CS550: Information State

For this homework, you will augment a dialogue manager for a banking system, built with the information state approach. In the information state approach, you need an IS engine and the system’s update rules.

You will be given a simple version of an IS engine that is implemented in Tcl. It uses a simple control strategy in which it runs understanding rules, deliberation rules, and action selection rules (when it has the turn). The first applicable understanding rule is applied, all applicable deliberation rules are applied, and the first applicable action section rule is applicable.

A number of IS variables are used by the update rules to communicate to the IS engine. These are:

**LastMove**: What was last said.

**LastSpeaker**: Who was the last speaker.

**NextMove**: What should the system say next.

**HaveTurn**: Should the system keep the turn after its current turn.

You will also be given the system’s update rules that implements the base functionality of the system. Through the questions in this homework, you will add additional functionality, including limited mixed initiative. Implementing the limited mixed initiative should help demonstrate how the IS approach lends itself towards building advanced dialogue systems.

For simplicity, we will not be including an ASR, parser, semantic interpreter, or speech synthesizer. So, the user will be interacting with the program by typing in and reading speech acts, rather than fully-formed English sentences or speech.

The initial set of update rules was given in the class lecture notes, and will allow you to have a very basic conversation, like the one in Table 1.

<table>
<thead>
<tr>
<th>System: query TransType</th>
<th>User: transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>System: query Amount</td>
<td>User: 50</td>
</tr>
<tr>
<td>System: query From</td>
<td>User: checking</td>
</tr>
<tr>
<td>System: query To</td>
<td>User: savings</td>
</tr>
<tr>
<td>System: bye</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Transcript

The code for the IS engine **ISEngine.tcl** and the initial system rules **ISAgent.tcl** are given in the course web site. Look over the initial system rules to make sure you understand what is going on and why. Try out the system by executing **ISAgent.tcl**, which in turn sources in **ISEngine.tcl**. You will see that it puts up two windows. The first is a console that shows a lot of debugging information that shows you what the IS engine just did. The second is a window in which the user can see what the system’s utterance was, and where the user can type in their response and press send.

For the above dialogue, you should be able to trace all of the update rules that were applied. This is shown in Table 2 for the first part of the dialogue from Table 1. For each update rule, I indicate if it is an understanding rules with ‘u’, deliberation rules with ‘d’, and action rules with ‘a’. I have also inserted columns for the names of variables so that you can use this table for the first question.

Also, the code in **ISEngine.tcl** is fairly simple. Look it over so that you can understand how it works.
<table>
<thead>
<tr>
<th>Update Rule</th>
<th>LastMove</th>
<th>QUD</th>
<th>QUD spkra</th>
<th>Agenda</th>
<th>NextMove</th>
</tr>
</thead>
<tbody>
<tr>
<td>d setup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a ask</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S query TransType</td>
<td>u self-ask</td>
<td>query TransType</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U transfer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>u other-answer</td>
<td>transfer</td>
<td>transfer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d transfer-to</td>
<td>transfer</td>
<td>transfer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d transfer-from</td>
<td>transfer</td>
<td>transfer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d transfer-amount</td>
<td>transfer</td>
<td>transfer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a ask</td>
<td>transfer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S query Amount</td>
<td>u self-ask</td>
<td>query Amount</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Transcript with Update Rules
Question 1: Base System

Finish specifying all of the update rules in Figure 2. Also, show the value of the variables \texttt{LastMove}, \texttt{QUD}, \texttt{QUDspeakers}, \texttt{Agenda}, \texttt{NextMove} after each update rule is done. For

Question 2: Invalid Answers (Part 1)

If user makes an invalid answer, such as answering “checking” to the amount query, explain what happens. Make sure you address the following: Why does the understanding rule \texttt{other-answer} not fire? Is any understanding rule applied? Why doesn’t the \texttt{ask} action rule fire to ask about the next item on the agenda? What action rule does fire and why?

Question 3: Invalid Answers (Part 2)

Instead of the observed behavior, let’s change the system so that it will behave normally if correct answers are given, but will re-ask the last question if an invalid answer is given. Create an action rule called \texttt{reask} that will be applicable if (a) there is something on the QUD and (b) if the top question on the QUD is by the system. The rule should reask the top question. Explain where this rule needs to be placed with respect to the other action selection rules. Give the definition of the rule. Also explain why you do not need an understanding rule for understanding the re-ask.

Here are a couple of rules-of-thumb to keep in formulating action selection rules, understanding rules, and deliberation rules. The effects of an action selection rule should change \texttt{NextMove} and \texttt{HaveTurn} and no other variables, and no other rule types should change these variables. Also, the preconditions of understanding rules should check \texttt{LastMove} and perhaps \texttt{LastSpeaker}, and that no other rule types should check these variables.

Question 4: Invalid Answers (Part 3)

If user makes an invalid answer, rather than just re-asking the question, the system should inform the user that the answer was not understood, and then re-ask the question.

You should do this using 4 rules. The first is an understanding rule that will fire if the \texttt{other-answer} rule from the base code does not fire. Instead of checking if \texttt{AnswerMatch} is false in the preconditions, you can achieve this by careful placement of the understanding rule. However, explicitly state what other conditions this rule should have. The effect of this rule should be to add to the agenda “inform could not understand your response”.

The second rule is a generic action rule \texttt{inform} that will see if the top item on the agenda is an inform, and if so, make this the next move. It will also not release the turn.

The third rule is a generic understanding rule \texttt{self-inform} that will see if the system just asked an inform that is on the top of the agenda. As the inform has been asked, it should remove it from the agenda.

The forth rule is the re-ask rule that you already added in an earlier question.

Give the three new rules that you added, and explain whether the position of each rule matters. If it does matter, explain where it must be place.

Question 5: Starting Over

If the user says “start over”, the system should start over with the banking application. You should have an understanding that will simply determine if the last move was “start over”, and if so, the system should set
all of the domain variables to "", as well as the agenda, QUD and QUd speakers, so as to reset the dialogue state. By resetting the dialogue state, the other deliberation rules will be applied, which will restart the dialogue.

Writeup 5.1. Marks 3
Hand in a copy of the rule that you added and explain where it should be positioned.

Question 6: Alternate Version of Start Over

Alter the functionality of the system so that when the user says 'start over', the system informs the user that they are starting over “inform starting over”, and then, in the next utterance (without asking the user to respond), restarts the dialogue. There are a number of different ways that this can be done. Try to think of a way that is consistent with how your other rules are working, and that might also make use of some of those rules.

Writeup 6.2. Marks 3
Hand in the code for your rules and explain what happens after the user says 'start over', in terms of what rules are applied. Also, explain the rationale for your solution.

Question 7: Checking the balance of the from-account

After the amount and from-account are known, you should check the balance of the from account to make sure it has enough money. If it doesn’t, you should inform the user of this, and reprompt for the amount.

Implement the above by adding a deliberation rule. If the amount in the from-account is not enough, add the following onto the agenda of “inform balance of from is only balance”, and set the amount to "".

Note that we want to make sure that the user is informed of the error before being re-asked for the account.

Note that you will be making use of the action selection rule inform that you added in the previous question as well as the understanding rule self-inform, and so only the deliberation rule is needed to solve this question.

In doing the above, you will want to check the value of something like is(balance(checking)). For some reason, Tcl has a problem processing associative array indices that look like associative array references. Try the following in a tclsh.

```tcl
set is(balance(checking)) 100
puts $is(balance(checking))
```

This ‘bug’ will make it difficult to use structured data in your information state. We can get around this problem by doing the following when trying to access of such an index.

```tcl
set is(balance(checking)) 100
puts $is([list balance(checking)])
```

Writeup 7.3. Marks 3
Hand in the rule that you added. Explain where you positioned the rule and why. Also, explain the overall set of rules that are executed in deal with an unallowed amount, from understanding the user’s unallowed amount up to when the system reprompts for the amount. Explain how the agenda is changing with each rule.

Question 8: User asking question

You will need a rule to understand the user’s question. This will be similar to the self-ask understanding rule; however, you should add an extra precondition to ensure that the question is one that the system can answer. For simplicity, allow the system to answer any question about what is in its information state.
You will also need an action rule for answering the question. Make sure that the preconditions only allow
the system to answer a **query** from **other**. Also, have the system keep the turn, which should force it to
automatically re-ask the previous question.

Also, set up an understanding rule for understanding the system’s answer. This is where you should update
the QUD and QUDspeakers to remove the user’s question.

**Question 9: User asking an invalid question**

Add an understanding rule to deal with the case where the user asks an invalid question. The effect of this
rule should be to add “inform could not understand” to the agenda. Note that the generic inform action
will take care of the rest, and that the system will automatically re-ask its previous question.