Sample Solutions for COSC1125/1127 Tutorial Week 7

The block world (fig. 7.18) may be represented by the following set of predicates:

STATE 1:
ontable(a) Λ ontable(c) Λ ontable(d) Λ on(b,a) Λ on(e,d) Λ gripping() Λ clear(b) Λ clear(c) Λ clear(e)

There are then a number of rules (or operators) to operate on states and produce new states. These operators typically include pickup, putdown, stack, and unstack. For example the unstack operator can be described as:

Rule1:
(∀X)(∀Y)(unstack(X,Y) → ((clear(Y) Λ gripping(X)) ← (on(X,Y) Λ (clear(X) Λ gripping()))).

If we apply unstack(e,d), a new state will be:

STATE 2:
ontable(a) Λ ontable(c) Λ ontable(d) Λ on(b,a) Λ **gripping(e)** Λ clear(b) Λ clear(c) Λ clear(e) Λ **clear(d)**

In the above we can see as a result of applying the operator, new predicates have been added to the new state, whereas other predicates such as on(e,d) Λ gripping() have been deleted from the new state because they are no longer valid. Operators such as this can be also described by using STRIPS.

You will notice that there are also predicates such as "ontable(a) Λ ontable(c) Λ ontable(d)" continue to remain true in the new state. The question here is how we can ensure this is indeed the case as we generate more and more new states. This is where frame axioms can be used.

Frame axioms are rules to tell what predicates describing a state are not changed by rule applications and are thus carried over intact to help describe the new state of the world. An example of frame axioms is:

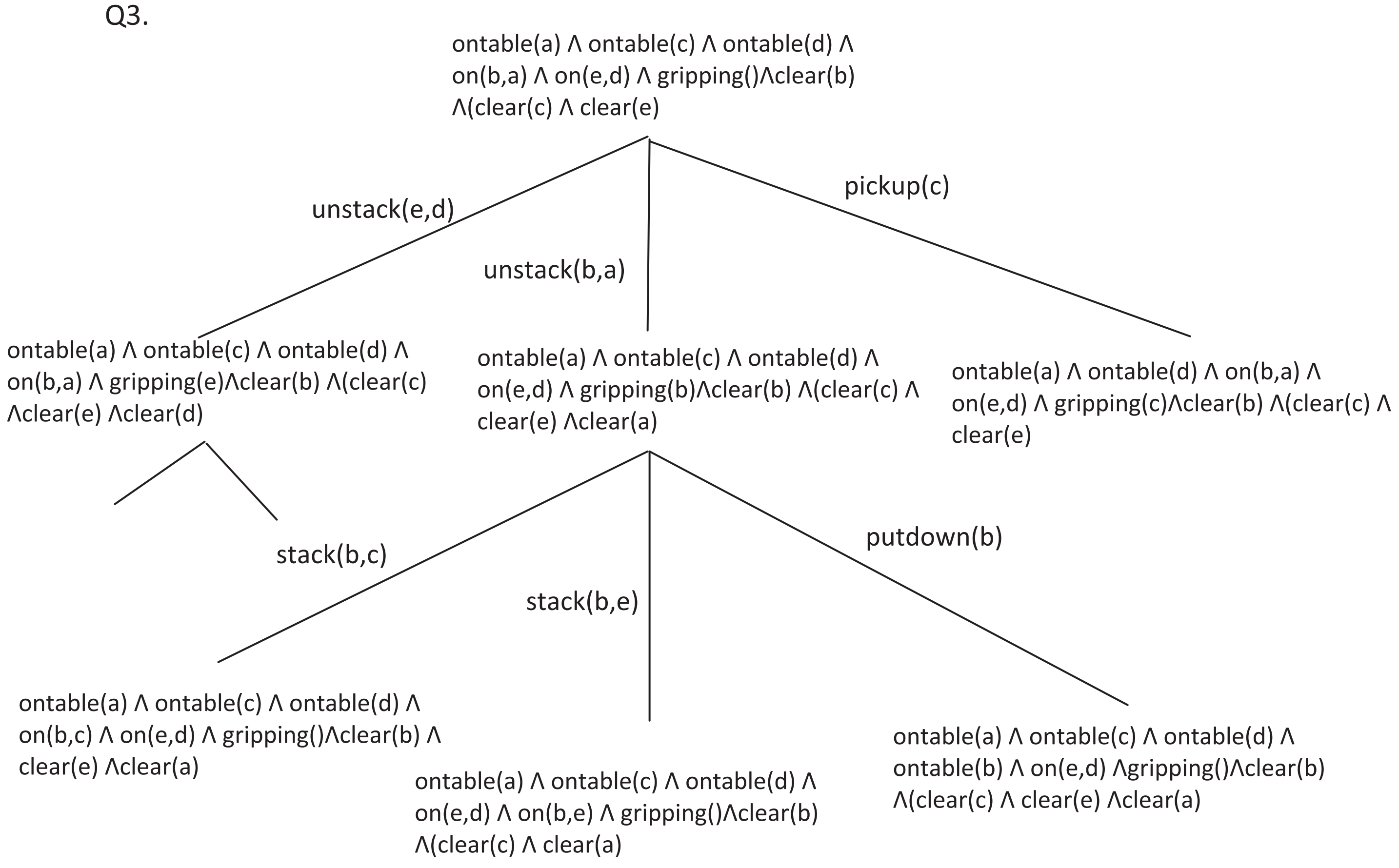
Rule 2:
(∀X)(∀Y) (∀Z)(unstack(Y,Z) → (ontable(X) ←ontable(X))).

The above rule says *ontable* is not affected by the *unstack* operator. By using this frame rule and the operator *unstack*, we can tell that when applying *unstack(e,d)* to STATE 1, we should continue to have "ontable(a) Λ ontable(c) Λ ontable(d)" in STATE 2, and update it with new predicates such as **gripping(e) Λ clear(d)**. Even for a simple block world example like this, we need a number of other frame axioms. For example,

(∀X)(∀Y) (∀Z)(stack(Y,Z) → (ontable(X) ←ontable(X)))
(∀X)(∀Y) (∀Z)(stack(Y,Z) → (clear(X) ←clear(X)))
(∀X) (∀Y)(∀Z)(pickup(X) → (on(Y, Z) ← on(Y,Z)))
(∀X) (∀Y)(pickup(X) → (ontable(Y) ← ontable(Y)))
(∀X) (∀Y)(∀Z)(putdown(X) → (on(Y, Z) ← on(Y,Z)))
(∀X) (∀Y)(putdown(X) → (ontable(Y) ← ontable(Y)))
…
etc.

After creating these axioms (which are straight forward and similar to those already created), we should question the complexity cost of adding this number of support axioms to a system in order to maintain a sound inference scheme. This issue can be addressed by using STRIPS, which maintains a *add* and *delete* list, to keep track of new predicates that should be added to the new states and as well as old predicates that should be deleted from the state.

Q3.


ontable(a) Λ ontable(c) Λ ontable(d) Λ on(b,a) Λ on(e,d) Λ gripping()Λclear(b) Λ(clear(c) Λ clear(e)
unstack(e,d)
unstack(b,a)
pickup(c)
ontable(a) Λ ontable(c) Λ ontable(d) Λ on(b,a) Λ gripping(e)Λclear(b) Λ(clear(c) Λclear(e) Λclear(d)
ontable(a) Λ ontable(c) Λ ontable(d) Λ on(e,d) Λ gripping(b)Λclear(b) Λ(clear(c) Λ clear(e) Λclear(a)
ontable(a) Λ ontable(d) Λ on(b,a) Λ on(e,d) Λ gripping(c)Λclear(b) Λ(clear(c) Λ clear(e)
stack(b,c)
stack(b,e)
putdown(b)
ontable(a) Λ ontable(c) Λ ontable(d) Λ on(b,c) Λ on(e,d) Λ gripping()Λclear(b) Λ clear(e) Λclear(a)
ontable(a) Λ ontable(c) Λ ontable(d) Λ on(e,d) Λ on(b,e) Λ gripping()Λclear(b) Λ(clear(c) Λ clear(a)
ontable(a) Λ ontable(c) Λ ontable(d) Λ ontable(b) Λ on(e,d) Λgripping()Λclear(b) Λ(clear(c) Λ clear(e) Λclear(a)

The STRIPS representation of the operators:

	P: gripping() Λclear(X) Λontable(X)
pickup(X):	A: gripping(X)
	D: ontable(X) Λ gripping()
	P: gripping(X)
putdown(X):	A: ontable(X) Λ gripping()Λ clear(X)
	D: gripping(X)
	P: clear(Y) Λgripping(X)
stack(X,Y):	A: on(X,Y) Λ gripping() Λ clear(X)
	D: clear(Y) Λ gripping(X)
	P: clear(X) Λgripping()Λ on(X,Y)
unstack(X,Y):	A: gripping(X) Λ clear(Y)
	D: gripping()Λ on(X,Y)

Q4.
a)
In a goal driven search, we know the goal state to start with. We can apply different operators to the goal state, and see what new states it can generate. In this case since the goal state is only two steps away from the initial state, you can easily identify that first unstack(e,c) and then secondly stack(e,d) will form a path to the initial state, therefore the reversed path would be the plan to achieve the required task. The plan is unstack(e,d) and stack(e,c).

In a data driven search, since we know the initial state for start, we can always try different operators, leading to different new states. From each new state, we can try these operators again, eventually we can identify the goal state. Since this is a rather small search space, a Breath-First-Search would be sufficient to find the path to the goal easily.

b)
Similar to the above, however the search space is larger than the previous example (it takes at least 8 steps to get to the goal state). It would be more appropriate to use heuristic search (e.g., measuring the distance of the current state from the goal state, and also checking if it is a repeated state), in order to guide the search towards the goal state.

Note for b) it can get really complicated in order to find the correct plan. For example we need to check if the preconditions of an operator are met or not, and the problem of having incompatible subgoals can also emerge. This goes beyond the scope of this subject.

Q5. Nothing really, because the property of color is not required for any preconditions of the operators.

Q6. similar to Q4. A plan such as the following can be generated using the goal-driven search:

unstack(C, A)
putdown(C)
pickup(B)
stack(B,C)
pickup(A)
stack(A,B)

You may use the STRIPS planner to see how the plan is generated.