From Approximate POMDP Planning Software

Main: PomdpX File Format

PomdpX File Format (version 1.0)

This page has been viewed 1923 times.

The most up-to-date version of this document is available at **http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.PomdpXDocumentation**.

Table of Contents

1. Overview

PomdpX is an XML file format for specifying models of Markov decision processes (MDPs), partially observable Markov decision processes (POMDPs), and mixed observability Markov decision processes (MOMDPs) [**1**]. PomdpX uses a factored model representation, which can be represented graphically as a dynamic Bayesian network (DBN):

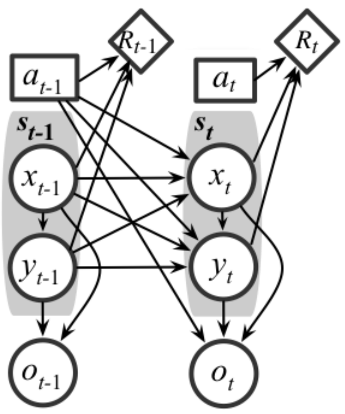

Figure 1.1. A MOMDP model. s_t represents the state, a_t represents the action, o_t represents the observation, and R_t represents the reward at time t. In a MOMDP model, the state variable s_t consists of two components: the fully observable state variable x_t and the partially observable state variable y_t.

PomdpX allows multiple state, action, observation, and reward variables to be specified in a model. A model must have at least one state, action, and reward variable. The observation variable is optional, depending on the type of the model (MDP, POMDP, or MOMDP). Each state variable must be specified as either partially observable (default) or fully observable. As a result, the PomdpX file format can specify any of the following models:

- MDPs, when all state variables are fully observable;
- POMDPs, when all state variables are partially observable;
- MOMDPs, when there are both fully and partially observable state variables.

The XML schema for PomdpX is available **here** for download.

2. PomdpX Tutorial

The purpose of this section is to provide a tutorial-like approach to using the PomdpX format. We make no assumptions about the user's familiarity with existing pomdp solvers.

2.1. Example Problem

We will be using a modified version of the *RockSample* problem [**2**] as our running example to encode into the PomdpX format. It models a rover on an exploration mission and it can achieve rewards by sampling rocks in its immediate area. Consider a map of size 1 × 3 as shown in Figure 2.1, with one rock at the left end and the terminal state at the right end. The rover starts off at the center and its possible actions are $A = \{West, East, Sample, Check\}$. The DBN for the *RockSample* problem is shown in Figure 2.2.

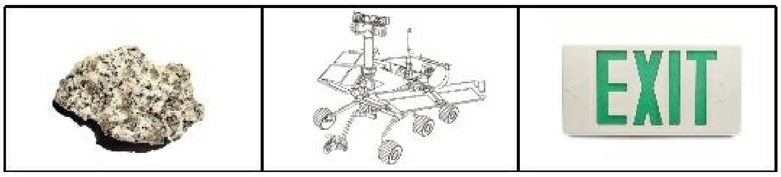

Figure 2.1. The 1 × 3 *RockSample* problem world.

This is a trivial problem but is adequate to showcase the salient features of PomdpX. As with the original version of the problem, the *Sample* action samples the rock at the rover's current location. If the rock is good, the rover receives a reward of 10 and the rock becomes bad. If the rock is bad, it receives a penalty of −10. Moving into the terminal area yields a reward of 10. A penalty of −100 is imposed for moving off the grid and sampling in a grid where there is no rock. All other moves have no cost or reward. The *Check* action returns a noisy observation from $O = \{Good, Bad\}$.

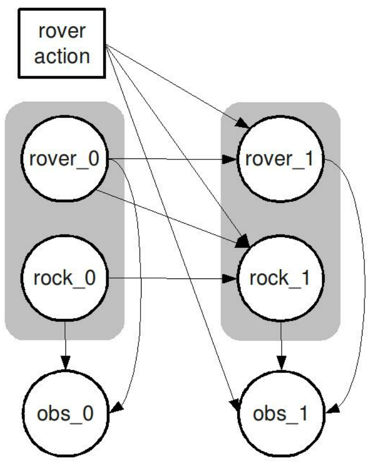

Figure 2.2. Dynamic Bayesian network of the *RockSample* problem. The rover's position is fully observed whereas the rock type is partially observed.

Example 1. A PomdpX document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pomdpx version="0.1" id="rockSample"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="pomdpx.xsd">
      <Description>            · · · </Description>
      <Discount>               · · · </Discount>
      <Variable>               · · · </Variable>
      <InitialStateBelief>     · · · </InitialStateBelief>
      <StateTransitionFunction> · · · </StateTransitionFunction>
      <ObsFunction>            · · · </ObsFunction>
      <RewardFunction>         · · · </RewardFunction>
</pomdpx>
```

2.2. File Format Structure

A PomdpX document consists of a header and a `pomdpx` root element which in turn contains child elements, as shown in Example 1 below. The first line of the document is an XML processing instruction which defines that the document adheres to the XML 1.0 standard and that the encoding of the document is ISO-8859-1. Other encodings such as UTF-8 are also possible.

2.2.1. <pomdpx> Tag

Continuing with the example above, the second line contains the root-element of a PomdpX document —the `pomdpx` element—which has the following attributes:

- `version`
- `id` – optional name for the specified model.
- `xmlns:xsi` – defines xsi as the XML Schema namespace.
- `xsi:noNamespaceSchemaLocation` – this is where we put our XML Schema

definition, `pomdpx.xsd`. The PomdpX input should be validated with this schema to ensure well-formedness.

The conventional ordering of the child elements is `Description`, `Discount`, `Variable` and thereafter: `InitialStateBelief`, `StateTransitionFunction`, `ObsFunction` and `RewardFunction`. However this ordering is not strictly required and one may permute their orderings. `Description` is an optional, short description of the specified model. The other child elements specify the POMDP tuple ($S, A, O, T , Z, R, \gamma$) and the initial belief b_0 .

In general these elements should all be present, and each can appear only once. `ObsFunction` may be omitted if there are no observation variables in the model. Similarly, `InitialBeliefState` may be omitted if all state variables are fully observed (for example an MDP model). `pomdpx`’s child elements are described in greater detail in the following subsections.

2.2.2. <Description> Tag

This is an optional tag that one may provide to give a brief description of the specified problem. For example:

Example 2. Contents of Description.

```
<Description> RockSample problem for map size 1 x 3.
Rock is at 0, Rover’s initial position is at 1.
Exit is at 2.
</Description>
```

2.2.3. <Discount> Tag

This specifies the discount factor γ. It has to be a real-valued number, for our RockSample problem, we will be using a discount factor of 0.95 and it is entered as shown:

Example 3. Contents of Discount.

```
<Discount> 0.95 </Discount>
```

2.2.4. <Variable> Tag

The state, action and observation variables which factorize the state *S*, action *A*, and observation *O* spaces are declared within the `Variable` element. Reward variables, *R* are also declared here. Example 4 gives the declaration of the variables for the *RockSample* problem.

Each state variable is declared with the <StateVar> tag. It contains the following attributes:

- `vnamePrev` – identifier for the variable's start state.
- `vnameCurr` – identifier for the variable's end state.
- `fullyObs` – set to true if the variable is fully observed. The default is false. Thus for the variable rock in Example 4, it is partially observed, as implied by the omission of the `fullyObs` attribute.

Example 4. Variable declaration. Defining *S*, *A*, *O*, and *R* variables.

```
<Variable>
     <StateVar vnamePrev="rover_0" vnameCurr="rover_1"
      fullyObs="true">
         <NumValues>3</NumValues>
     </StateVar>
     <StateVar vnamePrev="rock_0" vnameCurr="rock_1>"
         <ValueEnum>good bad</ValueEnum>
     </StateVar>
     <ObsVar vname="obs sensor">
         <ValueEnum>ogood obad</ValueEnum>
     </ObsVar>
     <ActionVar vname="action_rover">
         <ValueEnum>amw ame ac as</ValueEnum>
     </ActionVar>
     <RewardVar vname="reward rover" />
</Variable>
```

The possible values that a variable can assume are either specified with regards to the `<NumValues>` or `<ValueEnum>` tags.

If `<NumValues>` tags are used, we would give an integer to indicate the number of values/states for the variable. For instance, in the example, the *rover* is declared with three possible values. The values are subsequently referenced internally using numerals, starting from 0 and prepended with 's'. Hence the states for the rover variable would be s0, s1 and s2. When using `<NumValues>` it is up to the user to attach semantic meaning to the values, in our example, s0 denotes the left grid, s1 the center and s2 the right terminal grid.

If `<ValueEnum>` tags are used, the user will have to manually enumerate all the possible values/states the variable may take on. In our example, the *rock* has two possible values, it is either *good* or *bad*.

The observation and action variables are also declared similarly with the `<ObsVar>` and `<ActionVar>` tags respectively. Both require the attribute `vname` which serves as the identifier for the variable. The possible values that an observation or action can assume can also be specified with either `<NumValues>` or `<ValueEnum>`. If `<NumValues>` is used, 'o' and 'a' would be prepended to the values of observation and action variables respectively.

In the case of `<ValueEnum>`, the user will once again need to enumerate all possible values/states manually. In our example, for the *action_rover* variable, we enumerate all the four possible actions. 'amw' is a mnemonic for action move west and 'ac' stands for action check and so on.

Finally, reward variables are declared with the `<RewardVar>` tags which must contain the vname attribute. The `vname` serves as an identifier for the reward variable. The `<RewardVar>` is an empty XML tag and no values are specified. Note that we may use the XML shorthand of `<RewardVar vname="· · · " />` to close an empty tag here.

2.2.5. <InitialStateBelief> Tag

This is an optional tag. It specifies the initial belief b0, and may be omitted if all state variables are fully observed. The PomdpX format allows the initial belief to be specified as multiple multiplicative factors, with each `<CondProb>` tag specifying one of these factors. From our running *RockSample* problem, since the initial belief is not conditional on anything, it is factored as $b_0 = P(rover\_0|\emptyset)P(rock\_0|\emptyset)$. We will need two `<CondProb>` tags to specify it fully as shown below.

Example 5. Contents of `InitialStateBelief`.

```
  <InitialStateBelief>
      <CondProb>
         <Var>rover_0</Var>
         <Parent>null</Parent>
         <Parameter> · · · </Parameter>
      </CondProb>
      <CondProb>
         <Var>rock_0</Var>
         <Parent>null</Parent>
         <Parameter> · · · </Parameter>
      </CondProb>
   </InitialStateBelief>
```

The `<CondProb>` tag has no attributes and requires the following three children tags:

- `<Var>` – identifies the factor being specified. Only identifiers declared as `vnamePrev` of state variables are allowed here (see Section 2.2.4).
- `<Parent>` – the set of conditioning variables. Only the the keyword `null` and identifiers declared as `vnamePrev` of state variables are allowed here. Note however that PomdpX only allows certain combinations of `vnamePrev` identifiers to be specified in the `<Var>` and `<Parent>` tags. Referring to Figure 1.1, we only allow conditioning arrows from x_t (fully observed variables) to y_t (partially observed variables) and not the other way round. Specifically, a `vnamePrev` identifier is allowed as parent only if the variable is fully observed, and the `vnamePrev` identifier within the `<Var>` tag is partially observed. The keyword `null` may be used to signify the absence of any conditioning variables.
- `<Parameter>` – specifies the actual probabilities in the factor and is described in detail in Section 2.3.

2.2.6. <StateTransitionFunction> Tag

This specifies the transition function T , which in general is the multiplicative result of the individual transition functions of each state variable in the model. Each `<CondProb>` tag specifies the transition function for each state variable. For our *RockSample* problem, with reference to Figure 2.2, the overall transition function is: $P(rover\_1, rock\_1|action\_rover, rover\_0, rock\_0) = P(rover\_1|action\_rover, rover\_0) \times P(rock\_1|action\_rover, rover\_0, rock\_0)$.

This is translated to the following in PomdpX. One can see that it is very similar to its equational counterpart, only it has XML tags wrapped around it. We need to provide two CondProb elements, one each for the variable rover and rock.

Example 7. Contents of `StateTransitionFunction`.

```
 <StateTransitionFunction>
      <CondProb>
           <Var>rover_1</Var>
           <Parent>action_rover rover_0</Parent>
           <Parameter> · · · </Parameter>
      </CondProb>
      <CondProb>
           <Var>rock_1</Var>
           <Parent>action_rover rover_0 rock_0</Parent>
           <Parameter> · · · </Parameter>
      </CondProb>
 </StateTransitionFunction>
```

As described in 2.2.5, the `<Var>` tag identifies the state variable whose transition function is being specified. In this case, only identifiers declared as the `vnameCurr` attribute of state variables may be allowed here.

The identifiers within the `<Parent>` tag identify the conditioning variables in the transition function. They may be identifiers which had been declared as either the `vnamePrev` or `vnameCurr` attributes of state variables, or identifiers which had been declared as the `vname` attribute of action variables (see Section 2.2.4). There are no restrictions on the `vnamePrev` state identifiers and `vname` action identifiers that can appear within the `<Parent>` tag. Note however that analogous to the case for the initial belief, PomdpX only allows certain combinations of `vnameCur` identifiers to be specified in the `<Var>` and `<Parent>` tags. Specifically, one may only use `vnameCurr` identifiers within the `<Parent>` tag if the variable is fully observed, and the `vnameCurr` identifier within the `<Var>` tag is partially observed. We defer the description of `<Parameter>` tag to Section 2.3 as it is fairly involved.

2.2.7. <ObsFunction> Tag

This specifies the observation function *Z*, which in general is the multiplicative result of the individual observation functions of each observation variable in the model. Each `<CondProb>` tag specifies one of these individual observation functions. In the *RockSample* problem, the probability of an observation is conditional on taking an action and ending in a new state. Thus its parents are *action_rover*, *rover_1* and *rock_1*, as given in Example 8.

Example 8. Contents of `ObsFunction`.

```
<ObsFunction>
      <CondProb>
           <Var>obs_sensor</Var>
           <Parent>action_rover rover_1 rock_1</Parent>
           <Parameter> · · · </Parameter>
      </CondProb>
</ObsFunction>
```

For each CondProb element, the identifier within the `<Var>` tags identifies the observation variable whose observation function is being specified. The identifiers within the `<Parent>` tags identifies the conditioning variables in the observation function. Identifiers that appear within the `<Var>` tags must be identifiers which had been declared as the `vname` attribute of observation variables. Identifiers that appear within the `<Parent>` tags must be identifiers which had been declared as the `vnameCurr` attribute of state variables, or the `vname` attribute of action variables (see Section 2.2.4). Parameter specifies the actual probabilities in the function and will be described in Section 2.3.

2.2.8. <RewardFunction> Tag

This specifies the reward function *R*, which in general is the additive result of the individual reward functions of each reward variable in the model. Each `<Func>` tag specifies one of these individual reward functions. For our *RockSample* problem, the reward depends on the action taken at the current state, thus its parents are *action_rover*, *rover_0* and *rock_0*. This is shown in Example 9.

Example 9. Contents of `RewardFunction`.

```
<RewardFunction>
      <Func>
           <Var>reward_rover</Var>
           <Parent>action_rover rover_0 rock_0</Parent>
           <Parameter> · · · </Parameter>
      </Func>
</RewardFunction>
```

Similar to the `<CondProd>` tag, the `<Func>` tag has no attributes and requires the following three children tags to be defined:

- `<Var>` – this identifies the reward variable whose reward function is being specified. Only identifiers that had been declared as the `vname` attribute of reward variables may appear here.
- `<Parent>` – this identifies the domain of the reward function. All identifiers declared as `vnamePrev` or `vnameCurr` attributes of state variables, `vname` attribute of action variables or `vname` attribute of observation variables are allowed here.
- `<Parameter>` – specifies the actual values in the function and is described in detail in Section 2.3.

2.3. <Parameter> Tag

The `<Parameter>` tag is a fairly complicated component of PomdpX. It has an optional attribute called type, which has possible values `TBL` (default) and `DD`, short for *table* and *decision diagram*, respectively. We will describe how to encode the *RockSample* problem both in `TBL` and `DD`.

2.3.1. Table Type (TBL)

When the `<Parameter>` tag appears as a child of a CondProb element, it must contain `<Entry>` child tags. Each Entry element specifies the probability entry of a function table. The `<Entry>` tag itself must consist of the following:

- `<Instance>` – declares all the variables for the probability function. Each variable value must correspond to the identifiers that appear between the enclosing `<Parent>` tag, followed by the identifier that appears between the enclosing `<Var>` tag.
- `<ProbTable>` – specifies the actual numerical values of the probabilities. This is best illustrated by Example 10 below. With reference to Figure 2.2, we show the full encoding of the rock ’s transition function for the rover ’s action of moving *West*. From the example, the `<Var>` tag declares that we are defining the transition function for the variable *rock* (line 3). It is conditional on *action_rover*, *rover_0* and *rock_0*, which appear between the `<Parent>` tag

(line 4). The first `<Entry>` set (lines 6–9) specifies:

P (rock_1 = good|action_rover = amw, rover_0 = s0, rock_0 = good) = 1.0.

In this case, when *action_rover* is *amw*, and *rock_0* is good, *rock_1* will be good as well, since a move action will not disturb its state. Conversely, if action_rover is *amw*, and *rock_0* is *good* it is impossible for *rock_1* to be *bad* as specified by lines 18–29.

Note that order matters here and it might be the source of some subtle bugs if overlooked. As mentioned before, the conditioning variables declared between the `<Instance>` tag (first three elements in line 7) correspond to the order they appear in the enclosing `<Parent>` tag, the last element corresponds to the variable being defined. One may arbitarily re-order the conditioning variables as long as they match-up within the `<Parent>` and `<Instance>` tags and the last element is always the identifier defined by `<Var>`. The convention that we adopt is to declare actions, fully observed variables followed by partially observed variables.

Example 10. Contents of `Parameter type="TBL"`, within `CondProb`.

```
1. <StateTransitionFunction>
2.       <CondProb>
3.           <Var>rock_1</Var>
4.           <Parent>action_rover rover_0 rock_0</Parent>
5.           <Parameter type = "TBL">
6.               <Entry>
7.                    <Instance>amw s0 good good</Instance>
8.                    <ProbTable>1.0</ProbTable>
9.               </Entry>
10.              <Entry>
11.                   <Instance>amw s1 good good</Instance>
12.                   <ProbTable>1.0</ProbTable>
13.              </Entry>
14.              <Entry>
15.                   <Instance>amw s2 good good</Instance>
16.                   <ProbTable>1.0</ProbTable>
17.              </Entry>
18.              <Entry>
19.                   <Instance>amw s0 good bad</Instance>
20.                   <ProbTable>0.0</ProbTable>
21.              </Entry>
22.              <Entry>
23.                   <Instance>amw s1 good bad</Instance>
24.                   <ProbTable>0.0</ProbTable>
25.              </Entry>
26.              <Entry>
27.                   <Instance>amw s2 good bad</Instance>
28.                   <ProbTable>0.0</ProbTable>
29.              </Entry>
```

```
30.              <Entry>
31.                   <Instance>amw s0 bad good</Instance>
32.                   <ProbTable>0.0</ProbTable>
33.              </Entry>
34.              <Entry>
35.                   <Instance>amw s1 bad good</Instance>
36.                   <ProbTable>0.0</ProbTable>
37.              </Entry>
38.              <Entry>
39.                   <Instance>amw s2 bad good</Instance>
40.                   <ProbTable>0.0</ProbTable>
41.              </Entry>
42.              <Entry>
43.                   <Instance>amw s0 bad bad</Instance>
44.                   <ProbTable>1.0</ProbTable>
45.              </Entry>
46.              <Entry>
47.                   <Instance>amw s1 bad bad</Instance>
48.                   <ProbTable>1.0</ProbTable>
49.              </Entry>
50.              <Entry>
51.                   <Instance>amw s2 bad bad</Instance>
52.                   <ProbTable>1.0</ProbTable>
53.              </Entry>
54.          </Parameter>
55.      </CondProb>
56. </StateTransitionFunction>
```

It seems a bit daunting that it takes 56 lines just to declare the transition function for the rock for a simple 1 × 3 grid. And this only for the *rover*'s action of moving *West*. But XML is verbose by nature and that is the price to pay for interoperability and extensibility. However, PomdpX does provide several convenience features to ease the encoding task.

First and foremost, lines 18–41 are actually redundant since any entry not specified is assumed to be zero. Secondly, we observe that the first three `<Entry>` sets (lines 6–17) are very similar. They differ only in the state of *rover_0* and *s0* to *s2* are all the possible states of the rover. In such a situation, we may use the wildcard character “*”, which means that this is true for all possible values that could appear here. Therefore, lines 6–17 could be replaced by just one `<Entry>` tag, this is true for lines 42–53 too. Example 10 is re-written more succinctly and shown as Example 11.

Example 11. Usage of wildcard character *.

```
1. <StateTransitionFunction>
2.      <CondProb>
3.          <Var>rock_1</Var>
4.          <Parent>action_rover rover_0 rock_0</Parent>
5.          <Parameter type = "TBL">
6.              <Entry>
7.                   <Instance>amw * good good</Instance>
8.                   <ProbTable>1.0</ProbTable>
9.              </Entry>
10.             <Entry>
11.                  <Instance>amw * bad bad</Instance>
12.                  <ProbTable>1.0</ProbTable>
13.             </Entry>
14.         </Parameter>
15.     </CondProb>
16. </StateTransitionFunction>
```

As some probabilities of the *rock* ’s transition are zero, they may be conveniently left out. However in certain cases, some variables may have all non-zero transition probabilities. PomdpX specifically provides another special character “-” to handle this. The “-” character means cycle through all possible values that could appear here and match the listed probabilities (in `<ProbTable>`) accordingly. Hence, Example 11 can also be expressed as:

Example 12. Usage of character -.

```
1. <StateTransitionFunction>
2.      <CondProb>
3.          <Var>rock_1</Var>
```

```
4.          <Parent>action_rover rover_0 rock_0</Parent>
5.          <Parameter type = "TBL">
6.              <Entry>
7.                    <Instance>amw * good - </Instance>
8.                    <ProbTable>1.0 0.0</ProbTable>
9.              </Entry>
10.             <Entry>
11.                   <Instance>amw * bad - </Instance>
12.                   <ProbTable>0.0 1.0</ProbTable>
13.             </Entry>
14.         </Parameter>
15.     </CondProb>
16. </StateTransitionFunction>
```

Although it is not obvious here, one can imagine if the entries were both non- zero, the use of “-” would save us from having to specify another set of `<Entry>` tag.

With the introduction of the “-” character, the first <Entry> set (lines 6–9) in Example 12 is in effect specifying the following:

P (rock_1 = good|action_rover = amw, rover_0 = ∗, rock_0 = good) = 1.0 and *P (rock_1 = bad|action_rover = amw, rover_0 = ∗, rock_0 = good) = 0.0.*

There is also an implicit ordering in Example 12. For instance, the usage of “-” for the first `<Entry>` set (lines 6–9), considers the possible values of rock to be *good* first then *bad*, hence the `<ProbTable>` entries are listed as (1.0 0.0) rather than (0.0 1.0). This “internal” order is actually taken from the way rock is declared in the `<ValueEnum>` tag (see Section 2.2.4), in which its possible values were declared to be first *good* then *bad*.

In the quest for further compression, there is a final modification we can make to Example 12. We make the observation that the two `<Entry>` sets seem some- what complementary differing only in the states of *rock_0* and `<ProbTable>` entries. Thus employing the same trick for Example 12, we can replace the states of ''rock_0' with a “-”. This gives us Example 13.

Example 13. Usage of double -.

```
1. <StateTransitionFunction>
2.      <CondProb>
3.          <Var>rock_1</Var>
4.          <Parent>action_rover rover_0 rock_0</Parent>
5.          <Parameter type = "TBL">
6.              <Entry>
7.                   <Instance>amw * - - </Instance>
8.                   <ProbTable>1.0 0.0 0.0 1.0</ProbTable>
9.              </Entry>
10.         </Parameter>
11.     </CondProb>
12. </StateTransitionFunction>
```

By using double “-”, the single `<Entry>` set in Example 13 is equivalent to specifying the following:

P (rock_1 = good|action_rover = amw, rover_0 = ∗, rock_0 = good) = 1.0
P (rock_1 = bad|action_rover = amw, rover_0 = ∗, rock_0 = good) = 0.0
P (rock_1 = good|action_rover = amw, rover_0 = ∗, rock_0 = bad) = 0.0
and
P (rock_1 = bad|action_rover = amw, rover_0 = ∗, rock_0 = bad) = 1.0.

The `<ProbTable>` entries in Example 13 are in effect a 2 × 2 identity matrix. Hence our PomdpX format also allows for the keyword $identity_2$ to be used in lieu of having to enumerate all the ones and zeros (like line 8). Therefore Examples 13 and 14 are functionally equivalent.

Example 14. Usage of keyword `identity`.

```
1. <StateTransitionFunction>
2.      <CondProb>
3.          <Var>rock_1</Var>
```

```
         <Parent>action_rover rover_0 rock_0</Parent>
         <Parameter type = "TBL">
             <Entry>
                  <Instance>amw * - - </Instance>
                  <ProbTable>identity</ProbTable>
             </Entry>
         </Parameter>
     </CondProb>
</StateTransitionFunction>
```

Another recognized keyword which may also be used in the `<ProbTable>` tags is *uniform*. This is equivalent to the probability 1/n repeated n times, where *n* is the number of possible values that could appear here. For example, the `<Entry>` tag below,

Example 15. Usage of keyword `uniform`.

```
<InitialStateBelief>
     <CondProb>
         <Var>rock_0</Var>
         <Parent>null</Parent>
         <Parameter type = "TBL">
             <Entry>
                  <Instance> - </Instance>
                  <ProbTable>uniform</ProbTable>
             </Entry>
         </Parameter>
     </CondProb>
</InitialStateBelief>
```

gives: $P\ (rock\_0 = good|\varnothing) = 0.5$ *and* $P\ (rock\_0 = bad|\varnothing) = 0.5$, which specifies our initial belief that the rock has equal probability of being *good* or *bad*.

Besides being a child of the `CondProb` element, the `<Parameter>` tag may also appear as a child of the `Func` element which is used to define the reward function. In this case, the `<Entry>` tag within the `<Parameter>` must contain

the following:

- `<Instance>` – declares values of all the variables for the reward function. Each variable value must correspond to the identifiers that appear between the enclosing `<Parent>` tag.
- `<ValueTable>` – specifies the actual numerical reward.

Example 16. Contents of `Parameter type="TBL"`, within `Func`.

```
<RewardFunction>
     <Func>
         <Var>reward_rover</Var>
         <Parent>action_rover rover_0 rock_0</Parent>
         <Parameter type = "TBL">
             <Entry>
                  <Instance> ame s1 * </Instance>
                  <ProbTable>10</ProbTable>
             </Entry>
                .
                .
                .
         </Parameter>
     </Func>
</RewardFunction>
```

Example 16 shows a snippet defining the reward function for the rover. In this example, the `<Entry>` specifies:

$R_{reward\_rover}\ (action\_rover = ame,\ rover\_0 = s1,\ rock\_0 = *) = 10.$

By now, the wildcard character “*” should be familiar to the user. Its use here denotes the fact that the *rover* will obtain a reward of 10 moving East from s1 (to the terminal state), regardless of whether the *rock* is *good* or *bad*.

Note that the characters “*” and “-” can be used in a similar manner as described in the previous sections. However, the keywords uniform and identity cannot appear between `<ValueTable>` tags, since those keywords only make sense for probabilities and not rewards.

We reiterate here that any probability or value entries of a function table which are not specified within a `<Parameter>` tag are assumed to be zero. Fur- thermore, a particular probability or value entry can also be specified more than once. The definition that appears last within a `<Parameter>` tag is the one that will take effect. This is convenient for specifying exceptions to a more general specification. The full compact version of the PomdpX input file for the *RockSample* problem with `<Parameter type="TBL">` is given in Appendix A.

2.3.2. Decision Diagram (DD)

Note that the APPL parser does **not** support decision diagrams currently. However, decision diagrams are officially part of POMDPX, and we plan to support it in APPL in the future.

Decision diagrams are another way of describing the conditional probabilities of the variables. A decision diagram in PomdpX is represented as a rooted, directed, acyclic graph (DAG), which consists of `<Node>` tags and branches with `<Edge>` tags. Figure 2.3 shows a generic structure of the DAG used in PomdpX. Similar to `TBL`, the `<Parameter>` tag with `type = "DD"` may appear with the `<CondProb>` or `<Func>` tags. But unlike `TBL`, the syntax for the `DD` type is exactly the same within both `<CondProb>` and `<Func>` tags. Hence the following descriptions apply to `DD` in both cases.

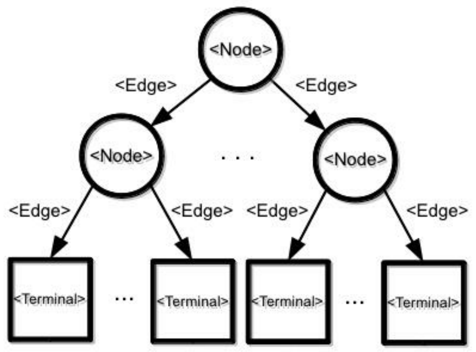

Figure 2.3. Generic structure of a DAG used in PomdpX. Intermediate nodes are circles and terminals are squares.

Example 17 shows a snippet of how the initial belief for the RockSample problem can be coded using `DD`. The discerning reader can immediately tell that using `DD` results in a more nested XML structure. This is so since we have to branch on each of the variables declared within the `<Var>` tag (line 1). However, there are convenience methods provided by PomdpX to alleviate this. We note that the `<Parameter type="DD">` tag can have two possible children: namely `<DAG>` or `<SubDAGTemplate>`. In Example 17, we are using `<DAG>` (line 4), whose purpose is to signify that we are explicitly defining the node- edge branching and terminal values. On the other hand, `<SubDAGTemplate>` allows us to reuse a pre-defined template at certain portions. We will describe `<SubDAGTemplate>` in detail later.

Within the `<DAG>` tag is where we will declare the variables as nodes and their possible values as edges. To declare a variable as a node, we use the `<Node>` tag. It requires an attribute `var` to indicate which state variable it is being defined for. For example, in line 7 of Example 17, we are defining it for the variable *rover_0*. The variable *rover_0* has three values, *s0* to *s2*, denoting its possible locations. These will be declared using the `<Edge>` tag which has a mandatory attribute `val` to indicate which value it is being defined for. These are given in lines 6, 7 and 17.

Example 17. Contents of `Parameter type="DD"`, within `CondProb`.

```
 1. <Var>rover_0 rock_0</Var>
 2. <Parent>null</Parent>
 3.  <Parameter type = "DD">
 4.     <DAG>
 5.        <Node var = "rover_0">
 6.            <Edge val="s0"><Terminal>0.0</Terminal></Edge>
 7.             <Edge val="s1">
 8.                <Node var = "rock_0">
 9.                    <Edge val = "good">
10.                       <Terminal>0.5</Terminal>
11.                   </Edge>
12.                   <Edge val = "bad">
13.                      <Terminal>0.5</Terminal>
14.                   </Edge>
15.                </Node>
16.          </Edge>
17.          <Edge val="s2"><Terminal>0.0</Terminal></Edge>
18.        </Node>
19.     </DAG>
20. </Parameter>
```

We find it easy to keep in mind that `<Node>` is for the declaration of variables, hence its attribute is `var` and `<Edge>`'s attribute is `val` since it is declaring for the values. In general, a decision diagram is described by starting from a node and branching to another node and so on, until a terminal state is reached. This is shown in lines 7–16. The order follows the sequence of variables declared within the `<Var>` tag. In Example 17, the DAG is rooted at *rover_0*. The `<Terminal>` tag is used to specify the terminating node of the DAG. It takes a real-valued number as argument. This may be either a conditional probability value or a reward.

One can imagine that with a large number of variables, the levels of `<Node>`-`<Edge>` branching will be tremendous. However, one can make use of context- sensitive independence to "short-circuit" this process. In Example 17, since it is not possible for the *rover_0* to start at *s0*, it is not necessary to branch on *rock_0* after taking the edge s0. Thus we can straightaway declare a `<Terminal>` value of 0.0 here (line 6). This applies for value *s2* also as shown in line 17. For value *s1*, since *rock_0* has a uniform probability of being *good* or *bad*, we declare the `<Terminal>` values of 0.5 each after taking the edges good and bad respectively (lines 7–16).

Another useful convenience method provided by PomdpX is the XML tag: `<SubDAG>`. It is an empty XML tag, but it has the following attributes:

- type – this declares the type of DAG being specified. There are four

possible types of `<SubDAG>`.
    * `deterministic` – this is a shorthand for `<Terminal>` value equals 1.0 for the value specified by `val`, and is used to specify that the DAG

is not noisy. See Example 18.
    * `persistent` – its usage is similar to the keyword `identity` in that DAGs specified as `persistent` would not change in their value. See Example 19.
    * `uniform` – its usage is exactly the same as when used in `<ProbTable>` (section 2.3.1), it means that the probabilities are equally distributed. See Example 20. In fact, Examples 17 and 20 are equivalent. Notice how the usage of `<SubDAG type="uniform">` shortens the node-edge branching significantly.
    * `template` – as the name suggests, a `<SubDAG>` declared as this type is modular and can be reused anywhere within the `<Parameter>` declaration. See Example 21.
- `var` – specifies the variable that it is declaring for. It is not valid for the type `template`.
- `val` – used only in conjunction with type `deterministic`. It specifies the value the particular variable takes on.
- `idref` – used only with type `template`. It specifies the `template` to be used by referencing the template through its identifier. The template

must have been declared with the `<SubDAGTemplate>`.

Example 18 gives the transition function for the *rover* for action *West*. Lines 9–19, show how one can declare the transition for the *rover* starting at *s0*. Since the movement of the *rover* is deterministic and non-noisy, there would be lots of redundant zero values being declared within the `<Terminal>` tags. With `<SubDAG type="deterministic">`, we could replace lines 9–19 by simply stating the `val` which is non-zero and the `var` for which it is defined for. Lines 21–28 illustrate the benefit of using this syntax which is more compact.

In situations where the state of a variable will not change from one instance to another. We may use `<SubDAG type="persistent">` as shown in Example 19, which describes a snippet of the *rock*'s transition function. With respect to the example, this simply means that the state of *rock_1* remains the same regardless of the move *West*, *East* and *Check* actions of the *rover*. Example 20 shows the usage of `<SubDAG>` of type `uniform`. Comparing it with Example 17, it is clear that the use of `<SubDAG>` and `uniform` type result in a more compact representation.

Example 18. Usage of `SubDAG type="deterministic"`.

```
<Var> rover_1</Var>
    <Parent>action_rover rover_0</Parent>
    <Parameter type = "DD">
        <DAG>
            <Node var = "action_rover">
                <Edge val = "amw">
                    <Node var = "rover_0>
                        <Edge val = "s0">
                            <Node var = "rover_1">
                                <Edge val = "s0">
                                     <Terminal>0.0</Terminal>
                                 </Edge>
                                 <Edge val = "s1">
                                     <Terminal>0.0</Terminal>
                                 </Edge>
                                 <Edge val = "s2">
                                     <Terminal>1.0</Terminal>
                                 </Edge>
                             </Node>
                         </Edge>
                         <Edge val = "s1">
                             <SubDAG type = "deterministic"
                                 var = "rover_1" val = "s0" />
                         </Edge>
                         <Edge val = "s2">
                             <SubDAG type = "deterministic"
                                     var = "rover_1" val = "s2" />
                         </Edge>
                     </Node>
                 </Edge>
                  ...
         </Node>
     </DAG>
 </Parameter>
```

Example 19. Usage of `SubDAG type="persistent"`.

```
<Var> rock_1</Var>
<Parent>action_rover rover_0 rock_0</Parent>
    <Parameter type = "DD">
        <DAG>
            <Node var = "action_rover">
                <Edge val = "amw">
                       <SubDAG type = "persistent" var = "rock_1" />
                </Edge>
                <Edge val = "ame">
                    <SubDAG type = "persistent" var = "rock_1" />
                </Edge>
                <Edge val = "ac">
                    <SubDAG type = "persistent" var = "rock_1" />
                </Edge>
                  .
                  .
                  .
         </Node>
     </DAG>
</Parameter>
```

Example 20. Usage of `SubDAG type="uniform"`.

```
<Var>rover_0 rock_0</Var>
<Parent>null</Parent>
    <Parameter type = "DD">
```

```
       <DAG>
           <Node var = "rover_0">
               <Edge val="s0"><Terminal>0.0</Terminal></Edge>
               <Edge val="s1">
                   <SubDAG type = "uniform" var = "rock_0" />
               </Edge>
              <Edge val="s2"><Terminal>0.0</Terminal></Edge>
        </Node>
    </DAG>
</Parameter>
```

Another convenience feature provided by PomdpX is the ability to modularize certain definitions for reuse. This is achieved with the `<SubDAGTemplate>` tag, it requires an attribute `id` to serve as an identifier for it. Its benefit is illustrated in Figure 2.4, where the three identical parts reuse the same definition provided by `<SubDAGTemplate>`. Example 21 shows the encoding of the observation function for the *rover* when it performs the action *Check*. By declaring the `<SubDAG>` of type `template` in line 10, it effectively means we are replacing it with lines 17–32. The full encoding of *RockSample* problem using `DD` type is given in Appendix B.

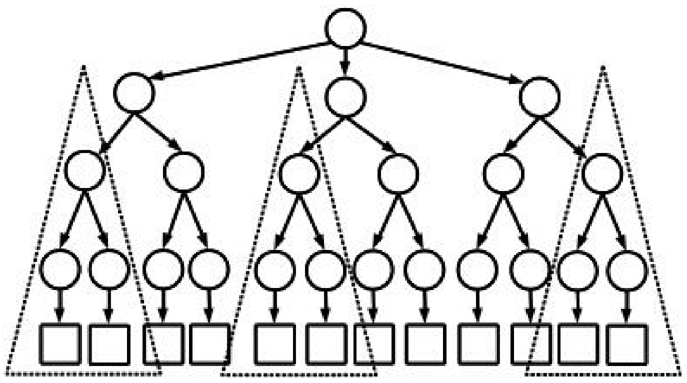

Figure 2.4 – Triangular portions of the decision diagram may use the same template definitions.

Example 21. Usage of `SubDAG type="template"`.

```
1. <Var>obs sensor</Var>
2. <Parent>action_rover rover_1 rock_1</Parent>
3. <Parameter type = "DD">
4.     <DAG>
5.         <Node var = "action_rover">
6.           ...
7.             <Edge val = "ac">
8.                 <Node var = "rover_1">
9.                     <Edge val = "s1">
10.                        <SubDAG type="template" idref="obs_rock"/>
11.                    </Edge>
12.                    ...
13.                </Node>
14.            </Edge>
15.        </Node>
16.    </DAG>
17. <SubDAGTemplate id = "obs_rock">
18.    <Node var="rock_1">
19.        <Edge val="good">
20.            <Node var="obs_sensor">
21.                <Edge val="ogood"><Terminal>0.8</Terminal></Edge>
22.                <Edge val="obad"><Terminal>0.2</Terminal></Edge>
23.            </Node>
24.        </Edge>
25.        <Edge val="bad">
26.            <Node var="obs_sensor">
27.                <Edge val="ogood"><Terminal>0.2</Terminal></Edge>
28.                <Edge val="obad"><Terminal>0.8</Terminal></Edge>
```

```
            </Node>
        </Edge>
    </Node>
</SubDAGTemplate>
</Parameter>
```

3. References

[1] S.C.W. Ong, S.W. Png, D. Hsu, and W.S. Lee. **POMDPs for robotic tasks with mixed observability**. In *Proc. Robotics: Science and Systems*, 2009.

[2] T. Smith and R. Simmons. Heuristic Search Value Iteration for POMDPs. In *Proc. Uncertainty in Artificial Intelligence*, 2004.

4. Appendix A

RockSample.pomdpx, `type="TBL"`

Full Specification of RockSample problem in PomdpX.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <pomdpx version="1.0" id="rockSample"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="pomdpx.xsd">
       <Description>RockSample problem for map size 1 x 3.
         Rock is at 0, Rover’s initial position is at 1.
         Exit is at 2.
       </Description>
       <Discount>0.95</Discount>
       <Variable>
            <StateVar vnamePrev="rover_0" vnameCurr="rover_1"
              fullyObs="true">
                <NumValues>3</NumValues>
            </StateVar>
            <StateVar vnamePrev="rock_0" vnameCurr="rock_1">
                <ValueEnum>good bad</ValueEnum>
            </StateVar>
            <ObsVar vname="obs_sensor">
                <ValueEnum>ogood obad</ValueEnum>
            </ObsVar>
            <ActionVar vname="action_rover">
                <ValueEnum>amw ame ac as</ValueEnum>
            </ActionVar>
            <RewardVar vname="reward_rover" />
       </Variable>
       <InitialStateBelief>
            <CondProb>
                <Var>rover_0</Var>
                <Parent>null</Parent>
                <Parameter type="TBL">
                      <Entry>
                          <Instance> - </Instance>
                          <ProbTable>0.0 1.0 0.0</ProbTable>
                      </Entry>
            </Parameter>
       </CondProb>
       <CondProb>
            <Var>rock_0</Var>
            <Parent>null</Parent>
            <Parameter type="TBL">
                <Entry>
                    <Instance>-</Instance>
                    <ProbTable>uniform</ProbTable>
                </Entry>
            </Parameter>
       </CondProb>
    </InitialStateBelief>
    <StateTransitionFunction>
        <CondProb>
            <Var>rover_1</Var>
            <Parent>action_rover rover_0</Parent>
```

```
                <Parameter type="TBL">
                    <Entry>
                        <Instance>amw s0 s2</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
                    <Entry>
                        <Instance>amw s1 s0</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
                    <Entry>
                        <Instance>ame s0 s1</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
                    <Entry>
                        <Instance>ame s1 s2</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
                    <Entry>
                        <Instance>ac s0 s0</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
                    <Entry>
                        <Instance>ac s1 s1</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
                    <Entry>
                        <Instance>as s0 s0</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
                    <Entry>
                        <Instance>as s1 s2</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
                    <Entry>
                        <Instance>* s2 s2</Instance>
                        <ProbTable>1.0</ProbTable>
                    </Entry>
            </Parameter>
        </CondProb>
        <CondProb>
            <Var>rock_1</Var>
            <Parent>action_rover rover_0 rock_0</Parent>
            <Parameter>
                <Entry>
                    <Instance>amw * - - </Instance>
                    <ProbTable>1.0 0.0 0.0 1.0</ProbTable>
                </Entry>
                <Entry>
                    <Instance>ame * - - </Instance>
                    <ProbTable>identity</ProbTable>
                </Entry>
                <Entry>
                    <Instance>ac * - - </Instance>
                    <ProbTable>identity</ProbTable>
                </Entry>
                <Entry>
                    <Instance>as * - - </Instance>
                    <ProbTable>identity</ProbTable>
                </Entry>
                <Entry>
                    <Instance>as s0 * - </Instance>
                    <ProbTable>0.0 1.0</ProbTable>
                </Entry>
            </Parameter>
        </CondProb>
    </StateTransitionFunction>
    <ObsFunction>
        <CondProb>
            <Var>obs sensor</Var>
            <Parent>action_rover rover_1 rock_1</Parent>
            <Parameter type="TBL">
                <Entry>
                    <Instance>amw * * - </Instance>
                    <ProbTable>1.0 0.0</ProbTable>
                </Entry>
                <Entry>
                    <Instance>ame * * - </Instance>
                    <ProbTable>1.0 0.0</ProbTable>
                </Entry>
                <Entry>
                    <Instance>as * * - </Instance>
                    <ProbTable>1.0 0.0</ProbTable>
                </Entry>
                <Entry>
                    <Instance>ac s0 - - </Instance>
                    <ProbTable>1.0 0.0 0.0 1.0</ProbTable>
                </Entry>
                <Entry>
                    <Instance>ac s1 - - </Instance>
                    <ProbTable>0.8 0.2 0.2 0.8</ProbTable>
                </Entry>
                 <Entry>
                      <Instance>ac s2 * - </Instance>
                      <ProbTable>1.0 0.0</ProbTable>
                 </Entry>
             </Parameter>
         </CondProb>
     </ObsFunction>
     <RewardFunction>
         <Func>
             <Var>reward_rover</Var>
             <Parent>action_rover rover_0 rock_0</Parent>
             <Parameter type="TBL">
                 <Entry>
                      <Instance>ame s1 *</Instance>
                      <ValueTable>10</ValueTable>
                 </Entry>
                 <Entry>
                      <Instance>amw s0 *</Instance>
                      <ValueTable>-100</ValueTable>
                 </Entry>
                 <Entry>
                      <Instance>as s1 *</Instance>
                      <ValueTable>-100</ValueTable>
                 </Entry>
                 <Entry>
                      <Instance>as s0 good</Instance>
                      <ValueTable>10</ValueTable>
                 </Entry>
                 <Entry>
                      <Instance>as s0 bad</Instance>
                      <ValueTable>-10</ValueTable>
                 </Entry>
             </Parameter>
         </Func>
     </RewardFunction>
 </pomdpx>
```

5. Appendix B

RockSample.pomdpx, `type="DD"`

Full Specification of RockSample problem in PomdpX.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pomdpx xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="0.1" id="rockSample"
\\xsi:noNamespaceSchemaLocation="pomdpx.xsd">
    <Description>RockSample problem for map size 1 x 3.
        Rock is at 0, Rover’s initial position is at 1.
        Exit is at 2.
    </Description>
    <Discount>0.95</Discount>
    <Variable>
        <StateVar vnamePrev="rover_0" vnameCurr="rover_1" fullyObs="true">
            <NumValues>3</NumValues>
        </StateVar>
        <StateVar vnamePrev="rock_0" vnameCurr="rock_1">
            <ValueEnum>good bad</ValueEnum>
        </StateVar>
        <ObsVar vname="obs_sensor">
            <ValueEnum>ogood obad</ValueEnum>
        </ObsVar>
        <ActionVar vname="action rover">
            <ValueEnum>amw ame ac as</ValueEnum>
        </ActionVar>
        <RewardVar vname="reward rover"/>
    </Variable>
    <InitialStateBelief>
        <CondProb>
            <Var>rover 0 rock 0</Var>
            <Parent>null</Parent>
            <Parameter type="DD">
                <DAG>
                    <Node var="rover 0">
                        <Edge val="s0">
                            <Terminal>0.0</Terminal>
                        </Edge>
                        <Edge val="s1">
                            <SubDAG type="uniform" var="rock 0"/>
                        </Edge>
                        <Edge val="s2">
                            <Terminal>0.0</Terminal>
                        </Edge>
                    </Node>
                </DAG>
            </Parameter>
        </CondProb>
    </InitialStateBelief>
    <RewardFunction>
        <Func>
            <Var>reward rover</Var>
            <Parent>action rover rover 0 rock 0</Parent>
            <Parameter type="DD">
                <DAG>
                    <Node var="action rover">
                        <Edge val="amw">
                            <Node var="rover 0">
                                <Edge val="s0">
                                    <Terminal>-100.0</Terminal>
                                </Edge>
                                <Edge val="s1">
                                    <Terminal>0.0</Terminal>
                                </Edge>
                                <Edge val="s2">
                                    <Terminal>0.0</Terminal>
                                </Edge>
                            </Node>
                        </Edge>
                        <Edge val="ame">
                            <Node var="rover 0">
                                <Edge val="s0">
                                    <Terminal>0.0</Terminal>
                                </Edge>
                                <Edge val="s1">
                                    <Terminal>10.0</Terminal>
                                </Edge>
                                <Edge val="s2">
                                    <Terminal>0.0</Terminal>
                                </Edge>
                            </Node>
                        </Edge>
                        <Edge val="ac">
                            <Terminal>0.0</Terminal>
                        </Edge>
                        <Edge val="as">
                            <Node var="rover 0">
                                <Edge val="s0">
                                    <Node var="rock 0">
                                        <Edge val="good">
                                            <Terminal>10</Terminal>
                                        </Edge>
                                        <Edge val="bad">
                                            <Terminal>-10</Terminal>
                                        </Edge>
                                    </Node>
                                </Edge>
                                <Edge val="s1">
                                    <Terminal>-100</Terminal>
                                </Edge>
                                <Edge val="s2">
                                    <Terminal>-100</Terminal>
                                </Edge>
                            </Node>
                        </Edge>
                    </Node>
                </DAG>
            </Parameter>
        </Func>
    </RewardFunction>
    <ObsFunction>
        <CondProb>
            <Var>obs sensor</Var>
            <Parent>action rover rover 1 rock 1</Parent>
            <Parameter type="DD">
                <DAG>
                    <Node var="action rover">
                        <Edge val="amw">
                            <SubDAG type="deterministic" var="obs sensor" val="ogood"/>
                        </Edge>
                        <Edge val="ame">
                            <SubDAG type="deterministic" var="obs sensor" val="ogood"/>
                        </Edge>
                        <Edge val="ac">
                            <Node var="rover 1">
                                <Edge val="s0">
                                    <Node var="rock 1">
                                        <Edge val="good">
                                            <SubDAG type="deterministic" var="obs sensor" val="ogood"/>
                                        </Edge>
                                        <Edge val="bad">
                                            <SubDAG type="deterministic" var="obs sensor" val="obad"/>
                                        </Edge>
                                    </Node>
                                </Edge>
                                <Edge val="s1">
                                    <SubDAG type="template" idref="obs rock"/>
                                </Edge>
                                <Edge val="s2">
                                    <SubDAG type="template" idref="obs rock"/>
                                </Edge>
                            </Node>
                        </Edge>
                        <Edge val="as">
                            <SubDAG type="deterministic" var="obs sensor" val="ogood"/>
                        </Edge>
                    </Node>
                </DAG>
                <SubDAGTemplate id="obs rock">
                    <Node var="rock 1">
                        <Edge val="good">
                            <Node var="obs sensor">
                                <Edge val="ogood">
                                    <Terminal>0.8</Terminal>
                                </Edge>
                                <Edge val="obad">
                                    <Terminal>0.2</Terminal>
                                </Edge>
                            </Node>
                        </Edge>
                        <Edge val="bad">
                            <Node var="obs sensor">
                                <Edge val="ogood">
                                    <Terminal>0.2</Terminal>
                                </Edge>
                                <Edge val="obad">
                                    <Terminal>0.8</Terminal>
                                </Edge>
                            </Node>
                        </Edge>
                    </Node>
                </SubDAGTemplate>
            </Parameter>
        </CondProb>
    </ObsFunction>
    <StateTransitionFunction>
```

```
<CondProb>
    <Var>rover 1</Var>
    <Parent>action rover rover 0</Parent>
    <Parameter type="DD">
        <DAG>
            <Node var="action rover">
                <Edge val="amw">
                    <Node var="rover 0">
                        <Edge val="s0">
                            <SubDAG type="deterministic" var="rover 1" val="s2"/>
                        </Edge>
                        <Edge val="s1">
                            <SubDAG type="deterministic" var="rover 1" val="s0"/>
                        </Edge>
                        <Edge val="s2">
                            <SubDAG type="deterministic" var="rover 1" val="s2"/>
                        </Edge>
                    </Node>
                </Edge>
                <Edge val="ame">
                    <Node var="rover 0">
                        <Edge val="s0">
                            <SubDAG type="deterministic" var="rover 1" val="s1"/>
                        </Edge>
                        <Edge val="s1">
                            <SubDAG type="deterministic" var="rover 1" val="s2"/>
                        </Edge>
                        <Edge val="s2">
                            <SubDAG type="deterministic" var="rover 1" val="s2"/>
                        </Edge>
                    </Node>
                </Edge>
                <Edge val="ac">
                    <SubDAG type="persistent" var="rover 1"/>
                </Edge>
                <Edge val="as">
                    <Node var="rover 0">
                        <Edge val="s0">
                            <SubDAG type="deterministic" var="rover 1" val="s0"/>
                        </Edge>
                        <Edge val="s1">
                            <SubDAG type="deterministic" var="rover 1" val="s2"/>
                        </Edge>
                        <Edge val="s2">
                            <SubDAG type="deterministic" var="rover 1" val="s2"/>
                        </Edge>
                    </Node>
                </Edge>
            </Node>
        </DAG>
    </Parameter>
</CondProb>
<CondProb>
    <Var>rock 1</Var>
    <Parent>action rover rover 0 rock 0</Parent>
    <Parameter type="DD">
        <DAG>
            <Node var="action rover">
                <Edge val="amw">
                    <SubDAG type="persistent" var="rock 1"/>
                </Edge>
                <Edge val="ame">
                    <SubDAG type="persistent" var="rock 1"/>
                </Edge>
                <Edge val="ac">
                    <SubDAG type="persistent" var="rock 1"/>
                </Edge>
                <Edge val="as">
                    <Node var="rover 0">
                        <Edge val="s0">
                            <SubDAG type="deterministic" var="rock 1" val="bad"/>
                        </Edge>
                        <Edge val="s1">
                            <SubDAG type="persistent" var="rock 1"/>
                        </Edge>
                        <Edge val="s2">
                            <SubDAG type="persistent" var="rock 1"/>
```

```
                                </Edge>
                            </Node>
                        </Edge>
                    </Node>
                </DAG>
            </Parameter>
        </CondProb>
    </StateTransitionFunction>
</pomdpx>
```

Retrieved from http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.PomdpXDocumentation
Page last modified on September 22, 2011, at 08:54 AM