cisc3665, fall 2011 / prof sklar
Lab V.2: Behaviors

Overview

This is the assignment for unit V, Game Theory and Behaviors, and should be submitted for a grade.

The purpose of this lab is to give you a chance to apply the concepts of deterministic and non-deterministic behaviors that we have discussed in lectures for unit V.

This assignment is worth **8 points**, or 8% of your term grade.

This assignment is due on DECEMBER 19, by 6AM.

Refer to the lecture notes and the readings from November 30 (Deterministic Behaviors) and December 5 (Non-deterministic Behaviors). These are posted on the class web page.

Record your answers on paper and bring a hardcopy to the final exam OR submit your answers electronically, using the electronic submission system.

1 Rule-based Behaviors

1.1 Below we explain how to perform inference ("decising making") using a rule base, in more detail than the lecture slides from November 30. Given a rule base, there are essentially two ways we can use the rules to make decisions ("perform inference"): *forward chaining*, which is data driven; and *backward chaining*, which is goal driven.
We start with the following assumptions:
Let `(c,a)` be a rule, as in `IF (c) THEN (a)`.
Let `fires(c,WM)` be true if condition c fires against (i.e., matches something in) working memory `WM`.

1.2 **Forward Chaining.** The *forward chaining algorithm* is as follows:

```
var WM : set of facts
var goal : goal we are searching for
var RuleBase : set of rules
var firedFlag : BOOLEAN
repeat
  firedFlag = FALSE
  for each (c,a) in RuleBase do
    if fires(c,WM) then
      if a == goal then
        return success
      end-if
      add a to WM
      set firedFlag to TRUE
    end-if
  end-for
until firedFlag = FALSE
return failure
```

For example, given the following:

```
RuleBase = { (princess sees obstacle ahead,princess moves away from obstacle),
             (princess sees frog ahead,princess moves next to frog),
             (princess moves next to frog,princess kisses frog),
             (princess kisses frog,frog turns into prince) }
```

```
WM = { (princess sees frog ahead) }
goal = { (frog turns into prince) }
firedFlag = FALSE
```

We can walk through the forward chaining algorithm as follows.

The first rule (c,a) in the RuleBase is (princess sees obstacle ahead,princess moves away from obstacle).
This rule does not fire because (princess sees obstacle ahead) is not in WM.

The second rule in the RuleBase is (princess sees frog ahead,princess moves next to frog).
This rule does fire because (princess sees frog ahead) is in WM.
We check to see if (princess moves next to frog) is equal to the goal (frog turns into prince).
It is not, so we add (princess moves toward frog) to WM:
WM = { (princess sees frog ahead),(princess moves next to frog) }
and we set firedFlag=TRUE.

Then we continue with the for each (c,a)... loop.
The next rule in the RuleBase is (princess moves next to frog,princess kisses frog).
This rule fires because (princess moves next to frog) is in WM.
We check to see if (princess kisses frog) is equal to the goal.
It is not, so we add to WM:
WM = { (princess sees frog ahead),(princess moves next to frog),(princess kisses frog) }
and we set firedFlag=TRUE.

Then we continue with the for each (c,a)... loop.
The next rule in the RuleBase is (princess kisses frog,frog turns into prince).
This rule fires because (princess kisses frog) is in WM.
We check to see if (frog turns into prince) is equal to the goal.
It is! So we return success.

1.3 **Backward Chaining.** Backward chaining means reasoning from *goals* back to *facts*. The idea is that this focuses the inference process. Thinking of the rules as building a tree connecting facts, in backward chaining, every path ends with the goal. Since, in general, there are more initial facts that goals, more of the paths built will be solutions than in forward chaining. The *backward chaining algorithm* is as follows:

```
var WM : set of facts
var RuleBase : set of rules
var firedFlag : BOOLEAN
function prove( g : goal )
  if g in WM then
    return TRUE
  if there is some (c,a) in WM such that a == g then
     for each precondition p in c do
        if not prove(p) then
          return FALSE
     return TRUE
  else
     return FALSE
end-function
```

For example, given the same problem as above, we can walk through the backward chaining algorithm as follows.

The goal (frog turns into prince) is not in WM, so we look for a rule that matches the goal.
We find (princess kisses frog,frog turns into prince),
and we recursively call prove((princess kisses frog)).

The goal (princess kisses frog) is not in WM, so we look for a rule that matches the goal.
We find (princess moves next to frog,princess kisses frog),
and we recursively call prove((princess moves next to frog)).

The goal (princess moves next to frog) is not in WM, so we look for a rule that matches the goal.
We find (princess sees frog ahead,princess moves next to frog),
and we recursively call prove((princess sees frog ahead)).

The goal (princess sees frog ahead) is in WM, so we update WM:
WM = { (princess sees frog ahead),(princess moves next to frog) }
and return TRUE to recursive call.

Inside recursive call, we update WM:
WM = { (princess sees frog ahead),(princess moves next to frog),(princess kisses frog) }
and return TRUE to recursive call.

Inside recursive call, we update WM:+
```
WM = { (princess sees frog ahead),(princess moves next to frog),(princess kisses frog),
       (frog turns into prince) }
```
and return TRUE (end of recursive calls).

Now it's your turn!

1.4 Forward Chaining

(1 point)

Given the following:

```
RuleBase = { (Kermit sees obstacle ahead,Kermit moves away from obstacle),
             (Kermit sees monster ahead,Kermit moves away from monster),
             (Kermit sees pond ahead,Kermit moves toward pond),
             (Kermit moves toward pond,Kermit sees fly),
             (Kermit sees fly,Kermit eats fly) }
WM = { (Kermit sees monster ahead) }
goal = { (Kermit eats fly) }
```

As I did above, "walk through" the FORWARD CHAINING algorithm to achieve the goal. Make sure you record all your steps, as I did above.

1.5 Backward Chaining

(1 point)

Walk through the BACKWARD CHAINING algorithm to achieve the goal, using the same Rule Base, and working memory and goal in the previous question. Make sure you record all your steps, as I did.

1.6 Predictive Reasoning

Given the following game scenario:
You need to get some food to eat in order to earn health points. The only food you can eat is in an orchard, but you're not sure if there is any food left in the orchard. The orchard is across the river, but you're not sure how to get to the orchard from the river. The NPC must take a boat to get across the river, but the boat might capsize while crossing the river.

Given the following probabilities:

$$\begin{aligned} Pr(food|orchard) &= 90\% \\ Pr(\neg food|orchard) &= 10\% \\ Pr(food|\neg orchard) &= 3\% \\ Pr(orchard|river) &= 75\% \\ Pr(orchard|\neg river) &= 1\% \\ Pr(river|boat) &= 50\% \\ Pr(river|\neg boat) &= 45\% \end{aligned}$$

where:
$Pr(food)$ represents the probability that you get food.
$Pr(orchard)$ represents the probability that you get to the orchard.
$Pr(river)$ represents the probability that you cross the river.
$Pr(boat)$ represents the probability that you get in the boat.

(1.6.a) **Draw the causal chain to represent this problem space.** *(1 point)*

(1.6.b) **Compute the probability that you get food, given that you get in the boat: $Pr(food|boat)$.** *(1 point)*

1.7 Diagnostic Reasoning

Given the following game scenario:
You need to get some gold in order to stay alive in the game. The gold is guarded by a monster in a tunnel. The monster has a baby, and she is very protective of her baby. Sometimes, the monster falls asleep, and you can sneak into the tunnel and grab some gold. Sometimes, the monster sleeps with her baby. Sometimes, the monster goes to the outhouse, and leaves the gold and the baby unguarded.

Given the following probabilities:

$$\begin{aligned} Pr(asleep) &= 20\% \\ Pr(gold|asleep) &= 90\% \\ Pr(gold|\neg asleep) &= 15\% \\ Pr(baby|asleep) &= 25\% \\ Pr(baby|\neg asleep) &= 2\% \end{aligned}$$

where:
$Pr(asleep)$ represents the probability that the monster is sleeping
$Pr(gold)$ represents the probability that the gold is left unguarded
$Pr(baby)$ represents the probability that the monster's baby is left unguarded

(1.7.a) **Draw the common cause network to represent this problem space.** *(1 point)*

(1.7.b) **Compute the probability that the gold is left unguarded,** $Pr(gold)$. *(0.5 points)*

(1.7.c) **Compute the probability that the baby is left unguarded,** $Pr(baby)$. *(0.5 points)*

(1.7.d) **BONUS QUESTION FOR EXTRA CREDIT:**
If you know that monster is asleep, how do these probabilities ($Pr(gold)$ **and** $Pr(baby)$**) change?** *(1 point)*

1.8 Explaining away

Given the following game scenario:
You are exploring a hostile planet, and you encounter a nasty alien. In order to stay alive, you must kill the alien. You can do that by using either poison or your lightsaber. However, you are unsure of whether the poison you have will disable the alien. You are also a bit wobbly with your lightsaber and don't always kill your opponent when you fight using it.

Given the following probabilities:

$$\begin{aligned} Pr(alien|lightsaber \wedge poison) &= 87\% \\ Pr(alien|\neg lightsaber \wedge poison) &= 12\% \\ Pr(alien|lightsaber \wedge \neg poison) &= 68\% \\ Pr(alien|\neg lightsaber \wedge \neg poison) &= 5\% \\ Pr(lightsaber) &= 77\% \\ Pr(poison) &= 10\% \end{aligned}$$

where:
$Pr(alien)$ is the probability that you will kill the alien
$Pr(lightsaber)$ is the probability that you will attack the alien with your light saber
$Pr(poison)$ is the probability that you will attack the alien with poison

(1.8.a) **Draw the common effect network to represent this problem space.** *(1 point)*

(1.8.b) **Compute the probability that you will kill the alien,** $Pr(alien)$. *(1 point)*