# A Hybrid Algorithm for LTL Games*

Saqib Sohail[1], Fabio Somenzi[1], and Kavita Ravi[2]

[1] University of Colorado at Boulder
{Saqib.Sohail,Fabio}@Colorado.EDU
[2] Cadence Design Systems
kravi@cadence.com

**Abstract.** In the game theoretic approach to the synthesis of reactive systems, specifications are often given in linear time logic (LTL). Computing a winning strategy to an infinite game whose winning condition is the set of LTL properties is the main step in obtaining an implementation. We present a practical hybrid algorithm—a combination of symbolic and explicit algorithm—for the computation of winning strategies for unrestricted LTL games that we have successfully applied to synthesize reactive systems with up to $10^{11}$ states.

## 1 Introduction

Great progress has been made in the verification of reactive systems over the last twenty years. The combination of sophisticated algorithms, powerful abstraction techniques, and rigorous design methodologies has made the verification of large hardware and software systems possible. Synthesis from specifications given as (temporal) logic formulae or automata [7, 11, 39] has proved a more difficult problem and has enjoyed less success in spite of the important applications that depend on efficient solutions of the synthesis problem. In particular, debugging and repair are promising fields in which techniques based on synthesis algorithms have found employment [23, 25].

Recent algorithmic advances in the determinization of Büchi automata and in the solution of parity games have renewed hope that realistic systems may be synthesized from their temporal specifications. In this paper we propose a hybrid approach to this problem that combines symbolic algorithms (operating on the characteristic functions of sets) and explicit algorithms (that manipulate individual set members).

Specifications are often made up of several relatively simple components—for instance, a collection of LTL properties. If that is the case, our approach scales well because it applies the expensive explicit processing steps to the individual components of the specification, and adopts symbolic techniques where they matter most—in the solution of the final generalized parity game. Preliminary experiments demonstrate that the approach is effective in dealing with rather large systems even in its current prototypical form. For instance, we were able to synthesize an optimal Nim player from a description of the game bookkeeping and the property that requires victory from each winning position.

* This work was supported in part by SRC contract 2006-TJ-1365.

F. Logozzo et al. (Eds.): VMCAI 2008, LNCS 4905, pp. 309–323, 2008.

Our approach converts each component of the specification into either a Büchi automaton or a parity automaton of minimum index. The Büchi automaton can be nondeterministic if it fair simulates the deterministic parity automaton obtained from its determinization by Piterman's procedure. We show that in that case the parity automaton must have a parity index less than or equal to two. The reactive system implementing the specification is derived by symbolically computing the winning strategies of a nondeterministic concurrent parity game obtained by composition of the several automata.

The rest of this paper is organized as follows. Section 2 recalls the notions on $\omega$ automata and games that are pertinent to this paper. Section 3 summarizes the algorithm. Section 4 discusses algorithmic choices for symbolic implementations. Section 5 discusses related work. Section 6 presents our experiment results and Sect. 7 concludes.

## 2 Automata and Games

A finite automaton on $\omega$-words $\langle \Sigma, Q, q_{\text{in}}, \delta, \alpha \rangle$ is defined by a finite alphabet $\Sigma$, a finite set of states $Q$, an initial state $q_{\text{in}} \in Q$, a transition function $\delta : Q \times \Sigma \to 2^Q$ that maps a state and an input letter to a set of possible successors, and an acceptance condition $\alpha$ that describes a subset of $Q^\omega$, that is, a set of infinite sequences of states. A run of automaton $M$ on $\omega$-word $w = w_0 w_1 \ldots$ is a sequence $q_0, q_1, \ldots$ such that $q_0 = q_{\text{in}}$, and for $i \geq 0$, $q_{i+1} \in \delta(q_i, w_i)$. A run is accepting iff (if and only if) it belongs to the set described by $\alpha$, and a word is accepted iff it has an accepting run in $M$. The subset of $\Sigma^\omega$ accepted by $M$ is the ($\omega$-regular) language of $M$. A *deterministic* automaton is such that $\delta(q, \sigma)$ is a singleton for all states $q \in Q$ and all letters $\sigma \in \Sigma$.

Several ways of specifying the acceptance condition $\alpha$ are in use. In this paper we are concerned with Büchi [6], co-Büchi, parity [36, 13], Rabin [40], and Streett [46] acceptance conditions. All these conditions are concerned with the sets of states that occur infinitely often in a run; for run $\rho$, this set is written $\mathsf{inf}(\rho)$. Büchi and co-Büchi acceptance conditions are both given as a set of states $F \subseteq Q$. A run $\rho$ is accepting for a Büchi (co-Büchi) condition iff $\mathsf{inf}(\rho) \cap F \neq \emptyset$ ($\mathsf{inf}(\rho) \cap F = \emptyset$). A parity acceptance condition is given as a function assigning a *priority* to each state of the automaton. Letting $[k] = \{i \mid 0 \leq i < k\}$, a parity condition of index $k$ is a function $\pi : Q \to [k]$. A run $\rho$ is accepting iff $\max\{\pi(q) \mid q \in \mathsf{inf}(\rho)\}$ is odd; that is, if the highest recurring priority is odd.

Rabin and Streett are given as a set of pairs of sets of states: $\{(U_1, E_1), \ldots, (U_k, E_k)\}$; $k$ is called the *index* of the condition. A run $\rho$ is accepted according to a Rabin (Streett) condition iff there exists $i$ such that $\mathsf{inf}(\rho) \cap U_i \neq \emptyset$ and $\mathsf{inf}(\rho) \cap E_i = \emptyset$ ( for all $i$, $\mathsf{inf}(\rho) \cap U_i = \emptyset$ or $\mathsf{inf}(\rho) \cap E_i \neq \emptyset$). Rabin and Streett acceptance conditions are complementary just as Büchi and co-Büchi are. A parity condition $\pi : Q \to [k]$ such that $k$ is even can be easily converted to a Rabin condition with $k/2$ pairs; hence, parity conditions are also known as *Rabin chain* conditions. It is also easy to translate $\pi$ to a Streett condition. A parity condition $\pi_c$ complementary to $\pi$ is obtained by letting $\pi_c(q) = \pi(q) + 1$ for all $q \in Q$.

Büchi, co-Büchi, and parity acceptance conditions may be *generalized*. A generalized Büchi condition consists of a collection $\mathcal{F} \subseteq 2^Q$ of Büchi conditions. A run $\rho$ is accepting for a generalized Büchi (co-Bc̈hi) condition iff it is accepting according to

each $F \in \mathcal{F}$ (some $F \in \mathcal{F}$). A generalized parity condition may be either conjunctive or disjunctive and is given as a collection $\Pi$ of priority functions. A run $\rho$ is accepting according to a conjunctive (disjunctive) condition $\Pi$ iff it is accepting according to each (some) $\pi \in \Pi$. Disjunctive and conjunctive generalized parity conditions are dual in the same sense in which Rabin and Streett conditions are and extend them just as Rabin and Streett conditions extend generalized co-Büchi and Büchi conditions.[1]

An $\omega$-regular automaton equipped with a Büchi acceptance condition is called a Büchi automaton; likewise for the other acceptance conditions. In this paper, we adopt popular three-letter abbreviations to designate different types of automata. The first letter of each abbreviation distinguishes nondeterministic (N) from deterministic (D) structures. The second letter denotes the type of acceptance condition: Büchi (B), co-Büchi (C), Rabin (R), Streett (S), and parity (P). The final letter indicates that the automata read infinite words (W). As examples, NBW designates a nondeterministic Büchi automaton (on infinite words), while DPW is the acronym for a deterministic parity automaton (also on infinite words).

Despite their similarity to automata on finite words DBWs are less expressive than NBWs and are not closed under complementation; accordingly, determinization is only possible in general by switching to a more powerful acceptance condition and complementation of NBWs cannot be accomplished by determinization followed by complementation of the acceptance condition. Piterman [37] has recently improved Safra's procedure [43] so as to produce a DPW (instead of a DRW) from an NBW. The construction extends the well-known subset construction for automata on finite words. Rather than labeling each state of the deterministic automaton with a subset of states of the NBW, it labels it with a tree of subsets. As a result, the upper bound on the number of states of the DPW derived from an NBW with $n$ states is $n^{2n+2}$. This fast-growing function discourages determinization of large NBWs. Concerning determinization, it should be noted that generalizing Büchi and co-Büchi conditions provides convenience and conciseness, but does not increase expressiveness. On the other hand, generalized Büchi games, just like Streett games, do not always admit memoryless strategies, to be discussed shortly.

Linear Time Logic (LTL) [49, 31] is a popular temporal logic for the specification of nonterminating reactive systems. LTL formulae are built from a set of atomic propositions, Boolean connectives, and basic temporal operators $\mathsf{X}$ (next), $\mathsf{U}$ (until), and $\mathsf{R}$ (releases). Derived operators $\mathsf{G}$ (always) and $\mathsf{F}$ (eventually) are usually included for convenience. Procedures exist (e.g., [17]) to translate an LTL formula into an NBW that accepts the language defined by the formula. On the one hand, if not all $\omega$-regular languages can be expressed in LTL, DBWs are not sufficient to translate all of LTL.[2]

Piterman's determinization procedure provides a way to find a DBW equivalent to an NBW whenever it exists. A set of states in an $\omega$-regular automaton $M$ is *essential* if it equals $\mathsf{inf}(\rho)$ for some run $\rho$ of $M$. A *positive chain* of length $m$ is a sequence of $m$ essential sets $R_1 \subset \cdots \subset R_m$ such that $R_i$ satisfies the acceptance condition of $M$

[1] Specifically, Rabin and Streett pairs can be seen as parity conditions with three colors, while co-Büchi and Büchi conditions can be seen as parity conditions with two colors.

[2] In fact, LTL formulae exist that describe $\omega$-regular languages with arbitrarily large Rabin indices. [30].

iff $i$ is odd. The *Rabin index* of an $\omega$-regular language $L$ is the minimum $k$ such that there exists a DRW with $k$ pairs recognizing $L$. The Rabin index $I(L)$ of language $L$ is related to the maximal length $\Xi(M)$ of a positive chain in a deterministic automaton $M_L$ accepting $L$ by the equation $I(L) = \lfloor(\Xi(M_L) + 1)/2\rfloor$ [48]. Carton and Maceiras have devised an algorithm that finds $I(L)$ given a DPW that recognizes $L$ in time $O(|Q|^2|\Sigma|)$ [9]. Moreover, every DPW $M$ that recognizes $L$ can be equipped with a new parity condition $\pi : Q \to [\Xi(M) + 1]$ without changing the accepted language. The following procedure therefore yields a DBW from an NBW if one exists: Convert NBW $N$ to an equivalent DPW $D$ by Piterman's procedure. Compute $\Xi(D)$ with the algorithm of Carton and Maceiras. If $\Xi(D) \leq 1$ the equivalent parity condition with $\leq 2$ priorities computed together with $\Xi(D)$ can be interpreted as a Büchi acceptance condition; otherwise no DBW equivalent to $N$ exists. (If $\Xi(D) = 0$, $N$ accepts the empty language.)

Deterministic $\omega$-regular automata can be used to define infinite games [47] in several ways. Here we consider turn-based and input-based two-player games, in which Player 0 (the *antagonist*) and Player 1 (the *protagonist*) move a token along the transitions of the automaton. If the resulting infinite sequence of states is accepted by the automaton, Player 1 wins, otherwise Player 0 wins. In *turn-based* games the set of states $Q$ is partitioned into $Q_0$ (antagonist states) and $Q_1$ (protagonist states). Each player moves the token from its states by choosing a letter from $\Sigma$ and the corresponding successor according to $\delta$. In *input-based* games, the alphabet $\Sigma$ is the Cartesian product $\Sigma_0 \times \Sigma_1$ of an antagonist alphabet and a protagonist alphabet. In state $q$, Player $i$ chooses a letter $\sigma_i \in \Sigma_i$. The token is then moved to $\delta(q, (\sigma_0, \sigma_1))$. Turn-based games are games of perfect information, whereas in input-based games a player may have full, partial, or no advance knowledge of the other player's choices. The amount of information available to one player obviously affects its ability to win the game. If one player has knowledge of the move of the other, then input-based games are easily reduced to turn-based games and are therefore determinate. In our input-based games we assume that Player 1 has no advance knowledge of the other player's choices, while Player 0 sees the opponent's moves.

The existence and computation of winning strategies are central problems in the study of infinite games. A strategy is a function that defines which letter a player should choose at each move. A strategy for Player $i$ in a turn-based game can be defined equivalently as either a function $\tau_i : Q^* \times Q_i \to \Sigma$, or as a function: $\tau_i : Q_i \times S_i \to \Sigma \times S_i$. The set $S_i$ is the player's *memory*; according to its cardinality, strategies are classified as *infinite memory*, *finite memory*, and *memoryless* (or *positional*). For input-based games in which Player 1 plays without knowing the opponent's choices but Player 0 knows what Player 1 has chosen, a strategy for Player 1 is defined as either a function $\tau_1 : Q^* \times Q \to \Sigma_1$ or a function $\tau_1 : Q \times S_1 \to \Sigma_1 \times S_1$ and a strategy for Player 0 is defined as either a function $\tau_0 : Q^* \times Q \times \Sigma_1 \to \Sigma_0$ or a function $\tau_0 : Q \times S_0 \times \Sigma_1 \to \Sigma_0 \times S_0$. A strategy $\tau_i$ is *winning* for Player $i$ from a given state of the automaton iff victory is secured from that state regardless of the opponent's choices if Player $i$ plays according to $\tau_i$.

The acceptance condition of the automaton translates into the *winning condition* of the game. We consider Büchi, co-Büchi, and parity games, their counterparts with

generalized winning conditions, as well as Rabin and Streett games. All these games are *determinate* [32]; that is, from each state of the automaton if a player has no winning strategy then the opponent has a winning strategy. Büchi, co-Büchi, generalized co-Büchi, Rabin, parity, and disjunctive generalized parity games admit memoryless strategies. The others—generalized Büchi, Streett, and conjunctive generalized parity games—admit finite memory strategies [50].

If the winning condition of an infinite game is given as an LTL formula on the states of the automaton we have an *LTL game*. Such a game can be solved by translating the formula into an equivalent deterministic $\omega$-automaton and composing it with the graph of the given automaton. As recalled above, not all LTL formulae have an equivalent DBW (or DCW for that matter). Therefore, determinization to some more powerful type of automaton is required in general. With Piterman's improvement of Safra's construction [37], the parity condition is the natural choice.

If an NBW derived from the given LTL formula is used in solving an LTL game, there is in general no guarantee that a winning solution will be found if one exists. (See [19] for an example.) Henzinger and Piterman [21, Theorem 4.1] have shown, however, that a nondeterministic automaton may still be used with the guarantee of a winning solution if it fair simulates an equivalent deterministic automaton. (This result subsumes [19, Theorem 1] about *trivially determinizable* NBWs.)

An $\omega$-regular automaton $P = \langle \Sigma, Q_P, q_{P\text{in}}, \delta_P, \alpha_P \rangle$ *fair simulates* [20] another such automaton $A = \langle \Sigma, Q_A, q_{A\text{in}}, \delta_A, \alpha_A \rangle$ with the same alphabet if Player 1 has a winning strategy for the following turn-based game: Initially, the protagonist token is placed on $q_{P\text{in}}$ and the antagonist token is placed on $q_{A\text{in}}$. At each turn, let $p \in Q_P$ be the state with the protagonist token and let $a \in Q_A$ be the state with the antagonist token. Player 0 chooses a letter $\sigma \in \Sigma$ and moves the $A$ token to one of the states in $\delta_A(a, \sigma)$. Player 1 then moves the $P$ token to one of the states in $\delta_P(p, \sigma)$. Player 1 wins if either the run of $A$ is not in $\alpha_A$ or the run of $P$ is in $\alpha_P$. A winning strategy for Player 1 is a function $\tau : (Q_A \times Q_P \times \Sigma)^+ \to Q_P$ that is consistent with $\delta_P$ ($\forall a \in Q_A, p \in Q_P, \sigma \in \Sigma \,.\, \tau(a, p, \sigma) \in \delta_P(p, \sigma)$) and that guarantees victory regardless of the opponent's choices.

When a game played on an $\omega$-regular automaton has nondeterministic transitions one needs to define which player is in charge of resolving nondeterminism. In an input-based game derived from an LTL game, Player 1, whose objective is to force the run of the automaton to satisfy the LTL formula, chooses the next state from $\delta(q, (\sigma_0, \sigma_1))$. A nondeterministic automaton for a given language can be much more compact than a deterministic one. Hence, [21, Theorem 4.1] may lead to considerable savings in the computation of winning strategies. On the negative side, we offer the following theorem, which implies that an NBW can only be used if an equivalent DBW exists.

**Theorem 1.** *Let $N$ be an NBW and $D$ an equivalent DPW. Let $D$ be of minimum index $k > 2$. Then $N$ does not fair simulate $D$.*

*Proof.* We assume that $N$ has a strategy $\tau$ to simulate $D$ and show that this leads to a contradiction. Since the winning condition of the simulation game is the disjunction of two parity conditions, we can assume that $\tau$ is memoryless. Since $k > 2$ there is a chain of essential sets

$$R_1 \subset R_2 \subset \cdots \subset R_{k-1}$$

with $R_{2i+1}$ accepting and $R_{2i}$ rejecting. Let $p$ be a state of $D$ in $R_1$. (By the chain property, $p$ is also in $R_2$.) Let $u \in \Sigma^*$ be a word that takes $D$ from the initial state to $p$. Let $v \in \Sigma^* \setminus \{\epsilon\}$ be a word that takes $D$ from $p$ back to $p$ while visiting all states of $R_1$ at least once. Finally, let $w \in \Sigma^* \setminus \{\epsilon\}$ be a word that takes $D$ from $p$ back to $p$ while visiting all states of $R_2$ at least once. The existence of $p$, $u$, $v$, and $w$ is guaranteed because $R_1$ and $R_2$ are essential sets.

We construct an ultimately periodic word $x$ as follows. We initialize $x$ to $u$ and $\Gamma = \emptyset$. Let $q_0^1$ be the state reached by $N$ when reading $x$ and following $\tau$. Append copies of $v$ to $x$ and extend the run of $x$ in $D$ and $N$. The run of $D$ will reach $p$ every time a copy of $v$ is added. The run of $N$ will go through states $q_1^1, q_2^1, \ldots$. Stop as soon as, for some $j > 0$ and $i < j$, $q_i^1 = q_j^1$. Call $q^1$ this repeating state, and add it to $\Gamma$. (Repetition is inevitable because $N$ is finite.) Append $w$ to $x$ and call $q_0^2$ the state reached by $N$ after running over the updated $x$. ($D$ will be in $p$.) Now append more copies of $v$ to $x$ until there are $j$ and $i < j$ such that $q_i^2 = q_j^2$. Call $q^2$ this repeating state. If $q^2$ is not in $\Gamma$ add it, append $w$ to $x$ and repeat; otherwise stop.

This process must terminate because $|\Gamma|$ grows at each iteration. At termination, $q_i^n = q^m$ for some $q^m \in \Gamma$ and $m < n$. We let $x = yz^\omega$, with $y$ the prefix of $x$ up to the first occurrence of $q^m$ and $z$ the segment between the first and second occurrence.

$D$ rejects $x$ because its essential set is $R_2$, but accepts $x' = yv^\omega$ whose essential set is $R_1$. Consider now $N$. We know that the run of $N$ on $x'$ according to $\tau$ must repeatedly go through $q^m$. Since $x'$ is accepted by $D$, the segments of the run in $N$ between occurrences of $q^m$ must visit some accepting state of $N$. However, this implies that $x$ is also accepted because the run of $x$ also goes through $q^m$ when $D$ is in $p$, and $q^m$ is in $\Gamma$ because it was seen twice in conjunction with $p$ while applying $v$ repeatedly. Since $D$ and $N$ are equivalent the assumption that $\tau$ exists is contradicted and the theorem is proved. □

For lack of space we do not present the extension of Theorem 1 to simulations between two arbitrary parity automata.

## 3 Algorithm

We describe an algorithm that takes as input a collection of LTL formulae and NBWs over an alphabet $\Sigma = \Sigma_0 \times \Sigma_1$. The input is converted into a conjunctive generalized parity game with one parity function for each formula and automaton. At each turn, Player 0 chooses a letter from $\Sigma_0$ and Player 1 chooses a letter from $\Sigma_1$. The objective of Player 1 is to satisfy the conjunctive generalized parity acceptance condition. A winning strategy for Player 1 from the initial state of the parity automaton thus corresponds to an implementation of a reactive system that reads inputs from alphabet $\Sigma_0$ and produces outputs from alphabet $\Sigma_1$. The reactive system satisfies all the linear-time properties given as LTL formulae or Büchi automata from its initial state. If no such winning strategy exists, there exists no implementation of the given specification.

As an initial step, all LTL formulae are converted to NBWs (using Wring [45]). The objective of the algorithm is to be efficient for practical specifications, which

often consist of the conjunction of many simple properties. While in theory one could conjoin all the NBWs to obtain one large NBW and then determinize it, the high complexity of determinization makes this approach infeasible. Instead, each NBW that is not also a DBW is converted to a DPW individually with Piterman's procedure [37]. This keeps the size of the resulting DPWs reasonably small, as discussed in Sect. 6. A parity condition of minimum index is then computed for each DPW by the procedure of [9].

If the parity index is 2, then fair simulation between the NBW and the DPW (which is in fact in this case a DBW), is checked with the algorithm of [26] (implemented as in [14, 18]). If the DPW is simulated by the NBW, the latter replaces the former. (See Theorem 1. From a practical standpoint it should be noted that the NBW is made complete before checking for fair simulation, because otherwise the check is guaranteed to fail.) If there is no fair simulation, the DPW is optionally simplified. (This is done after reducing the index, to increase the chance of simplification.) All the processing up to this point is done by explicit (i.e., non-symbolic) algorithms. At the end of this phase, each specification has been translated into one of the following: a DBW, a DPW, or an NBW that simulates an equivalent DBW.

The next step of processing reduces the collection of automata to a generalized parity game [10]. The transition structures of the automata are converted into one symbolic transition relation—as is customary in symbolic model checking. Effective ways of avoiding blow-up in the composition of the transition relations are well-known [15, 41, 35]. The parity (or Büchi) conditions for all the automata collectively form the generalized parity condition.

We use the "classical" algorithm described in [10] to compute winning strategies for generalized parity conditions. This algorithm is based on [22], which in turn extends Zielonka's algorithm for parity conditions [50].[3] The generalized parity algorithm selects one parity condition and tries to prove all states winning for Player 1, using the maximum color from the selected priority function and recurring on a subgame for the remaining colors and parity conditions.

If Player 1 wins in all the states, the algorithm proceeds to the next parity condition. If, on the other hand, Player 0 has some winning states, the algorithm restarts on a game graph that is pruned of the winning states of Player 0.

At a given recursion level, each parity condition produces a sub-strategy for Player 1. Therefore, Player 1 uses a counter to rotate among these sub-strategies. For a fixed order in the priority conditions, the total memory required is bound by $\prod_{1 \leq i \leq k}(k-i+1)^{d_i}$, where $2d_i + 1$ is the number of colors of the $i$-th priority condition.

The choice of this algorithm over the dominion-based algorithm in [10, 27] is for two reasons. The first is the unsuitability of the dominion-based algorithm to symbolic implementation as discussed in Sect. 4. The second is that the dominion-based algorithm has better asymptotic bounds only when the number of colors is comparable to the number of states. However, in our application, this is seldom, if ever, the case.

The basic *attractor* computation for each player in the generalized parity algorithm is based on an extension of the $\mathsf{MX}$ operator discussed in [23]. Specifically, the set of

[3] The algorithm in [10] contains a bug that is easily fixed by reverting the termination condition of the inner loop to the one given in [22].

states that Player $i$ can control to a target set of states $T \subseteq Q$ (the attractor of $T$ for Player $i$) is given by:

$$\mathsf{MX}_1\, T = \{q \mid \exists \sigma_1 \in \Sigma_1 \,.\, \forall \sigma_0 \in \Sigma_0 \,.\, \exists q' \in \delta(q, (\sigma_0, \sigma_1)) \,.\, q' \in T\}$$
$$\mathsf{MX}_0\, T = \{q \mid \forall \sigma_1 \in \Sigma_1 \,.\, \exists \sigma_0 \in \Sigma_0 \,.\, \forall q' \in \delta(q, (\sigma_0, \sigma_1)) \,.\, q' \in T\} \ .$$

This formulation implies that Player 1 chooses $\sigma_1$ from $\Sigma_1$ first, then Player 0 chooses $\sigma_0$ from $\Sigma_0$, and finally nondeterminism is resolved in favor of Player 1. (That is, all nondeterminism is due to the NBWs representing the properties.) Since Player 1 has no knowledge of the upcoming opponent's move, it has a winning strategy only if a Moore-style implementation of the specification exists.

## 4 Practical Symbolic Algorithms

In the context of verification and synthesis of reactive systems, symbolic algorithms are broadly defined as those algorithms that employ characteristic functions to represent sets. The use of Binary Decision Diagrams [5] to manipulate Boolean functions, in particular, is typically associated with the idea of symbolic algorithms. Techniques like the symbolic model checking algorithm of McMillan [33] have significantly contributed to the success of formal verification thanks to their ability to deal—albeit not with uniform efficiency—with sets of size well beyond the capabilities of more conventional, explicit algorithms. While such successes encourage the use of symbolic algorithms, not all algorithms are amenable to symbolic implementation, leading to a conflict between the choice of one with lowest complexity and one that is symbolic-friendly. Our approach is to use the best algorithm in terms of worst-case complexity that is efficiently implementable in a symbolic manner. In this Section, we try to identify some algorithmic features that are best suited to symbolic implementations.

Obviously, algorithms that process the elements of a large set one by one draw only modest benefit from representing such a set symbolically and are limited to manipulating relatively small sets. Some algorithms resort to picking a (hopefully) small number of seed elements from a large set. Examples are provided by the subquadratic symbolic algorithms for Strongly Connected Component (SCC) analysis [2, 16] that grow an SCC with symbolic techniques starting from an individual seed state. (In this context, complexity refers to the number of *images* and *preimages* computed.) While these algorithms are rightfully considered symbolic, it should be noted that for cycle detection in very large graphs they tend to be outperformed by "more symbolic" algorithms based on computing a hull of the SCCs [42] in spite of their better complexity bounds. Closer to the subject of this paper, one can compare two variants of McNaughton's algorithm [34] (adapted to parity automata by Zielonka [50]) that appear in [47] and [27]. The algorithm in [27] is "more symbolic" than the one in [47] because it computes the attractor of all states of maximum priority at once instead of picking one state from that set and computing its attractor. Notice that the computation of the attractor, which is at the heart of the generalized parity algorithm, is very similar to the fixpoint computations performed in symbolic model checking, and therefore eminently suitable for symbolic implementation. Consequently, one can also leverage various techniques to speed up symbolic model checking in the implementation of the algorithms in [50, 22, 10].

Both variants of McNaughton's algorithm, on the other hand, are far more amenable to symbolic implementation than the algorithms that have superseded them in terms of asymptotic performance. Consider the small progress measure algorithm of Jurdziński [26]. If an SCC of the game graph contains a losing position, the number of iterations is bounded from below by the number of states of a certain priority in the SCC. For a large graph that number may be large enough to prevent termination in practice, even with the optimization of [12]. Another problem with the algorithm of [26] is the need to attach and manipulate a vector of integers to every state of the game graph. As the number of distinct measures increases in the course of the computation, the size of the decision diagrams representing the map from states to measures may easily blow up. Similar observations apply to the algorithms of [1] and the small dominion versions of the ones in [10, 27]. Therefore, we use an explicit implementation of the algorithm of [26] to check fair simulations of small automata derived from individual properties, but prefer the algorithm in [10] for the analysis of large games obtained compositionally.

Besides avoiding explicit enumeration of the elements of large sets, successful symbolic algorithms also limit the arity of relations that are represented symbolically. Even ternary relations, as encountered in the computation of the transitive closure of a graph, tend to impact performance significantly. Finally, the need to represent arbitrary subsets of the powerset of a large set puts a very severe limit on the size of the problems that can be handled by a symbolic algorithm. While $10^{20}$ states have become a rather conservative estimate of what can be done in model checking, algorithms that allocate one BDD variable to each state in a set so as to represent collections of subsets by the characteristic functions of their occurrence vectors (as required by the algorithms of [30, 21]) are limited in most cases to a hundred states or less. For such reasons we prefer an explicit implementation of Piterman's improved determinization procedure [37] to the approach of [21].

## 5 Related Work

In Sect. 4 we have discussed the relation and indebtedness of our work to [26, 1, 27, 22, 21, 10]. An alternative to our approach is to translate the set of parity conditions arising out of the Piterman's procedure to Streett conditions that are then converted to a single parity condition with the algorithm of [8]. However, the algorithm of [22, 10] has better worst-case complexity.

The approaches of [19] and [38, 3] are symbolic, but are restricted in the class of specifications that can be dealt with. In [38] it is noted that checking the realizability of a formula of the form

$$\bigwedge_{1\le i\le m} (\mathsf{G}\,\mathsf{F}\,J_i^1) \rightarrow \bigwedge_{1\le i\le n} (\mathsf{G}\,\mathsf{F}\,J_i^2) \ , \tag{1}$$

where each $J_i^j$ is propositional (a generalized Reactivity(1) formula) can be done in time proportional to $(nm|\Sigma|)^3$, where $\Sigma$ is the set of atomic propositions in (1). Moreover, formulae of the form

$$\bigwedge_{1\le i\le p} \mathsf{G}\,B_i \ , \tag{2}$$

where the only temporal operator allowed in each $B_i$ is the next-time operator $\mathsf{X}$, effectively correspond to the description of transition relations and can be directly regarded as the description of the game graph. This second observation applies also to our approach, while a specification like (1), can be translated into a DPW with $O(mn)$ states, $O(mn|\Sigma|)$ transitions, and 3 colors. Therefore, the class of specifications handled by [38] can be handled efficiently by our algorithm, which on the other hand, can deal with full LTL.

An approach to LTL synthesis that does not require determinization was introduced in [30, 29]. An implementation is described in [24]. From an LTL formula one derives a universal co-Büchi tree automaton (UCT), and transforms it into an NBT. A witness to language nonemptiness, if it exists, corresponds to a winning strategy. While this "Safraless" approach has the same worst-case complexity as the "Safraful" approach based on determinization, its proponents claim three main advantages for it. First, that its implementation is simpler than that of the determinization procedure. Second, that the Safraless approach lends itself to symbolic implementation. Third, that optimizations can be applied at various steps of the Safraless approach to combat the state explosion. Our implementation of Piterman's procedure, however, is only a few hundred lines of code and took only a few days to write and debug. Concerning the symbolic implementation, the Safraless approach requires the manipulation of sets of sets of states. As discussed in Sect. 4, this greatly reduces the effectiveness of a symbolic implementation. The ability to apply intermediate optimizations is beneficial, and the approach is efficient when the UCT is weak. On the other hand, for strong UCT, the approach is practically unable to prove nonrealizability [24]. It appears that only a thorough experimental comparison, beyond the scope of this paper, could ascertain the relative practical strengths of the Safraless and determinization-based approaches.

The work of Jobstmann *et al.* [25] addresses the extraction of efficient implementations from the winning strategies, which is something we do not address in this paper.

Sebastiani and Tonetta [44] present a procedure that strives to produce a deterministic Büchi automaton from an LTL formula. Their approach may improve the performance of our translator by reducing the number of instances in which Piterman's determinization procedure is invoked. The procedure of [44], however, is heuristic. Consider the following LTL formula:

$$\varphi = p \wedge (p \,\mathsf{U}\, \mathsf{G}((\neg p \rightarrow \mathsf{X}\, p) \wedge (p \rightarrow \mathsf{X}\, \neg p))) \ .$$

Let $\psi = \mathsf{G}((\neg p \rightarrow \mathsf{X}\, p) \wedge (p \rightarrow \mathsf{X}\, \neg p))$ and $\theta = p \,\mathsf{U}\, \psi$. Expansion yields

$$\begin{aligned}
\psi &= (p \wedge \mathsf{X}(\neg p \wedge \psi)) \vee (\neg p \wedge \mathsf{X}(p \wedge \psi)) \\
\varphi &= (p \wedge \mathsf{X}(\neg p \wedge \psi)) \vee (p \wedge \mathsf{X}\, \theta) \\
\neg p \wedge \psi &= \neg p \wedge \mathsf{X}(p \wedge \psi) \\
p \wedge \psi &= p \wedge \mathsf{X}(\neg p \wedge \psi) \\
\theta &= (p \wedge \mathsf{X}(\neg p \wedge \psi)) \vee (\neg p \wedge \mathsf{X}(p \wedge \psi)) \vee (p \wedge \mathsf{X}\, \theta) \ .
\end{aligned}$$

This is a set of closed covers. The cover for $\varphi$ is nondeterministic and branch postponement is not applicable. Hence, the MoDeLLA algorithm would (initially) produce a nondeterministic automaton. However, there exists a deterministic Büchi automaton for

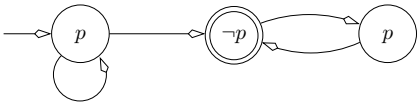


**Fig. 1.** Deterministic automaton for $\varphi = p \wedge (p \,\mathsf{U}\, \mathsf{G}((\neg p \rightarrow \mathsf{X}\, p) \wedge (p \rightarrow \mathsf{X} \neg p)))$

$\varphi$, shown in Fig. 1. Even though the procedure of [44] does not guarantee a deterministic automaton whenever one exists for the given LTL formula, branch postponement tends to increase the chances that the NBW will fair-simulate the DBW produced by determinization.

## 6 Experiments

The procedure described in Sect. 3 has been implemented as an extension of Wring [45] and Vis [4]. In this section we report on some preliminary experiments intended to test the claim of practicality and scalability of the proposed approach. Each of the following specifications was synthesized, and the the resulting model was verified against the specification. In these experiments we disabled the use of NBWs in the parity game.

*Generalized Büchi.* In this experiment, Player 1 seeks to satisfy a generalized Büchi condition $(\mathsf{G}\,\mathsf{F}\, b) \wedge (\mathsf{G}\,\mathsf{F}\, c)$ in a simple automaton. The strategy uses one bit of memory.

*NBW7.* In this experiment the specification is an NBW for $(\mathsf{F}\,\mathsf{G}\, p) \wedge (\mathsf{G}\,\mathsf{F}\, q)$. The language is not in DBW or DCW and as a result the NBW is translated into a DPW of index 3. In the game, Player 1 controls both $p$ and $q$ and wins from all states of the automaton. Since there is only one parity condition, the strategy is memoryless.

*Simple Arbiter.* Three DBWs specify this simple synchronous arbiter that grants request even if they are not persistent, guarantees fairness, and produces no spontaneous grants. Player 0 chooses the values of the two request signals and Player 1 chooses the values of the grant signals. The specification is symmetric with respect to the two clients, and the strategies computed by the program come in symmetric pairs.

*Round Robin Arbiter.* This experiment synthesizes an arbiter from a collection of 10 safety properties from [28]. This is a more detailed specification than the one of the simple arbiter, as it prescribes, for example, how ties should be broken. Once again, Player 1 controls the grant signals, while the opponent controls the requests.

*Combination Lock.* In this experiment the objective is to synthesize the finite state machine controller of a combination lock. Given a counter that can be incremented or decremented by a user and represents the dial of the lock, and a set of seven properties that prescribe that the lock opens iff the correct combination is entered, Player 1 seeks a strategy for the update of the control machine of the lock, while Player 0 operates the counter under a fairness constraint.

*Nim Player.* The game of Nim is played with several piles of counters. A move consists of taking one or more counters from one pile. The player who removes the last counter wins. The Sprague-Grundy theory of impartial games applies to it and it is known which player has a winning strategy from a given position and how to play according to such strategy. In this experiment, the specification is split between a model that does the game bookkeeping and plays one side of the game, and an LTL property:

$$\mathsf{G}((\neg \mathtt{turn} \wedge \mathtt{winning}) \rightarrow \mathsf{F}\,\mathtt{win}) \ ,$$

which says that when the environment moves from a winning position, it always wins. Satisfying this property entails synthesizing an optimal player. The bookkeeper has a fixed number of piles, but chooses nondeterministically how many counters to place on each pile at the start of a play. This choice is given to Player 0 in the game, while Player 1 plays the environment.

**Table 1.** Experimental data

| name | Spec | | Int | | colors | bits | $\sigma$ | | $\sigma_o$ | | CPU (s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LTL | NBW | DBW | DPW | | | edges | nodes | edges | nodes | pp | sol |
| GB | 0 | 2 | 2 | 0 | 4 | 1 | 10 | 8 | 8 | 3 | 0 | 0 |
| NBW7 | 0 | 1 | 0 | 1 | 3 | 0 | 10 | 7 | 6 | 1 | 0 | 0 |
| simple | 0 | 3 | 3 | 0 | 6 | 6 | 44 | 53 | 398 | 12 | 0 | 0.01 |
| rrobin | 10 | 0 | 10 | 0 | 20 | 17 | 250 | 112 | 16588800 | 66 | 13.55 | 0.1 |
| lock | 7 | 0 | 5 | 2 | 16 | 13 | 3196 | 302 | 1277280 | 178 | 2.71 | 1.35 |
| nim | 0 | 1 | 1 | 0 | 2 | 1 | 8.29e+11 | 3410 | 1.87e+11 | 298 | 0 | 22.63 |

The results of the experiments are summarized in Table 1. For each game the number of LTL formulae and NBWs in the specification are given, followed by two columns that give the statistics of processing each formula or automaton. The remaining columns describe the resulting generalized parity game and its solution by reporting the total number of different priorities (or colors) of the parity acceptance condition, the number of binary state variables of the strategy automaton, the number of edges in the union of all winning strategies computed for Player 1 ($\sigma$), the size of the corresponding BDD, the number of edges in an optimized winning strategy ($\sigma_o$), and the size of the corresponding BDD. Finally, the time to preprocess (pp) and solve the game and compute the optimized strategy (sol) is given. (The optimization of the strategy [3] never takes more than 0.17 s in the experiments of Table 1.) The number of edges in the optimized strategy may be higher than in the bundle of strategy returned by the algorithm because edges from positions that are losing for Player 1 may be added if they help reduce the size of the BDD. Our algorithm does not yet optimize the number of bits in the strategy automaton, which is therefore far from optimal.

## 7 Conclusion

We have presented an algorithm for the computation of strategies for LTL games and, in general, for games whose winning conditions are given by a set of LTL formulae

and Büchi automata. The solution involves determinization, but only on the individual components of the specification. Since these components are typically small, our approach appears to scale well and is capable of handling games with large numbers of states. To that effect symbolic techniques are applied where it matters most—after the individual automata have been composed. The initial experimental results encourage us to continue in the development of this algorithm so that it may address larger and more realistic problems.

## References


[1] Björklund, H., Sandberg, S., Vorobyov, S.: A discrete subexponential algorithm for parity games. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 663–674. Springer, Heidelberg (2003)

[2] Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 37–54. Springer, Heidelberg (2000)

[3] Bloem, R., et al.: Specify, compile, run: Hardware form PSL. In: 6th International Workshop on Compiler Optimization Meets Compiler Verification. Electronic Notes in Theoretical Computer Science (2007), `http://www.entcs.org/`

[4] Brayton, R.K., et al.: VIS: A system for verification and synthesis. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 428–432. Springer, Heidelberg (1996)

[5] Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)

[6] Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proceedings of the 1960 International Congress on Logic, Methodology, and Philosophy of Science, pp. 1–11. Stanford University Press (1962)

[7] Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Trans. Amer. Math. Soc. 138, 295–311 (1969)

[8] Buhrke, N., Lescow, H., Vöge, J.: Strategy construction in infinite games with Streett and Rabin chain winning conditions. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, pp. 207–225. Springer, Heidelberg (1996)

[9] Carton, O., Maceiras, R.: Computing the Rabin index of a parity automaton. Theoretical Informatics and Applications 33, 495–505 (1999)

[10] Chatterjee, K., Henzinger, T.A., Piterman, N.: Generalized Parity Games. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 153–167. Springer, Heidelberg (2007)

[11] Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)

[12] de Alfaro, L., Faella, M.: Accelerated algorithms for 3-color parity games with an application to timed games. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 108–120. Springer, Heidelberg (2007)

[13] Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proc. 32nd IEEE Symposium on Foundations of Computer Science. pp. 368–377 (October 1991)

[14] Etessami, K., Wilke, T., Schuller, A.: Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, Springer, Heidelberg (2001)

[15] Geist, D., Beer, I.: Efficient model checking by automated ordering of transition relation partitions. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 299–310. Springer, Heidelberg (1994)

[16] Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected componenets in a linear number of symbolic steps. In: Symposium on Discrete Algorithms, Baltimore, MD (January 2003)
[17] Gerth, R., et al.: Simple on-the-fly automatic verification of linear temporal logic. In: Protocol Specification, Testing, and Verification, pp. 3–18. Chapman and Hall, Boca Raton (1995)
[18] Gurumurthy, S., Bloem, R., Somenzi, F.: Fair Simulation Minimization. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, Springer, Heidelberg (2002)
[19] Harding, A., Ryan, M., Schobbens, P.-Y.: A new algorithm for strategy synthesis in LTL games. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 477–492. Springer, Heidelberg (2005)
[20] Henzinger, T., Kupferman, O., Rajamani, S.: Fair simulation. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 273–287. Springer, Heidelberg (1997)
[21] Henzinger, T.A., Piterman, N.: Solving games without determinization. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 394–409. Springer, Heidelberg (2006)
[22] Horn, F.: Streett games on finite graphs. In: Workshop on Games in Design and Verification, Edimburgh, UK (July 2005)
[23] Jin, H., Ravi, K., Somenzi, F.: Fate and free will in error traces. Software Tools for Technology Transfer 6(2), 102–116 (2004)
[24] Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: Formal Methods in Computer Aided Design (FMCAD 2006), San Jose, CA, pp. 117–124 (November 2006)
[25] Jobstmann, B., Griesmayer, A., Bloem, R.: Program repair as a game. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 226–238. Springer, Heidelberg (2005)
[26] Jurdziński, M.: Small progress measures for solving parity games. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000)
[27] Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, FL, pp. 117–123 (January 2006)
[28] Katz, S., Grumberg, O., Geist, D.: Have I written enough properties?" — A method of comparison between specification and implementation. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 280–297. Springer, Heidelberg (1999)
[29] Kupferman, O., Piterman, N., Vardi, M.Y.: Safraless compositional synthesis. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 31–44. Springer, Heidelberg (2006)
[30] Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: Foundations of Computer Science, Pittsburgh, PA, pp. 531–542 (October 2005)
[31] Lichtenstein, O., Pnueli, A.: Checking that finite state concurrent programs satisfy their linear specification. In: Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages, New Orleans, pp. 97–107 (January 1985)
[32] Martin, D.A.: Borel determinacy. Annals of Mathematics 102, 363–371 (1975)
[33] McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers, Boston (1994)
[34] McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic 65, 149–184 (1993)
[35] Moon, I.-H., Hachtel, G.D., Somenzi, F.: Border-block triangular form and conjunction schedule in image computation. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 73–90. Springer, Heidelberg (2000)
[36] Mostowski, A.W.: Regular expressions for infinite trees and a standard form of automata. In: Skowron, A. (ed.) SCT 1984. LNCS, vol. 208, pp. 157–168. Springer, Heidelberg (1985)

[37] Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: 21st Symposium on Logic in Computer Science, Seattle, WA, pp. 255–264 (August 2006)
[38] Piterman, N., Pnueli, A., Sa´ar, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005)
[39] Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. Symposium on Principles of Programming Languages (POPL 1989), pp. 179–190 (1989)
[40] Rabin, M.O.: Automata on Infinite Objects and Church's Problem. In: Regional Conference Series in Mathematics, American Mathematical Society, Providence (1972)
[41] Ranjan, R.K., et al.: Efficient BDD algorithms for FSM synthesis and verification. In: Presented at IWLS 1995, Lake Tahoe, CA (May 1995)
[42] Ravi, K., Bloem, R., Somenzi, F.: A comparative study of symbolic algorithms for the computation of fair cycles. In: Johnson, S.D., Hunt Jr, W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 143–160. Springer, Heidelberg (2000)
[43] Safra, S.: Complexity of Automata on Infinite Objects. PhD thesis, The Weizmann Institute of Science (March 1989)
[44] Sebastiani, R., Tonetta, S.: More deterministic" vs. "smaller" Büchi automata for efficient LTL model checking. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 126–140. Springer, Heidelberg (2003)
[45] Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000)
[46] Streett, R.S.: Propositional dynamic logic of looping and converse is elementarily decidable. Information and Control 54, 121–141 (1982)
[47] Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)
[48] Wagner, K.: On $\omega$-regular sets. Information and Control 43(2), 123–177 (1979)
[49] Wolper, P., Vardi, M.Y., Sistla, A.P.: Reasoning about infinite computation paths. In: Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, pp. 185–194 (1983)
[50] Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200(1–2), 135–183 (1998)