Concurrent Reachability Games*

Luca de Alfaro Thomas A. Henzinger Orna Kupferman

Department of EECS, University of California at Berkeley,
Berkeley, CA 94720-1770, USA
Email:{dealfaro,tah,orna}@eecs.berkeley.edu

Abstract

An open system can be modeled as a two-player game between the system and its environment. At each round of the game, player 1 (the system) and player 2 (the environment) independently and simultaneously choose moves, and the two choices determine the next state of the game. Properties of open systems can be modeled as objectives of these two-player games. For the basic objective of reachability —can player 1 force the game to a given set of target states?— there are three types of winning states, according to the degree of certainty with which player 1 can reach the target. From type-1 states, player 1 has a deterministic strategy to always reach the target. From type-2 states, player 1 has a randomized strategy to reach the target with probability 1. From type-3 states, player 1 has for every real $\varepsilon > 0$ a randomized strategy to reach the target with probability greater than $1 - \varepsilon$.

We show that for finite state spaces, all three sets of winning states can be computed in polynomial time: type-1 states in linear time, and type-2 and type-3 states in quadratic time. The algorithms to compute the three sets of winning states also enable the construction of the winning and spoiling strategies. Finally, we apply our results by introducing a temporal logic in which all three kinds of winning conditions can be specified, and which can be model checked in polynomial time. This logic, called Randomized ATL, is suitable for reasoning about randomized behavior in open (two-agent) as well as multi-agent systems.

*This work was partially supported by the SRC contract 97-DC-324.041, by ARO under the MURI grant DAAH04-96-1-0341, by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the DARPA/NASA grant NAG-2-1214, and by the NSF grant CCR-9504469.

1. Introduction

One of the central problems in system verification is the *reachability* question: given an initial state s and a target state t, can the system get from s to t? The dynamics of a closed system, which does not interact with its environment, can be modeled by a state-transition graph, and the reachability question reduces to graph reachability, which can be solved in linear time and is complete for NLOGSPACE [Jon75]. By contrast the dynamics of an open system, which does interact with its environment, is best modeled as a game between the system and the environment.

In some situations, it may suffice to have the system and the environment take turns to make moves, yielding a *turn-based* model. In this case, the game graph is an AND-OR graph. A (deterministic) strategy for the AND player maps every path that ends in an AND state to a successor state, and similarly for the OR player. Thus the reachability question (can the system get from s to t no matter what the environment does?) reduces to AND-OR graph reachability (does the OR player have a strategy so that for all strategies of the AND player, the game, if started in s, reaches t?). This problem can again be solved in linear time and is complete for PTIME [Imm81]. With respect to AND-OR graph reachability, randomized strategies are not more powerful than deterministic strategies. A *randomized* strategy for the AND player maps every path that ends in an AND state to a probability distribution on the successor states, and similarly for the OR player. In turn-based models, it can be seen that the AND-OR graph reachability question has the same answer as the probabilistic question "does the OR player have a randomized strategy so that for all randomized strategies of the AND player, the game, if started in s, reaches t with probability 1?".

The turn-based model is naive, because in realistic concurrency models, in each state, the system and the environment independently choose moves, and the parallel execu-

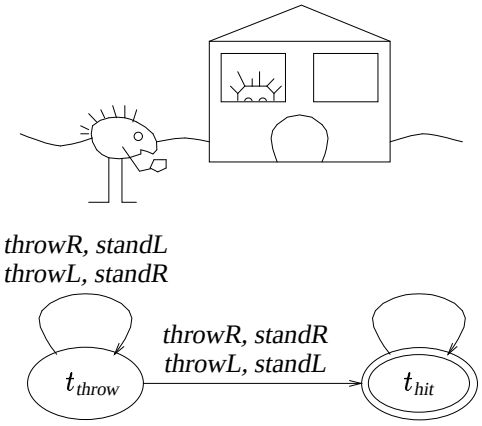

Figure 1. Game LEFT-OR-RIGHT**.**

tion of the moves determines the next state. Such a *simultaneous* game is a natural model for synchronous systems where the moves are chosen truly simultaneously, as well as for distributed systems in which the moves are not revealed until their combined effect is apparent. In particular, the modeling of synchronization between processes often requires the consideration of simultaneous games.

The simultaneous case is more general than the turn-based one, and deterministic strategies no longer tell the whole story about the reachability question. The fact that randomized strategies can be more powerful than deterministic ones is illustrated by the game LEFT-OR-RIGHT, depicted in Figure 1. Initially, the game is at state t_{throw}. At each round, player 1 can choose to throw a snowball either at the left window (move *throwL*) or at the right window (move *throwR*). Independently and simultaneously, player 2 must choose to stand behind either the left window (move *standL*) or the right window (move *standR*). If the snowball hits player 2, the game proceeds to the target state t_{hit}; otherwise, another round of the game is played from t_{throw}.

For each move of player 1, player 2 has a countermeasure. If we consider only deterministic strategies, then for every strategy of player 1, there is (exactly one) strategy of player 2 such that t_{hit} is never reached. Hence, if we base our definitions on deterministic strategies, we obtain the answer NO to the reachability question. The situation of player 2, however, is not nearly as safe as this negative answer implies. If player 1 chooses at each round the window at which to throw the snowball by tossing a coin, then player 2 will be hit with probability 1, regardless of her strategy.

The coin-tossing criterion used by player 1 to select the move is an example of a randomized strategy, and the game illustrates the value of randomized strategies for winning reachability games. If player 1 adopts a deterministic strategy, the moves he plays during the game are completely determined by the history of the game, which is visible also to player 2. Once player 1 has chosen a deterministic strategy, player 2 can choose her strategy to counteract every move of player 1, as if she were able to see it before choosing her own move. Randomized strategies postpone the choice of the move until the game is being played, precluding this type of spying behavior. Another way of thinking about randomized strategies is through the concept of *initial randomization.* The choice of a randomized strategy is equivalent to the choice of a probability distribution over the set of deterministic strategies [Der70]. By choosing such a distribution, rather than a single strategy, player 1 prevents player 2 from tailoring her strategy to counteract the strategy chosen by player 1. The greater power of randomized strategies is a well-known fact in game theory, and it has its roots in von Neumann's minimax theorem [vN28].

Once we consider randomized strategies, we can answer the reachability question with three kinds of affirmative answers. The first kind of answer is the answer *sure:*

> Player 1 has a strategy so that for all strategies of player 2, the game, if started in s, always reaches t.

To establish this type of answer, it suffices to consider deterministic strategies only. The second, weaker kind of answer is the answer *almost sure:*

> Player 1 has a strategy so that for all strategies of player 2, the game, if started in s, reaches t with probability 1.

To establish this type of answer, it is necessary to consider randomized strategies, as previously discussed. The third, yet weaker kind of answer is the answer *limit sure:*

> For every real $\varepsilon > 0$, player 1 has a strategy so that for all strategies of player 2, the game, if started in s, reaches t with probability greater than $1 - \varepsilon$.

The three kinds of answers form a proper hierarchy, in the sense that there are cases in which almost-sure reachability holds whereas sure reachability does not, and cases in which limit-sure reachability holds whereas almost-sure reachability does not. Note that the second gap does not appear in reachability problems over Markov chains, or Markov decision processes [KSK66, BT91]. While the game LEFT-OR-RIGHT witnesses the first gap, the second gap is witnessed by the game HIDE-OR-RUN, adapted from [KS81] and depicted in Figure 2. The target state is s_{home}, and the interesting part of the game happens at state s_{hide}. At this state, player 1 is hiding behind a small hill, while

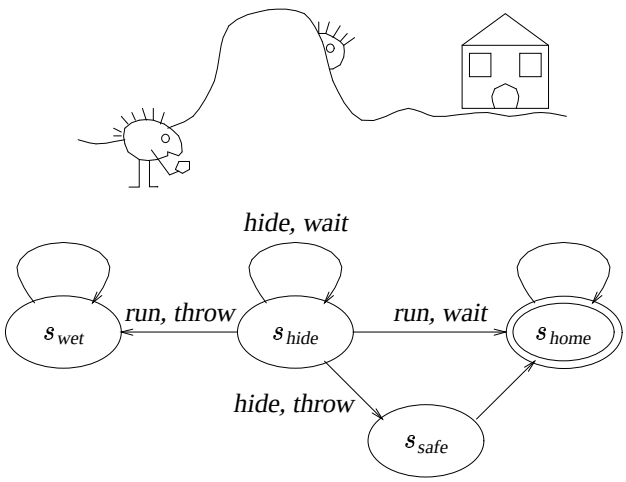

Figure 2. Game HIDE-OR-RUN**.**

player 2 is trying to hit him with a snowball. Player 1 can choose between hiding or running, and player 2 can choose between waiting and throwing her only snowball. If player 1 runs and player 2 throws the snowball, then player 2 is hit, and the game proceeds to state s_{wet}. If player 1 runs and player 2 waits, then player 1 gets home, and the game proceeds to state s_{home}. If player 1 hides and player 2 throws the snowball, then player 1 is no more in danger, and the game proceeds to state s_{safe}. Finally, if player 1 hides and player 2 waits, the game stays at s_{hide}.

In this game, from state s_{hide} player 1 does not have a strategy (randomized or deterministic) that ensures reaching s_{home} with probability 1: in order to reach home regardless of the strategy of player 2, player 1 may have to take a chance and run while player 2 is still in possession of the snowball. On the other hand, if player 1 runs with very small probability at each round, it becomes very difficult for player 2 to time her snowball to coincide with the running of player 1 —and a badly timed snowball enables player 1 to reach s_{home}. Thus, if player 1 runs at each round with probability p, when p goes to 0, he is able to reach s_{home} with probability approaching 1 [KS81]. Hence, the answer to the reachability question is limit sure but not almost sure.

In this paper, we study simultaneous reachability games, and we consider strategies for the players that can be both randomized and history-dependent. We focus on *deterministic* games, in which the current state and the players' moves uniquely determine the successor state; the more general case of *probabilistic games*, in which the successor state is chosen according to a probability distribution, is similar, and has been described in [dAHK98].

The contributions of the paper are as follows. First, we provide efficient algorithms that, given a finite simultaneous game and a set R of target states, determine the sets *Sure*(R), *Almost*(R) and *Limit*(R) of initial states for which the answer to the reachability question is sure, almost sure, and limit sure, respectively. The set *Sure*(R) can be determined in linear time [TW68, Bee80]. By contrast, the determination of the sets *Almost*(R) and *Limit*(R) requires quadratic time. All three algorithms are formulated as nested fixed-point computations, and can be implemented using symbolic state-space traversal methods [BCM+92]. Our algorithms also enable the effective construction of winning strategies for player 1, and spoiling strategies for player 2, for the three types of answers. The correctness proofs for the algorithms, and for all results of the paper, can be found in [dAHK98].

Second, we characterize the three kinds of reachability in terms of the time (i.e., the number of rounds) required to reach a target state, and in terms of the types of winning and spoiling strategies available to the two players. In particular, while the time to target is bounded from *Sure*(R), only the expected time to target can be bounded from *Almost*$(R) \setminus$ *Sure*(R). From *Limit*$(R) \setminus$ *Almost*(R), neither the time to target nor the expected time to target are bounded. We also show that the spoiling strategies for almost-sure reachability must in general have infinite memory, in contrast with the situation for Markov decision processes [Der70] and for limit-sure reachability [KS81].

Third, we introduce a temporal logic for the specification of open systems, which can be used both for two-agent systems (system vs. environment) and for more general, multi-agent systems. The logic, called *Randomized ATL* (RATL), is an extension of the logic *Alternating Temporal Logic* of [AHK97]. Both logics let us specify that a set of agents has strategies to ensure that the paths of the global system satisfy given temporal properties. The logic ATL considers only deterministic strategies; hence its semantics is defined on the basis of the sure answer for reachability questions. The logic RATL considers instead randomized strategies, and it distinguishes between three kinds of satisfaction for path properties: sure satisfaction (as in ATL), almost-sure satisfaction, and limit-sure satisfaction. The proper hierarchy between sure, almost-sure and limit-sure reachability implies a proper hierarchy for the three kinds of satisfaction. We show that this hierarchy collapses in the special case of safety properties, such as invariance. Our algorithms for solving the reachability question for simultaneous games lead to a symbolic, quadratic-time model-checking algorithm for RATL.

Related work. Polynomial-time algorithms to compute the sets *Almost*(R) and *Limit*(R) are already known for

one-player games and for turn-based games.

A *one-player* game is a game in which only one player can choose among more than one move. While deterministic one-player games are equivalent to graphs, and are thus easily analyzed, probabilistic one-player games are equivalent to Markov decision processes. In Markov decision processes, standard arguments concerning the existence of optimal strategies show that $\mathit{Almost}(R) = \mathit{Limit}(R)$; moreover, this set can be computed in polynomial time by a reduction to linear programming [Der70]. If the only player is player 1, which attempts to reach set R, we can also compute $\mathit{Almost}(R)$ using the polynomial-time algorithms independently proposed by [dA97, Var95], which avoid the use of linear programming. If the only player is player 2, to compute $\mathit{Almost}(R)$ it suffices to compute the set of states of a Markov decision process from which R is reached with probability 1 under all strategies. This can be done with the polynomial-time algorithm of [HSP83, Var85, CY88], which again avoids the use of linear programming.

In deterministic turn-based games the three types of winning states coincide: that is, $\mathit{Sure}(R) = \mathit{Almost}(R) = \mathit{Limit}(R)$. The problem of computing these sets is equivalent to the previously mentioned AND-OR reachability problem, and the existence of memoryless deterministic winning and spoiling strategies follows from an analysis of the algorithms. In probabilistic turn-based games, we have $\mathit{Almost}(R) = \mathit{Limit}(R)$, and these sets can be computed in polynomial time [Yan98]. The probem of computing these sets can also be reduced to the one of solving *switching-controller undiscounted games*, but this reduction does not yield a polynomial-time algorithm [VTRF83a]. The problem of deciding which player has the greatest probability of winning is in NP $\cap$ CO-NP [Con92].

For general reachability games with finite sets of states and actions, [KS81] showed the existence of memoryless ε-optimal strategies for both players. While these results imply the existence of memoryless winning and spoiling strategies for limit-sure reachability, they do not provide methods for the effective construction of these strategies. The maximal probability with which player 1 can force a visit to R can be computed witn a successive approximation method proposed for total reward stochastic games [TV87, FV97]. However, we are not aware of previous criteria for efficiently deciding whether the sequence of approximations converges to 1. Surveys of algorithms for general stochastic games can be found in [RF91, FV97].

2. Reachability Games

A (two-player) *game structure* $G = \langle S, \mathit{Moves}, \Gamma_1, \Gamma_2, p \rangle$ consists of the following components:

- A finite state space S.
- A finite set *Moves* of moves.
- Two move assignments $\Gamma_1, \Gamma_2 : S \mapsto 2^{\mathit{Moves}} \setminus \emptyset$. For $i \in \{1, 2\}$, assignment Γ_i associates with each state $s \in S$ the non-empty set $\Gamma_i(s) \subseteq \mathit{Moves}$ of moves available to player i at state s. For technical convenience, we assume that $\Gamma_i(s) \cap \Gamma_j(t) = \emptyset$ unless $i = j$ and $s = t$, for all $i, j \in \{1, 2\}$ and $s, t \in S$, so that all moves are distinct.
- A transition function $\delta : S \times \mathit{Moves} \times \mathit{Moves} \mapsto S$, which associates with every state $s \in S$ and all moves $a_1 \in \Gamma_1(s)$ and $a_2 \in \Gamma_2(s)$ a successor state $\delta(s, a_1, a_2) \in S$.

At every state $s \in S$, player 1 chooses a move $a_1 \in \Gamma_1(s)$, and simultaneously and independently player 2 chooses a move $a_2 \in \Gamma_2(s)$; the game then proceeds to the successor state $\delta(s, a_1, a_2)$. A *path* of G is an infinite sequence $\overline{s} = s_0, s_1, s_2, \ldots$ of states in S such that for all $k \geq 0$, there are moves $a_1^k \in \Gamma_1(s_k)$ and $a_2^k \in \Gamma_2(s_k)$ such that $s_{k+1} = \delta(s_k, a_1^k, a_2^k)$. We denote by Ω the set of all paths.

A *reachability game* (or *game*, for short) $\mathcal{G} = \langle G, R \rangle$ consists of a game structure G and a set $R \subseteq S$ of *target states;* the set R itself is called the *target set.* The goal of player 1 in the game $\mathcal{G}$ is to reach a state in the target set R, and the goal of player 2 is to prevent this. Thus, a reachability game is a special case of a *recursive game* [Eve57].

We say that a game structure G is *turn-based* if at every state at most one player can choose among multiple moves; that is, for every state $s \in S$ there exists at most one $i \in \{1, 2\}$ with $|\Gamma_i(s)| > 1$.

In the following, we consider a game $\mathcal{G} = \langle \langle S, \mathit{Moves}, \Gamma_1, \Gamma_2, p \rangle, R \rangle$, unless otherwise noted. To simplify the presentation of the results, we assume that the target set R is *absorbing*; that is, we assume that for all $s \in R$ and all $a_1 \in \Gamma_1(s)$ and $a_2 \in \Gamma_2(s)$, we have $\delta(s, a_1, a_2) \in R$. If R is not absorbing, it is trivial to obtain an equivalent game with an absorbing target set. We define the *size* of the game $\mathcal{G}$ to be equal to the number of entries of the transition function δ; specifically, $|\mathcal{G}| = \sum_{s \in S} |\Gamma_1(s)||\Gamma_2(s)|$.

2.1. Strategies

For a finite set A, a *probability distribution* on A is a function $p : A \mapsto [0, 1]$ such that $\sum_{a \in A} p(a) = 1$. We denote the set of probability distributions on A by $\mathcal{D}(A)$. A *strategy* for player $i \in \{1, 2\}$ is a mapping $\pi_i : S^+ \mapsto \mathcal{D}(\mathit{Moves})$ that associates with every nonempty finite sequence $\sigma \in S^+$ of states, representing the past history of

the game, a probability distribution $\pi_i(\sigma)$ used to select the next move. Thus, the choice of the next move can be history-dependent and randomized. The strategy π_i can prescribe only moves that are available to player i: for all sequences $\sigma \in S^*$ and states $s \in S$, we require that if $\pi_i(\sigma s)(a) > 0$ then $a \in \Gamma_i(s)$. We denote by Π_i the set of all strategies for player $i \in \{1, 2\}$. Given a state $s \in S$ and two strategies $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$, we define $\mathit{Outcomes}(s, \pi_1, \pi_2) \subseteq \Omega$ to be the set of paths that can be followed when the game starts from s and the players use the strategies π_1 and π_2. Formally, $s_0, s_1, s_2, \ldots \in \mathit{Outcomes}(s, \pi_1, \pi_2)$ if $s_0 = s$ and if for all $k \geq 0$ there exist moves $a_1^k \in \Gamma_1(s_k)$ and $a_2^k \in \Gamma_2(s_k)$ such that $\pi_1(s_0, \ldots, s_k)(a_1^k) > 0$, $\pi_2(s_0, \ldots, s_k)(a_2^k) > 0$, and $s_{k+1} \in \delta(s_k, a_1^k, a_2^k)$.

Once the starting state s and the strategies π_1 and π_2 for the two players have been chosen, the game is reduced to an ordinary stochastic process. Hence, the probabilities of events are uniquely defined, where an *event* $\mathcal{A} \subseteq \Omega$ is a measurable set of paths. For an event $\mathcal{A} \subseteq \Omega$, we denote by $\Pr_s^{\pi_1,\pi_2}(\mathcal{A})$ the probability that a path belongs to $\mathcal{A}$ when the game starts from s and the players use the strategies π_1 and π_2. In particular, given a subset $R \subseteq S$ of states, we denote the event of reaching R by $(\Diamond R) = \{s_0, s_1, s_2, \ldots \in \Omega \mid \exists k \,.\, s_k \in R\}$. Then, $\Pr_s^{\pi_1,\pi_2}(\Diamond R)$ is the probability of reaching R when the game starts at s, and playes 1 and 2 use strategies π_1 and π_2, respectively.

Similarly, for a measurable function f that associates a number in $\mathbb{R} \cup \{\infty\}$ with each path, we denote by $\mathrm{E}_s^{\pi_1,\pi_2}\{f\}$ the expected value of f when the game starts from s and the strategies π_1 and π_2 are used. In particular, we denote by $T_{\Diamond R}$ the measurable function that associates with each path $\overline{s} : s_0, s_1, s_2, \ldots$ the time $\min\{k \geq 0 \mid s_k \in R\}$ of first passage in R. Then, $\mathrm{E}_s^{\pi_1,\pi_2}\{T_{\Diamond R}\}$ is the expected time to reach R, when the game starts at s, and playes 1 and 2 use strategies π_1 and π_2, respectively.

Types of strategies. We distinguish the following types of strategies:

- A strategy π is *deterministic* if for all $\sigma \in S^+$ there exists $a \in \mathit{Moves}$ such that $\pi(\sigma)(a) = 1$.
- A strategy π is *counting* if $\pi(\sigma_1 s) = \pi(\sigma_2 s)$ for all $s \in S$ and all $\sigma_1, \sigma_2 \in S^*$ with $|\sigma_1| = |\sigma_2|$; that is, the strategy depends only on the current state and the number of past rounds of the game.
- A strategy π is *finite-memory* if the distribution chosen at every state $s \in S$ depends only on s itself, and on a finite number of bits of information about the past history of the game.
- A strategy π is *memoryless* if $\pi(\sigma s) = \pi(s)$ for all $s \in S$ and all $\sigma \in S^+$.

2.2. Classification of Winning States

A *winning state* of game $\mathcal{G}$ is a state from which player 1 can reach the target set R with probability arbitrarily close to 1. We distinguish three classes of winning states:

- The class $\mathit{Sure}(R)$ of *sure-reachability states* consists of the states from which player 1 has a strategy to force the game to R. Precisely, $s \in \mathit{Sure}(R)$ iff there is $\pi_1 \in \Pi_1$ such that for all $\pi_2 \in \Pi_2$ we have $\mathit{Outcomes}(s, \pi_1, \pi_2) \subseteq (\Diamond R)$.
- The class $\mathit{Almost}(R)$ of *almost-sure-reachability states* consists of the states from which player 1 has a strategy to reach R with probability 1. Precisely, $s \in \mathit{Almost}(R)$ iff there is $\pi_1 \in \Pi_1$ such that for all $\pi_2 \in \Pi_2$ we have $\Pr_s^{\pi_1,\pi_2}(\Diamond R) = 1$.
- The class $\mathit{Limit}(R)$ of *limit-sure-reachability states* consists of the states from which for every real $\epsilon > 0$, player 1 has a strategy to reach R with probability at least $1 - \epsilon$. Precisely, $s \in \mathit{Limit}(R)$ iff $\sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \Pr_s^{\pi_1,\pi_2}(\Diamond R) = 1$.

Clearly, $\mathit{Sure}(R) \subseteq \mathit{Almost}(R) \subseteq \mathit{Limit}(R)$. There are games for which both inclusions are strict. The strictness of the inclusion $\mathit{Sure}(R) \subseteq \mathit{Almost}(R)$ follows from the well-known fact that randomized strategies are more powerful than deterministic strategies [vN28], and is witnessed by the state t_{throw} of the game LEFT-OR-RIGHT. The strictness of the inclusion $\mathit{Almost}(R) \subseteq \mathit{Limit}(R)$ is witnessed by the state s_{hide} of the game HIDE-OR-RUN [KS81].

Winning and spoiling strategies. A *winning strategy for sure reachability* is a strategy π_1 for player 1 that acts as a witness to all states in $\mathit{Sure}(R)$; that is, for all states $s \in \mathit{Sure}(R)$ and all strategies π_2 of player 2, $\mathit{Outcomes}(s, \pi_1, \pi_2) \subseteq (\Diamond R)$. Similarly, a *winning strategy for almost-sure reachability* is a strategy π_1 for player 1 such that for all states $s \in \mathit{Almost}(R)$ and all strategies π_2 of player 2, $\Pr_s^{\pi_1,\pi_2}(\Diamond R) = 1$. A *winning strategy family for limit-sure reachability* is a family $\{\pi_1[\varepsilon] \mid \varepsilon > 0\}$ of strategies for player 1 such that for all reals $\varepsilon > 0$, all states $s \in \mathit{Limit}(R)$, and all strategies π_2 of player 2, $\Pr_s^{\pi_1[\varepsilon],\pi_2}(\Diamond R) \geq 1 - \varepsilon$.

A *spoiling strategy for sure reachability* is a strategy π_2 for player 2 that acts as a witness to all states $s \notin \mathit{Sure}(R)$ and all strategies of player 1; that is, for all states $s \notin \mathit{Sure}(R)$ and all strategies π_1 of player 1, $\mathit{Outcomes}(s, \pi_1, \pi_2) \not\subseteq (\Diamond R)$. Similarly, a *spoiling strategy for almost-sure reachability* is a strategy π_2 for player 2

	Reachability		
	Sure	Almost	Limit
Complexity	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Winning strategies	DM	M	M
Spoiling strategies	M	C	M
Time	Bnd	Unb	Unb
Expected time	Bnd	Bnd	Unb

Table 1. Overview of results about sure, almost-sure, and limit-sure reachability. The input size of the game is indicated by n. The abbreviations DM, M, C stand for deterministic memoryless, (randomized) memoryless, and (randomized) counting, respectively; the abbreviations Bnd and Unb stand for bounded and unbounded.

such that for all states $s \notin \mathit{Almost}(R)$ and all strategies π_1 of player 1, $\Pr_s^{\pi_1,\pi_2}(\Diamond R) < 1$. Finally, a *spoiling strategy for limit-sure reachability* is a strategy π_2 for player 2 such that there exists a real $q > 0$ such that for all states $s \notin \mathit{Limit}(R)$ and all strategies π_1 of player 1, $\Pr_s^{\pi_1,\pi_2}(\Diamond R) \leq 1 - q$.

We will show that for all three types of reachability, winning and spoiling strategies always exist.

2.3. Time to Reachability

For a state $s \in S$ and an integer $m \geq 0$, we say that the *time from s to R is bounded by m* if there exists a strategy π_1 for player 1 such that for all strategies π_2 of player 2, $\sup\{T_{\Diamond R}(\overline{s}) \mid \overline{s} \in \mathit{Outcomes}(s,\pi_1,\pi_2)\} \leq m$. If the time from s to R is not bounded by any integer m, we say that the *time from s to R is unbounded*. We say that the *expected time from s to R is bounded* if there exists an integer $m \geq 0$ and a strategy π_1 for player 1 such that for all strategies π_2 of player 2, we have $\mathrm{E}_s^{\pi_1,\pi_2}\{T_{\Diamond R}\} < m$. Given a subset $U \subseteq S$ of states, we generalize these definitions to U: the time (or the expected time) to R is bounded from U iff it is bounded from all $s \in U$.

2.4. Overview of Our Results

In Table 1 we present an overview of the main results on reachability games that are presented in this paper. The first row lists the complexity of the algorithms for computing the sets of winning states with respect to the three types of reachability. The second and the third row list the types of winning and spoiling strategies available to the players. For each type of reachability, we list the tightest class of strategies that always contains at least one such winning and spoiling strategy (according to the classification of Section 2.1). The last two rows state whether the time to the target, and the expected time to the target, are bounded on the sets of winning states.

3. Computing the Winning States

In this section we present algorithms for computing the three sets $\mathit{Sure}(R)$, $\mathit{Almost}(R)$, and $\mathit{Limit}(R)$.

3.1. Sure-Reachability States

To compute the set $\mathit{Sure}(R)$, we introduce the notion of *move sub-assignments*, and the functions *Pre* and *Stay*.

A *move subassignment* γ for player 1 is a mapping $\gamma : S \mapsto 2^{\mathit{Moves}}$ that associates with each state $s \in S$ a subset $\gamma(s) \subseteq \Gamma_1(s)$ of moves. We use move subassignments to limit the set of moves that player 1 can play during the game. We denote by Δ the set of all move subassignments for player 1.

The function $\mathit{Pre}_1 : 2^S \times \Delta \mapsto 2^S$ is defined by

$$\mathit{Pre}_1(U,\gamma) = \Big\{ s \in S \;\Big|\; \exists a_1 \in \gamma(s) \,.\, \forall a_2 \in \Gamma_2(s) \,.\, \delta(s,a_1,a_2) \in U \Big\}.$$

Intuitively, $\mathit{Pre}_1(U,\gamma)$ is the set of states from which player 1 can be sure of being in U after one round using only moves from γ, regardless of the move chosen by player 2. The function $\mathit{Pre}_2 : 2^S \times \Delta \mapsto 2^S$ is defined in a similar way:

$$\mathit{Pre}_2(U,\gamma) = \Big\{ s \in S \;\Big|\; \exists a_2 \in \Gamma_2(s) \,.\, \forall a_1 \in \gamma(s) \,.\, \delta(s,a_1,a_2) \in U \Big\}.$$

Note that in both $\mathit{Pre}_1(U,\gamma)$ and $\mathit{Pre}_2(U,\gamma)$ the subassignment γ refers to player 1. The function $\mathit{Stay}_1 : 2^S \mapsto \Delta$ is defined such that for all states $s \in S$,

$$\mathit{Stay}_1(U)(s) = \Big\{ a_1 \in \Gamma_1(s) \;\Big|\; \forall a_2 \in \Gamma_2(s) \,.\, \delta(s,a_1,a_2) \in U \Big\}.$$

Intuitively, $\mathit{Stay}_1(U)$ is the largest move subassignment for player 1 that guarantees that the game is in U after one round, regardless of the move chosen by player 2.

The set $\mathit{Sure}(R)$ can be computed using the following algorithm.

Algorithm 1
Input: Reachability game $\mathcal{G} = \langle G, R \rangle$.
Output: Sure-reachability set $\mathit{Sure}(R)$.

> **Initialization:** Let $U_0 = R$.
> **Repeat** For $k \geq 0$, $U_{k+1} = U_k \cup \mathit{Pre}_1(U_k, \Gamma_1)$
> **Until** $U_{k+1} = U_k$.
> **Return:** U_k.

The algorithm is identical to the one used for turn-based games, and it can be implemented to run in time linear in the size of the game [TW68, Bee80]. The algorithm can also be implemented symbolically as a fixed-point computation. The theorem below summarizes some basic facts about the set $\mathit{Sure}(R)$.

Theorem 1 *For a reachability game with target set R:*

1. *Algorithm 1 computes the set $\mathit{Sure}(R)$. The algorithm can be implemented to run in time linear in the size of the game.*
2. *Player 1 has a memoryless deterministic winning strategy for sure reachability.*
3. *Player 2 has a memoryless spoiling strategy for sure reachability. This spoiling strategy cannot in general be deterministic.*
4. *For every state $s \in \mathit{Sure}(R)$, the time from s to R is bounded by the size of the state space.*

If $R = U_0, U_1, \ldots, U_m = \mathit{Sure}(R)$ is the sequence of sets computed by Algorithm 1, a deterministic memoryless winning strategy consists in playing at each $s \in U_{k+1} \setminus U_k$ a fixed move in $\mathit{Stay}(U_k)(s)$, where $0 \leq k < m$. A simple memoryless spoiling strategy for player 2 consists in choosing a move uniformly at random from the available moves at each state. To see that deterministic spoiling strategies may not exist in general, it suffices to consider the state t_{throw} of the game LEFT-OR-RIGHT.

Reachability in turn-based games. If a reachability game with target set R is turn-based, then player 2 has a deterministic spoiling strategy π_2 such that $\Pr_s^{\pi_1,\pi_2}(\Diamond R) = 0$ for all strategies $\pi_1 \in \Pi_1$ for player 1 and all states $s \notin \mathit{Sure}(R)$. Such a spoiling strategy simply chooses at each $s \notin \mathit{Sure}(R)$ one of the moves $b \in \Gamma_2(s)$ such that $\delta(s, a, b) \notin \mathit{Sure}(R)$ for all $a \in \Gamma_1(s)$ [Tho95].

This observation leads to the fact that in turn-based games we have $\mathit{Sure}(R) = \mathit{Almost}(R) = \mathit{Limit}(R)$, i.e. the three notions of reachability coincide. Another consequence of the above observation is that deterministic turn-based reachability games have "0-1 laws"; that is, for all states $s \in S$ of a turn-based game,

$$\sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \Pr_s^{\pi_1,\pi_2}(\Diamond R) \in \{0, 1\}.$$

This 0-1 law only applies to deterministic, turn-based games. As an example of a (non-turn-based) deterministic game without a 0-1 law, consider a one-round version of the game LEFT-OR-RIGHT. After the only round, the game moves from the state t_{throw} either to the state t_{hit} or to the state t_{missed}. Then,

$$\sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \Pr_{t_{\mathit{throw}}}^{\pi_1,\pi_2}(\Diamond\{t_{\mathit{hit}}\}) = \frac{1}{2}\,.$$

3.2. Almost-Sure-Reachability States

The algorithm for the computation of the set $\mathit{Almost}(R)$ uses the function *Safe*. For $i \in \{1, 2\}$, the function $\mathit{Safe}_i \colon 2^S \times \Delta \mapsto 2^S$ associates with each state set $U \subseteq S$ and each move subassignment $\gamma \subseteq \Delta$ the largest subset $V \subseteq U$ such that $\mathit{Pre}_i(V, \gamma) \subseteq V$. The set $\mathit{Pre}_i(V, \gamma)$ can be computed as the limit of the decreasing sequence $U_0 = V, U_1, U_2, \ldots$, where we take $U_{k+1} = V \cap \mathit{Pre}_i(U_k, \gamma)$ for $k \geq 0$. Hence, the set V is the largest subset of U that player i can be sure of not leaving at any time in the future, regardless of the moves chosen by the other player, given that player 1 chooses moves only according to γ. Using an appropriate data structure, as suggested in [TW68, Bee80], the computation of $\mathit{Safe}_i(V, \gamma)$ can be implemented to run in linear time. The algorithm can also be implemented symbolically as a nested fixed-point iteration.

The set $\mathit{Almost}(R)$ can be computed using the following algorithm. The algorithm has running time quadratic in the size of the game, and it can be implemented symbolically as a nested fixed-point computation.

Algorithm 2
Input: Reachability game $\mathcal{G} = \langle G, R \rangle$.
Output: Almost-sure-reachability set $\mathit{Almost}(R)$.

> **Initialization:** Let $U_0 = S$, $\gamma_0 = \Gamma_1$.
> **Repeat** For $k \geq 0$, let
>
> $$C_k = \mathit{Safe}_2(U_k \setminus R, \gamma_k),$$
> $$U_{k+1} = \mathit{Safe}_1(U_k \setminus C_k, \gamma_k),$$
> $$\gamma_{k+1} = \mathit{Stay}_1(U_{k+1})$$
>
> **Until** $U_{k+1} = U_k$.
> **Return:** U_k.

The algorithm can be understood as follows. The set C_0 is the largest subset of $S \setminus R$ to which player 2 can confine the game. Player 1 must avoid entering C_0 at all costs: if C_0 is entered with positive probability, R will not be reached with probability 1. The set U_1 is the largest set of states from which player 1 can avoid entering C_0. The move subassignment γ_1 then associates with each state the set of moves that player 1 can select in order to avoid leaving U_1. Since $\gamma_1 \subseteq \Gamma_1$, by choosing only moves from γ_1, player 1 may lose some of the ability to resist confinement. The set C_1 is the largest subset of $U_1 \setminus R$ to which player 2 can confine the game, under the assumption that player 1 uses only moves from γ_1. The set U_2 is then the largest subset of U_1 from which player 1 can avoid entering C_1, and the subassignment $\gamma_2 \subseteq \gamma_1$ guarantees that player 1 never leaves U_2. The computation of C_k, U_{k+1}, and γ_{k+1}, for $k \geq 0$, continues in this way, until we reach $m > 0$ such that:

- if player 1 chooses moves only from γ_m, the game will never leave U_m;
- player 2 cannot confine the game to $U_m \setminus R$, even if player 1 chooses moves only from γ_m.

At this point, we have $U_{m+1} = U_m = \mathit{Almost}(R)$.

Theorem 2 *For a reachability game with target set R:*

1. *Algorithm 2 computes the set $\mathit{Almost}(R)$. The algorithm can be implemented to run in time quadratic in the size of the game.*
2. *Player 1 has a memoryless winning strategy for almost-sure reachability. This winning strategy cannot in general be deterministic.*
3. *Player 2 has a counting spoiling strategy for almost-sure reachability. This spoiling strategy cannot in general be deterministic, nor finite-memory.*
4. *For every state $s \in \mathit{Almost}(R)$, the expected time from s to target R is bounded.*

If $S = U_0, U_1, \ldots, U_m = \mathit{Almost}(R)$ and $\gamma_1, \ldots, \gamma_m$ are the sequences of sets and move sub-assignments computed by the algorithm, a memoryless winning strategy for player 1 consists in playing at each $s \in U_m$ a move chosen uniformly at random from $\gamma_m(s)$.

Game HIDE-OR-RUN is an example of a game that does not have a finite-memory spoiling stragety. In fact, it can be seen that for each finite-memory strategy of player 2, player 1 has a strategy to get from s_{hide} to s_{home} with probability 1. To construct an infinite-memory spoiling strategy, we proceed as follows. Consider the two memoryless strategies π_2^1 and π_2^2 for player 2 defined by

$$\begin{cases} \pi_2^1(s_{\mathit{hide}})(\mathit{throw}) = 0 \\ \pi_2^1(s_{\mathit{hide}})(\mathit{wait}) = 1 \end{cases} \qquad \begin{cases} \pi_2^2(s_{\mathit{hide}})(\mathit{throw}) = \frac{1}{2} \\ \pi_2^2(s_{\mathit{hide}})(\mathit{wait}) = \frac{1}{2}. \end{cases}$$

The strategy π_2^1 is effective against the strategies of player 1 that wait till player 2 throws the snowball before running. On the other hand, the strategy π_2^2 is effective against the strategies of player 1 that may run before having seen player 2 throw the snowball. To obtain a spoiling strategy, which must work in all cases, we "mix" strategies π_2^1 and π_2^2, as if player 2 could secretly flip a coin before the game starts to decide which of the two strategies to use. The idea of flipping a coin before the game starts to determine which strategy to use is known as *initial randomization*, and it constrasts with *on-the-fly randomization*, that is the process of flipping coins during the game to choose the move to be played. Our definition of strategy allows only on-the-fly randomization. Nevertheless, from [Der70] we know that the initial randomization between finitely many strategies $\pi^1, \pi^2, \ldots, \pi^m$ can be simulated by a single strategy π that uses on-the-fly randomization only. However, there is a price to pay: even when strategies $\pi^1, \pi^2, \ldots, \pi^m$ are memoryless, strategy π may need infinite memory. In our case, by mixing the strategies π_2^1 and π_2^2 with equal probability, we obtain the strategy π_2, defined for all $k > 0$ by

$$\pi_2({s_{\mathit{hide}}}^k)(\mathit{wait}) = 2^{(-1/2^k)},$$

where ${s_{\mathit{hide}}}^k$ is the path consisting of k states s_{hide}. It is easy to check that if player 2 plays according to π_2, then he eventually throws a snowball with probability 1/2, consistently with the fact that π_2 is the "equal probability mix" of π_2^1 and π_2^2. Note that π_2 is an infinite-memory, counting strategy.

3.3. Limit-Sure-Reachability States

Similarly to the algorithm for almost-sure reachability, the algorithm for limit-sure reachability computes a decreasing sequence $U_0 = S$, U_1, U_2, ... of candidate winning states; the set $\mathit{Limit}(R)$ is the limit of this sequence. At each iteration $k \geq 0$, the algorithm determines a set $C_k \subseteq U_k \setminus R$ of states from which player 1 cannot force a visit to R with probability arbitrarily close to 1, and assigns $U_{k+1} = U_k \setminus C_k$.

The set C_k is also determined in an iterative fashion. Initially, we set $C_k^0 = U_k \setminus R$; then, we remove states from this set, computing a decreasing sequence C_k^0, C_k^1, C_k^2, ... that converges to C_k. To understand how this latter sequence is computed, consider the stage of these iterations when sets U_k and C_k^j have been computed, and consider a

state $s \in C_k^j$. From the point of view of player 1, the situation from s is as follows. By construction, the states in $S \setminus U_k$ are not winning states, so that player 1 must avoid leaving U_k. Moreover, as $C_k^j \cap R = \emptyset$, player 1 must also avoid being trapped in C_k^j. Hence, player 1 must try to escape from C_k^j, and at the same time avoid leaving U_k.

Denote by $\xi_1 \in \mathcal{D}(\Gamma_1(s))$ and $\xi_2 \in \mathcal{D}(\Gamma_2(s))$ the distributions used by players 1 and 2 at s, respectively. Given a subset $V \subseteq S$ of states, indicate by $p(s, \xi_1, \xi_2)(V)$ the probability of going from s to V in one round under distributions ξ_1 and ξ_2. Consider the ratio

$$\frac{p(s, \xi_1, \xi_2)(S \setminus C_k^j)}{p(s, \xi_1, \xi_2)(S \setminus U_k)} \tag{1}$$

between the probabilities of escaping from C_k^j and of leaving U_k. If player 1 can choose ξ_1 to make the ratio (1) arbitrarily large, then any attempt of player 2 to confine the game to C_k^j can involve s only in a transitory fashion: in fact, infinitely many visits to s would lead to escaping from C_k^j with arbitrarily high probability, while losing the game by leaving U_k with negligible probability. On the other hand, if the ratio (1) cannot be made arbitrarily large, then player 2 can choose ξ_2 so that, at each visit to s, the probability of leaving C_k^j is compensated by a proportional probability of leaving U_k. In this case, player 1 cannot use state s to escape from C_k^j.

These considerations motivate our definition of *limit-escape states*. Given the sets $U \subseteq S$ and $C \subseteq U$, and a state $s \in C$, we say that s is *limit escape* with respect to C and U if

$$\sup_{\xi_1 \in \mathcal{D}(\Gamma_1(s))} \inf_{\xi_2 \in \mathcal{D}(\Gamma_2(s))} \frac{p(s, \xi_1, \xi_2)(S \setminus C)}{p(s, \xi_1, \xi_2)(S \setminus U)} = \infty \ . \tag{2}$$

A state s is then removed from C_k^j to form C_k^{j+1} iff it is limit escape with respect to C_k^j and U_k.

Let us illustrate the algorithm for limit-sure reachability on game HIDE-OR-RUN. The algorithm first computes $C_0 = \{s_{wet}\}$ and $U_1 = \{s_{hide}, s_{safe}, s_{home}\}$. The state s_{safe} is easily eliminated from $C_1^0 = \{s_{hide}, s_{safe}\}$, leading to $C_1^1 = \{s_{hide}\}$. At s_{hide}, player 1 can play either *hide* or *run*. To escape from C_1^1 and reach s_{home} with arbitrarily high probability, player 1 must be "patient" and choose move *run* with sufficiently low probability at each round. Precisely, for every $0 < \varepsilon < 1$, define the distribution $\xi_1[\varepsilon] \in \mathcal{D}(\Gamma_1(s))$ by:

$$\xi_1[\varepsilon](run) = \varepsilon, \qquad \xi_1[\varepsilon](hide) = 1 - \varepsilon \ . \tag{3}$$

By using distribution $\xi_1[\varepsilon]$ and letting $\varepsilon \to 0$, player 1 can make the ratio (1) diverge (for $k = j = 1$): in fact,

$$\lim_{\varepsilon \to 0} \inf_{\xi_2 \in \mathcal{D}(\Gamma_2(s))} \frac{p(s, \xi_1[\varepsilon], \xi_2)(S \setminus C_1^1)}{p(s, \xi_1[\varepsilon], \xi_2)(S \setminus U_1)}$$

$$= \lim_{\varepsilon \to 0} \inf_{0 \le q \le 1} \frac{\varepsilon(1-q) + (1-\varepsilon)q}{\varepsilon q} = \lim_{\varepsilon \to 0} \frac{1-\varepsilon}{\varepsilon} = \infty \ .$$

The divergence of the ratio between the one-round probability of escape and the one-round probability of capture enables player 1 to eventually escape with probability arbitrarily close to 1. To verify this, let $\pi_1[\varepsilon]$ be the memoryless strategy for player 1 that uses distribution $\xi_1[\varepsilon]$ at state s_{hide}. Once $\pi_1[\varepsilon]$ is fixed, results on Markov decision processes ensure that the optimal strategy for player 2 to avoid reaching R is memoryless (and also deterministic) [Der70]. Hence, simple calculations show that

$$\inf_{\pi_2 \in \Pi_2} \Pr_{s_{hide}}^{\pi_1[\varepsilon], \pi_2}(\Diamond\{s_{home}\}) = 1 - \varepsilon \ .$$

The fact that $s_{hide} \in \mathit{Limit}(R)$ follows by taking the limit $\varepsilon \to 0$ in this equation. This confirms that $s_{hide} \in \mathit{Limit}(R) \setminus \mathit{Almost}(R)$, as we mentioned in the introduction.

There is a relation between the computation of the sets C_k in the algorithms for almost-sure and limit-sure reachability. In Algorithm 2, the set C_k is computed by $C_k = \mathit{Safe}_2(U_k \setminus R, \gamma_k)$. If we expand the computation of C_k, we see that C_k is again computed as the fixpoint of a decreasing sequence C_k^0, C_k^1, C_k^2, ... For $j \ge 0$, a state s is removed from C_k^j if there is ξ_1 such that for all ξ_2, the numerator of (1) is non-zero, and the denominator is 0. In this case, player 1 from s can use ξ_1 to escape C_k^j with positive probability, while not risking a retreat from U_k. Such an s is called a *safe-escape state.* For almost-sure reachability, player 1 must use safe escape, because in order to reach the target with probability 1 he cannot risk to lose. For limit-sure reachability, player 1 can instead use limit escape: as long as the ratio between risk (of retreat) and escape (towards the target) can be made arbitrarily large, the player can reach the target with probability arbitrarily close to 1.

3.3.1 Computing Limit-Escape States

The following algorithm determines whether a state is limit escape.

Algorithm 3
Input: Game structure G, two sets $C \subseteq U \subseteq S$ of states, and a state $s \in C$.
Output: YES if s is limit escape with respect to C and U, NO otherwise.

Initialization: Let $\mathcal{B}_{-1} = \emptyset$.
Repeat For $k \ge 0$, let

$$\mathcal{A}_k = \big\{a \in \Gamma_1(s) \mid \forall b \in \Gamma_2(s) \ . \ \text{if } \delta(s,a,b) \notin U \text{ then } b \in \mathcal{B}_{k-1}\big\},$$

$$\mathcal{B}_k = \big\{b \in \Gamma_2(s) \mid \exists a \in \mathcal{A}_k \ . \ \delta(s,a,b) \notin C\big\}$$

Until $\mathcal{A}_{k+1} = \mathcal{A}_k$ and $\mathcal{B}_{k+1} = \mathcal{B}_k$.
Return: YES if $\mathcal{B}_k = \Gamma_2(s)$, NO otherwise.

We say that a move $a \in \Gamma_1(s)$ is *labeled* if $a \in \mathcal{A}_k$; if a is labeled we define $\ell(a) = \min\{i \mid a \in \mathcal{A}_i\}$. Similarly, we say that a move $b \in \Gamma_2(s)$ is labeled if $b \in \mathcal{B}_k$. The algorithm declares the state s limit escape w.r.t. U and C iff all the moves $\Gamma_2(s)$ for player 2 at s are labeled. When Algorithm 3 is given as input state s_{hide} of game HIDE-OR-RUN and $C = \{s_{hide}\}$, $U = \{s_{hide}, s_{safe}, s_{home}\}$, it labels the moves of player 1 at s_{hide} with

$$\ell(hide) = 0 \qquad \ell(run) = 1\ . \tag{4}$$

If a state s is declared limit escape, then also all moves in $\Gamma_1(s)$ are labeled, and their labels provide us with an ε-indexed family $\xi_1[\varepsilon]$ of distributions that make the ratio (2) diverge. Precisely, for $0 < \varepsilon < 1/(2|\Gamma_1(s)|)$, the distribution $\xi_1[\varepsilon]$ plays move $a \in \Gamma_1(s)$ with probability $\varepsilon^{\ell(a)}$ if $\ell(a) > 0$, and it plays all the moves in $\{a \in \Gamma_1(s) \mid \ell(a) = 0\}$ uniformly at random with the remaining probability. From (4), we see that the distribution constructed in this fashion for the state s_{hide} of the game HIDE-OR-RUN coincides with the one given in (3).

3.3.2 Computing Limit-Sure Reachability States

Given the target set R and a subset $U \subseteq S$ with $R \subseteq U$, the following algorithm computes the largest subset $Cage(U) = C \subseteq U \setminus R$ that does not contain any limit-escape state with respect to C and U. Set C is computed as the limit of the previously described decreasing sequence $C^0, C^1, C^2, \ldots$.

Algorithm 4
Input: Reachability game $\mathcal{G} = \langle G, R\rangle$, and $U \subseteq S$ with $R \subseteq U$.
Output: $Cage(U) \subseteq S$.

Initialization: Let $C^0 = U \setminus R$.
Repeat For $j \geq 0$, let $C^{j+1} =$
$\{s \in C^j \mid s$ not limit escape w.r.t. C^j and $U\}$
Until $C^{j+1} = C^j$.
Return: C^j.

The set $Limit(R)$ can be computed using the following algorithm, which uses the computation of *Cage* as a subroutine.

Algorithm 5
Input: Reachability game $\mathcal{G} = \langle G, R\rangle$.
Output: Limit-sure-reachability set $Limit(R)$.

Initialization: Let $U_0 = S$.
Repeat For $k \geq 0$, let $U_{k+1} = U_k \setminus Cage(U_k)$
Until $U_{k+1} = U_k$.
Return: U_k.

The following theorem summarizes the results on limit-sure reachability.

Theorem 3 *For a reachability game with target set R:*

1. *Algorithm 5 computes set $Limit(R)$. The algorithm can be implemented to run in time quadratic in the size of the game.*
2. *Player 1 has a family of memoryless winning strategies for limit-sure reachability. These winning strategies cannot in general be deterministic.*
3. *Player 2 has a memoryless spoiling strategy for limit-sure reachability. This spoiling strategy cannot in general be deterministic.*

To obtain a version of the algorithm that runs in quadratic time it is necessary to optimize the implementation of Algorithm 4; the optimized version is given in [dAHK98]. Results 2 and 3 are from [KS81]; the construction of the winning and spoiling strategies is explained in [dAHK98].

To see that deterministic memoryless winning strategies may not exist in general, it suffices to consider the state t_{throw} of the game LEFT-OR-RIGHT. To see that deterministic memoryless spoiling strategies may not exist in general, it suffices to consider again the one-round version of the game LEFT-OR-RIGHT, in which after the only round the game moves from the state t_{throw} either to the state t_{hit} or to the state t_{missed}. Then, it is immediate to check that $Limit(t_{hit}) = \{t_{hit}\}$; moreover, by considering the state t_{throw} we see that there are no deterministic spoiling strategies.

4. Randomized ATL

For the specification and verification of open systems, [AHK97] introduced the temporal logic *Alternating Temporal Logic* (ATL). The logic ATL is interpreted over multi-player game structures, and includes formulas of the form $\langle\langle A\rangle\rangle\theta$, which asserts that a team A of players (called *agents*) has a strategy to ensure that all outcomes of the game satisfy the specification θ. The semantics of the logic ATL is defined with respect to deterministic strategies only. Consequently, in a two-player game structure, if φ_R is a formula defining the target set R, then the formula $\langle\langle Player1\rangle\rangle\Diamond\varphi_R$ is true exactly in all the sure-reachability states.

In this section, we generalize the logic to *Randomized ATL* (RATL). The logic RATL is defined with respect to randomized strategies, and distinguishes between three kinds of satisfaction for path properties: sure satisfaction, almost-sure satisfaction, and limit-sure satisfaction; correspondingly, the single quantifier $\langle\langle\ \rangle\rangle$ of ATL is replaced

by the three quantifiers $\langle\langle\ \rangle\rangle_{sure}$, $\langle\langle\ \rangle\rangle_{almost}$, and $\langle\langle\ \rangle\rangle_{limit}$. For example, the formula $\langle\langle \mathit{Player1} \rangle\rangle_{almost} \Diamond \varphi_R$ will be true exactly in the almost-sure-reachability states.

Formally, a *system* $\mathcal{S} = \langle n, S, \mathit{Moves}, \Gamma, \delta, Q, L \rangle$ consists of a number $n > 0$ of agents, a finite state space S, a finite set *Moves* of moves, a move assignment $\Gamma : S \times \{1, \ldots, n\} \mapsto 2^{\mathit{Moves}} \setminus \emptyset$, a transition function $\delta : S \times \mathit{Moves}^n \mapsto S$, a finite set Q of propositions, and a function $L : S \mapsto 2^Q$ that labels each state with the propositions that are true in the state. Thus, a system with n agents is a labeled n-player game structure: at every state $s \in S$, each agent $i \in \{1, \ldots, n\}$ chooses a move $a_i \in \Gamma(s, i)$, and the game proceeds to the state $\delta(s, a_1, \ldots, a_n)$. Typically, the agents model individual processes, or components, of a reactive program. The *paths* of $\mathcal{S}$ are defined in analogy to two-player game structures. A *strategy* π_A for a (possibly empty) set $A = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ of agents is a mapping $\pi_A : S^+ \mapsto \mathcal{D}(\mathit{Moves}^k)$ such that $\pi_A(\sigma s)(a_1, \ldots, a_k) > 0$ implies $a_j \in \Gamma(s, i_j)$ for all $1 \leq j \leq k$. Given a set A of agents, we denote by Π_A the set of strategies for A.

The temporal logic RATL is defined with respect to a set Q of propositions and a set $\Sigma = \{1, \ldots, n\}$ of agents. A Randomized ATL formula is one of the following:

- q, for propositions $q \in Q$.
- $\neg\varphi$ or $\varphi \vee \psi$, where φ and ψ are RATL formulas.
- $\langle\langle A \rangle\rangle_{win} \bigcirc \varphi$ or $\langle\langle A \rangle\rangle_{win} \Box \varphi$ or $\langle\langle A \rangle\rangle_{win}\, \varphi\, \mathcal{U}\, \psi$, where $A \subseteq \{1, \ldots, n\}$ is a set of agents, *win* $\in \{\mathit{sure}, \mathit{almost}, \mathit{limit}\}$ is a type of winning condition, and φ and ψ are RATL formulas.

The operators $\langle\langle\ \rangle\rangle_{win}$ are *path quantifiers*, and $\bigcirc$ ("next"), $\Box$ ("always"), and $\mathcal{U}$ ("until") are the usual *temporal operators* [MP91]. We interpret RATL formulas over the states of a system $\mathcal{S}$ that has the same sets of agents and propositions used to define the formulas. The subformulas of RATL of the form $\bigcirc\varphi$, $\Box\varphi$, or $\varphi\,\mathcal{U}\,\psi$ are called *path subformulas*, and they are interpreted over the paths of $\mathcal{S}$. For a path subformula θ, we denote by $[\![\theta]\!]$ the event consisting of all the paths that satisfy θ, as defined by the standard semantics of the temporal operators. Subformulas of RATL of the form p, $\neg\varphi$, $\varphi \vee \psi$, or $\langle\langle A \rangle\rangle_{win}\, \theta$ are called *state subformulas*, and they are interpreted over the states of $\mathcal{S}$. For a state subformula φ, we write $s \models \varphi$ to indicate that the state s satisfies φ. We present here only the semantics for state subformulas of the form $\langle\langle A \rangle\rangle_{win}\, \theta$; the propositional and boolean cases are standard. For a path subformulas θ, we define:

- $s \models \langle\langle A \rangle\rangle_{sure}\, \theta$ iff there exists $\pi_A \in \Pi_A$ such that for all $\pi_{\Sigma\setminus A} \in \Pi_{\Sigma\setminus A}$ we have $\mathit{Outcomes}(s, \pi_A, \pi_{\Sigma\setminus A}) \subseteq [\![\theta]\!]$.
- $s \models \langle\langle A \rangle\rangle_{almost}\, \theta$ iff there exists $\pi_A \in \Pi_A$ such that for all $\pi_{\Sigma\setminus A} \in \Pi_{\Sigma\setminus A}$ we have $\Pr_s^{\pi_A, \pi_{\Sigma\setminus A}}([\![\theta]\!]) = 1$.
- $s \models \langle\langle A \rangle\rangle_{limit}\, \theta$ iff

$$\sup_{\pi_A \in \Pi_A} \inf_{\pi_{\Sigma\setminus A} \in \Pi_{\Sigma\setminus A}} \Pr_s^{\pi_A, \pi_{\Sigma\setminus A}}([\![\theta]\!]) = 1.$$

In particular, the logic ATL is the fragment of RATL where the only path quantifier is $\langle\langle A \rangle\rangle_{sure}$.

If $s \models \langle\langle A \rangle\rangle_{win}\theta$, for *win* $\in \{\mathit{sure}, \mathit{almost}, \mathit{limit}\}$, then the winning strategies provide a controller C for the set of agents A. When the controller C is composed with the set A of agents, the resulting system is guaranteed to satisfy θ with *win* confidence. If *win* = *sure*, then the controller can always be chosen to be deterministic and memoryless. If *win* $\in \{\mathit{almost}, \mathit{limit}\}$, then the controller can still be chosen to be memoryless, but it may need to be randomized.

From the classification of winning states in Section 2, it follows that $s \models \langle\langle A \rangle\rangle_{sure}\, \theta$ implies $s \models \langle\langle A \rangle\rangle_{almost}\, \theta$, which in turn implies $s \models \langle\langle A \rangle\rangle_{limit}\, \theta$; the reverse implications do not necessarily hold. Interestingly, the implications can be strict only for path subformulas θ of the form $\varphi\,\mathcal{U}\,\psi$, which specify liveness-like properties (such as reachability). By contrast, for path subformulas θ of the form $\bigcirc\varphi$ and $\Box\varphi$, which specify safety-like properties, the three winning conditions are equivalent.

Theorem 4 *Consider a path formula θ of the form $\bigcirc\varphi$ or $\Box\varphi$. Then, for every state s of a system $\mathcal{S}$, we have $s \models \langle\langle A \rangle\rangle_{sure}\, \theta$ iff $s \models \langle\langle A \rangle\rangle_{almost}\, \theta$ iff $s \models \langle\langle A \rangle\rangle_{limit}\, \theta$.*

The *model-checking problem* for RATL asks, given a system $\mathcal{S}$ and an RATL formula φ, for the set of states of $\mathcal{S}$ that satisfy φ. A model-checking algorithm for RATL can proceed bottom-up on the state subformulas of φ, as in CTL and ATL model checking [CE81, QS81, AHK97]. The non-trivial cases are $\langle\langle A \rangle\rangle_{sure}\, \varphi\, \mathcal{U}\, \psi$, $\langle\langle A \rangle\rangle_{almost}\, \varphi\, \mathcal{U}\, \psi$, and $\langle\langle A \rangle\rangle_{limit}\, \varphi\, \mathcal{U}\, \psi$. The subformula $\langle\langle A \rangle\rangle_{sure}\, \varphi\, \mathcal{U}\, \psi$ can be checked as in ATL. In order to check the other two subformulas, we first construct a two-player game structure, in which player 1 corresponds to the set A of agents, and player 2 corresponds to the set $\Sigma \setminus A$. We define the target set to be $R = \{s \in S \mid s \models \psi\}$. If R is not absorbing, we modify locally the game structure to make it absorbing. To check the subformula $\langle\langle A \rangle\rangle_{almost}\, \varphi\, \mathcal{U}\, \psi$, we modify Algorithm 2, so that $C_0 = \{s \in S \mid s \not\models \varphi \vee \psi\}$. To check the subformula $\langle\langle A \rangle\rangle_{limit}\, \varphi\, \mathcal{U}\, \psi$, we modify Algorithm 5, so that $U_0 = \{s \in S \mid s \models \varphi \vee \psi\}$. Intuitively, while in the $\Diamond R$ reachability game player 1 only has to avoid states in which player 2 can keep him away from the target set R, in the $\varphi\,\mathcal{U}\,\psi$ game player 1 also has to avoid states that satisfy neither φ nor ψ.

Theorem 5 *The model-checking problem for RATL specifications can be solved in time quadratic in the size of the system and linear in the size of the formula.*

Acknowledgments. We thank Rajeev Alur, Jerzy Filar, Christos Papadimitriou, T.E.S. Raghavan, Valter Sorana, Moshe Vardi, and Mihalis Yannakakis for helpful discussions and pointers to the literature.

References

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. 38th IEEE Symposium on Foundations of Computer Science*, pages 100-109, 1997.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Information and Computation*, 95(2):142-170, 1992.

[Bee80] C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. on Database Systems*, 5:241–259, 1980.

[BT91] D.P. Bertsekas and J.N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. of Op. Res.*, 16(3):580–595, 1991.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *Lect. Notes in Comp. Sci.*, pages 52–71. Springer-Verlag, 1981.

[CS91] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. In *Computer Aided Verification*, volume 575 of *Lect. Notes in Comp. Sci.*, pages 48–58. Springer-Verlag, 1991.

[Con92] A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

[CY88] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Proc. 29th IEEE Symp. Found. of Comp. Sci.*, pages 338–354, 1988.

[dAHK98] L. de Alfaro, T.A. Henzinger, and O. Kupferman. Concurrent reachability games. Technical Report UCB/ERL M98/33, University of California at Berkeley, 1998.

[dA97] L. de Alfaro. *Formal verification of probabilistic systems*. PhD thesis, Stanford University, 1997. Technical Report STAN-CS-TR-98-1601.

[Der70] C. Derman. *Finite State Markovian Decision Processes*. Acedemic Press, 1970.

[Eve57] H. Everett. Recursive games. In *Contributions to the Theory of Games III*, volume 39 of *Annals of Mathematical Studies*, pages 47–78, 1957.

[FV97] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

[HSP83] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Trans. Prog. Lang. Sys.*, 5(3):356–380, July 1983.

[Imm81] N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384–406, 1981.

[Jon75] N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–75, 1975.

[KS81] P.R. Kumar and T.H. Shiau. Existence of value and randomized strategies in zero-sum discrete-time stochastic dynamic games. *SIAM J. Control and Optimization*, 19(5):617–634, 1981.

[KSK66] J.G. Kemeny, J.L. Snell, and A.W. Knapp. *Denumerable Markov Chains*. D. Van Nostrand Company, 1966.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

[vN28] J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Math. Annal*, 100:295–320, 1928.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, Lecture Notes in Computer Science, volume 137, pages 337–351. Springer-Verlag, 1981.

[RF91] T.E.S. Raghavan and J.A. Filar. Algorithms for stochastic games — a survey. *ZOR — Methods and Models of Op. Res.*, 35:437–472, 1991.

[Tho95] W. Thomas. On the synthesis of strategies in infinite games. In *Proc. of 12th Annual Symp. on Theor. Asp. of Comp. Sci.*, volume 900 of *Lect. Notes in Comp. Sci.*, pages 1–13. Springer-Verlag, 1995.

[TV87] F. Thuijsman and O.J. Vrieze. The bad match, a total reward stochastic game. *Operations Research Spektrum*, 9:93–99, 1987.

[TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–81, 1968.

[Var85] M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. Proc. 26th IEEE Symp. on Foundations of Computer Science, pages 327–338, 1985.

[Var95] M.Y. Vardi. Infinite games against nature. Unpublished manuscript, 1995.

[VTRF83a] O.J. Vrieze, S.H. Tijs, T.E.S. Raghavan, and J.A. Filar. A finite algorithm for the switching controller stochastic game. *OR Spectrum*, 5:15–24, 1983.

[Yan98] M. Yannakakis. Personal communication, 1998.