r_j. Intuitively, when the memory of the strategy is (j, k), player C tries to move to states for which $r_{j\oplus 1}[0]$ is lower and updates her memory to $(j \oplus 1, 1)$. If player C cannot do that, she tries to decrease $r_j[0]$ or $r_j[k]$. When $r_j[k]$ reaches 1, we are ensured that we are in a J^C_k state and player C updates her memory to $(j, (k \bmod m) + 1)$. More formally we have the following.

Let $f : N \times M \times \Sigma_{cor} \to N \times M \times \Sigma_G$ be the strategy for player C. Let s be some state such that $r_j(s) = \infty$, then for every $k \in M$, $f(j, k, s)$ is undefined. Let s be some state such that $r_j(s) \neq \infty$. The strategy f chooses the first possible option from the following.

1. If there exists a C-successor t of s such that forall A-successors s' of t we have $r_{j\oplus 1}(s')[0] < r_j(s)[0]$ then $f(j, k, s) = (j \oplus 1, 1, t)$.

2. If there exists a C-successor t of s such that forall A-successors s' of t we have $r_j(s')[0] < r_j(s)[0]$ then $f(j, k, s) = (j, 1, t)$.

3. If $r_j(s)[k] = 1$ and there exists a C-successor t of s such that forall A-successors s' of t we have $r_j(s')[0] = r_j(s)[0]$ then $f(j, k, s) = (j, (k \bmod m) + 1, t)$.

4. If $r_j(s)[k] \neq 0$ and there exists a C-successors t of s such that forall A-successors s' of t we have $r_j(s')[0] = r_j(s)[0]$ and $r_j(s')[k] < r_j(s)[k]$ then $f(j, k, s) = (j, k, t)$.

We show that this strategy is feasible and that it is a winning strategy for player C.

Claim 11 *Let s be a state in Σ_{cor}. If $r_j(s) \neq \infty$ and $f(j, k, s) = (j', k', t)$ then forall A-successors s' of s, $r_{j'}(s') \neq \infty$.*

Proof: Let $r_j(s)[0] = i$. From Equation 4 it follows that there exists a C-successor t of s such that either forall A-successors s' of t we have $s' \in Z^{i-1}_{j\oplus 1}$ or forall A-successors s' of t we have $s' \in Z^i_j$. ∎

Claim 12 *Let s be a state in Σ_{cor}. If $r_j(s) \neq \infty$ and $f(j, k, s) = (j', k', t)$ then one of the following holds forall A-successors s' of t:*

- $r_{j'}(s')[0] < r_j(s)[0]$
- $r_j(s')[0] = r_j(s)[0]$ *and either* $s \models J^C_k$ *or* $r_j(s')[k] < r_j(s)[k]$.

Proof: From Claim 11 we know that player C can control the game so that forall s' we have $r'_j(s') \neq \infty$. From Equation 4 we know that player C can control the game so that $r'_j(s')[0] \leq r_j(s)[0]$. Suppose $r_j(s)[0] = l$ and $r_j(s)[k] = 1$. Then $s \in X^{j,k}_{l,1}$. Hence, either $s \in \bigcirc Z^{i-1}_{j\oplus 1}$ or $s \in \neg J^A_j \wedge J^C_k \wedge \bigcirc Z^i_j$. In the first case player C can control the game so that it reaches a location s' such that $r_{j\oplus 1}(s')[0] < r_j(s)[0]$. In the second case $s \models \neg J^A_j \wedge J^C_k$ and player C can control the game to reach a state s' such that $r_j(s')[0] \leq r_j(s)[0]$. Suppose $r_j(s)[k] = a > 1$ then $s \in X^{j,k}_{l,a} \setminus X^{j,k}_{l,a-1}$. Clearly, $s \models \neg J^A_j$ and player C can control the game so that it reaches a state s' such that $r_j(s')[k] < r_j(s)[k]$. ∎

Let s be a state in Z_1.

Claim 13 *Every s-run compliant with f is winning for player C.*

Proof: Let $g_f : \langle(n_0, m_0), s_0\rangle, \langle(n_1, m_1), s_1\rangle, \ldots$ denote the outcome of f from state s (that is, all the states in g_f are the correlated states in the s-run compliant with f). From Claim 11 we know that for every $i \geq 0$ we have $r_{n_i}(s_i) \neq \infty$. From Claim 12 we know that there exists some i such that forall $i' \geq i$ we have $n_{i'} = j$ for some $j \in N$ and $r_j(s_{i'})[0] = l$ for some $l \in \mathbb{N}$. Clearly, it cannot be the case that $s_{i'} \models J_j^A$ for $i' \geq i$. Suppose by contradiction that there exists a point $a > i$ such that forall $a' \geq a$ we have $s_{a'} \not\models J_k^C$. We know that $r_j(s_{a'})[m_{a'}]$ is defined. According to Claim 12 $r_j(s_{a'})[m_{a'}]$ decreases. There exists a point $a' > a$ such that $r_j(s_{a'})[m_{a'}] = 1$, and $m_{a'+1} = (m_{a'} \bmod m) + 1$. Similarly, according to Claim 12 $r_j(s_{a'})[(m_{a'} \bmod m) + 1]$ decreases. So the game reaches some point where J_k^C holds in contradiction to the assumption. We conclude that $g_f \Downarrow_{V_A}$ is not a computation (J_j^A is visited finitely often) and that $g_f \Downarrow_{V_C}$ is a computation. Hence, player C wins. ▟

□