BRSU

Planning and Scheduling
Assignment 2
Representations

Bastian Lang

October 13, 2015

1 Remark

For part 1 and 4 of this assignment I only rewrote the problem, I did not correct the given description. For a *pickup* or an *unstack* operation the robot hand would have to be empty, i.e. holding = nil. But this is not listed as a precondition in the given problem description, so I did not add it in my solution.
Also I did not add a definition of the constant symbols and the state variables as this has not been done by the provided classical representation as well.

2 Exercises

3 Rewrite the problem as a set-theoretic planning problem.

s_0 = {on-c1-table, on-c3-c2, clear-c3, on-c2-table, clear-c1}
g = {on-c1-c2, on-c2-c3}

3.1 pickup

Rule: For every block cX exchange x of the classical representation with cX.

pickup-c1
precond: on-c1-table, clear-c1
effects: ¬ on-c1-table, ¬ clear-c1, holding-c1

pickup-c2
precond: on-c2-table, clear-c2
effects: not-on-c2-table, not-clear-c2, holding-c2

pickup-c3
precond: on-c3-table, clear-c3
effects: not-on-c3-table, not-clear-c3, holding-c3

3.2 PUTDOWN

Rule: For every block cX exchange x of the classical representation with cX.

putdown-c1
precond: holding-c1
effects: on-c1-table, clear-c1,not-holding-c1

putdown-c2
precond: holding-c2
effects: on-c2-table, clear-c2,not-holding-c2

putdown-c3
precond: holding-c3
effects: on-c3-table, clear-c3,not-holding-c3

3.3 UNSTACK

Rule: For every two blocks cX and cY exchange x of the classical representation with cX and y with cY.

unstack-c1-c2
precond: on-c1-c2, clear-c1

effects: not-on-c1-c2, not-clear-c1, holding-c1, clear-c2

unstack-c1-c3
precond: on-c1-c3, clear(c1)
effects: ¬on-c1-c3, ¬clear-c1, holding-c1, clear-c3

unstack-c2-c3
precond: on-c2-c3, clear(c2)
effects: not-on-c2-c3, not-clear-c2, holding-c2, clear-c3

3.4 STACK

Rule: For every two blocks cX and cY exchange x of the classical representation with cX and y with cY.

stack-c1-c2
precond: holding-c1, clear-c2
t- effects: clear-c1, on-c1-c2, not-clear-c2, not-holding-c1

stack-c1-c3
precond: holding-c1, clear-c3
effects: clear-c1, on-c1-c3, not-clear-c3, not-holding-c1

stack-c3-c2
precond: holding-c3, clear-c2
effects: clear-c3, on-c3-c2, not-clear-c2, not-holding-c3

4 WHY ARE THERE SEPARATE OPERATORS FOR PUTDOWN AND STACK, RATHER THAN A SINGLE OPERATOR FOR BOTH?

Because in this problem there is always an empty spot on the table. So to put down a block the only precondition to check is if the block is being held right now.
To stack a block upon another block, there is one more precondition to check: there may not already be another block upon the second block.

5 In the DWR domain, why do we not need two operators analogous to putdown and stack for placing containers onto a pile with a crane?

In DWR containers may only be placed on piles. So the pile has always to be specified. Therefore we cannot have a *take* or *put* operation that does not specify the pile.
I *guess* that by choosing nil or the pile itself as the *d* in the *put* operation (d = object the chosen container is put upon) one can address an empty pile. But then also top(p,p) should be valid - a pile being the top of itself. Otherwise there are two operations missing in the slide set(putting and taking from an empty pile).

6 Rewrite the problem as a state-variable planning problem.

s_0 = {pos(c1) = table, pos(c3) = c2, clear(c3) = 1, pos(c2) = table, clear(c2) = 0 }
g = {pos(c1) = c2, pos(c2) = c3}

6.1 pickup

pickup(x: block)
preconds: pos(x) = table, clear(x) = 1
effects: pos(x) = nil, clear(x) = 0, holding = x

6.2 putdown

putdown(x: block)
preconds: holding = x
effects: pos(x) = table, clear(x) = 1, holding = nil

6.3 unstack

unstack(x: block, y: block)
preconds: pos(x) = y, clear(x) = 1
effects: pos(x) = nil, clear(x) = 0, holding = x, clear(y) = 1

6.4 stack

stack(x: block, y: block)
preconds: holding = x, clear(y) = 1
effects: clear(x) = 1, pos(x) = y, clear(y) = 0, holding = nil