Play to Test[⋆]

Andreas Blass[⋆⋆1], Yuri Gurevich[2], Lev Nachmanson[2], and Margus Veanes[2]

[1] University of Michigan, Ann Arbor, MI, USA
ablass@umich.edu
[2] Microsoft Research, Redmond, WA, USA
gurevich,levnach,margus@microsoft.com

Abstract. Testing tasks can be viewed (and organized!) as games against nature. We study reachability games in the context of testing. Such games are ubiquitous. A single industrial test suite may involve many instances of a reachability game. Hence the importance of optimal or near optimal strategies for reachability games. One can use linear programming or the value iteration method of Markov decision process theory to find optimal strategies. Both methods have been implemented in an industrial model-based testing tool, Spec Explorer, developed at Microsoft Research.

1 Introduction

If you think of playful activities, software testing may be not the first thing that comes to your mind, but it is useful to see software testing as a game that the tester plays with the implementation under test (IUT). We are not the first to see software testing as a game [2] but our experience with building testing tools at Microsoft leads us to a particular framework.

An industrial tester typically writes an elaborate test harness around the IUT and provides an application program interface (API) for the interaction with the IUT. You can think of the API sitting between the tester and the IUT. It is symmetric in the sense that it specifies the methods that the tester can use to influence IUT and the methods that the IUT can use to pass information to the tester. From tester's point of view, the first methods are *controllable actions* and the second methods are *observable actions*.

The full state of the IUT is hidden from the tester. Instead, the tester has a model of the IUT's behavior. A model state is given by the values of the model variables which can be changed by means of actions whether controllable or observable. But this is not the whole story. In addition, there is an implicit division of the states into *active* and *passive*; in other words there is an implicit Boolean state variable "the state is active". The initial state is active but, whenever the model makes a transition to a target state where an observable action is enabled, the target state is passive; the target state is active otherwise. At a passive state, the tester waits for an observable action.

[⋆] Microsoft Research Technical Report MSR-TR-2005-04; January 29, 2005; revised April 5, 2005.

[⋆⋆] Much of the research reported here was done while the first author was a visiting researcher at Microsoft Research.

If nothing happens within a state-dependent timeout, the tester interprets the timeout itself as a default observable action which changes the passive state into an active state with the same values of the explicit variables. At an active state the tester applies one of the enabled controllable actions. Some active states are *final*; this is determined by a predicate on state variables. The tester has an option of finishing the game whenever the state is final.

We presume here that the model has already been tested for correctness. We are testing IUT for the conformance to the model. Here are some examples of how you detect nonconformance. Suppose that the model is in a passive state s. If only actions a, b are enabled in s but you observe an action c, different from a and b, then you witness a violation of the conformance relation. If the model tells you that any non-timeout action enabled in s returns a positive integer but the IUT throws an exception or returns -1, then, again, you have discovered a conformance violation. This kind of conformance relation is close to the one studied by de Alfaro [9].

In a given passive state the next observable action and its result are not determined uniquely in general. What are the possible sources of the apparent nondeterminism? One possible source is that the IUT interacts with the outside world in a way that is hidden from the tester. For example, it is in many cases not desirable for the tester to control the scheduling of the execution threads of a multithreaded IUT; it may be even impossible in the case of a distributed IUT. Another possible source of nondeterminism is that the model state is more abstract than the IUT state. For example, the model might use a set to represent a collection of elements that in reality is ordered in one way or another in the IUT.

The group on Foundations of Software Engineering at Microsoft Research developed a tool, called Spec Explorer, for writing, exploring, and validating software models and for model-based testing of software. Typically the model is more abstract and more compact than the IUT; nevertheless its state space can be infinite or very large. It is desirable to have a finite state space of a size that allows one to explore the state space. To this end, Spec Explorer enables the tester to generate a finite but representative set of parameters for the methods. Also, the tester can indicate a collection of predicates and other functions with finite (and typically small) domains and then follow only the values of these functions during the exploration of the model [11]. These and other ways of reducing the state space are part of a cohesive finite state machine (FSM) generation algorithm implemented in the Spec Explorer tool; the details fall outside the scope of this paper. The tool is briefly described in [12]; a better description of it is in preparation. The tool is available from [1].

The game that we are describing is an example of so-called games against nature which is a classical area in optimization and decision making under uncertainty going back all the way to von Neumann [22]. Only one of the two players, namely the tester, has a goal. The other player is disinterested and makes random choices. We make a common assumption that the random choices are made with respect to a known probability distribution. How do we know the probability distribution? In fact, we usually don’t. Of course symmetry considerations are useful, but typically they are insufficient to determine the probability distribution. One approximates the probability distribution by experimentation.

The tester may have various goals. Typically they are cover-and-record goals e.g. visit every state (or every state-to-state transition) and record everything that happened in the process. Here we study *reachability games* where the goal is to reach a final state. It is easy to imagine scenarios where a reachability game is of interest all by itself. But we are interested in reachability games primarily because they are important auxiliary games. In an industrial setting, the tester often runs test suites that consist of great many test segments. The state where one test segment naturally ends may be inappropriate for starting the next segment because various shared resources have been acquired or because the state is littered with ancillary data. The shared resources should be freed and the state should be cleaned up before the segment is allowed to end. Final states are such clean states where a new segment can be initiated. And so the problem arises of arriving at one of the final states.

It is a priori possible that no final state is reachable from the natural end-state of a test segment. In such a case it would be impossible to continue a test suite. Spec Explorer avoids such unfortunate situations by pruning the FSM so that that it becomes *transient* in the following sense: from every state, at least one final state is reachable (unless IUT crashes). The pruning problem can be solved efficiently using a variation of [8, Algorithm 1] (which is currently implemented in Spec Explorer), or the improved algorithm in [7, Section 4].

The tester cannot run a great many test segments by hand. The testing activity at Microsoft gets more and more automated. The Spec Explorer tool plays an important role in the process. Now is the time to expose a simplification that we made above speaking about the tester making moves. It is a testing tool (TT) that makes moves. The tester programs a game strategy into the TT.

The reachability games are so ubiquitous that it is important to compute optimal or nearly optimal strategies for them. You compute a strategy once for a given game and then you use it over and over a great many times. Since reachability games are so important for us, we research them from different angles.

In Section 2, reachability games are formulated, analyzed and solved by means of linear programming. We associate a state dependent cost with each action. The optimal strategy minimizes the expected total cost which is the sum of the costs incurred during the execution. The main result of the section is related to [8, Theorem 9]; see Section 4 in this connection. In Section 3 we observe that a reachability game can be viewed as a negative Markov decision process with infinite horizon [20]; the stopping condition is the first arrival at a final state. This allows one to solve any reachability problem using the well known value iteration method. Theorem 7.3.10 in [20] guarantees the convergence. Finally, Section 4 is devoted to related work.

Often the value iteration method works faster than the simplex method, but linear programming has its advantages and sheds some more light on the problem. In general, the applicability of one method does not imply the applicability of the other. In particular and somewhat surprisingly, linear programming is not applicable to negative Markov decision problems in general according to [20, page 324].

Spec Explorer makes use of both, linear programming and value iteration, to generate strategies. Recall that strategy generation happens upon completion of FSM generation and a possible elimination of states from which no final state is reachable. The step of getting from the model program to a particular test graph is illustrated with the following example.

Example: Chat Session We illustrate here how to model a simple reactive system. This example is written in the AsmL specification language [15]. The chat session lets a client post messages for the other clients. The state of the system is given by the tuple `(clients,queue,recipients)`, where `clients` is the set of all clients of the session, `queue` is the queue of pending (sender,text) messages, and `recipients` is the set of remaining recipients of the first message in the queue called the *current* message.

```
var clients as Set of Integer
var queue as Seq of (Integer,String)
var recipients as Set of Integer
```

Posting a message is a *controllable* action. The action is enabled if the Boolean expression given by the `require` clause holds. Notice that the second conjunct of the enabling condition is trivially true if the queue is empty.

```
Post(sender as Integer, text as String)
  require sender in clients and
          forall msg in queue holds msg.First <> sender
  if queue.IsEmpty then recipients := clients - {sender}
  queue := queue + [(sender,text)]
```

Delivery of a message is an *observable* action. The current message must be delivered to all the clients other than the sender. Upon each delivery, the corresponding receiver is removed from the set of recipients. If there are no more recipients for the current message, the queue is popped and the next message (if any) becomes the current one. In other words, the specification prescribes that the current message must be delivered to all the recipients before the remainder of the queue is processed.

```
Deliver(msg as (Integer, String), recipient as Integer)
  require not queue.IsEmpty and then
          queue.Head = msg and recipient in recipients
  if recipients.Size = 1 then
    if queue.Length = 1 then recipients := {}
    else recipients := clients - {queue.Tail.Head.First}
    queue := queue.Tail
  else recipients := recipients - {recipient}
```

A good example of a natural finality condition in this case is `queue.IsEmpty`, specifying any state where there are no pending messages to be delivered.

If we configure the chat session example in Spec Explorer so that the initial state is $(\{0,1\},[],\{\})$ with two clients 0 and 1, where client 0 only posts “hi”, and client 1 only posts “bye”, then we get the test graph illustrated in Figure 1. The initial state is s_1, and that is also the only final state with the above finality condition.

Finally let us note that this paper addresses a relatively easy case when all the states are known in advance. The more challenging (and important) case is on-the-fly testing where new states are discovered as you go. In a sense, this paper is a warmup before tackling on-the-fly testing.

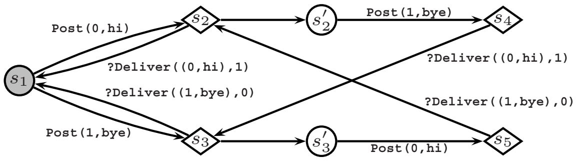

Fig. 1. Sample test graph generated by Spec Explorer from the chat model; diamonds represent passive states; ovals represent active states; links to s_2' and s_3' represent transitions to active mode; observable actions are prefixed by a question mark.

2 Reachability games and linear programming

We use a modification of the definition of a test graph in [19] to describe nondeterministic systems. A *test graph* G has a set V of *vertices* or *states* and a set E of directed *edges* or *transitions*. The set of states splits into three disjoint subsets: the set V^{a} of *active* states, the set V^{p} of *passive* states, and the set V^{g} of *goal* states. Without loss of generality, we may assume that V^{g} consists of a single goal state g such that no edge exits from g; the reduction to this special case is obvious.

There is a *probability function* p mapping edges exiting from passive nodes to positive real numbers such that, for every $u \in V^{\mathrm{p}}$,

$$\sum_{(u,v)\in E} p(u,v) = 1. \tag{1}$$

Notice that this implies that for every passive state there is at least one edge starting from it, and we assume the same for active states. Finally, there is a *cost function* $\boldsymbol{c}$ from edges to positive reals. One can think about the cost of an edge as, for example, the time for IUT to execute the corresponding function call. Formally, we denote by G the tuple

$$(V, E, V^{\mathrm{a}}, V^{\mathrm{p}}, g, p, \boldsymbol{c}).$$

We assume also that for all $u, v \in V$ there is at most one edge from u to v. (This is not necessarily the case in applications; the appropriate reduction is given in Section 2.3.) Thus $E \subset V \times V$. It is convenient to extend the cost function to $V \times V$ by setting $\boldsymbol{c}(u,v) = 0$ for all $(u,v) \notin E$.

2.1 Reachability game

Let $G = (V, E, V^{\mathrm{a}}, V^{\mathrm{p}}, g, p, \boldsymbol{c})$ be a test graph and u a vertex of it. The *reachability game* $R(u)$ over G is played by a testing tool (TT) and an implementation under test (IUT). The vertices of G are the states of $R(u)$, and u is the initial state. The current state of the game is indicated by a marker. Initially the marker is at u. If the current state v is active then TT moves the marker from v along one of graph edges. If the current state v is passive then IUT picks an edge (v,w) with probability $p(v,w)$ and moves the marker from v to w. TT wins if the marker reaches g. With every transition e the cost $\boldsymbol{c}(e)$ is added to the total game cost.

A *strategy* for (the player TT in) G is a function S from V^{a} to V such that $(v, S(v)) \in E$ for every $v \in V^{\mathrm{a}}$. Let $R(u, S)$ be the subgame of $R(u)$ when TT plays according to S.

We would like to evaluate strategies and compare them. To this end, for every strategy S, let $M_S[v]$ be the expected cost of the game $R(v, S)$. Of course, the expected cost may diverge, in which case we set $M_S[v] = \infty$. We say that M_S is *defined* if $M_S[v] < \infty$ for all v. If, for example, c reflects the durations of transition executions then M_S reflects the expected game duration. The expected cost function satisfies the following equations.

$$\begin{aligned} &M_S[g] = 0 \\ &M_S[u] = c(u, S(u)) + M_S[S(u)] \quad \text{for } u \in V^{\mathrm{a}} \\ &M_S[u] = \sum_{(u,v)\in E} \{p(u, v)(c(u, v) + M_S[v])\} \quad \text{for } u \in V^{\mathrm{p}} \end{aligned} \tag{2}$$

We call a strategy S *optimal* if $M_S[v] \le M_{S'}[v]$ for every strategy S' and every $v \in V$, or, more concisely, if $M_S \le M_{S'}$ for every strategy S'. How can we construct an optimal strategy? Our plan is to show that the cost vector M of an optimal strategy is an optimal solution of a certain linear programming problem. This will allow us to find such an M. Then we will define a strategy S such that, for all active states u,

$$c(u, S(u)) + M[S(u)] = \min_{(u,v)\in E}\{c(u, v) + M[v]\}. \tag{3}$$

We will define transient test graph and prove that the strategy S is optimal when the test graph is transient.

Let us suppose from here on that the set V of states is $\{0, 1, ..., n-1\}$ and that the goal state $g = 0$. Consider a strategy S over G. We denote by P_S the following $n \times n$ matrix of non-negative reals:

$$P_S[u, v] = \begin{cases} p(u, v), & \text{if } u \in V^{\mathrm{p}} \text{ and } (u, v) \in E\text{ ;} \\ 1, & \text{if } u \in V^{\mathrm{a}} \text{ and } v = S(u) \text{ or if } u = v = 0; \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

$P_S[u, v]$ is the probability of the move (u, v) when the game $R(u, S)$ is in state u, except that there is no move from state 0. (We could have added an edge $(0, 0)$ in which case there would be no exception.) So P_S is a *probability matrix* (also called a *stochastic matrix*) [10] since all entries are nonnegative and each row sum equals 1.

A strategy S is called *reasonable* if for every $v \in V$ there exists a number k such the probability to reach the goal state within at most k steps in the game $R(v, S)$ is positive. Intuitively, a reasonable strategy may be not optimal but eventually it has some chance of leading the player to the goal state.

Lemma 1. *A strategy S is reasonable for a test graph G if and only if, for some k, there exists, for each vertex v, a path P_v of length at most k from v to the goal state such that, whenever an active vertex w occurs in P_v, then the next vertex in P_v is $S(w)$.*

Proof. The "only if" half is obvious, because a play in $R(v,S)$ that reaches the goal state in at most k steps traces out a path P_v of the required sort. For the "if" half, recall that all the edges of G have positive probabilities. Thus, P_v has a positive probability of being traced by a play of $R(v,S)$, and so this game has a positive probability of reaching g in at most k steps. □

A nonempty subset U of V is *closed* if the game never leaves U after starting at any vertex in U. If S is reasonable, no subset U of $V-\{g\}$ is closed under the game $R(u,S)$, for any $u \in U$. This property is used to establish the following facts.

We let P'_S denote the *minor* of P_S obtained by crossing out from P_S row 0 and column 0.

Lemma 2. *Let S be a reasonable strategy. Then*

$$\lim_{k\to\infty} {P'_S}^k = 0 \tag{5}$$

and

$$\sum_{k=0}^{\infty} {P'_S}^k = (I - P'_S)^{-1}. \tag{6}$$

Proof. This follows from [10, Proposition M.3] but we present the proof here for the sake of completeness. The top row of P_S has 1 as the first element followed by a sequence of zeroes. Therefore for each $k \geq 0$ the power ${P'_S}^k$ equals the minor of ${P_S}^k$ obtained by removal of row 0 and column 0. The element ${P_S}^k[u,v]$ is the probability to get from u to v in exactly k moves. Since the strategy S is reasonable, for every u there exists an integer k such that ${P_S}^k[u,0] > 0$ and therefore the sum of the u-th row of ${P'_S}^k$ is < 1. The same is true for ${P'_S}^n$ for any $n > k$; that can be proved by induction using the fact that the only transition from 0 is to 0. Therefore there exists an integer k and a positive real number $\alpha < 1$ such that the sum of every row of ${P'_S}^k$ is at most α. But then ${P'_S}^l$ has row sums at most α^m for any $m > 0$ and $l \geq mk$ which proves the convergence in (5) and also existence of the sum $\sum_{i=0}^{\infty} {P'_S}^i$.

Now we can prove (6). For any $j > 0$ we have the equality

$$(I - P'_S)\sum_{i=0}^{j} {P'_S}^i = (\sum_{i=0}^{j} {P'_S}^i)(I - P'_S) = I - {P'_S}^{j+1}.$$

Upon taking the limit as $j \to \infty$ we get

$$(I - P'_S)\sum_{i=0}^{\infty} {P'_S}^i = (\sum_{i=0}^{\infty} {P'_S}^i)(I - P'_S) = I,$$

which proves (6). □

Reasonable strategies can be characterized in terms of their cost vectors as follows.

Lemma 3. *A strategy S is reasonable if and only if M_S is defined. Moreover, if M_S is defined then $M'_S = (I - P'_S)^{-1} b'_S$ where M'_S and b'_S are the projections to the set $V - \{0\}$ of the expected cost vector M_S and the "immediate cost" vector b_S defined by*

$$b_S[u] = \sum_{v \in V} P_S[u,v] c(u,v) \quad (\forall u \in V).$$

Proof. We first prove the direction ($\Rightarrow$). Assume S is a reasonable strategy and let us show that M_S exists. By using reasonableness of S we can find a natural number k such that for any vertex $v \in V$ the probability to finish the game $R(v,S)$ within k steps is positive and greater than some positive real number a. Let b be the largest cost of any edge. Let us consider an arbitrary $v \in V - \{0\}$. For every natural number m let us denote by A_m the event that the game $R(v,S)$ ends within km steps but not within $k(m-1)$ steps, and for every integer $l \geq 0$ let B_l be the event that the game does not end within kl steps. Using P to denote probability of events, we obviously have $P(A_m) \leq P(B_{m-1})$ for $m > 0$, and $P(B_l) \leq (1-a)^l$ for $l \geq 0$. In particular, the probability of the intersection of all the B_l's is 0, and so the A_m's constitute a partition of almost all of the possible plays of the game. Now we can estimate $M_S[v]$ from above as follows;

$$M_S[v] \leq \sum_{m=1}^{\infty} P(A_m) kmb \leq kb \sum_{l=0}^{\infty} P(B_l)(l+1) \leq kb \sum_{l=0}^{\infty} (l+1)(1-a)^l = kb/a^2,$$

which is finite. So we have proved that $M_S[v]$ is defined for every $v \in V - \{0\}$ and, of course, $M_S[0] = 0$ so M_S is defined. This is enough for the proof of ($\Rightarrow$) but we need more information about M_S.

We now prove the direction ($\Leftarrow$) by contraposition. Assume that S is not reasonable. We need to show that, for some $v \in V$, $M_S[v] = \infty$. Indeed, let v be such a vertex that for every $k > 0$ the probability to reach the goal vertex in k moves in the game $R(v,S)$ is zero. Let $A \subset V$ be the set of vertices that can be reached in the game $R(v,S)$. Let $\beta = \min_{u,w \in A, (u,w) \in E} c(u,w)$. Then $\beta > 0$ and any run of the game of length l will have cost at least βl. Now starting from v all runs are infinite, hence have infinite cost. So $M_S[v]$ is undefined which is a contradiction.

Finally, we check the formula $M'_S = (I - P'_S)^{-1} b'_S$. Equation (2) tells us that, for each $u \in V - \{0\}$,

$$M'_S[u] = \sum_{(u,v) \in E} \big(P_S[u,v](c(u,v) + M_S[v]) \big).$$

The $c(u,v)$ terms in this sum give $b'_S[u]$. The remaining terms give

$$\sum_{(u,v) \in E} P_S[u,v] M_S[v] = \sum_{v \in V - \{0\}} P'_S[u,v] M'_S[v],$$

where the restriction to $v \neq 0$ is justified because $M_S[0] = 0$. Thus, we have in matrix form

$$M'_S = b'_S + P'_S M'_S.$$

Since Lemma 2 assures us that $I - P'_S$ is invertible, we can algebraically transform the last equation to the desired $M'_S = (I - P'_S)^{-1} b'_S$. □

A vertex v of a test graph is called *transient* if the goal state is reachable from v. We say that a test graph is *transient* if all its non-goal vertices are transient. There is a close connection between transient graphs and reasonable strategies.

Lemma 4. *A test graph is transient if and only if it has a reasonable strategy.*

Proof. Let G be a transient test graph. We construct a reasonable strategy T. Using transience of G, we fix, for every $v \in V$, a shortest path P_v to 0 (shortest in terms of number of edges). We can arrange also that if w is a state that occurs in P_v then P_w is a suffix of P_v. For each state v define $T(v)$ as the immediate successor of v in P_v. We show that T is a reasonable strategy. Let v be a vertex in V. We need to show that there exists a number k such that the probability to reach the goal state within at most k steps in the game $R(v, T)$ is positive. Let P_v be the sequence $(v_0, v_1, \ldots, v_k)$, where k is the length of P_v, $v_0 = v$, and $v_k = 0$. If v_i is active, then $v_{i+1} = T(v_i)$ and the probability p_i of going from v_i to v_{i+1} is 1. If v_i is passive then the edge (v_i, v_{i+1}) in E has probability $p_i > 0$. The probability that $R(v, T)$ follows the sequence P_v is the product of all the p_i, so it is positive. A fortiori, the probability that it gets to the goal state in k steps is positive.

To prove the other direction assume that G has a reasonable strategy S. Then for each $v \in V$ the game $R(v, S)$ eventually moves the marker to the goal vertex thus creating a path from v to g. □

In practice, the probabilities and costs in a test graph may not be known exactly. It is therefore important to know that, as long as the graph is transient, the optimal cost is robust, in the sense that it is not wildly sensitive to small changes in the probabilities and costs. This sort of robustness is, of course, just continuity, which the next lemma establishes.

Lemma 5. *For transient test graphs, the optimal cost vector M is a continuous function of the costs $c(u, v)$ and the probabilities $p(u, v)$.*

Proof. Throughout this proof, “continuous” means as a function of the costs $c(u, v)$ and the probabilities $p(u, v)$.

Temporarily consider any fixed, reasonable strategy S for the given test graph. Thanks to Lemma 1, S remains reasonable when we modify the probabilities (and costs) as long as they remain positive.

The formula for b_S in Lemma 3 shows that this vector is continuous. So is the matrix $I - P'_S$. Since the entries in the inverse of a matrix are, by Cramer’s rule, rational functions of the entries of the matrix itself, we can infer the continuity of $(I - P'_S)^{-1}$ and therefore, by Lemma 3, the continuity of M'_S. Since the only component of M_S that isn’t in M'_S is 0, we have shown that M_S is continuous.

Now un-fix S. The optimal cost vector M is simply the componentwise minimum of the M_S, as S ranges over the finite set of reasonable strategies. Since the minimum of finitely many continuous, real-valued functions is continuous, the proof of the lemma is complete. □

Of course, we cannot expect the optimal strategy to be a continuous function of the costs and probabilities. A continuous function from the connected space of cost-and-probability functions to the finite space of strategies would be constant, and we certainly cannot expect a single strategy to be optimal independently of the costs and probabilities. Nevertheless, the optimal strategies are robust in the following sense.

Suppose S is optimal for a given test graph, and let an arbitrary $\varepsilon > 0$ be given. Then after any sufficiently small modification of the costs and probabilities, S will still be within ε of optimal. Indeed, the continuity, established in the proof of Lemma 5, of the function M_S and of its competitors $M_{S'}$ arising from other strategies, ensures that, if we modify the costs and probabilities by a sufficiently small amount, then no component M_S will increase by more than $\varepsilon/2$ and no component of any $M_{S'}$ will decrease by more than $\varepsilon/2$. Since $M_S \leq M_{S'}$ before the modification, it follows that $M_S \leq M_{S'} + \varepsilon$ afterward.

A similar argument shows that, if S is strictly optimal for a test graph G, in the sense that any other S' has all components of $M_{S'}$ strictly larger than the corresponding components of M_S, then S remains strictly optimal when the costs and probabilities are modified sufficiently slightly. Just apply the argument above, with ε smaller than the minimum difference between corresponding components of M_S and any $M_{S'}$.

2.2 Linear programming

Ultimately, our goal is to compute optimal strategies for a given test graph G. We start by formulating the properties of the expected cost vector M as the following optimization problem. Let $\mathbf{d}$ be the constant row vector $(1, ..., 1)$ of length $|V| = n$.

LP: Maximize $\mathbf{d}M$, i.e. $\sum_{u \in V} M[u]$, subject to $M \geq 0$ and

$$\begin{cases} M[0] \leq 0 \\ M[u] \leq c(u,v) + M[v] \quad \text{for } u \in V^{\mathrm{a}} \text{ and } (u,v) \in E \\ M[u] \leq \sum_{(u,v)\in E}\{p(u,v)(c(u,v) + M[v])\} \quad \text{for } u \in V^{\mathrm{p}} \end{cases}$$

Let us denote the inequalities above by the family $\{\rho_i\}_{i \in \{0\} \cup \{(u,v) \in E : u \in V^{\mathrm{a}}\} \cup V^{\mathrm{p}}}$. We say that a solution M of LP is *tight* for ρ_i if the left hand side and the right hand side of ρ_i are equal, i.e., there is no slackness in the solution of ρ_i. We will use the following lemma.

Lemma 6. *If LP has an optimal solution M then for all active states u there is an edge $(u, v) \in E$ such that M is tight for $\rho_{(u,v)}$, and for all passive states u, M is tight for ρ_u.*

We use the *dual problem* in the proof of Lemma 6. The inequalities LP can be written in matrix form as $AM \leq \mathbf{b}$ and we can formulate the dual optimization problem as follows.

DP: Minimize $X\mathbf{b}$, subject to $XA \geq \mathbf{d}$ and $X \geq 0$.

Let J be the following set of indices.

$$J \overset{\text{def}}{=} \{0\} \cup \{(u,v) \in E : u \in V^{\mathrm{a}}\} \cup V^{\mathrm{p}}.$$

It is convenient to order J in a fixed sequence $(i_0, i_1, \ldots, i_{k-1})$ such that $i_0 = 0$. We refer to the ordinal of $i \in J$ in this sequence by $\bar{\imath}$. The inequalities of LP can be written in normalized form as follows.

$$\rho_i \overset{\text{def}}{=} \sum_{j=0}^{n-1} A(\bar{\imath}, j) M[j] \leq \mathbf{b}(\bar{\imath}), \quad i \in J,$$

where A is a $k \times n$ matrix and $\mathbf{b}$ is a column vector of length k.

It is helpful in understanding the notions to consider a simple example first.

Example 1. Consider the test graph G in Figure 3. The LP associated to G has the following inequalities:

$$\begin{aligned}
&M[0] \leq 0 \\
&M[1] \leq c(1,2) + M[2] \\
&M[1] \leq c(1,0) + M[0] \\
&M[2] \leq \tfrac{1}{3}(c(2,1) + M[1]) + \tfrac{2}{3}(c(2,0) + M[0])
\end{aligned}$$

The LP in matrix form looks like:

$$\overbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 0 \\ -\frac{2}{3} & -\frac{1}{3} & 1 \end{pmatrix}}^{A} \overbrace{\begin{pmatrix} M[0] \\ M[1] \\ M[2] \end{pmatrix}}^{M} \leq \overbrace{\begin{pmatrix} 0 \\ c(1,2) \\ c(1,0) \\ \frac{2}{3}c(2,0) + \frac{1}{3}c(2,1) \end{pmatrix}}^{\mathbf{b}}$$

The dual problem is to minimize $X\mathbf{b}$, subject to $X \geq 0$ and

$$\overbrace{\begin{pmatrix} x_0 & x_{(1,2)} & x_{(1,0)} & x_2 \end{pmatrix}}^{X} A \geq \overbrace{\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}}^{\mathbf{d}}$$

We can write it in the form of inequalities:

$$\begin{aligned}
x_0 &\geq 1 + x_{(1,0)} + \tfrac{2}{3}x_2 \\
x_{(1,2)} + x_{(1,0)} &\geq 1 + \tfrac{1}{3}x_2 \\
x_2 &\geq 1 + x_{(1,2)}
\end{aligned}$$

Intuitively, x_e can be understood as a flow where the strategy follows an edge with a greater flow.

Proof (Lemma 6). Assume LP has an optimal solution M. By the Duality Theorem (see e.g. [10, Proposition G.8]), DP has an optimal solution X. By expanding $XA \geq \mathbf{d}$ we get for each active state u an inequality of the form

$$\sum_{(u,v) \in E} X(\overline{(u,v)}) - \sum_{i \in J} a_{u,i} X(\bar{\imath}) \geq 1, \quad \text{where all } a_{u,i} \geq 0, \tag{7}$$

and for each passive state i we get an inequality of the form

$$X(\bar{\imath}) - \sum_{j \in J} a_{i,j} X(\bar{\jmath}) \geq 1, \text{where all } a_{i,j} \geq 0. \tag{8}$$

Let u be an active state. From (7) follows that some $X(\overline{(u,v)}) > 0$, which by Complementary Slackness [10, Proposition G.9] implies that M is tight for the corresponding inequality $\rho_{(u,v)}$.

Let i be a passive state. From (8) follows that $X(\bar{\imath}) > 0$, which by Complementary Slackness implies that M is tight for the corresponding inequality ρ_i. □

The following characterization of transient test graphs is the main result of this section.

Theorem 1. *The following statements are equivalent for all test graphs G.*

(a) G is transient.
(b) G has a reasonable strategy.
(c) LP for G has a unique optimal solution M. Moreover, $M = M_S$ for some strategy S and the strategy S is optimal.

Proof. (a) ⇔ (b) is Lemma 4. We prove (b) ⇒ (c). Assume G has a reasonable strategy S. To see that LP is feasible, note that $M = 0$ is a feasible solution of LP. By Lemma 3, we know that M_S is defined. We show first that any feasible solution M of LP is bounded by M_S, i.e.,

$$M \leq M_S. \tag{9}$$

Let M be any feasible solution of LP. Let M' be the projection of M onto the set $V - \{0\}$; let P_S and P'_S be defined as above. LP ensures that

$$M'[u] \leq \sum_{v \in V} P'_S[u,v](c(u,v) + M[v]) \quad (\forall u \in V').$$

The sum of the terms $P'_S[u,v]c(u,v)$ here is $b'_S[u]$ as defined in Lemma 3. In the sum of the remaining terms $P'_S[u,v]M[v]$, we can restrict v to range over V' because $M[0] \leq 0$. Thus, we get the matrix inequality $M' \leq b'_S + P'_S M'$, which is equivalent to $(I - P'_S)M' \leq b'_S$. By Lemma 2 the inverse matrix $(I - P'_S)^{-1}$ exists and all its entries are non-negative. So our inequality will be preserved if we multiply it by $(I - P'_S)^{-1}$ on the left. The result is $M' \leq (I - P'_S)^{-1} b'_S$. Thus (9) follows by using Lemma 3 since $M_S[0] = M[0] = 0$.

Since LP is feasible and bounded it has an optimal solution M^*. Lemma 6 ensures that there exists a strategy S^* such that

$$M^*[u] = c(u, S^*(u)) + M^*[S^*(u)] \text{ for } u \in V^{\mathrm{a}},$$
$$M^*[u] = \sum_{(u,v) \in E} \{p(u,v)(c(u,v) + M^*[v])\} \text{ for } u \in V^{\mathrm{p}},$$

which by (2) implies that $M^* = M_{S^*}$. By (9) it follows that $M_{S^*} \leq M_S$ for any strategy S, and hence S^* is optimal.

To see that the optimal solution M^* is unique, suppose M^+ were another optimal solution to LP. As in the preceding paragraph, it would give us a strategy S^+ such that $M^+ = M_{S^+}$ and S^+ is optimal. As both S^* and S^+ are optimal, each of M_{S^*} and M_{S^+} is $\leq$ the other. So they are equal, and this means that $M^* = M^+$.

Finally, note that (c) ⇒ (b) by Lemma 3 since M_S is defined. □

Now we presume that the test graph is transient and show how to construct an optimal strategy. By applying Theorem 1 and solving LP, find the cost vector M of some optimal strategy O. In our notation, $M = M_O$. Construct strategy S so that equation (3) is satisfied for every active state u.

Proposition 1. *The constructed strategy S is optimal.*

Proof. First we check that S is reasonable. Let G_S be the graph obtained from G by removing all edges (u, v) such that u is active and $v \neq S(u)$. It is easy to see that S is unreasonable if and only if G_S has a closed vertex set that does not contain 0. By contradiction, assume that U is such a set. By Lemmas 3 and 4 the optimal strategy O is reasonable. Choose a vertex $u \in U$ such that $M[u] = \min_{u' \in U} M[u']$, and let $v = S(u)$. Since U is a closed subset of G_S, $v \in U$. By the construction of S, $c(u, v) + M[v] \leq c(u, O(u)) + M[O(u)] = M[u]$. As $c(u, v) > 0$, we get $M[v] < M[u]$, which contradicts our choice of u.

Second we prove that $M_S = M$ and so M is optimal. It suffices to prove that $M'_S \leq M'$ where $'$, as before, signifies the projection to $V - \{0\}$. By Lemma 2, P'_S is invertible. By Lemma 3, $M'_S = (I - P'_S)^{-1} b'_S$. By the construction of S, we have $b'_S + P'_S M' \leq b'_O + P'_O M' = M'$, so that $b'_S \leq (I - P'_S) M'$. By equation (6), all entries of $(I - P'_S)^{-1}$ are non-negative, so we have $(I - P'_S)^{-1} b'_S \leq M'$. Thus $M'_S \leq M'$. □

Notice that, even though an optimal strategy S yields a unique cost vector M_S, S itself is not necessarily unique. Consider for example a test graph without passive states and with edges $\{1 \xrightarrow{1} 2,\ 2 \xrightarrow{10} 0,\ 1 \xrightarrow{10} 3,\ 3 \xrightarrow{1} 0\}$ that are annotated with costs; clearly both of the two possible strategies are optimal.

2.3 Graph transformation

We made the assumption that for each two vertices in the graph there is at most one edge connecting them. Let us show that we did not lose any generality by assuming this. For an active state u and for any $v \in V$ let us choose an edge leading from u to v with the smallest cost and discard all the other edges between u and v. For a passive state u replace the set of multiple edges D between u and v with one edge e such that $p(e) = \sum_{e' \in D} p(e')$ and $c(e) = (\sum_{e' \in D} p(e') c(e'))/p(e)$. This merging of multiple edges into a single edge does not change the expected cost of one step from u. The graph modifications have the following impact on LP. With removal of the edges exiting from active states we drop the corresponding redundant inequalities. The introduction of one edge for a passive state with changed c and p functions does not change the coefficients before $M[v]$ in LP in the inequality corresponding to passive states and therefore does not change the solution of LP.

2.4 Graph compression

Every test graph is equivalent, as far as our optimization problems are concerned, to one in which no edge joins two passive vertices. The idea is to replace the edges leaving a passive vertex u in the following manner. Consider all paths emanating from u, passing through only passive vertices, but then ending at an active vertex or the goal vertex. Each such path has a probability, obtained by multiplying the probabilities of its edges, and it has a cost, obtained by adding the costs of its edges. Replace each such path by a single edge, from u to the final, active vertex in the path; give this new edge the same probability and cost that the path had. If this replacement process produces several edges joining the same pair of vertices, transform them to a single edge as in Subsection 2.3. The details of test graph compression are given in the appendix.

One may wonder if such a compression is worthwhile. The answer depends to a great extent on the topology of the test graph. It may sometimes pay off to apply the compression to certain subgraphs of the full test graph, rather than to the whole test graph. Let us illustrate a fairly common situation that arises in testing highly concurrent systems where the compression would reduce the number of states and edges. We revisit the chat model above and extend it as follows. There is an additional state variable `nClients` representing the number of clients entering in the chat session, so the state is given by the tuple `(nClients,clients,queue,recipients)`. There is a new *controllable* action `Start` that starts the entering phase of clients by updating `nClients`. There is also a new *observable* action `Enter` representing the event of a client entering the session. A client that has already entered the session cannot enter it again.

```
var nClients as Integer
Start(n as Integer)
  require nClients = 0 and n > 0
  nClients := n
Enter(c as Integer)
  require nClients > 0 and c in {0..nClients-1} - clients
  clients := clients + {c}
```

Assume also that the enabling condition (`require` clause) of the `Post` action is extended with the condition that the entering phase was started and that all clients have entered the session, i.e., `nClients > 0` **`and`** `clients.Size = nClients`. So the "posting" phase is not started until all clients have entered the session. Suppose that the initial state s_0 is $(0, \emptyset, [], \emptyset)$. By generating the FSM from the model program with 3 clients, the initial part of the test graph up to the posting phase that starts in state s_1 is illustrated in Figure 2.a. The compression of the subgraph between the states s_0 and s_1 would yield the subgraph shown in Figure 2.b with a single passive state p and a transition from p to s_1 representing the composed event of all three clients having entered the session in some order.

The effect of the compression algorithm is in some cases, such as in this example, similar to partial order reduction. Obviously, reducing the size of the test graph improves feasibility of the linear programming approach. However, for large graphs we use the value iteration algorithm, described next. Due to the effectiveness of value iteration the immediate payoff of compression is not so clear, unless compression is simple

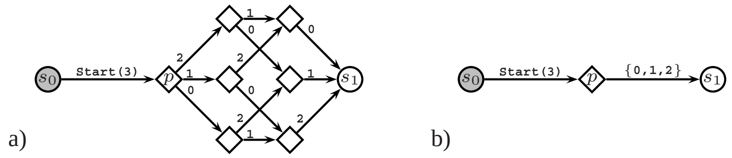

Fig. 2. a) Test subgraph obtained by exploring the extended chat model with 3 clients up to the posting phase; transitions from passive states (diamonds) are labeled by the respective client entering the session. b) Same subgraph after compression.

and the number of states is reduced by an order of magnitude. We are still investigating the practicality of compression and it is not yet implemented in the Spec Explorer tool.

3 Value iteration

Value iteration is the most widely used algorithm for solving discounted Markov decision problems (see e.g. [20]). Reachability games give rise to non-discounted Markov decision problems. Nevertheless the value iteration algorithm applies; this is a practical approach for computing strategies for transient test graphs. Test graphs, modified by inserting a zero-cost edge $(0, 0)$, correspond to a subclass of negative stationary Markov decision processes (MDPs) with an infinite horizon, where rewards are negative and thus regarded as costs, strategies are stationary, i.e. time independent, and there is no finite upper bound on the number of steps in the process. The optimization criterion for our strategies corresponds to the *expected total reward criterion*, rather than the expected discounted reward criterion used in discounted Markov decision problems.

Let $G = (V, E, V^{\mathrm{a}}, V^{\mathrm{p}}, g, p, c)$ be a test graph modified by inserting a zero-cost edge $(0, 0)$. The classical value iteration algorithm works as follows on G.

Value iteration Let $n = 0$ and let M^0 be the zero vector with coordinates V so that every $M^0[u] = 0$. Given n and M^n, we compute M^{n+1} (and then increment n):

$$M^{n+1}[u] = \begin{cases} \min_{(u,v)\in E}\{c(u, v) + M^n[v]\}, & \text{if } u \in V^{\mathrm{a}}; \\ \sum_{(u,v)\in E} p(u, v)(c(u, v) + M^n[v]), & \text{if } u \in V^{\mathrm{p}}; \\ 0, & \text{if } u = 0. \end{cases} \tag{10}$$

Value iteration for negative MDPs with the expected total reward criterion, or *negative Markov decision problems* for short, does not in general converge to an optimal solution, even if one exists. However, if there exists a strategy for which the expected cost is finite for all states [20, Assumption 7.3.1], then value iteration *does* converge for negative Markov decision problems [20, Theorem 7.3.10]. In light of lemmas 3 and 4, this implies that value iteration converges for transient test graphs. Let us make this more precise, as a corollary of Theorem 7.3.10 in [20].

Corollary 1. *Let G be a transient test graph as above. For any $\varepsilon > 0$, there exists N such that, for all $n \geq N$ and all states $u \in V$, $M^*[u] - M^n[u] < \varepsilon$, where M^* is the optimal cost vector.*

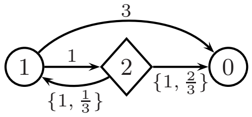

Fig. 3. Sample test graph; transitions from active states are labeled by their costs; transitions from passive states are labeled by their costs and probabilities.

The iterative process, generally speaking, does not reach a fixed point in finitely many iterations. Consider the test graph in Figure 3. It is not difficult to calculate that the infinite sequence $(M^n[1])_{n=1}^{\infty}$ computed by (10) is

$$1,\ 2,\ 2\tfrac{1}{3}\ 2\tfrac{2}{3},\ 2\tfrac{7}{9},\ 2\tfrac{8}{9},\ 2\tfrac{25}{27},\ 2\tfrac{26}{27},\ \ldots,\ 2\tfrac{3^i-2}{3^i},\ 2\tfrac{3^i-1}{3^i},\ \ldots$$

that converges to $M^*[1] = 3$.

When should we terminate the iteration? Given a cost vector M let S_M denote any strategy defined so that equation (3) is satisfied for every active state u. Further, let $S_n = S_{M^n}$. Observe that the total number of possible strategies is finite and that any non-optimal strategy occurs only finitely many time in the sequence $S_0, S_1, \ldots$. Thus, from some point on, every S_n is optimal. In reality, the desired n is typically not that large because the convergence of the computed costs towards the optimal costs is exponentially fast. For practical purposes, the iteration process halts when the additional gain is absorbed in rounding errors.

Test graphs are negative MDPs

For clarity, we define here formally a mapping from test graphs to negative MDPs. Let $G = (V, E, V^{\mathrm{a}}, V^{\mathrm{p}}, g, p, c)$ be a test graph. The set of states of the MDP is V and the set of transitions is $E \cup \{(g, g)\}$. For every state $u \in V$ define A_u as the following set of *allowable* or *enabled actions* in u:

$$A_u \stackrel{\text{def}}{=} \begin{cases} \{(u,v) : (u,v) \in E \cup \{(g,g)\}\}, & \text{if } u \in V^{\mathrm{a}} \cup \{g\}; \\ \{u\}, & \text{otherwise.} \end{cases}$$

The probability $p_a(u, v)$ of an action a taking the system from state u to state v is thus:

$$p_a(u,v) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } a = (u,v) \in E \cup \{(g,g)\} \text{ and } u \in V^{\mathrm{a}} \cup \{g\}; \\ p(u,v), & \text{if } a = u \in V^{\mathrm{p}} \text{ and } (u,v) \in E; \\ 0, & \text{otherwise.} \end{cases}$$

We can define the cost $c(a)$, or equivalently the *negative reward* $r(a) = -c(a)$, of an action a as follows. If a is an edge, its cost is already given as the cost of that edge. If a is a passive state then $c(a) = \sum_{(a,v)\in E} p(a,v)c(a,v)$. Notice that the cost of an action a that is a passive state can be defined independently of the target states of the transitions emanating from a, and this does not affect the optimization problem at hand since

$$\sum_{(u,v)\in E} \{p(u,v)(c(u,v) + M[v])\} = c(u) + \sum_{(u,v)\in E} p(u,v)M[v].$$

With these definitions, the value iteration step (10) above can be written using the standard formulation:

$$M^{n+1}[u] = \min_{a \in A_u} \left\{ c(a) + \sum_{v \in V} p_a(u,v) M^n[v] \right\}.$$

4 Related work

Extension of the FSM-based testing theory to nondeterministic and probabilistic FSMs got some attention a while ago [13, 24]. The use of games for testing is pioneered in [2]. A recent overview of using games in testing is given in [23].

An implementation that conforms to the given specification can be viewed as a refinement of the specification. In study [9], based on [3], the game view is proposed as a general framework for dealing with refining and composing systems. Models with controllable and observable actions correspond to interface automata in [9].

Model-based testing allows one to test a software system using a specification (a.k.a. model) of the system under test [5]. There are other model-based testing tools [4, 16–18, 21]. To the best of our knowledge, Spec Explorer is currently alone in supporting the game approach to testing. Our models are Abstract State Machines [14]. In Spec Explorer, the user writes models in AsmL [15] or in Spec# [6].

The technical development in Section 2 is based on classical techniques that were used to prove that linear programming works for MDPs with the discounted reward criterion [10, Theorem 2.3.1], even though we consider the total reward criterion here. For (total reward) negative Markov decision problems linear programming is not applicable in general according to [20, page 324]. The additional insight we needed is that transiency is a necessary and sufficient condition on test graphs under which linear programming works. The main result of Section 2, Theorem 1, was obtained before we learned about Alfaro's Theorem 9 [8], which shows that linear programming works for negative MDP after eliminating non-transient vertices, and that the optimal solution of the LP is unique.

One may wonder how transient stochastic games [10, Section 4.2] are related to transient test graphs. A transient stochastic game is a game between two players that will stop with probability 1 no matter which strategies are used. This condition gives rise to a proper subclass of transient test graphs where *all* strategies are reasonable. Recall that a test graph is transient if and only if there *exists* a reasonable strategy. An unreasonable strategy is for example a strategy that takes you back and forth between two active states.

Appendix: Elimination of passive states

Each test graph can be viewed as a negative MDP [Section 3]. However here our goal is to replace any test graph G by an equivalent Markov decision process (MDP) such that the MDP states are the active states of the graph. This replacement amounts to eliminating passive states from the picture, replacing them with the probability distributions that are part of an MDP.

For the most part, we follow the notation of [20]. Thus, the MDP we construct will have

- a set S of *states*,
- for each $s \in S$ a set A_s of *actions* available in state s,
- for each $s \in S$ and $a \in A_s$ a probability distribution $p(-|s,a)$ on S, and
- for each $s \in S$ and $a \in A_s$ a *cost* $c(s,a)$ (whose negative is the reward, denoted by $r(s,a)$ in [20]).

In addition, our MDP will have a *goal state* $g \in S$. A *strategy* for such an MDP is a function assigning to each non-goal state s one of the actions in A_s. Given a strategy σ and a starting state $s \in S$, the resulting random *run* of the MDP is the sequence $(s_0, s_1, \dots)$ of states obtained as follows. $s_0 = s$. If $s_n \neq g$, then s_{n+1} is chosen at random subject to the probability distribution $p(-|s_n, \sigma(s_n))$. That is, an action $a = \sigma(s_n)$ is chosen (deterministically) according to σ, and then the next state is obtained randomly from the associated distribution. If $s_n = g$, then the run ends with s_n. The *cost* of the run is the sum of the costs $c(s_n, \sigma(s_n))$ of the individual steps of the run. We shall design our MDP so that the optimization problem "Find a strategy that minimizes the expected cost of the run" is equivalent to the optimization problem for the game associated to a test graph G.

In detail, this notion of equivalence means the following for our MDP.

- S consists of the active vertices and the goal vertex of G.
- For each active vertex s of G, the actions in A_s are the outgoing edges from s in G. (We need not define A_g, since our runs always end as soon as they reach g.) Notice that what we have already said ensures that strategies in our MDP are the same as strategies for TT in G.
- If R is a run of the game on G in which TT uses strategy S, then by omitting the passive vertices from R we get a run $D(R)$ of the MDP in which we use S.
- The probability of any run R' in the MDP equals the sum of the probabilities of all the runs R, in the game on G, for which $D(R) = R'$.
- The expected cost of any strategy is the same in the MDP as in G.

The idea of the construction is roughly as follows. In any run of the game on G, consider the segments that begin at an active vertex v, go through some number (possibly zero) of passive vertices, and arrive at an active vertex v' or at the goal vertex (i.e., at a state of the MDP). Such a segment begins with TT's choice of an outgoing edge from v. The rest of the segment is out of TT's control; it consists of random choices made by the IUT. We want to view TT's choice of the edge leaving v as an action (as indicated in the description of the MDP above), and we intend to view the subsequent random choices by IUT as implementing a certain probability distribution on the possible vertices v' at which the segment could have ended. Our task is to describe this probability distribution precisely, in a manner amenable to computation, and to assign costs in such a way that the definition of equivalence is satisfied.

Before undertaking this task, we should note that it is sometimes impossible. It could happen that runs of the game on G get stuck in passive vertices and never reach another state of the MDP.

Definition 1 A *trap* in a test graph is a nonempty set T of passive vertices such that all outgoing edges from vertices in T lead to vertices in T. That is, it is a closed set consisting entirely of passive vertices

Since a trap is a special kind of closed set and cannot contain the goal vertex g of G, it is clear that a transient test graph cannot contain a trap. Since our interest is in transient graphs (as others have no optimal strategies), we assume from now on that G **has no traps**. Under this assumption, we construct the equivalent MDP as follows.

The state are the active vertices of G and the goal vertex. If s is a state of G, then A_s is the set of outgoing edges from s in G. Notice that, by these definitions, we have satisfied the first three clauses in the definition of equivalence; it remains to define the probability distributions $p(-|s,a)$ and the cost function c so as to satisfy the remaining two clauses. We begin with the probability distributions.

Fix a state $s \neq g$ and an action $a \in A_s$. So a is an edge leaving s in G, say the edge (s,w). We let $p(s'|s,a)$ be the probability that, starting from w (the head of a) and making random moves in G according to the given probabilities at passive vertices, the first non-passive vertex (i.e., the first state of the MDP) that we encounter is $s\,'$. In this definition, we regard w itself as being encountered, right at the start of the path; thus, if w happens to be a state, then $p(-|s,a)$ gives probability 1 to w.

By using this definition of $p(-|s,a)$ for all actions a, we satisfy the fourth clause in the definition of equivalence. Indeed, the probability of any single step in $R\,'$ is exactly the total probability of all the segments (from an active vertex, through passive ones, to an active one or the goal) that would, had they occurred in R, have produced that step in $D(R)$. Since different steps in any R' and different segments in any R are probabilistically independent, the desired result follows.

We must, however, verify two things about the $p(-|s,a)$'s, one to make the definition legitimate, and one to make it reasonable. To make it legitimate, we must verify that, for each non-goal state s, the probabilities assigned to its actions constitute a probability distribution, i.e, that $\sum_{s'} p(s'|a,s) = 1$. To make it reasonable, we must provide a way to compute the probability distributions. Notice that both requirements are trivial if w is not passive, as then our distribution gives probability 1 to w and 0 to all other vertices. So we need only verify the two requirements when w is passive. We attack the second requirement first.

With s,a and thus w fixed, and assuming that w is passive, consider the set X of all vertices reachable in G from w by a path consisting only of passive vertices except possibly for the last vertex in the path. Let Y be the set of passive vertices in X, and let Z be the rest of X. Thus, Z is a subset of the set of states of our MDP, and it is clear from the definition of $p(-|s,a)$ that this distribution is concentrated on Z. Let P be the

$X \times Z$ matrix whose entries are defined as follows. $P_{x,z}$ is the probability that, starting at vertex x of G and moving according to the probabilities given by the test graph G, the first non-passive vertex we encounter is z. As before, the starting point x counts as encountered. Notice that the probabilities $p(z|s,a)$ are given by one row of the matrix P, namely the row indexed by w. We shall show how to compute the whole matrix P; then we shall have in particular a computation of the desired $p(z|s,a) = P_{w,z}$.

If x happens not to be passive, then, since x is the first non-passive vertex encountered in any run that starts at x, we have $P_{x,x} = 1$ and $P_{x,z} = 0$ for all $z \neq x$. In the non-trivial case, where x is passive, we have

$$P_{x,z} = \sum_{(x,u)\in E} p(x,u) P_{u,z},$$

where E is the edge set of G and p is the probability distribution given as part of the test graph G. The right side of this equation amounts to a matrix product, but there is a discrepancy in that x ranges only over Y (since the equation is for the non-trivial case that x is a passive vertex), whereas u ranges over all of X. To write the equation in a convenient form, which also includes the trivial case that $x \in Z$, we adopt the following conventions. Let us order the set X (which indexes the rows of our matrix P) so that all elements of Y precede all elements of Z, and let us divide P (and other matrices where X is involved in the indexing) into blocks according to the partition of X into Y and Z. Thus, P is regarded as consisting of two blocks,

$$P = \begin{pmatrix} P' \\ I \end{pmatrix},$$

where I is the $Z \times Z$ identity matrix giving the trivial entries of P, while P' is the $Y \times Z$ matrix consisting of the non-trivial entries. Now we can write the equation above for the non-trivial entries and the description of the trivial entries as a single matrix equation:

$$\begin{pmatrix} P' \\ I \end{pmatrix} = \begin{pmatrix} L & M \\ O & O \end{pmatrix} \begin{pmatrix} P' \\ I \end{pmatrix} + \begin{pmatrix} O \\ I \end{pmatrix},$$

where O denotes a zero matrix (of a size appropriate for the context) and where L and M contain the probabilities from G, i.e, $L_{x,u} = p(x,u)$ when both x and u are passive, and $M_{x,u} = p(x,u)$ when x is passive but u is an active vertex or the goal. Thus, we can solve for P,

$$P = \begin{pmatrix} P' \\ I \end{pmatrix} = \left[\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} - \begin{pmatrix} L & M \\ O & O \end{pmatrix} \right]^{-1} \begin{pmatrix} O \\ I \end{pmatrix},$$

provided the inverse here exists, i.e., provided that 1 is not an eigenvalue of $\begin{pmatrix} L & M \\ O & O \end{pmatrix}$.

To see that this proviso is satisfied, we proceed by contradiction. (The following is a standard argument, but we include it here for completeness.) Suppose we had a non-zero column vector which, when multiplied on the left by our matrix $\begin{pmatrix} L & M \\ O & O \end{pmatrix}$, produces again the same column vector. Clearly, the bottom block of components of such an eigenvector must be zero, since the bottom part of the matrix is zero. The top block of our eigenvector, the part indexed by Y, is a column vector Q such that $LQ = Q$. Since $Q \neq O$, we can arrange, replacing Q with $-Q$ if necessary, that there is at least one strictly positive entry in Q. Let m be the largest of these entries, and let $Y' \subseteq Y$ be the (nonempty) set of indices where it occurs, i.e., $Y' = \{y \in Y : Q_y = m\}$. For each $y \in Y'$, the eigenvalue equation $LQ = Q$ gives us

$$m = Q_y = \sum_{u \in Y} p(y,u) Q_u \leq \sum_{u \in Y} p(y,u) m \leq$$

$$\leq \sum_{u \in X} p(y,u) m = \sum_{(y,u) \in E} p(y,u) m = m.$$

Here the first inequality comes from the definition of m as the largest entry in Q, and the second follows from $Y \subseteq X$ because all p's and m are non-negative. The next to last equality follows from the fact that, as y is a passive vertex in $Y \subseteq X$, any edge of G leaving y points to a vertex u in X, by definition of X. The last equality is just the fact that $p(y,-)$ is a probability distribution on the set of these u's. The displayed chain of equations and inequalities, beginning and ending with m, implies that both of the inequalities in the chain must actually be equalities. This means (since m and all p's are positive) that every edge (y,u) leaving y in G must point to a vertex u that is in Y (for the sake of the second inequality) and has $Q_u = m$ (for the sake of the first inequality). That is, u must be in Y'. We have shown that every edge leaving any vertex in Y' points to a vertex in Y'. This means that Y' is a trap, contrary to our assumption that G has no traps.

This contradiction concludes the proof of the formula above for P, thus showing that P and in particular the probabilities $p(z|s,a)$ needed in our MDP can be computed from the data in the test graph G by means of elementary matrix arithmetic.

We must still show that $p(-|s,a)$ is a probability distribution, i.e., that

$$1 = \sum_z p(z|s,a) = \sum_z P_{w,z}.$$

It is at least as easy to prove more, namely that every row (not just row w) of the matrix P adds up to 1. That is, we shall prove that $PJ_Z = J_X$, where J_Z denotes the column vector of 1's indexed by Z, and analogously for J_X. In view of our formula for P, what must be proved is

$$J_X = PJ_Z = \left[\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} - \begin{pmatrix} L & M \\ O & O \end{pmatrix} \right]^{-1} \begin{pmatrix} O \\ I \end{pmatrix} J_Z =$$

$$\left[\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} - \begin{pmatrix} L & M \\ O & O \end{pmatrix} \right]^{-1} \begin{pmatrix} O \\ J_Z \end{pmatrix}, \quad (11)$$

where we have multiplied out a trivial matrix product. This equation simplifies, by "cross-multiplying" (i.e., getting rid of the inverse on the right) to

$$\left[\begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} - \begin{pmatrix} L & M \\ O & O \end{pmatrix} \right] J_X = \begin{pmatrix} O \\ J_Z \end{pmatrix}.$$

The left side simplifies, since $J_X = \begin{pmatrix} J_Y \\ J_Z \end{pmatrix}$, to

$$\begin{pmatrix} J_Y \\ J_Z \end{pmatrix} - \begin{pmatrix} L & M \\ O & O \end{pmatrix} J_X,$$

and so, by transposing some terms, we bring the equation we want into the form

$$\begin{pmatrix} L & M \\ O & O \end{pmatrix} J_X = \begin{pmatrix} J_Y \\ O \end{pmatrix}.$$

The bottom block here is trivial, and the top block says simply that $p(y, -)$ is a probability distribution for every $y \in Y$. Thus, the desired equation is true, and $p(-|s, a)$ is a probability distribution.

This completes the construction of our MDP's probability distributions and the verification of their claimed properties. It remains to define a cost function that satisfies the last clause in the definition of equivalence.

Consider any action a at a state $s \neq g$ of our MDP. So a is an edge (s, w) of G. The cost $c(s, a)$ that we assign to a is to be the expectation of the random variable R_a defined using G as follows. Start with a marker at vertex s, move it along the edge a to w (incurring a cost $c(v, w)$), and then continue moving the marker, at random according to the probability distribution from G, until it encounters a non-passive vertex. (This "continue moving" would involve no moves at all if w happened not to be passive, since, as usual, w counts as encountered.) The additional moves of the marker, if any, will also incur some costs, and we let R_a be the total cost of all the moves, starting from s with move a, and ending when another non-passive vertex is encountered.

The value of R_a is almost surely finite; that is, with probability 1 the marker will encounter another non-passive vertex. This follows from our verification above that $p(-|s, a)$ is a probability distribution; $p(z|s, a)$ is the probability that the marker, moving as just described, first encounters a non-passive vertex at z, so $\sum_z p_a(z)$ is the probability that the marker encounters some non-passive vertex.

We need more, namely that the expectation of R_a is finite, so that it can be used as $c(s, a)$. We also need an efficient way to compute this expectation. We attack the latter issue first — assuming that $c(s, a)$ is finite, how can we compute it? Afterward, we shall verify the required finiteness.

Assuming finiteness for the time being, and using the notations X, Y, Z as above, define C_x, for $x \in X$, to be the expectation of the total cost S_x incurred if one starts at x and moves randomly, according to the probabilities in G, until one encounters a non-passive vertex. As usual, we consider x to be encountered, so if $x \in Z$ then $C_x = 0$. The crucial one of these costs is C_w because $c(s, a) = c(v, w) + C_w$, but we shall obtain it by computing the entire matrix (of one column) C consisting of all the C_x's.

We have already observed that $C_x = 0$ for $x \in Z$. For $x \in Y$, we split the expected cost C_x into the expected cost incurred in the first move, from x to some u, and the costs incurred subsequently, while moving from u until we encounter a non-passive vertex. The first of these costs is

$$D_x = \sum_{(x,u) \in E} p(x, u) c(x, u),$$

and the second is

$$\sum_{(x,u)\in E} p(x,u)C_u.$$

In both sums, u ranges over both passive and non-passive vertices, but in the second sum only the non-passive vertices contribute non-zero terms. Thus, we have

$$C_x = \sum_{(x,u)\in E} p(x,u)C_u + D_x,$$

or in matrix form, keeping only the non-trivial rows, the ones indexed by passive vertices,

$$C = LC + D.$$

Having already verified that $\begin{pmatrix} I & O \\ O & I \end{pmatrix} - \begin{pmatrix} L & M \\ O & O \end{pmatrix}$ is invertible, we know that $I - L$ is invertible, so we can solve for C:

$$C = (I - L)^{-1}D.$$

This solves the problem of computing C, and in particular $c(s,a) = C_w$, under the assumption that all entries of C are finite. It remains to show that this assumption is correct.

For this purpose, we consider "approximations" $C_x^{(n)}$ defined exactly like C_x except that we replace the random variables S_x with $S_x^{(n)}$ counting only the costs of the first n moves. Note that $S_x = \sup_n S_x^{(n)}$. Note also that each $C_x^{(n)}$ is obviously finite, being at most n times the maximum of the cost function c of G. We have $C_x^{(0)} = 0$ and

$$C_x^{(n+1)} = \sum_{(x,u)\in E} p(x,u)C_u^{(n)} + D_x.$$

Since all costs in G were non-negative, $S_x^{(n)}$ and $C_x^{(n)}$ are obviously increasing functions of n for each x. We shall show, by induction on n, that $C_x^{(n)} \le \big[(I-L)^{-1}D\big]_x$, i.e., that the column vectors $C^{(n)}$ are majorized componentwise by the vector C as computed in the preceding paragraph. (Of course, $C^{(n)}$ is majorized by the actual C, but we won't know that this agrees with what was computed in the preceding paragraph until we complete the present proof that the actual C is finite.) Once this is done, we can invoke the monotone convergence theorem (probably overkill, but it works) to conclude that the expectation C_x of $S_x = \sup_n S_x^{(n)}$ is finite, as required.

It remains to carry out the induction to prove that $C_x^{(n)} \le \big[(I-L)^{-1}D\big]_x$. The induction step is easy; given this inequality for n, we have, in matrix notation,

$$\begin{aligned} C^{(n+1)} = LC^{(n)} + D \le L(I-L)^{-1}D + D = \\ L(I-L)^{-1}D + (I-L)(I-L)^{-1}D = (I-L)^{-1}D. \end{aligned}$$

But we must verify the basis for the induction, namely that $(I-L)^{-1}D$ has non-negative entries. (It is, of course, obvious, that the actual expected cost vector has non-negative entries, but we're still proving that this is the same as $(I-L)^{-1}D$.)

For this, it suffices to show that all eigenvalues of L are smaller than 1 in absolute value, for then $(I-L)^{-1}$ equals the infinite sum $\sum_n L^n$, which has non-negative entries because L does. The verification that all eigenvalues of L are smaller than 1 in absolute value proceeds by contradiction and is very similar to the proof above that 1 is not an eigenvalue of $\begin{pmatrix} L & M \\ O & O \end{pmatrix}$.

Suppose there were an eigenvalue λ with $|\lambda| \geq 1$, and let Q be a non-zero column vector with $LQ = \lambda Q$. Among all the entries of Q, let m be one with $|m|$ as large as possible, let Y' be the set of indices y for which $Q_y = m$, and let Y'' be the possibly larger set of indices y with $|Q_y| = |m|$. For $y \in Y'$, we have

$$\lambda m = \lambda Q_y = (LQ)_y = \sum_{(y,u)\in E, u\in Y} p(y,u)Q_u.$$

The right side here is a weighted average of some entries Q_u of Q and possibly 0 (the latter if there are $(y,u) \in E$ with $u \in Z$), so the maximality of $|m|$ implies that this right side has absolute value at most $|m|$, and that the absolute value can equal $|m|$ only if all the terms occurring with non-zero weight in the average are equal. Thus, we have $|\lambda| = 1$ and all edges $(y,u) \in E$ have $u \in Y''$. Repeating the argument for any other entries m' of Q that have the same absolute value as m, we find that all outgoing edges from Y'' lead to vertices in Y''. This means that Y'' is a trap, contrary to our standing assumption.

References

1. Spec Explorer. URL:http://research.microsoft.com/specexplorer, released January 2005.
2. R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *Proc. 27th Ann. ACM Symp. Theory of Computing*, pages 363–372, 1995.
3. R. Alur, T. A. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *Proceedings of the Ninth International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *LNCS*, pages 163–178. Springer, 1998.
4. C. Artho, D. Drusinsky, A. Goldberg, K. Havelund, M. Lowry, C. Pasareanu, G. Rosu, and W. Visser. Experiments with test case generation and runtime analysis. In Börger, Gargantini, and Riccobene, editors, *Abstract State Machines 2003*, volume 2589 of *LNCS*, pages 87–107. Springer, 2003.
5. M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes. Towards a tool environment for model-based testing with AsmL. In Petrenko and Ulrich, editors, *Formal Approaches to Software Testing, FATES 2003*, volume 2931 of *LNCS*, pages 264–280. Springer, 2003.
6. M. Barnett, R. Leino, and W. Schulte. The Spec# programming system. In M. Huisman, editor, *Cassis International Workshop, Marseille*, LNCS. Springer, 2004.
7. K. Chatterjee, M. Jurdziński, and T. Henzinger. Simple stochastic parity games. In *CSL 03: Computer Science Logic*, Lecture Notes in Computer Science 2803, pages 100–113. Springer-Verlag, 2003.
8. L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *International Conference on Concurrency Theory*, volume 1664 of *LNCS*, pages 66–81. Springer, 1999.
9. L. de Alfaro. Game models for open systems. In N. Dershowitz, editor, *Verification: Theory and Practice: Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *LNCS*, pages 269 – 289. Springer, 2004.
10. J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., 1996.
11. W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. In *ISSTA'02*, volume 27 of *Software Engineering Notes*, pages 112–122. ACM, 2002.
12. W. Grieskamp, N. Tillmann, and M. Veanes. Instrumenting scenarios in a model-driven development environment. *Information and Software Technology*, 46(15):1027–1036, December 2004.
13. S. Gujiwara and G. V. Bochman. Testing non-deterministic state machines with fault-coverage. In J. Kroon, R. Heijunk, and E. Brinksma, editors, *Protocol Test Systems*, pages 363–372, 1992.
14. Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
15. Y. Gurevich, B. Rossman, and W. Schulte. Semantic essence of AsmL. *Theoretical Computer Science*, 2005. To appear in special issue dedicated to FMCO 2003, preliminary version available as Microsoft Research Technical Report MSR-TR-2004-27.
16. A. Hartman and K. Nagin. Model driven testing - AGEDIS architecture interfaces and tools. In *1st European Conference on Model Driven Software Engineering*, pages 1–11, Nuremberg, Germany, December 2003.
17. C. Jard and T. Jéron. TGV: theory, principles and algorithms. In *The Sixth World Conference on Integrated Design and Process Technology, IDPT'02*, Pasadena, California, June 2002.

18. V. V. Kuliamin, A. K. Petrenko, A. S. Kossatchev, and I. B. Bourdonov. UniTesK: Model based testing in industrial practice. In *1st European Conference on Model Driven Software Engineering*, pages 55–63, Nuremberg, Germany, December 2003.
19. L. Nachmanson, M. Veanes, W. Schulte, N. Tillmann, and W. Grieskamp. Optimal strategies for testing nondeterministic systems. In *ISSTA'04*, volume 29 of *Software Engineering Notes*, pages 55–64. ACM, July 2004.
20. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Mathematical Statistics. A Wiley-Interscience, New York, 1994.
21. J. Tretmans and E. Brinksma. TorX: Automated model based testing. In *1st European Conference on Model Driven Software Engineering*, pages 31–43, Nuremberg, Germany, December 2003.
22. J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 1944.
23. M. Yannakakis. Testing, optimization, and games. In *Proceedings of the Nineteenth Annual IEEE Symposium on Logic In Computer Science, LICS 2004*, pages 78–88. IEEE, 2004.
24. W. Yi and K. G. Larsen. Testing probabilistic and nondeterministic processes. In *Testing and Verification XII*, pages 347–61. North Holland, 1992.