CS472 Foundations of Artificial Intelligence
Fall 2000
Assignment 4 Solutions

1. **Situation Calculus (15 Points)** Using the Situation Calculus as described in R&N beginning on p. 204 you are to formalize the blocks world domain. The objects in this domain are blocks, tables, and situations. The predicates are:

   On(x,y,s) ClearTop(x,s) Block(x) Table(x)

   The only action is PutOn(x,y) where x must be a block whose top is clear of other blocks, and y is either the table or a different block with a clear top.

   (a) (2 pts) Write an axiom or axioms describing the action PutOn.
   Axiom 1: $\forall a, x, y, s$ **On(x,y,Result(a,s))** $\iff$ **[(a=PutOn(x,y)** $\wedge$ **ClearTop(x,s)** $\wedge$ **(ClearTop(y,s)** $\vee$ **Table(y)))** $\vee$ **(On(x,y,s)** $\wedge \neg\exists z$ **z** $\neq$ **x** $\wedge$ **a = PutOn(x,z))**
   Axiom 2: $\forall x, s$ **(**$\forall y \neg$**On(y,x,s)** $\iff$ **ClearTop(x,s))**

   (b) (2 pts) Describe the initial state S_0 where two blocks A, B are on the table T, and a third block C is on top of A.
   Block(A) Block(B) Block(C) Table(T) On(A,T,S_0**) On(B,T,**S_0**) On(C,A,**S_0**)**

   (c) (2 pts) Generate a query to a theorem prover that will generate a plan to create a three-block stack where C is on top of B, and B is on top of A.
   $\exists s$ **On(C,B,s)** $\wedge$ **On(B,A,s)**

   (d) (4 pts) Give a plan generated by the query above.
   Result(PutOn(C,B), Result(PutOn(B,A), Result(PutOn(C,T),S_0 **)))**

   (e) (5 pts) Show formally that the plan above follows from the situation S_0 and the axiom(s) for PutOn.
   There is no fixed solution for this. See grader's comments.

   Grader's Comments

   (a) Code *1: The most common mistake is to leave out the check for **a = PutOn(x,y)**. Specifying the action being performed is very important here. Without the action, having the other conditions will not result in a change.

   (b) Students typically do well in this part.

   (c) The common mistake here is to ask the prover **On(C,B)** $\wedge$ **On(B,A)**. The situation argument cannot be omitted since the On(X,Y) property changes over time in the blocks world domain.

(d) Students typically do well in this part too.

(e) Code *2: The derivation is not detailed and formal enough. Listing out part (c) and each of the PutOn action axioms alone will not suffice for full credits. For each action, students have to first list out the necessary preconditions for Axiom 1 to be applied, and verify that **they can be satisfied**. After the application of Axiom 1, Axiom 2 should be used to figure out which blocks have clear tops. Many people did not provide the intermediate link to demonstrate exactly how their plan will take situation S_0 to the eventual goal specified in part (d).

2. **Knowledge Base Design (40 Points)**

(a) Design a knowledge base to capture, in logical form, the knowledge of the following sentence in order to answer a series of questions about it.

> Yesterday John went to the Lansing Tops and bought two pounds of potatoes and a steak.

The sentences in the knowledge base should have a straightforward logical structure (e.g. statements that objects have certain properties, that objects are related in a certain way, that all objects satisfying one property also satisfy another, etc.). Hint: Consider the following when designing your knowledge base:

- What classes, individuals, relations, and so on would you need? (You will need events and temporal ordering, among other things.)
- Where does everything fit in a more general hierarchy?
- What are the constraints and interrelationships among them?
- How detailed must you be about each of the various concepts?

The knowledge base that you construct must be capable of answering a list of questions given below. Not every question deals with information explicitly stated in the sentence above. As a result, you will have to add to the knowledge base enough background knowledge to answer the questions. Try to make your representation be as general as possible: e.g. don't say "People buy food from Tops." because that doesn't help with people who shop at other supermarkets; don't say "Joe made stew with his steak and potatoes," because that doesn't generalize to anything else; and don't turn the questions into answers. Many of the additional assertions may be only approximately correct in reality, but the idea is to extract the common sense that lets a person answer the questions at all.

(b) Sketch the chains of reasoning that would be necessary to answer each of the following questions based on your knowledge base from part (a).

- Does John have money before going to the supermarket? [Yes]
- Does John have less money after going to the supermarket? [Yes]
- What does John plan to do with the potatoes? [Eat them]
- Did John buy any meat? [Yes]

- Is John a vegan? [No]
- Did John see anyone in the supermarket? [Yes]
- Does Tops sell toothpaste? [Yes]
- Is John a child or is he an adult? [An adult]

Point Allocation
Check do they encode all the infomation explicit in the sentence given. (8 points)
Check if the lines of reasoning are plausible. (24 points)
Check does their knowledge base generalize beyond just what is given and what is asked. (8 points)

Grader's Comments

Most people had the right idea when constructing the database. Not as easy as it appears to be on first glance does it. People tended to forget to put details when encoding the sentence such as the events occured yesterday, or that he bought 2lbs. of potatos. Also people tended to forget time or situation when dealing with the have money issue. For example if he went to the store and spent all his money then Have(Money) would be true before he went to the store but not true after. So you really need Have(x, s) where x is the item and s is the situation or time. Temporal Order does matter (and you were told that it does matter). Also we we just looking for the lines of reasoning that went into answering the questions, instead of the logical steps it took you knowledge base to prove the answer followed logically, but since many of you just put the logical steps we tended to give full credit, execpt where your failed to explain why an axiom was in your knowledge base other than to just answer the question.

3. **Planning (25 Points)** Consider a version of the milk/bananas/drill shopping problem described in chapter 11 in R &N and in class.

   (a) (2 pts) Modify the description of the *Buy* action (p. 350) so that the agent needs to have a credit card to buy anyting.
   Op(Action:Buy(x), Precondition: At(Agent, Store) $\wedge$ Sells (store,x) $\wedge$ Have(cc) $\wedge$ CreditCard(cc), Effect: Have(x))

   (b) (2 pts) Write a *PickUp* operator that enables the agent to *Have* an object if it is portable and at the same location as the agent.
   Op(Action:PickUp(x), Precondition: Portable(x) $\wedge$ At(agent, loc) $\wedge$ At(x,loc), Effect: Have(x))

   (c) (10 pts) Assume that CC is a credit card which is at home but *Have*(CC) is initially false. Construct a partially ordered plan that achieves the goal state from the initial state both of which are described on p. 341 of R &N.

   (d) (11 pts) Explain in detail what will happen during the planning process when the agent explores a partial plan in which it leaves home with out a credit card.

Even if you left without the credit card you would have an unfufilled precondition for the buy step: namely Have(CC). This can be achieved by adding a PickUp(CC) step. It must come before because it satisifies a precondition, and you can satisify its precondition by linking it to the start step. The Go step threatens the condition At(agent, home) of the causal link between the start state and the PickUp step, so the conflict must be resolved. It can be resolved by placing the PickUp step before the Go step in the ordering. So even though we planned in the "wrong" order, we didn't have to backtrack in the planning, because in partial-order planning the order of the steps in not necessarily the order in which the steps will occur in the final plan.

Grading info: Almost all students got (a) and (b) right. I would like to make some notes here, **again**. First of all, make sure you use the correct capitalization of letters. Letter case matters when writting formulas in FOL (again, I took no points out, but...) A more major mistake is to nest predicates. $At(Location(x))$ simply does not parse. One way to make it parse is to define *Location* as a function (as opposed to predicate), which nobody did. Even if you do, in declarative languages people usually say something like $let\ l = location(x)\ in\ At(l)$.

There were a couple of minor mistakes in (c):

- No preconditions mentioned (-1 point)
- No differentiation between causal and order links (-2 points)
- No (the right type of) link between "PickUp" and "Go(HWS)" (-2 points)
- Not representation of solution to the planning problem (-2 points)
- Not a partial order (-3 points)

The grading for (d) was much more coarse. Minor mistakes like not arguing about the (proper type of) link between "PickUp" and "Go(HWS)", backtraking and sending the agent back home were penalized 2 points each. Note that the agent does not cycle going here and there. The point is to produce a plan **before** the agent even left home, and then for her to follow the plan. When explanations were very unclear, 5 points were taken.

4. **Planning - Shakey's World (30 Points)** Shakey's World is described in R&N pp. 360 - 362.

   (a) (12 pts) Translate Shakey's six actions into STRIPS format.
   Op(Action: Go(y), Precondition:At(Shakey, x) $\wedge$ In(x,r) $\wedge$ In(y,r) $\wedge$ On(Shakey, Floor), Effect: At (Shakey, y) $\wedge$ $\neg$(At(Shakey, x)))

   Op(Action: Push(b,x,y), Precond: At(Shakey, x) $\wedge$ On(Shakey, Floor) $\wedge$ At(b,x) $\wedge$ In(x,r) $\wedge$ In(y,r) $\wedge$ Pushable(b), Effect: At(b,y) $\wedge$ At(Shakey, y) $\wedge$ $\neg$ At(b,x) $\wedge$ $\neg$At(Shakey, x))

Op(Action:Climb(b), Precond: At(Shakey, x) ∧ On(Shakey, Floor) ∧ At(b,x) ∧ Climbable(x), Effect: On(Shakey, b) ∧ ¬On(Shakey, Floor))

Op(Action:Down(b), Precond:On(Shakey, b) ∧ Climbable(b), Effect: On(Shakey, Floor) ∧ ¬ On(Shakey, b))

Op(Action: TurnOn(l), Precond: On(Shakey, b) ∧ At(b,x) ∧ At(l,x), Effect: TurnedOn(l))

Op(Action: TurnOff(l), Precond: On(Shakey, b) ∧ At(b,x) ∧ At(l,x), Effect: ¬TurnedOn(l))

Grader's notes: Most students did well on this problem. Many had the right idea, but didn't get the effects quite right, forgetting, for example, to include ¬ (At(Shakey, x)) as an effect for the Go action. The Down action required Climbable(b), or ¬ On(Shakey, Floor), or At(b,x) ∧ At(Shakey,x), as a precondition, in order to ensure that b mentioned in Down(b) wasn't the floor.

(b) (10 pts) Manually construct a plan for Shakey to get Box2 into Room2 from the starting configuration in Figure 11.5 on p. 361.
Go(door3)
Go(corridor)
Go(door1)
Go(room1)
Push(Box2, room1, door1)
Push(Box2, door1, corridor)
Push(Box2, corridor, door2)
Push(Box2, door2, room2)

Grader's notes: Shakey's world wasn't specified very well in the text, so there were quite a few variants of this basic plan. Almost all plans got full points, unless they forgot to include the doors, which served as connection points between rooms.

(c) (8 pts) Suppose Shakey has n boxes in a room and needs to move them all into another room. What is the complexity of the planning process in terms of n? Explain.
The complexity of the planning process depends on how restricted what language you are using to generate the plan. For example simle planning such as using STRIPS is known to be PSPACE-complete (Bylander 1992).

Grader's notes: This question was too ambiguous, so we didn't grade it. In general the planning problem is PSPACE-complete, which means, intuitively, that an unlucky plan can be exponentially long. But Shakey's world, as presented in the text, is simple enough that a good planner should be able to quickly find a short plan. A few students mixed up the complexity of the planning process with the length of the plan itself – this was a definite error, so you should make sure that you understand the difference between the two.

5. **Knowledge-Based Agent Design (100 Points)** This is the (Java) programming portion of the assignment. You will design agents to play capture-the-flag. Your agents will compete against the agents of other groups in the class and against agents designed by one of the TAs (Jason). Downloadable code, a description of the game, sample agents, a demo, and a description of what to turn in are available from the course web site. As noted at the top of this handout, each group should turn in only ONE version of this part of the assignment (in the format described at the web site).