The Young Modeler's Handbook

David V. Pynadath

September 4, 2013

I never satisfy myself until I can make a mechanical model of a thing. If I can make a mechanical model I can understand it.

Lord Kelvin

Chapter 1

Modeling a Scenario

O brave new world,
That has such people in't.

Shakespeare

1.1 Hello World

The *world*, as you might expect, is the entire universe of the simulation. It contains all of the global aspects of the simulation state, including the agents themselves.

```
from psychsim.world import World

world = World()
```

An *agent* is an autonomous decision-making entity in the world. The only requirement for an agent is that it have a name.

```
from psychsim.agent import Agent

free = Agent('Freedonia')
world.addAgent(free)
```

1.2 State

The *state* of a simulation captures the dynamic process by which it evolves. As in a *factored MDP*, we divide an overall state vector, $\vec{S}$, into a set of orthogonal features, $S_0 \times S_1 \times \cdots \times S_n$. Each feature captures a distinct aspect of the simulation state. Rather than refer to each such feature by its index, PsychSim instead allows you to refer to each feature by a more meaningful *key*, as in Python's dictionary keys. Keys are treated internally as unstructured strings, but you may find it useful to make use of the the following types of structured keys.

1.2.1 Unary Keys

It can be useful to describe a state feature as being local to an individual agent. Doing so does *not* limit the dependency or influence of the state feature in any way. However, it can be useful to define state features as local to an agent to make the system output more readable. For example, the following defines a state feature "troops" for the previously defined agent "Freedonia".

```
world.defineState(free.name,'troops')
```

For state features which are not local to any agent, a null agent name (i.e., `None`) indicates that the state feature will pertain to the world as a whole. Again, from the system's perspective, this makes no difference, but it can be useful to distinguish global and local state in presentation.

```
world.defineState(None,'treaty)
```

1.2.2 Binary Keys

There can also be state that pertains to the *relationship* between two agents.

```
world.defineRelation(free.name,sylv.name,'trusts')
```

The order in which the agents appear in this definition *does* matter, as reversing the order will generate a reference to a different element of the state vector.

1.2.3 Domains

By default, a state feature is assumed to be real-valued, in $[-1, 1]$. However, both the "defineState" and "defineRelation" methods take optional arguments to modify the valid ranges of the feature. The range of possible values does not affect the execution of the simulation, as simulation normalizes the values back to the $[-1, 1]$ range during decision-making.

```
world.defineState(free.name,'troops',lo=0,hi=50000)
```

It is also possible to specify that a state feature take on only integral values with the optional "domain" argument.

```
world.defineState(free.name,'troops',int,lo=0,hi=50000)
```

One can also define a boolean state feature using the same argument, in which case the optional "lo" and "hi" arguments are obviously moot.

```
world.defineState(None,'treaty,bool)
```

It is also possible to define an enumerated list of possible state features.

```
phases = ['offer','respond','rejection','end','paused',
          'engagement']
world.defineState(None,'phase',list,phases)
```

The domains also work within "defineRelation" calls as well. In other words, relationships can take on the same sets of values as unary state features.

1.3 Actions

1.3.1 Legality

```
tree = makeTree({'if': equalRow(stateKey(None,'phase'),'offer'),
                 True: True,
                 False: False})
free.setLegal(action,tree)
```

1.3.2 Dynamics

1.3.3 Termination

Termination conditions specify when scenario execution should reach an absorbing end state (e.g., when a final goal is reached, when time has expired). A termination condition is a PWL function (Section 1.7) with boolean leaves.

```
world.addTermination(makeTree({'if': trueRow(stateKey(None,'trea
                                True: True, False: False}))
```

This condition specifies that the simulation ends if a "treaty" is reached. Multiple conditions can be specified, with termination occurring if any condition is true.

1.4 Reward

```
    goalFTroops = maximizeFeature(stateKey(free.name,'troops'))
    free.setReward(goalFTroops,1.)
    goalFTerritory = maximizeFeature(stateKey(free.name,'territo
    free.setReward(goalFTerritory,1.)
```

1.5 Models

A *model* in the PsychSim context is a potential configuration of an agent that may apply in certain worlds or decision-making contexts. All agents have a "True" model that represents their real configuration, which forms the basis of all of their decisions during execution.

It also possible to specify alternate models that represent perturbations of this true model, either to represent the dynamics of the agent's configuration or to represent the perceptions other agents have of it.

```
free.addModel('friend')
```

1.5.1 Model Attribute: `static`

1.6 Observations

1.7 PWL functions

Chapter 2

Executing a Scenario

```
world.step()
```

2.1 Algorithms

2.1.1 Decision Making

2.1.2 Belief Update

World has $\Pr(M_0)$

World generates an observation ω

But M_0 cannot generate ω

$$\begin{aligned}
\Pr(M_1 = m) &= \Pr(M_1 = m | M_0, \omega) && (2.1)\\
&= \frac{\Pr(M_1 = m, \omega | M_0)}{\Pr(\omega | M_0)} && (2.2)\\
&\propto && (2.3)
\end{aligned}$$

And done.