CS 289	Knowledge representation and reasoning	
Fall 1995	Stuart Russell	Answers, Weeks 1–3

2.1 A performance measure is used by an outside observer to evaluate how successful an agent is. It is a function from histories to a real number. A utility function is used by an agent itself to evaluate how desirable states or histories are. In our framework, the utility function may not be the same as the performance measure; furthermore, an agent may have no explicit utility function at all, whereas there is always a performance measure.

2.3 We'll choose the domain of writing and debugging a computer program.

We'll take the percepts to be strings of characters, whether transmitted on a stream or displayed on a screen. Some of these represent the specification for the problem, others represent the result of actions. The actions are creating and editing files, running the compiler, and running the resulting program. At a low level, these actions all involve transmitting a stream of characters. The goal is to create a program that, with high probability, satisfies the specification (or at least most of it). The environment responds to the compilation action by generating a list of errors and warnings, and perhaps a generated binary file, and responds to the run-program action by generating the output of the program.

The environment is highly inaccessible. The agent sees only a character (or screen) at a time, whereas the relevant state may include all of the memory and file system. This means that the agent must maintain a lot of internal state as it interacts with the computer. Fortunately, most systems provide nondestructive commands that can be used to reveal the system's state, although some aspects (such as the contents of the cache) are very hard to ascertain. The environment is non-deterministic because of other users that may be competing for the computer's and the network's resources, and perhaps altering files, etc. Also because of the possibility that a component will fail and produce erroneous outputs (or no output at all). It is non-episodic because rewards are obtained only at the end of a long (sometimes seemingly infinite) sequence of actions. It is dynamic: other users can modify files, etc., and there is usually some cost of time. Digital computers are discrete, but some peripheral devices may be continuous.

6.1 Given a sentence S, construct a truth table with one row for each possible combination of truth values for the propositions in S, such that the last column of the table is S itself. If this last column has all true values, then the sentence is valid; if it has any true values then it is satisfiable; if all values are false then it is unsatisfiable.

6.3

We use the `validity` function from `logic/prop.lisp` to determine the validity of each sentence:

```
> (validity "Smoke => Smoke")
VALID
> (validity "Smoke => Fire")
SATISFIABLE
> (validity "(Smoke => Fire) => (~Smoke => ~Fire)")
SATISFIABLE
> (validity "Smoke | Fire | ~Fire")
VALID
```

6.5 From the first two statements, we see that if it is mythical, then it is immortal; otherwise it is a mammal. So it must be either immortal or a mammal, and thus horned. That means it is also magical. However, we can't deduce anything about whether it is mythical. Using the propositional reasoning code:

```
> (setf kb (make-prop-kb))
#S(PROP-KB SENTENCE (AND))
> (tell kb "Mythical => Immortal")
T
> (tell kb "~Mythical => ~Immortal ^ Mammal")
T
> (tell kb "Immortal | Mammal => Horned")
T
> (tell kb "Horned => Magical")
T
> (ask kb "Mythical")
NIL
> (ask kb "~Mythical")
NIL
> (ask kb "Magical")
T
> (ask kb "Horned")
T
```

The failure to prove anything about mythical, given that we have a complete inference procedure, means that there must be models of the KB where mythical is true and models where it is false. This can be seen by inspection of the truth table for the KB.

6.7 There are 4 models for (a), corresponding to the four possible truth values for C and D. Similarly, there are only 2 models for (c), one where D is true, and one where D is false. There are 12 models for (b), because there are 4 possible values for C and D, multiplied by 3 ways in which $A \vee B$ can be made true.

The main point is that as you add constraints (e.g., by going from $A \wedge B$ to $A \wedge B \wedge C$ you get fewer models, and as you relax constraints (e.g., by going from $A \wedge B$ to $A \vee B$) you get more models.

7.2 In this exercise, it is best not to worry about details of tense and larger concerns with consistent ontologies and so on. The main point is to make sure you understand connectives and quantifiers and the use of predicates, functions, constants, and equality. Let the basic vocabulary be as follows:
$Takes(x, c)$: student x takes course c;
$Fails(x, c)$: student x fails course c;
$Score(x, c)$: the score obtained by student x in course c;
H: a specific history course; B: a specific biology course;
$Dislikes(x, y)$: person x dislikes person y
$Shaves(x, y)$: person x shaves person y
$Student(x)$, $Person(x)$, $Man(x)$, $Woman(x)$, $Barber(x)$, $Vegetarian(x)$, $Smart(x)$, $Professor(x)$, $Politician(x)$: predicates satisfied by members of the corresponding categories.
$Fools(x, y, t)$: person x fools person y at time t.

a. Not all students take both History and Biology.
$\neg\forall x \;\; Student(x) \Rightarrow (Takes(x, H) \land Takes(x, B))$.

b. Only one student failed History.
$\exists x \;\; Student(x) \land Fails(x, H) \land \forall y \;\; Student(y) \land Fails(x, H) \Rightarrow x = y$.

c. Only one student failed both History and Biology.
$\exists x \;\; Student(x) \land Fails(x, H) \land Fails(x, B) \land \forall y \;\; Student(y) \land Fails(x, H) \land Fails(x, B) \Rightarrow x = y$.

d. The best score in History was better than the best score in Biology.
$\exists x \;\; \forall y \;\; Score(x, H) > Score(y, B)$.

e. Every person who dislikes all vegetarians is smart.
$\forall x \;\; Person(x) \land (\forall y \;\; Vegetarian(y) \Rightarrow Dislikes(x, y)) \Rightarrow Smart(x)$.

f. No person likes a smart vegetarian.
$\forall x, y \;\; Person(x) \land Smart(y) \land Vegetarian(y) \Rightarrow \neg Likes(x, y)$.

g. There is a woman who likes all men who are not vegetarians.
$\exists x \;\; Woman(x) \land \forall y \;\; Man(y) \land \neg Vegetarian(y) \Rightarrow Likes(x, y)$.

h. There is a barber who shaves all men in town who do not shave themselves.
$\exists x \;\; Barber(x) \land \forall y \;\; Man(y) \land \neg Shaves(y, y) \Rightarrow Shaves(x, y)$.

i. No person likes a professor unless the professor is smart.
$\forall x, y \;\; Person(x) \land Professor(y) \land \neg Smart(y) \Rightarrow \neg Likes(x, y)$.

j. Politicians can fool some of the people all of the time, and they can fool all of the people some of the time, but they can't fool all of the people all of the time.

$$\begin{aligned}
&\forall x \;\; Politician(x) \Rightarrow \\
&\quad (\exists y \;\; \forall t \;\; Person(y) \land Fools(x, y, t)) \land \\
&\quad (\exists t \;\; \forall y \;\; Person(y) \Rightarrow Fools(x, y, t)) \land \\
&\quad \neg(\forall t \;\; \forall y \;\; Person(y) \Rightarrow Fools(x, y, t))
\end{aligned}$$

7.11 This question takes you through the initial stages of developing a logical representation for actions that incorporates more and more realism. Implementing the reasoning tasks in a theorem-prover is also a good idea. Although the use of logical reasoning for the initial task—finding a route on a graph—may seem like overkill, it is interesting that we can keep making the situation more complicated simply by describing those added complications, with no additions to the reasoning system.

a. $At(Robot, Arad, S_0)$.

b. $\exists\, s \;\; At(Robot, Bucharest, s)$.

c. The successor-state axiom should be mechanical by now:

$$\begin{aligned}\forall\, a, x, y, s \;\; At(Robot, y, Result(a, s)) \;\; \Leftrightarrow \;\; & [(a = Go(x, y) \\ & \wedge DirectRoute(x, y) \wedge At(Robot, x, s)) \\ \vee \;\; & (At(Robot, y, s) \\ & \wedge \neg(\exists\, z \;\; a = Go(y, z) \wedge z \neq y))]\end{aligned}$$

d. To represent the amount of fuel the robot has in a given situation, use the function $Fuel(Robot, s)$. Let $Distance(x, y)$ denote the distance between cities x and y, measured in units of fuel consumption. Let $Full$ be a constant denoting the fuel capacity of the tank.

e. The initial situation is described by $At(Robot, Arad, S_0) \wedge Fuel(Robot, s) = Full$. The above axiom for location is extended as follows (note that we do not say what happens if the robot runs out of gas):

$$\begin{aligned}\forall\, a, x, y, s \;\; At(Robot, y, Result(a, s)) \;\; \Leftrightarrow \;\; & [(a = Go(x, y) \\ & \wedge DirectRoute(x, y) \wedge At(Robot, x, s) \\ & \wedge Distance(x, y) < Fuel(Robot, s)) \\ \vee \;\; & (At(Robot, y, s) \\ & \wedge \neg(\exists\, z \;\; a = Go(y, z) \wedge z \neq y))] \\ \forall\, a, x, y, s \;\; Fuel(Robot, Result(a, s)) = f \;\; \Leftrightarrow \;\; & [(a = Go(x, y) \\ & \wedge DirectRoute(x, y) \wedge At(Robot, x, s) \\ & \wedge Distance(x, y) < Fuel(Robot, s) \\ & \wedge f = Fuel(Robot, s) - Distance(x, y)) \\ \vee \;\; & (f = Fuel(Robot, s) \\ & \wedge \neg(\exists\, v, w \;\; a = Go(v, w) \wedge v \neq w))]\end{aligned}$$

f. The simplest way to extend the representation is to add the predicate $GasStation(x)$, which is true of cities with gas stations. The $Fillup$ action is described by adding another clause to the above axiom for $Fuel$, saying that $f = Full$ when $a = FillUp$.