

# POMDPStressTesting.jl Example: Walk1D

Robert J. Moss

*Computer Science, Stanford University*

MOSSR@CS.STANFORD.EDU

## Abstract

In this self-contained tutorial, we define a simple problem for adaptive stress testing (AST) to find failures. This problem, called Walk1D, samples random walking distances from a standard normal distribution  $\mathcal{N}(0, 1)$  and defines failures as walking past a certain threshold (which is set to  $\pm 10$  in this example). AST will either select the seed which deterministically controls the sampled value from the distribution (i.e. from the transition model) or will directly sample the provided environmental distributions. These action modes are determined by the seed-action or sample-action options. AST will guide the simulation to failure events using a notion of distance to failure, while simultaneously trying to find the set of actions that maximizes the log-likelihood of the samples.

## 1. Gray-box Simulator and Environment

The simulator and environment are treated as gray-box because we need access to the state-transition distributions and their associated likelihoods.

**Parameters.** First, we define the parameters of our simulation.

```
1 @with_kw mutable struct Walk1DParams
2     startx::Float64 = 0 # Starting x-position
3     threshx::Float64 = 10 # +- boundary threshold
4     endtime::Int64 = 30 # Simulate end time
5 end
```

**Simulation.** Next, we define a GrayBox.Simulation structure.

```
1 @with_kw mutable struct Walk1DSim <: GrayBox.Simulation
2     params::Walk1DParams = Walk1DParams() # Parameters
3     x::Float64 = 0 # Current x-position
4     t::Int64 = 0 # Current time ±
5     distribution::Distribution = Normal(0, 1) # Transition distribution
6 end
```

### 1.1 GrayBox.environment

Then, we define our GrayBox.Environment distributions. When using the ASTSampleAction, as opposed to ASTSeedAction, we need to provide access to the sampleable environment.

```
1 GrayBox.environment(sim::Walk1DSim) = GrayBox.Environment(:x => sim.distribution)
```

## 1.2 GrayBox.transition!

We override the transition function from the GrayBox interface, which takes an environment sample as input. We apply the sample in our simulator, and return the log-likelihood.

```
1 function GrayBox.transition!(sim::Walk1DSim, sample::GrayBox.EnvironmentSample)
2     sim.t += 1 # Keep track of time
3     sim.x += sample[:x].value # Move agent using sampled value from input
4     return logpdf(sample)::Real # Summation handled by 'logpdf()'
5 end
```

## 2. Black-box System

The system under test, in this case a simple single-dimensional moving agent, is always treated as black-box. The following interface functions are overridden to minimally interact with the system, and use outputs from the system to determine failure event indications and distance metrics.

### 2.1 BlackBox.initialize!

Now we override the BlackBox interface, starting with the function that initializes the simulation object. Note, each interface function may modify the `sim` object in place.

```
1 function BlackBox.initialize!(sim::Walk1DSim)
2     sim.t = 0
3     sim.x = sim.params.startx
4 end
```

### 2.2 BlackBox.distance!

We define how close we are to a failure event using a non-negative distance metric.

```
1 BlackBox.distance!(sim::Walk1DSim) = max(sim.params.threshx - abs(sim.x), 0)
```

### 2.3 BlackBox.isevent!

We define an indication that a failure event occurred.

```
1 BlackBox.isevent!(sim::Walk1DSim) = abs(sim.x) >= sim.params.threshx
```

### 2.4 BlackBox.isterminal!

Similarly, we define an indication that the simulation is in a terminal state.

```
1 BlackBox.isterminal!(sim::Walk1DSim) =
2     BlackBox.isevent!(sim) || sim.t >= sim.params.endtime
```

## 2.5 BlackBox.evaluate!

Lastly, we use our defined interface to evaluate the system under test. Using the input sample, we return the log-likelihood, distance to an event, and event indication.

```

1 function BlackBox.evaluate!(sim::Walk1DSim, sample::GrayBox.EnvironmentSample)
2     logprob::Real = GrayBox.transition!(sim, sample) # Step simulation
3     distance::Real = BlackBox.distance!(sim) # Calculate miss distance
4     event::Bool = BlackBox.isevent!(sim) # Check event indication
5     return (logprob::Real, distance::Real, event::Bool)
6 end

```

## 3. AST Setup and Running

Setting up our simulation, we instantiate our simulation object and pass that to the Markov decision process (MDP) object of the adaptive stress testing formulation. We use Monte Carlo tree search (MCTS) with progressive widening on the action space as our solver. Hyperparameters are passed to MCTSPWSolver, which is a simple wrapper around the POMDPs.jl implementation of MCTS. Lastly, we solve the MDP to produce a planner. Note we are using the ASTSampleAction.

```

1 function setup_ast(seed=0)
2     # Create gray-box simulation object
3     sim::GrayBox.Simulation = Walk1DSim()
4
5     # AST MDP formulation object
6     mdp::ASTMDP = ASTMDP{ASTSampleAction}(sim)
7     mdp.params.debug = true # record metrics
8     mdp.params.top_k = 10 # record top k best trajectories
9     mdp.params.seed = seed # set RNG seed for determinism
10
11     # Hyperparameters for MCTS-PW as the solver
12     solver = MCTSPWSolver(n_iterations=1000, # number of algorithm iterations
13                           exploration_constant=1.0, # UCT exploration
14                           k_action=1.0, # action widening
15                           alpha_action=0.5, # action widening
16                           depth=sim.params.endtime) # tree depth
17
18     # Get online planner (no work done, yet)
19     planner = solve(solver, mdp)
20
21     return (planner, mdp::ASTMDP)
22 end

```

After setup, we *playout* the planner and output an action trace of the best trajectory.

```

1 (planner, mdp) = setup_ast()
2 action_trace = playout(mdp, planner)

```

We can also *playback* specific trajectories and print intermediate  $x$ -values.

```
1 final_state = playback(mdp, action_trace, sim->sim.x)
```

Finally, we can print metrics associated with the AST run for further analysis.

```
1 failure_rate = print_metrics(mdp)
```

## 4. Solvers

The solvers provided by the POMDPStressTesting.jl package include the following.

```
1 # Reinforcement learning
2   MCTSPWSolver
3 # Deep reinforcement learning
4   TRPOSolver
5   PPOSolver
6 # Stochastic optimization
7   CEMSolver
8 # Baselines
9   RandomSearchSolver
```

## 5. Reward Function

The AST reward function gives a reward of 0 if an event is found, a reward of negative distance if no event is found at termination, and the log-likelihood during the simulation.

$$R(p, e, d, \tau) = \begin{cases} 0 & \text{if } \tau \wedge e \\ -d & \text{if } \tau \wedge \neg e \\ \log(p) & \text{otherwise} \end{cases}$$

```
1 function R(p,e,d,τ)
2   if τ && e
3     return 0
4   elseif τ && !e
5     return -d
6   else
7     return log(p)
8   end
9 end
```