COMP3620/COMP6320 Artificial Intelligence
Tutorial 3: Knowledge Representation and Reasoning

April 17-20, 2018

Exercise 1

Define the standard connectives in terms of $\rightarrow$ and $\bot$.

Answer

The key point is that $\neg A$ is definable as $A \rightarrow \bot$. Then standard equivalences can be used to get $\wedge$ and $\vee$ using $\neg$ as required. In detail:

$$\begin{array}{rlll} \neg A & \text{is} & A \rightarrow \bot & \\ A \vee B & \text{is} & \neg A \rightarrow B & \text{i.e.} \quad (A \rightarrow \bot) \rightarrow B \\ A \wedge B & \text{is} & \neg(A \rightarrow \neg B) & \text{i.e.} \quad (A \rightarrow (B \rightarrow \bot)) \rightarrow \bot \end{array}$$

Exercise 2

Put into clause form and refute by resolution:

$$\begin{array}{rcll} \Gamma & = & \{ & \forall x(\neg Q(x) \rightarrow P(x)), \\ & & & \neg\exists y\, P(y), \\ & & & Q(a) \rightarrow \exists x(R(x) \wedge \neg Q(x)) \quad \} \end{array}$$

Answer

First get the quantifiers to the front (prenex normal form):

$$\begin{array}{l} \forall x(\neg Q(x) \rightarrow P(x)), \\ \forall y\, \neg P(y), \\ \exists x(Q(a) \rightarrow (R(x) \wedge \neg Q(x))) \end{array}$$

Next remove the existential quantifier and use a name (skolem constant) instead:

$$\begin{array}{l} \forall x(\neg Q(x) \rightarrow P(x)), \\ \forall y\, \neg P(y), \\ Q(a) \rightarrow (R(b) \wedge \neg Q(b)) \end{array}$$

Delete the universal quantifiers, and put the propositional parts into clause form:

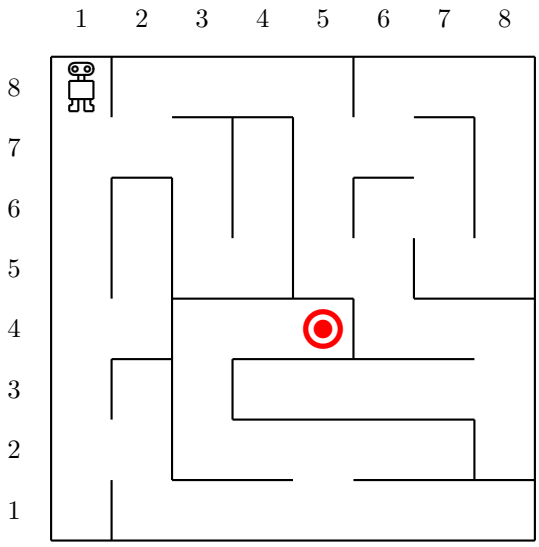
$$\begin{array}{rcll} \Delta & = & \{ & Q(x) \vee P(x), \\ & & & \neg P(y), \\ & & & \neg Q(a) \vee R(b) \\ & & & \neg Q(a) \vee \neg Q(b) \quad \} \end{array}$$

A resolution derivation could be written something like this:

1.	$Q(x) \vee P(x)$	given	
2.	$\neg P(y)$	given	
3.	$Q(x)$	from 1, 2	unifier $\{y \leftarrow x\}$
4.	$\neg Q(a) \vee \neg Q(b)$	given	
5.	$\neg Q(b)$	from 3, 4	$\{x \leftarrow a\}$
6.	$\bot$	from 3, 5	$\{x \leftarrow b\}$

Exercise 3

The following is a simple gridworld problem in the form of a maze:



The robot in row 8, column 1 needs to find its way to the goal at row 4, column 5. The actions available to it in any given state are to move one square up, down, left or right—but it cannot exit the grid or walk through the walls.

Decide what is involved in describing this problem purely in terms of logic. The best approach is to think of some things you need to say, then see what vocabulary (predicates, function symbols, names) you need for that. The questions to be considered include:

- What objects are in the domain of discourse?
- What is invariant between states, and what vocabulary is required to describe it?
- What is required to define a specific state?
- How can we represent logically the relationship between an arbitrary state and its successor, and relate that to the possible actions?

Do not forget to include in the problem description the specification of the initial state and the goal.

For problems like this, it is convenient to use functions as well as relations. For example, the position of the robot in a state is a function of the state, not just a relation between the state and various places in the grid.

You can assume some basic properties of numbers, so you don't need to say specifically that row r4 is r3+1, etc. You can also assume that the states are ordered, so for any state `s` (except maybe the last one) there is a next state `next(s)` for instance.

You do not have to write out every detail of the problem, especially as parts of it are a bit boring, but the closer you get to a full logical description the better. If you are happy with your encoding of this problem in terms of logic, and want to explore it further, just for fun, you may like to point your browser at
`https://l4f.cecs.anu.edu.au`
and in particular at
`https://l4f.cecs.anu.edu.au/puzzles/logician/traditional-maze`

Answer

This is a harder question, and there is no unique correct answer. Here is a sample encoding of the entire problem, assuming that rows `r1, ..., r8` and columns `c1, ..., c8` are objects in the domain, along with actions `up, down, left, right` and `no_op` and a finite sequence of states `s1, ..., sN` for an unknown value of `N`. We assume that the total order on rows, columns and states is given (perhaps because we use the numbers themselves as objects). If that has to be axiomatised as well, it adds rather uninteresting stuff to the encoding, so we just assume that "$<$" and "`next`" make sense.

The other vocabulary used is:

function symbols	
`xpos(s)`	column where the robot is in state `s`
`ypos(s)`	row where the robot is in state `s`
predicates	
`isState(s)`	object `s` is a state
`isCol(x)`	object `x` is a column
`isRow(y)`	object `y` is a row
`isAct(a)`	object `a` is an action
`goes(s,a)`	action `a` happens in state `s`
`rightwall(x,y)`	no move right from cell with coordinates `x, y`
`downwall(x,y)`	no move down from cell with coordinates `x, y`
`leftwall(x,y)`	no move left from cell with coordinates `x, y`
`upwall(x,y)`	no move up from cell with coordinates `x, y`
names (constants)	
`initial`	name of initial state
`final`	name of last (goal) state

With that vocabulary, we can start to write some basic axioms:

$$\forall s\ \forall x\ (xpos(s)=x \rightarrow (isState(s) \wedge isCol(x)))$$
$$\forall s\ \forall y\ (ypos(s)=y \rightarrow (isState(s) \wedge isRow(y)))$$
$$\forall s\ \forall y\ (goes(s,a) \rightarrow (isState(s) \wedge isAct(a)))$$

$$\neg\exists s\ (isState(s) \wedge (s < initial \vee s > final))$$
$$goes(final,no\_op)$$

The preconditions of the five actions are quite simple:

$$\forall s\ (goes(s,up) \rightarrow \neg upwall(xpos(s),ypos(s)))$$
$$\forall s\ (goes(s,down) \rightarrow \neg downwall(xpos(s),ypos(s)))$$
$$\forall s\ (goes(s,left) \rightarrow \neg leftwall(xpos(s),ypos(s)))$$
$$\forall s\ (goes(s,right) \rightarrow \neg rightwall(xpos(s),ypos(s)))$$
$$\forall s\ (goes(s,no\_op) \rightarrow s=final)$$

The postconditions and frame axioms can be taken together:

$$\forall s\ (goes(s,up) \rightarrow (ypos(next(s)) = ypos(s)+1 \wedge xpos(next(s)) = xpos(s)))$$
$$\forall s\ (goes(s,down) \rightarrow (ypos(next(s)) = ypos(s)-1 \wedge xpos(next(s)) = xpos(s)))$$
$$\forall s\ (goes(s,left) \rightarrow (xpos(next(s)) = xpos(s)-1 \wedge ypos(next(s)) = ypos(s)))$$
$$\forall s\ (goes(s,right) \rightarrow (xpos(next(s)) = xpos(s)+1 \wedge ypos(next(s)) = ypos(s)))$$

The initial and goal states are easy to specify:

$$xpos(initial) = c1 \wedge ypos(initial) = r8$$
$$xpos(final) = c5 \wedge ypos(final) = r4$$

That leaves the specification of where all the walls are. This is rather tedious, and I wouldn't expect people to write it out completely in the tutorial. It's important to think about how it could be expressed, though.

```
∀x ∀y (rightwall(x,y) → isRow(y))
∀x ∀y (upwall(x,y) → isCol(x))
∀x ∀y (leftwall(x,y) ↔ (x=c1 ∨ rightwall(x-1,y)))
∀x ∀y (downwall(x,y) ↔ (y=r1 ∨ upwall(x,y-1)))

∀x (rightwall(x,r1) ↔ (x=c1 ∨ x=c8))
∀x (rightwall(x,r2) ↔ (x=c2 ∨ x=c7 ∨ x=c8))
∀x (rightwall(x,r3) ↔ (x=c1 ∨ x=c2 ∨ x=c3 ∨ x=c8))
∀x (rightwall(x,r4) ↔ (x=c2 ∨ x=c5 ∨ x=c8))
∀x (rightwall(x,r5) ↔ (x=c1 ∨ x=c2 ∨ x=c4 ∨ x=c6 ∨ x=c8))
∀x (rightwall(x,r6) ↔ ¬(x=c6))
∀x (rightwall(x,r7) ↔ (x=c3 ∨ x=c4 ∨ x=c7 ∨ x=c8))
∀x (rightwall(x,r8) ↔ (x=c1 ∨ x=c5 ∨ x=c8))
∀y (upwall(c1,y) ↔ (y=r8))
∀y (upwall(c2,y) ↔ (y=r3 ∨ y=r6 ∨ y=r8))
∀y (upwall(c3,y) ↔ (y=r1 ∨ y=r4 ∨ y=r7 ∨ y=r8))
∀y (upwall(c4,y) ↔ ¬(y=r5 ∨ y=r6))
∀y (upwall(c5,y) ↔ (y=r2 ∨ y=r3 ∨ y=r4 ∨ y=r8))
∀y (upwall(c6,y) ↔ (y=r1 ∨ y=r2 ∨ y=r3 ∨ y=r6 ∨ y=r8))
∀y (upwall(c7,y) ↔ (y=r5 ∨ y=r6))
∀y (upwall(c8,y) ↔ (y=r1 ∨ y=r4 ∨ y=r8))
```

Many variations on this type of logical encoding are possible: there is no way I can list them all. One can use relations in place of the functions `xpos` and `ypos`; it is then necessary to add a constraint saying the robot can't be in two places at once. I have taken them to be partial functions, with no value for arguments other than states, but they could be taken to have a null value in the junk cases instead. Then, different ways of representing the blocked moves (the walls) are possible. Making a wall into a relation between two neighbouring cells is somehow natural, and OK for a piece of logic like this, but a bad idea from an efficiency point of view, as "`wall`" then becomes a 4-place relation.