POMDPStressTesting.jl Example: Walk1D

Robert J. Moss MOSSR@CS.STANFORD.EDU
Computer Science, Stanford University

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 passed a certain threshold (which is set to ± 10 in this example). AST will select the seed which deterministically controls the sampled value from the distribution (i.e. from the transition model). AST will try to find the set of seeds that maximizes the log-probability of the samples, while simultaneously guiding the simulation to failures using a notion of distance to failure.

1. Parameters and Simulation

First, we define the parameters of our simulation.

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

Next, we define a `BlackBox.Simulation` structure.

```
# Implement abstract BlackBox.Simulation
@with_kw mutable struct Walk1DSim <: BlackBox.Simulation
    params::Walk1DParams = Walk1DParams() # Parameters
    x::Float64 = 0 # Current x-position
    t::Int64 = 0 # Current time
    distribution::Distribution = Normal(0, 1) # Transition model
end
```

2. BlackBox.initialize!

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

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

3. BlackBox.transition!

Continuing with the BlackBox interface overriding, we define how the simulation will transitions between steps, returning the log-probability.

```
function BlackBox.transition!(sim::Walk1DSim)
    sim.t += 1 # Keep track of time
    sample = rand(sim.distribution) # Sample value from distribution
    logprob = logpdf(sim.distribution, sample) # Get log-probability of sample
    sim.x += sample # Move agent
    return logprob::Real
end
```

4. BlackBox.distance!

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

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

5. BlackBox.isevent!

We define an indication that a failure event occurred.

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

6. BlackBox.isterminal!

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

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

7. BlackBox.evaluate!

Lastly, we use our already defined interface to evaluate the system under test. Returning the log-probability $\log(p)$, distance to an event d, and event indication e as output.

```
function BlackBox.evaluate!(sim::Walk1DSim)
    logprob::Real  = BlackBox.transition!(sim) # Step simulation
    distance::Real = BlackBox.distance!(sim) # Calculate miss distance
    event::Bool    = BlackBox.isevent!(sim) # Check event indication
    return (logprob::Real, distance::Real, event::Bool)
end
```

8. AST Setup and Running

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

```julia
function setup_ast()
    # Create black-box simulation object
    sim::BlackBox.Simulation = Walk1DSim()

    # AST MDP formulation object
    mdp::ASTMDP = ASTMDP(sim)
    mdp.params.debug = true # record metrics
    mdp.params.seed = 1
    mdp.params.top_k = 10

    # Hyperparameters for MCTS-PW as the solver
    solver = MCTSASTSolver(depth=sim.params.endtime,
                           exploration_constant=10.0,
                           k_action=0.1,
                           alpha_action=0.85,
                           n_iterations=1000)

    policy = solve(solver, mdp)

    return (policy, mdp, sim)
end
```

Now we setup the AST run.

```julia
(policy, mdp, sim) = setup_ast()
```

Then *playout* the policy and ouput an action trace of the best trajectory.

```julia
action_trace = playout(mdp, policy)
```

We can also *playback* specific trajectories and display intermediate states along the way (in this case, the x value of the agent).

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

Embedded Julia source file.