A landscape with games in the background

Igor Walukiewicz
LaBRI
Université Bordeaux-1 and CNRS
351, cours de la Libération
33405 Talence Cedex, France

Abstract

An overview of applications of two player path-forming games to verification and synthesis is given. Several extensions of the standard model of finite games with regular winning conditions are discussed. One direction is that of considering non-regular winning conditions. The other concerns the ways games are played, in particular probabilistic and multi-player games.

1 Introduction

We will discuss applications of two player path-forming games on graphs. In most cases they will be particular cases of Borel games [55, 39]. The research reported here is not the mainstream of the classical game theory for at least two reasons: (i) the winning conditions are a bit unusual as they are motivated by the model-checking problem and automata theory (ii) we are interested in the quality of winning strategies. We will give an overview of problems in verification and synthesis were games play a fundamental role. We want to stress that this article is not a survey as it is impossible to do it in a small number of pages. In consequence, an disproportionally big part of the article is devoted to author's results. Other aspects of the subject were covered, among others, in recent invited talks at CSL'03 [23] and at STACS'04 [34].

In what concerns verification, we will mostly concentrate on the model-checking problem, that is the problem of verifying if a given formula holds in a given state of a given model. We will discuss a close connection between this problem and the problem of deciding a winner in some games. This connection turns out to be very robust and of great help even in extensions such like pushdown systems [75] or probabilistic concurrent games [23].

For the synthesis part, we will see that, here too, we can reduce synthesis problems to problems of solving games. Unlike for model-checking, there are many different approaches to the synthesis problem varying significantly, at least at the first sight. Among others, we will discuss a proposal of an extension of the μ-calculus that is capable of expressing in a unified way many of the problems found in the literature.

In the final part of this overview we will consider some recent generalizations. We will present results on various non-regular winning conditions. We will also discuss extensions of the game model both to probabilistic and to multi-player settings.

2 Intuitions and definitions

Intuitions about applications of games we are interested in can be already seen on the level of propositional logic. Consider the task of verifying whether a given propositional formula is true under a given valuation. The solution can be formulated in terms of *model-checking game* between two players who we call Adam and Eve. The goal of Eve is to show that a given formula φ is true under a given valuation V. Assume that all negations in φ appear only before propositional variables. Positions in the game are pairs (V, ψ), where ψ is a subformula of φ. In a position (V, ψ), if ψ is a disjunction then Eve picks a disjunct that she claims to be true under V. If ψ is a conjunction then Adam picks a conjunct that he claims to be false. Afterwards, the process repeats with a chosen subformula in place of ψ. Arriving at a propositional variable or its negation the winner is decided by consulting the valuation. The formula φ is true under V iff Eve has a winning strategy from (V, φ). Admittedly, the structure of this game is very simple but it is enough to add recursive definitions of objects (cf. Section 3) to appreciate advantages of this formulation.

Observe that the above game is not useful for answering the satisfiability question. For this we need to consider a different game that we call *satisfiability-checking game*. The players now deal with sets of formulas $\{\psi_1, \ldots, \psi_k\}$ and the goal of Eve is to show that the conjunction of the formulas in the set is satisfiable. At each turn Eve can replace a disjunction in the current set by one of its disjuncts. Adam can replace a conjunction by the two conjuncts. At the end, when there are no formulas to reduce, Eve wins iff there is no variable and its negation in the set. A formula φ is satisfiable iff Eve has a winning strategy from the position $\{\varphi\}$. An important point here is that we can consider sets of formulas and not multisets, i.e., we have an implicit contraction rule. While it is obvious in this setting, in general this is linked with existence of positional strategies in appropriate class of games. Satisfiability-checking games (or rather their duals) are the starting point of recent game-theoretic approaches to proof-theory [31, 9]. Let us make a digression about extensions of satisfiability-checking games to μ-calculi. While in this case each single game is still simple, the structure of the class of such games, from the point of view of proof-theory, is not well understood yet. This structure is very interesting and nontrivial even in a simpler context where the conjunction and disjunction are free operators [67, 68]. This is also related with fixpoint hierarchy questions [12, 5, 6, 76].

Differences between model-checking and satisfiability-checking games are rather subtle and not well visible in the game structure, in particular with such a high level description. In the above simple examples the most striking difference is that Adam has not much to say in the satisfiability-checking game. This is just a peculiarity of the extremely simple setting. A more pertinent difference between the two kinds of games is that in satisfiability-checking games we deal with sets of formulas and the winning conditions become more complicated.

We will argue below that extensions of model-checking game play a prominent role in verification and are invaluable tools in designing verification algorithms. The satisfiability-checking games appear in the context of synthesis. In a sense that can be made precise these games describe all the models of a formula, i.e., all the systems satisfying the specification. While the formulation of a synthesis problem is more complicated than this, still the connection with satisfiability-checking games is very strong.

We finish this section with the formal definition of games we are interested in. A *game* G is a tuple $\langle V_E, V_A, T \subseteq (V_E \cup V_A)^2, Acc \subseteq V^\omega \rangle$ where Acc is a set defining the *winning condition* and $\langle V_E \cup V_A, T\rangle$ is a graph with the vertices partitioned into those of Eve and those of Adam. We say that a vertex v' is a *successor* of a vertex v if $T(v, v')$ holds.

A *play* between Eve and Adam from some vertex $v \in V = V_E \cup V_A$ proceeds as follows: if $v \in V_E$ then Eve makes a choice of a successor, otherwise Adam chooses a successor; from this successor the same rule applies and the play goes on forever unless one of the parties cannot make a move. The player who cannot make a move looses. The result of an infinite play is an infinite path $v_0v_1v_2\ldots$ This *path is winning* for Eve if it belongs to Acc. Otherwise Adam is the winner.

A *strategy* σ for Eve is a function assigning to every sequence of vertices $\vec{v}$ ending in a vertex v from V_E a vertex $\sigma(\vec{v})$ which is a successor of v. A *play respecting* σ is a sequence $v_0v_1\ldots$ such that $v_{i+1} = \sigma(v_i)$ for all i with $v_i \in V_E$. The *strategy* σ *is winning for Eve* from a vertex v iff all the plays starting in v and respecting σ are winning. A *vertex is winning* if there exists a strategy winning from it. The strategies for Adam are defined similarly.

A *strategy with memory* M is a triple:

$$c : M \times V_E \to \mathcal{P}(V), \quad up : M \times V \to M, \quad m_0 \in M$$

The role of the initial memory element m_0 and the memory update function up is to abstract some information from the sequence $\vec{v}$. This is done by iteratively applying up function:

$$up^*(m, \varepsilon) = m \quad \text{and} \quad up^*(m, \vec{v}v) = up^*(up(m, \vec{v}), v)$$

This way each sequence $\vec{v}$ of vertices is assigned a memory element $up^*(m_0, \vec{v})$. Then the choice function c defines a strategy by $\sigma(\vec{v}v) = c(up^*(m_0, \vec{v}), v)$. The strategy is *memoryless* iff $\sigma(\vec{v}) = \sigma(\vec{w})$ whenever $\vec{v}$ and $\vec{w}$ end in the same vertex; this is a strategy with a memory M that is a singleton.

In most of the cases in this paper the winning conditions $Acc \subseteq V^\omega$ will be *Muller conditions*: that is, there will be a colouring $\lambda : V \to Colours$ of the set of vertices with a finite set of colours and a set $\mathcal{F} \subseteq \mathcal{P}(Colours)$ that define the winning sequences by:

$$\vec{v} \in Acc \quad \text{iff} \quad \mathrm{Inf}_\lambda(\vec{v}) \in \mathcal{F}$$

where $\mathrm{Inf}_\lambda(\vec{v})$ is the set of colours appearing infinitely often on $\vec{v}$.

An important special case is a *parity condition*. It is a condition determined by a function $\Omega : V \to \{0, \ldots, d\}$ in the following way:

$$Acc = \{v_0v_1\ldots \in V^\omega : \liminf_{i\to\infty} \Omega(v_i) \text{ is even}\}$$

Hence, in this case, the colours are natural numbers and we require that the smallest among those appearing infinitely often is even. This condition was discovered by Mostowski [56] and is the most useful form of Muller conditions. It is the only Muller condition that guarantees existence of memoryless strategies [28, 57, 53]. It is closed by negation (the negation of a parity condition is a parity condition). It is universal in the sense that very game with a Muller condition can be reduced to a game with a parity condition [56].

The main results about games that we need are summarized in the following theorem.

Theorem 1 ([51, 28, 57])
Every game with Muller winning conditions is determined, i.e., every vertex is winning for one of the players. It is algorithmically decidable who is a winner from a given vertex in a finite game. In a parity game a player has a memoryless winning strategy from each of his winning vertices.

3 Verification

In this section we will be mostly interested in the model-checking problem. We will start from the concrete example of the μ-calculus model-checking and then consider the extensions and the restrictions of the problem.

Formulas of the μ-calculus over the sets $\mathrm{Prop} = \{p_1, p_2, \dots\}$ of *propositional constants*, $Act = \{a, b, \dots\}$ of *actions*, and $Var = \{X, Y, \dots\}$ of *variables*, are defined by the following grammar:

$$F := \mathrm{Prop} \mid \neg\mathrm{Prop} \mid \mathit{Var} \mid F \vee F \mid F \wedge F \mid \langle Act\rangle F \mid [Act]F \mid \mu\, Var.F \mid \nu\, Var.F$$

Note that we allow negations only before propositional constants. This is not a problem as we will be interested in *sentences*, i.e., formulas where all variables are bound by μ or ν. In the following, $\alpha, \beta, \dots$ will denote formulas.

Formulas are interpreted in *transition systems* of the form $\mathcal{M} = \langle S, \{R_a\}_{a\in Act}, \rho\rangle$, where: S is a nonempty set of *states*, $R_a \subseteq S \times S$ is a binary relation interpreting the action a, and $\rho : \mathrm{Prop} \to \mathcal{P}(S)$ is a function assigning to each propositional constant a set of states where this constant holds.

For a given transition system $\mathcal{M}$ and an *assignment* $V : Var \to \mathcal{P}(S)$, the set of states in which a formula φ is true, denoted $\| \varphi \|_V^{\mathcal{M}}$, is defined inductively as follows:

$$\| p \|_V^{\mathcal{M}} = \rho(p) \qquad \| \neg p \|_V^{\mathcal{M}} = S - \rho(p)$$
$$\| X \|_V^{\mathcal{M}} = V(X)$$
$$\| \langle a\rangle\alpha \|_V^{\mathcal{M}} = \{s : \exists s'.R_a(s,s') \wedge s' \in \| \alpha \|_V^{\mathcal{M}}\}$$
$$\| \mu X.\alpha(X) \|_V^{\mathcal{M}} = \bigcap\{S' \subseteq S : \| \alpha \|_{V[S'/X]}^{\mathcal{M}} \subseteq S'\}$$
$$\| \nu X.\alpha(X) \|_V^{\mathcal{M}} = \bigcup\{S' \subseteq S : S' \subseteq \| \alpha \|_{V[S'/X]}^{\mathcal{M}}\}$$

We have omitted here the obvious clauses for boolean operators and for $[a]\alpha$ formula. We will omit V in the notation if α is a sentence and will sometimes write $\mathcal{M}, s \vDash \alpha$ instead of $s \in \| \alpha \|^{\mathcal{M}}$.

The *model-checking problem for the μ-calculus* is: given a sentence α and a finite transition system $\mathcal{M}$ with a distinguished state s^0 decide if $\mathcal{M}, s^0 \vDash \alpha$.

3.1 From model-checking to games

The solution of the model-checking problem goes via games and is a direct extension of the model-checking game for the propositional logic. Positions in the game are of the form (s, β) where s is a state of a given structure and β is a subformula of a given sentence. In addition to the rules from the propositional game we have rules handling modalities and fixpoints. In a position $(s, \langle a\rangle\beta)$, Eve must find a state t such that $(s,t) \in R_a$ and the new position becomes (t, β). In a position $(s, [a]\beta)$ Adam has to do the same. From a position of the form $(s, \mu X.\beta(X))$ or $(s, \nu X.\beta(X))$ the game proceeds to $(s, \alpha(\mu X.\beta(X)))$ and $(s, \alpha(\nu X.\beta(X)))$ respectively. A play ends in a position of the form (s, p) or $(s, \neg p)$ and the winner depends on whether p is true in s or not.

A major difference between this game and the one for propositional logic is that now the game may not terminate due to the rules handling fixpoints. Hence, it is necessary to decide who is the winner of an infinite play. For this each fixpoint subformula is assigned a rank (natural number) in such a way that μ-subformulas have odd ranks, ν-subformulas have even ranks and if γ is a subformula of β then γ has a bigger rank than β. We refer to [70] for the omitted details. After assignment of ranks the winning condition can be expressed by a parity condition: the smallest rank appearing infinitely often is even.

Thus the model-checking problem can be reduced in linear time to the problem of solving parity games. The reduction works in log-space. As it is well-known that the problem is PTIME-hard, from the algorithmic point of view we do not loose anything in the translation.

It should be also noted that the same kind of reduction can be applied to CTL and (with some more work) to CTL*, ECTL*, ...The schema of translation actually works for most programme logics over transition systems. Such a translation is often fruitful as it exposes the real combinatorial structure hidden in the syntax of the logic. In [43] some of these reductions are done in detail (the authors instead of games prefer to use the terminology of alternating automata over one letter alphabet.)

Let us remark that the above translation does not assume anything about the system. Thus the reduction works for model-checking over any class of labelled graphs closed under product with finite graphs (in order to handle the formula component of the reduction). Because of this we can concentrate on the problem of solving parity games over particular class of structures and get the results on the corresponding model-checking problem automatically.

An example is that of pushdown systems. In this case a graph is not finite but it is finitely defined as the graph of configurations of a push-down automaton. To define a game each state of the automaton is assigned a rank and is designated to belong to Eve or Adam. Eve's positions are configurations with Eve's states and analogously for Adam. The ranks define the parity winning condition. Such games can be solved in EXPTIME [75]. Coupled with the reduction described above this gives EXPTIME algorithm for the model-checking problem. The pushdown model-checking problem is actually EXPTIME-complete. Hence, once again, we do not loose anything by translating into games. Looking at the structure of obtained games one can show that the problem remains EXPTIME-complete if we consider CTL instead of the μ-calculus [74]. Only when we prohibit the use of "on all paths eventually" modality, the resulting fragment of CTL is called EF, we obtain PSPACE-completeness result [74].

Going further one can consider higher-order pushdown systems [29, 40]. These are very similar to pushdown systems but have higher-order stack. For example a stack of level 2 is a stack of stacks of level 1. The additional power is given by a new operation $push_2$ of pushing a copy of the topmost 1-level stack. Of course there are still ordinary push and pop operations that work on the topmost 1-level stack. Just to give an example, with a stack of level 2 a pushdown automaton can recognize the language $\{a^i b^j a^i b^j : i, j \in \mathbb{N}\}$. Somehow surprisingly it turns out that the graphs of such systems are closely related to graphs in Caucal hierarchy [19, 16]. This hierarchy was introduced in the context of decidability of monadic second-order logic and is obtained by repeated use of operations of unwinding and inverse rational mapping. Through results on games on higher-order pushdown systems we get that the model-checking for the mu-calculus over systems with k-level stack is k-EXPTIME complete [13, 15].

It is maybe worthwhile here to make a connection to different area namely to game semantics of programming languages. For example, a game semantics of a program in 3rd level idealized Algol can be described by a deterministic pushdown automaton [59]. This gives a possibility to apply the model-checking techniques in the new area [2]. It would be interesting to compare this with by now standard applications of push-down model checking to programme analysis [30, 63].

Let us finish with some results showing upper bounds on the possible extensions. Push-down systems are prefix word-rewriting systems (via representation of a stack as a word). For a bigger class of prefix tree-rewriting systems only very limited classes of games are decidable [45, 46]. Finally, for the class of well-structured systems we get decidability only for reachability and safety games and with important restrictions on the structure [1].

3.2 From parity games to model checking

The match between parity games and model-checking works both ways. Above we were reducing model-checking to parity games but it is also possible to go the other way around. For every parity condition there is a μ-calculus formula defining Eve's winning positions in every game with this winning condition [28]. Interestingly, these formulas turn out to be very robust. Their natural generalization characterizes the winning positions in much larger class of concurrent probabilistic games [25, 23] (cf. Section 5.2.1).

Let us observe that the discovery of this perfect match, is the result of gradual understanding of the subject. This understanding is far from being complete. As an example let us mention the case of logics for traces. While by now we know some number of them and understand them quite well [26, 77], we have no reduction to an appropriate class of games. A more formal way of saying this is that we still do not know a good notion of alternating automaton on traces. This should be some generalization of asynchronous automaton where dualization gives an automaton accepting the complement of the language. The situation is also not clear in the real-time setting. There, it is still not settled what is the right level of expressive power. Timed automata are too strong from this point of view because they are not closed under complement. Both deterministic timed automata [50] and event-clock automata [3, 36] have some advantages, but there are one too many and moreover there are other appealing proposals and directions to explore [38, 37].

4 Synthesis

The synthesis problem is to construct a system from a given specification. This general statement can be made precise in a variety of ways. The simplest interpretation is that a specification is a formula and a system is a model for it. In the extreme, we can think of a propositional formula as a specification and of a valuation of variables as a model. In this case our satisfiability-checking game from Section 2 may be considered as a solution of the synthesis problem. Observe that it has an appealing property that every satisfying valuation is a result of some winning strategy in this game. Complicating a little, we can take μ-calculus formulas as specifications and transition systems as models. We will need to refine our game by adding rules for dealing with new constructs. These are variations of the ones for the model-checking game. The wining condition becomes slightly complicated because we need to deal with sets of formulas. After setting everything correctly we get the same situation as in the first case: every model of a formula can be obtained from a winning strategy in the satisfiability-checking game for this formula [58].

These reductions are by far not the end of the story. There is another approach to formulating the synthesis problem. A game itself can be considered as a specification of a reactive system. One player, say Eve, takes the role of the controller and the other player that of the environment. The objective is that the system must satisfy the specification no matter what the environment does. If the specification is given as a winning condition in the game then the controller is nothing else but a winning strategy in the game against the environment. This setting is not as far from the previous one as it appears. We will see it while examining even more elaborate formulation of the problem.

Among several ways of specifying the synthesis problem we will use here the formalism from discreet event system control theory of Ramadge and Wonham [62, 41, 17]. This formalism is easy to explain and quite general. For example, a simple extension of it to distributed synthesis setting [66, 7] can easily encode the formalizations in the style proposed by Pnueli and Rosner [60, 42].

In the control theory of discreet event systems a *process* (also called a *plant*) is a deterministic non-complete finite state automaton over an alphabet Σ of events, which defines all possible sequential behaviours of the process. Some of the states of the plant are termed *marked*.

The alphabet Σ is the disjoint union of two subsets: the set Σ_{cont} of controllable events and the set Σ_{unc} of uncontrollable events. Σ is also the disjoint union of the sets Σ_{obs} of observable events and Σ_{uno} of unobservable events.

A controller is a process R which satisfies the following two conditions:

(C) For any state q of R, and for any uncontrollable event a, there is a transition from q labelled by a.

(O) For any state q of R, and for any unobservable event a, if there is a transition from q labelled by a then this transition is a loop over q.

In other words, a controller must react to any uncontrollable event and cannot detect an occurrence of an unobservable event.

If P is a process and R is a controller, the *supervised system* is the product $P \times R$. Thus, if this system is in the state (p, r) and if for some controllable event a, there is no transition labelled by a from r, the controller forbids the supervised system to perform the event a. On the other hand, if an unobservable event occurs in the supervised system, the state of the controller does not change, as if the event had not occurred.

Of course, a supervised system has less behaviours than its plant alone. One can a priori define a set of admissible behaviours of the plant, and the control problem is to find a controller R such that all behaviours of the supervised system are admissible. For instance, one can demand that some dangerous states are never reachable, or that one can always go back to the initial state of the plant. More formally, the basic control problem is the following:

> Given a plant P and a set S of behaviours, does there exist a controller R satisfying (C) and (O) such that the behaviours of the supervised system $P \times R$ are all in S?

and the synthesis problem is to construct such a controller if it does exist. Some variants of this problem take into account the distinction between terminal and non terminal behaviours (in [62] called marked and non marked) of the plant.

In their works Ramadge and Wonham are mainly interested in finding *maximal* controllers, i.e., controllers such that the behaviours of the supervised system are exactly the admissible behaviours of the plant. A less restrictive problem than finding maximal controllers is finding controllers such that the set of supervised behaviours lies between a set of admissible behaviours and a set of required behaviours, for instance, to discard controllers which forbid everything.

Indeed, all these constraints on the behaviour of the supervised system, amount to saying that all paths in the system are in some regular language, and/or that all words of a given language are paths of the system, can be expressed by formulas of the μ-calculus. Therefore, the Ramadge-Wonham's approach can be extended by using any formula of the μ-calculus to specify the desired property of the supervised system [7]. Hence, the control problem becomes

> Given a plant P and a formula α , does there exist a controller R satisfying (C) and (O) such that $P \times R$ satisfies α?

An example of such a formula α characterizing the controller is the following. Let a, c, f be three events where only c is controllable. The event f symbolizes a failure of the device which controls c so that after an occurrence of f, the event c becomes uncontrollable. The formula expressing this phenomenon is $\nu x.(\langle a\rangle x \wedge [c]x \wedge \langle f\rangle \nu y.(\langle a\rangle y \wedge \langle c\rangle y \wedge \langle f\rangle y))$. Another example is the case where only one out of two events c_1 and c_2 is controllable at a time. This is expressed by $\nu x.(\langle a\rangle x \wedge (((\langle c_1\rangle x \wedge [c_2]x) \vee ([c_1]x \wedge \langle c_2\rangle x))$.

It turns out, moreover, that the condition (C) is expressible in the μ-calculus by the formula $\nu x.(\bigwedge_{b\in\Sigma_{unc}}\langle b\rangle x \wedge \bigwedge_{c\in\Sigma_{cont}}[c]x)$. It remains to deal with the condition (O) which, unfortunately, is not expressible in the μ-calculus because it is not invariant under bisimulation. That is why we extend the μ-calculus into a *loop μ-calculus*. This extension consists in associating with each event a a basic proposition $\circlearrowleft_a$ whose interpretation is that there is an a-transition from a state that ends in the same state, i.e., it is a self-loop. This way the condition (O) can be expressed as $\nu x.(\bigwedge_{a\in\Sigma_{obs}}[a]x \wedge \bigwedge_{a\in\Sigma_{uno}}([a]\mathit{false} \vee \circlearrowleft_a)$. Besides, we can express that an observable event becomes unobservable after a failure: $\nu x.(\cdots \wedge [a]x \wedge \langle f\rangle \nu y.(\cdots \wedge ([a]\mathit{false} \vee \circlearrowleft_a) \wedge \langle f\rangle y))$, or that at most one out of two events a and b is observable: $[a]\mathit{false} \vee \circlearrowleft_a \vee [b]\mathit{false} \vee \circlearrowleft_b$.

Summarizing, the general form of a control problem is:

(*) Given a plant P and two formulas α and β, does there exist a controller R satisfying β such that $P \times R$ satisfies α?

where α and β are loop μ-calculus formulas. Paper [7] describes the construction of a loop μ-calculus formula α/P that is satisfied by precisely those controllers R for which $P \times R \vDash \alpha$. This way a process R is a solution of the synthesis problem (*) if and only if $R \vDash (\alpha/P) \wedge \beta$.

Therefore, all control problems of the form (*) are indeed satisfiability problems in the loop μ-calculus and the synthesis problems amount to finding models of loop formulas. Fortunately, the loop μ-calculus has properties very similar to the ordinary one and finding such models can be reduced to finding winning strategies in parity games [7]. Reciprocally, finding a winning strategy is itself a control problem: your moves are controllable and the moves of your opponent are not.

We have not discussed till now the complexity issues. The satisfiability checking for the loop μ-calculus is EXPTIME-complete, hence not worse than for the standard one. Nevertheless for the issues of feasibility one would probably prefer to work with automata rather than μ-calculus. A solution of a synthesis problem involves several constructions on formulas (or equivalently on automata). It is not a surprise that in this case it would be very helpful to have good procedures for decreasing the size of formulas or automata. There are also numerous issues connected with reductions between different acceptance conditions [73].

In the next section we will consider the task of synthesizing several controllers. Here we finish with a short remark on synthesis for real time systems. There are several new issues that arise here as one wants to avoid pathological situations like controller preventing time to diverge, see [24] for a discussion. If the conditions do not talk about time then the region construction allows to get the solution in a pretty much the same way as for the untimed case [8, 24]. If specifications are given by a timed automaton then there are numerous variants of the problem depending on the variant of automaton used and whether the automaton specifies desired or undesired behaviours. Very roughly, the problem is decidable if one limits the granularity of the controller (which limits the constants that can occur in its guards). It is also decidable for deterministic specifications but only when all the events are observable [27, 18, 11].

5 Going further

All the results we have described so far use reductions to parity games over finite or finitely presented graphs. In this last section we will give some examples where it is necessary to explore new kinds of games. One direction calls for more complex winning conditions. The other concerns extensions of the way the game is played.

5.1 Generalizing the class of winning conditions.

Let us come back to the model of push-down games (cf. Section 3.1). As noted in [71, 14] there exists "natural winning conditions exploiting the infinity of pushdown transition graphs". The condition proposed in [14] is *strict unboundedness*: no configuration appears infinitely often on the play.

In [10] we propose a new winning condition for push-down games that we call unboundedness: there is no bound on the size of the stack during the play. We consider Boolean combinations of this condition and the parity condition, for example, such a condition can say that a stack is unbounded and some state appears infinitely often. We characterize conditions for which there is a strategy with finite memory for both players. We show that the problem of deciding a winner in pushdown games with Boolean combinations of Büchi and unboundedness conditions is EXPTIME-complete (in the size of the automaton defining the game graph). While the method from [10] becomes prohibitively complicated when dealing with boolean combinations of parity conditions and unboundedness, a different approach [33] gives an EXPTIME algorithm in this more general case.

These results lead to a natural question: what is a good class of winning conditions for push-down games? All the conditions given by finite automata can be reduced to games with parity conditions alone. If conditions are given by push-down automata then the problem of determining the winner becomes undecidable. It is so even if an automaton defining the winning condition is deterministic. Recently a new class of pushdown automata, called visibly pushdown automata, has been defined [4]. It has numerous good closure properties and is capable of encoding strict unboundedness condition. It would be only natural to investigate pushdown games with winning conditions defined by visibly pushdown automata. Let us also remark that the class of interesting conditions for push-down games is apparently quite large. In [69] a family of decidable winning conditions of arbitrary high finite Borel complexity has been constructed.

The above results suggest that it may be worthwhile to investigate for what kind of winning conditions there exist finite memory strategies in all possible graphs. In [35] we have considered Muller conditions over infinite number of colours. A particular case of such a condition is when colours are natural numbers and the winner is decided by looking at the parity of the smallest number appearing infinitely often (additionally we can assume that Eve wins if there is no such number). For example, strict unboundedness condition is a condition of this form. It turns out that this infinite kind of a parity condition is the only type of condition that guarantees the existence of memoryless strategies in all games. It is important to add that for this result to hold all positions of the game need to have a colour assigned. If we permit partial assignments of colours or put coloring on edges of the game graph and not on positions then only ordinary (i.e. finite) parity conditions admit memoryless strategies [22].

5.2 Generalizing the game model

5.2.1 Concurrent probabilistic games

Apart from generalizing the class of winning conditions, one can generalize the rules according to which games are played. Instead of playing in turns, the players may be required to make the moves simultaneously (as in "paper, scissors, stone" game). The other extension is that transitions may be probabilistic, i.e., giving a probability distribution on the successor states.

Here is a definition encompassing the two extensions at the same time [25]. A game is a tuple $G = \langle V, M, \Gamma_E, \Gamma_A, \theta, Acc\rangle$ where V is a set of positions, M the set of moves, $\Gamma_E, \Gamma_A : V \to (\mathcal{P}(M) - \{\emptyset\})$ are choice of move functions for Eve and Adam respectively, and $\theta : V \times M \times M \to \mathcal{D}(V)$ is a function giving for each position and a pair of moves a probability distribution of reaching a next position. Finally, Acc is an accepting condition which is a parity condition in all the cases considered in the literature. At every position v, Eve and Adam choose moves $m_e \in \Gamma_E(v)$ and $m_a \in \Gamma_A(v)$ respectively and the game proceeds to a position v' with the probability $\theta(v, m_e, m_a)(v')$.

A position v is *deterministic* if for every $m_e, m_a \in M$ there is $v' \in V$ with $\theta(v, m_e, m_a)(v') = 1$. A position v is *Adam independent* if $\Gamma_A(v)$ is a singleton. Hence, the games we have considered before are the special case when all the positions are deterministic and either Adam or Eve independent. *Markov decision processes* are the special case where all positions are Adam independent. *Perfect-information stochastic games* are obtained by requiring that each position should be Adam or Eve independent and moreover if a position is not deterministic then it should be both Adam and Eve independent. In other words, these are ordinary turn based games with additional randomized positions where a successor is chosen according to the distribution assigned there.

Changing the notion of a game suggests also changing the notion of a strategy. An example of "paper, scissors, stone" game shows that there are concurrent games where none of the players has a winning strategy understood as a function form the history of a play to positions; such strategies are called *pure* in this context. More interesting are *randomized strategies* that suggest a probability distribution instead of a single next move. Having these, we need also to talk about the probability of winning when Eva and Adam choose their randomized strategies. (see [25] for the definition). A *value for Eva* of the game is the supremum over all Eva's strategies of infimum over all Adam's strategies of the probability of winning when playing the two chosen strategies. The value for Adam is defined dually. The quantitative determinacy result of Martin [52] states the values for Eva and Adam sum up to 1. In [23] de Alfaro and Majumdar show how to calculate the values of a game using appropriate extension of the μ-calculus.

This general model of games may be too powerful if we are concerned with the quality of the strategies. First, in some cases there may exist only ε-optimal strategies, i.e., guaranteeing the probability of winning ε-smaller than optimal. Moreover even those may need to be randomized and use infinite memory [23]. It turns out that in perfect-information parity stochastic games the situation is much better and both players have optimal pure and memoryless strategies [20, 78]. The values of such games are always rational numbers and there is an exponential-time algorithm for computing them.

5.2.2 Distributed games

In Section 4 we have discussed the problem of synthesizing a single controller. In a distributed system one can have multiple processes. The system specifies possible interactions between the processes and the environment and also the interactions among the processes themselves. The synthesis problem here is to find a program for each of the processes such that the overall behaviour of the system satisfies a given specification. We call this *distributed synthesis problem (DSP)*.

The distributed synthesis problem has been considered by Pnueli and Rosner in the setting of an architecture with fixed channels of communication among processes [60]. They have shown that distributed synthesis is undecidable for most classes of architectures. They have obtained decidability result for a special class of hierarchical architectures called *pipelines* which was recently extended to branching time specifications over two-way pipelines and one-way rings [42]. These are essentially the only architectures for which the problem is decidable [47, 48]. In [49] it is suggested to overcome this undecidability result by restricting the class of allowed controllers.

The other approach to distributed synthesis, initiated roughly at the same time as the work of Pnueli and Rosner, comes from control theory of discrete event systems [62, 44, 66]. A system is given as a plant (deterministic finite state automaton) and the distributed synthesis problem is to synthesize a number of controllers, each being able to observe and control only a specific subset of actions of the plant. While the original problem refers only to safety properties, an extension to the μ-calculus specifications has also been considered [7, 65, 64]. Except for some special cases, the problem turns out to be undecidable ([7, 72]). It is one of the important goals of the area of decentralized control synthesis to identify conditions on a plant and a specification such that DSP is decidable.

As we have seen in Section 4, game theory provides an approach to solving the (centralized) synthesis problem. An interaction of a process with its environment can be viewed as a game between two players and the synthesis problem reduces to finding a finite-state winning strategy for Eve. The winning strategy can then be implemented as the required program. This approach does not extend directly to DSP because there we have more than two parties.

In [54] we suggest an approach to DSP by directly encoding the problem game-theoretically. We extend the notion of games to n players playing a game against a single hostile environment. (Here, it will be convenient to change the nomenclature and talk about games between players and environment rather than between Eves and Adam.) We call this model *distributed games*. In this model, there are no explicit means of interaction among players. Any such interaction must take place through the environment. Moreover, each player has only a local view of the global state of the system. Hence, a *local strategy* for a player is a function of its local history (of player's own states and the partial view of the environment's states). A *distributed strategy* is a collection of local strategies; one for each of the players. The environment in distributed games, on the other hand, has access to the global history. Any play in a distributed game consists of alternating sequence of moves of (some of) the players and of the environment.

Distributed synthesis in this model amounts to finding a distributed winning strategy. This means finding a collection of local strategies that can win against the global environment. A side effect of the requirement that the players need to win together is that they need to implicitly communicate when they make their moves. The card game of bridge is a good example of the kind of implicit communication we have in mind. When $n = 1$, distributed games reduce to the usual two-party games.

Let us define the model more formally. A *local game* is thus any game $G = \langle P, E, T\rangle$ as defined in Section 2, where P are positions for the player (Eve) and E are the positions for the environment (Adam). There is no winning condition in this game and we require that it is bipartite, i.e., a successor of a player position is always an environment position and vice versa.

Let $G_i = \langle P_i, E_i, T_i\rangle$, for $i = 1, \ldots, n$, be local games. A *distributed game* constructed from $G_1, \ldots, G_n$ is $\mathcal{G} = \langle P, E, T, Acc \subseteq (E \cup P)^\omega\rangle$ where:

1. $E = E_1 \times \cdots \times E_n$.
2. $P = (P_1 \cup E_1) \times \cdots \times (P_n \cup E_n) \setminus E$.
3. From a player's position, we have $(x_1, \ldots, x_n) \to (x'_1, \ldots, x'_n) \in T$ if and only if $x_i \to x'_i \in T_i$ for all $x_i \in P_i$ and $x_i = x'_i$ for all $x_i \in E_i$.
4. From environment's position, if we have $(x_1, \ldots, x_n) \to (x'_1, \ldots, x'_n) \in T$ then for every x_i, either $x_i = x'_i$ or $x'_i \in P_i$ and moreover $(x_1, \ldots, x_n) \neq (x'_1, \ldots, x'_n)$
5. Acc is any winning condition.

Notice that there is an asymmetry in the definition of environment's and players' moves. In a move from players' to environment's position, all components which are players' positions must change, and the change respects transitions in local games. In the move from environment's to players' position, all components are environment's positions but only some of them need to change; moreover these changes need not to respect local transitions. Hence, while global moves of the player are a kind of free product of moves in local games, it is not the case for the environment. The moves from environment positions are the only part of a distributed game that is not determined by the choice of components, i.e., of local games. This freedom makes it possible to encode different communication patterns and other phenomena.

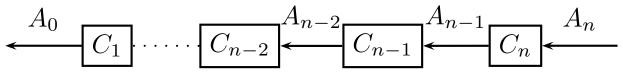
We interpret a distributed game as a game of n players against environment. This intuition will become clear when we will define the notions of views and local strategies.

For an n-tuple η and $i = 1, \ldots, n$, let $\eta[i]$ denote the i-th component of η. Similarly, for a sequence $\vec{v} = \eta_1\eta_2\ldots$ of n-tuples, let $\vec{v}[i] = \eta_1[i]\eta_2[i]\ldots$ denote the projection of the sequence on the i-th component.

From the definition of the moves it is easy to observe that given a play $\vec{v}$ in a distributed game $\mathcal{G}$, the projection of $\vec{v}$ to the positions of the i-local game, $\vec{v}[i]$, is of the form $e_0^+ p_0 e_1^+ p_1 \ldots$ Note that the player's positions do not repeat since as soon as the local game moves to a player position, he reacts immediately with an environment position. The *view of process* i of $\vec{v}$ is $view_i(\vec{v}) = e_0 p_0 e_1 p_1 \ldots$

An *i-local strategy* is a strategy in the game G_i. A *distributed (player's) strategy* σ is a tuple of local strategies $\langle\sigma_1, \ldots, \sigma_n\rangle$. A distributed strategy σ defines a strategy in $\mathcal{G}$ by $\sigma(\vec{v} \cdot (x_1, \ldots, x_n)) = (e_1, \ldots, e_n)$ where $e_i = x_i$ if $x_i \in E_i$ and $e_i = \sigma_i(view_i(\vec{v} \cdot x_i))$ otherwise. We call σ the *global strategy* associated with the given distributed players' strategy.

Let us se how to code a pipeline synthesis problem into distributed games. A pipeline is a sequence of processes communicating via unidirectional channels:



We assume that the alphabets $A_0, \ldots, A_n$ are disjoint. The execution follows in rounds. Within a round, processes get inputs and produce outputs in a step-wise fashion. At the beginning of a round, process C_n gets input $a_n \in A_n$ from the environment and gives an output $a_{n-1} \in A_{n-1}$. In the next step, this output is given as input to process C_{n-1} and so on. When C_1 has given an output, the round finishes and another round starts.

A *local controller* for the i-th component is a function $f_i : (A_i)^* \to A_{i-1}$. A sequence $a_0 b_0 a_1 b_1 \cdots \in (A_i \cdot A_{i-1})^\omega$ *respects* f_i if $f_i(a_0 a_1 \ldots a_j) = b_j$ for all j.

A *pipeline controller* F is a tuple of local controllers $\langle f_1, \ldots, f_n \rangle$, one for each component. An execution of the pipeline is a string in $(A_n A_{n-1} \cdots A_0)^\omega$. An execution $\vec{v}$ respects F if $\vec{v}|(A_i \cup A_{i-1})$ respects f_i, for all $i = 1, \ldots, n$.

Let $\Sigma = \bigcup_{i=0\ldots,n} A_i$. A controller F defines a set of Σ-labeled paths $\mathcal{L}(F)$ which is the set of all the executions respecting F.

The *pipeline synthesis problem* is: given a pipeline over alphabets $A_0, \ldots, A_n$ and a deterministic parity word automaton $\mathcal{A}$ over the alphabet $\Sigma = \bigcup_{i=0\ldots,n} A_i$, find a pipeline controller $F = \langle f_1, \ldots, f_n \rangle$ such that $\mathcal{L}(F) \subseteq L(\mathcal{A})$.

A pipeline synthesis problem is coded as a distributed game $\mathcal{G} = \langle P, E, T, Acc \rangle$ constructed from local games $G_0, \ldots, G_n$, with G_0 taking the role of the automaton $\mathcal{A} = \langle Q, \Sigma, q^0, \delta : Q \times \Sigma \to Q, \Omega : Q \to \mathbb{N} \rangle$ and G_i the role of the i-th component C_i. The game G_0 is defined by :

- $P_0 = Q \times \Sigma^{n+1}$; $E_0 = Q$;
- $(q, w) \to q' \in T_0$ if $q' = \delta(q, w)$; and $q \to (q, w) \in T_0$ for all $w \in \Sigma^{n+1}$.

For each component $i = 1, \ldots, n$ we have the game G_i which is defined by: $P_i = A_i$; $E_i = (A_i \to A_{i-1})$; and there is a complete set of transitions between P_i and E_i. The intuition is that the game G_0 simulates the specification automaton $\mathcal{A}$ and each game G_i models a process which reads an input and responds with a declaration for the next round: a function of type $A_i \to A_{i-1}$ declares what will be the output depending on the input.

From an environment position $(q, f_1, \ldots, f_n)$, for a letter $a_n \in A_n$ we have a move to $((q, w(a_n)), a_1, \ldots, a_n)$ where $w(a_n) = a_n a_{n-1} \ldots a_0$ is a word such that $a_{i-1} = f_i(a_i)$. With this definition of environment moves we code the pipeline communication model. Indeed the letters emitted on the channels are determined by iteratively using the declaration functions f_i.

The winning condition Acc is the set of sequences such that the projection on the states in the first component satisfies the the parity condition of the automaton $\mathcal{A}$. Here we need to assume some special form of $\mathcal{A}$. This is because we “jump” over the states by using $\delta(q, w)$ for w a word of $n + 1$ letters. We need to be sure that while doing this we do not jump over states of small priority. As the length of w is fixed we can easily guarantee this. The initial position is $\eta_0 = (q^0, a_1, \ldots, a_n)$ for some arbitrarily chosen letters $a_1, \ldots, a_n$.

It is not difficult to show that there is a distributed winning strategy in $\mathcal{G}$ from η_0 iff the pipeline synthesis problem has a solution. A distributed winning strategy gives a controller for the pipeline.

The main technical results about these games [54] are two theorems allowing their simplification. In general it is not decidable to check whether there is a distributed winning strategy in a finite distributed game. The simplification theorems allow to reduce the number of players and to reduce nondeterminism in the game. For example, these theorems would allow to simplify the above game into a game between one player and the environment (where the existence of a winning strategy is decidable). In some other examples, after simplification we get a game where environment has no choice of moves; and this case is also decidable. This technique is enough to solve decidable cases of distributed control problems considered in the literature: pipelines [60, 42], local specifications and double flanked pipelines [48, 47], communicating state machines [49], decentralized control synthesis [66].

To make a link with the previous subsection, observe that distributed games are quite different from concurrent games. In both cases the players need to make moves without having full information, but in distributed games these payers are on the same side hence they want to cooperate. On the other hand it seems reasonable to introduce and study randomized strategies in the setting of distributed games. It is well-known from the theory of distributed computing that some distributed communication problems can have only randomized solutions [61, 32, 21].

References

[1] P. A. Abdulla, A. Bouajjani, and J. d’Orso. Deciding monotonic games. In *CSL’03*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–14, 2003.

[2] S. Abramsky, D. R. Ghica, A. S. Murawski, and C.-H. L. Ong. Applying game semantics to compositional software modeling and verification. In *TACAS’04*, volume 2988 of *Lecture Notes in Computer Science*, pages 421–435, 2004.

[3] R. Alur, L. Fix, and T. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 204, 1997.

[4] R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC'04*, 2004. To appear.

[5] A. Arnold. The mu-calculus alternation-depth hierarchy is strict on binary trees. *RAIRO–Theoretical Informatics and Applications*, 33:329–339, 1999.

[6] A. Arnold and L. Santocanale. Ambiguous classes in the games mu-calculus hierarchy. In *FOSSACS 03*, volume 2620 of *Lecture Notes in Computer Science*, pages 70–86, 2003.

[7] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

[8] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Contrloller synthesis for timed automata. In *Proc. IFAC Symp. System Structure and Control*, pages 469–474, 1998.

[9] A. Blass. A game semantics for linear logic. *Ann. Pure and Appl. Logic*, 56(1-3):183–220, 1992.

[10] A. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with the unboundedness and regular conditions. In *FSTTCS'03*, volume 2914 of *Lecture Notes in Computer Science*, pages 88–99, 2003.

[11] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In *CAV'03*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192, 2003.

[12] J. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195:133–153, 1997.

[13] T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *ICALP'03*, volume 2719 of *Lecture Notes in Computer Science*, pages 556–569, 2003.

[14] T. Cachat, J. Duparc, and W. Thomas. Solving pushdown games with a Sigma-3 winning condition. In *CSL'02*, volume 2471 of *Lecture Notes in Computer Science*, pages 322–336, 2002.

[15] T. Cachat and I. Walukiewicz. Complexity of games on higher-order pushdown automata. Manuscript, 2004.

[16] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS'03*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–124, 2003.

[17] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[18] F. Cassez, T. Henzinger, and J. Raskin. A comparison of control problems for timed and hybrid systems. In *Hybrid Systems Computation and Control (HSCC'02)*, number 2289 in Lecture Notes in Computer Science, pages 134–148, 2002.

[19] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176, 2002.

[20] K. Chatterjee, M. Jurdziński, and T. A. Henzinger. Quantitative stochastic parity games. In *SODA'04*, 2004. To appear.

[21] B. Chor and C. Dwork. Randomization in Byzantine agreement. In S. Micali, editor, *Randomness and Computation*, pages 433–498. JAI Press, Greenwich, 1989.

[22] T. Colcombet and D. Niwiński. On the positional determinacy of edge–labeled games. Submitted, 2004.

[23] L. de Alfaro. Quantitative verification and control via the mu-calculus. In *CONCUR'03*, volume 2761 of *Lecture Notes in Computer Science*, pages 102–126, 2003.

[24] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *CONCUR'03*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156, 2003.

[25] L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. In *STOC'01: 33rd Annual ACM Symposium on Theory of Computing*, pages 675–683, 2001.

[26] V. Diekert and P. Gastin. Pure future local temporal logics are expressively complete for mazurkiewicz traces. In *LATIN'04*, Lecture Notes in Computer Science, 2004. to appear.

[27] D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS'02*, volume 2285 of *Lecture Notes in Computer Science*, pages 571–582, 2002.

[28] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS'91*, pages 368–377, 1991.

[29] J. Engelfriet. Iterated push-down automata and complexity classes. In *15th STOC'83*, pages 365–373, 1983.

[30] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *FOSSACS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 14–30, 1999.

[31] W. Felscher. Dialogues as a foundation for intuitionistic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume III, pages 341–372. Reidel Publishing Company, 1986.

[32] M. Fisher. The consensus problem in unreliable distributed systems (a brief survey). In *Proc. International Conference on Foundations of Computation*, 1983.

[33] H. Gimbert. Parity and explosion games on context-free graphs. Manuscript, 2003.

[34] E. Grädel. Positional determinacy of infinite games. In *STACS'04*, volume 2996 of *Lecture Notes in Computer Science*, pages 4–18, 2004.

[35] E. Grädel and I. Walukiewicz. Positional determinacy of infnite games, 2004. Submitted.

[36] T. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In *ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591, 1998.

[37] Y. Hirshfeld and A. Rabinovich. A framework for decidable metrical logics. In *ICALP'99*, volume 1664 of *Lecture Notes in Computer Science*, pages 422–432, 1999.

[38] Y. Hirshfeld and A. Rabinovich. Quantitative temporal logic. In *CSL'99*, volume 1683 of *Lecture Notes in Computer Science*, pages 172–187, 1999.

[39] A. S. Kechris. *Classical Descriptive Set Theory*, volume 156 of *Graduate Texts in Mathematics*. Springer-Verlag, 1995.

[40] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS' 02*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222, 2002.

[41] R. Kumar and V. K. Garg. *Modeling and control of logical discrete event systems*. Kluwer Academic Pub., 1995.

[42] O. Kupferman and M. Vardi. Synthesizing distributed systems. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, 2001.

[43] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[44] F. Lin and M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Transactions on automatic control*, 33(12):1330–1337, 1990.

[45] C. Löding. Model-checking infinite state systems generated by ground tree rewriting. In *FOSSACS'02*, volume 2303 of *Lecture Notes in Computer Science*, pages 280–294, 2002.

[46] C. Löding. *Infinite Graphs Generated by Tree Rewriting*. PhD thesis, RWTH Aachen, 2003.

[47] P. Madhusudan. *Control and Synthesis of Open Reactive Systems*. PhD thesis, University of Madras, 2001.

[48] P. Madhusudan and P. Thiagarajan. Distributed control and synthesis for local specifications. In *ICALP'01*, volume 2076 of *Lecture Notes in Computer Science*, 2001.

[49] P. Madhusudan and P. Thiagarajan. A decidable class of asynchronous distributed controllers. In *CONCUR'02*, volume 2421 of *Lecture Notes in Computer Science*, 2002.

[50] O. Maler and A. Pnueli. On recognizable timed languages. In *FOSSACS'04*, volume 2987 of *Lecture Notes in Computer Science*, pages 348–362, 2004.

[51] D. Martin. Borel determinacy. *Ann. Math.*, 102:363–371, 1975.

[52] D. Martin. The determinacy of Blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

[53] R. McNaughton. Infinite games played on finite graphs. *Ann. Pure and Applied Logic*, 65:149–184, 1993.

[54] S. Mohalik and I. Walukiewicz. Distributed games. In *FSTTCS'03*, volume 2914 of *Lecture Notes in Computer Science*, pages 338–351, 2003.

[55] Y. Moschovakis. *Descriptive Set Theory*, volume 100 of *Studies in Logic*. North-Holand, 1980.

[56] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Fifth Symposium on Computation Theory*, volume 208 of *LNCS*, pages 157–168, 1984.

[57] A. W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdansk, 1991.

[58] D. Niwiński and I. Walukiewicz. Games for μ-calculus. *Theoretical Computer Science*, 163:99–116, 1996.

[59] C.-H. L. Ong. bservational equivalence of third-order idealized Algol is decidable. In *LICS'02*, pages 245–256, 2002.

[60] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *31th IEEE Symposium Foundations of Computer Science (FOCS 1990)*, pages 746–757, 1990.

[61] M. Rabin. Randomized Byzantine generals. In *FOCS'83*, pages 403–409, 1983.

[62] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(2):81–98, 1989.

[63] T. Reps, S. Schwoon, and S. Jha. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *10th Int. Static Analysis Symp.*, volume 2694 of *Lecture Notes in Computer Science*, pages 189–213, 2003.

[64] S. Riedweg. *Logiques pour le contrôle d'automatismes discrets*. PhD thesis, Université de Rennes 1, December 2003.

[65] S. Riedweg and S. Pinchinat. Quantified mu-calculus for control synthesis. In *MFCS'03*, volume 2747 of *Lecture Notes in Computer Science*, pages 642–651, 2003.

[66] K. Rudie and W. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. on Automat. Control*, 37(11):1692–1708, 1992.

[67] L. Santocanale. A calculus of circular proofs and its categorical semantics. In *FOSSACS 02*, volume 2303 of *Lecture Notes in Computer Science*, 2002.

[68] L. Santocanale. Free μ-lattices. *Journal of Pure and Applied Algebra*, 168(2-3):227–264, 2002.

[69] O. Serre. Games with winning conditions of high Borel complexity. In *ICALP'04*, Lecture Notes in Computer Science, 2004. to appear.

[70] C. Stirling. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, 2001.

[71] W. Thomas. On the synthesis of strategies in infinite games. In *STACS'95*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13, 1995.

[72] S. Tripakis. Undecidable problems in decentralized observation and control for regular languages. *Information Processing Letters*, 90(1):21–28, 2004.

[73] N. Wallmeier, P. Htten, and W. Thomas. Symbolic synthesis of finite-state controllers for request-response specifications. In *Conference on the Implementation and Application of Automata, CIAA'00*, volume 2759 of *Lecture Notes in Computer Science*, pages 11–22, 2003.

[74] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *FSTTCS'00*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138, 2000.

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

[76] I. Walukiewicz. Deciding low levels of tree-automata hierarchy. In *WoLLIC'02*, volume 67 of *ENTCS*, 2002.

[77] I. Walukiewicz. Local logics for traces. *Journal of Automata, Languages and Combinatorics*, 7(2):259–290, 2002.

[78] W. Zielonka. Perfect-information stochastic parity games. In *FOSSACS'04*, volume 2987 of *Lecture Notes in Computer Science*, pages 499 – 513, 2004.