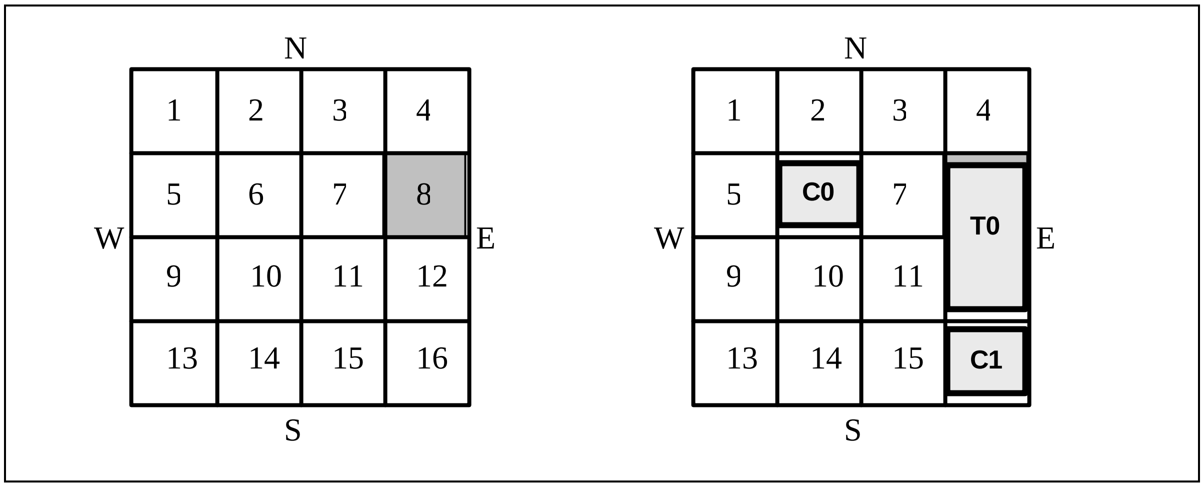
Homework #1: Partial Order Planning: Representation and Search
Distributed: September 26, 2001
Due: October 10, 2001

Problem 1 (20%): "Rush Hour"

"Rush Hour" is a popular puzzle-type game. The goal of the game is to slide plastic cars and trucks back and forth until a path is created for one of the cars to exit the game board. You are to write operator representations for a simplified version of Rush Hour and solve problems in this domain using a partial order planner.

Details: The board we will use is 4x4 squares, numbered 1 to 16 (see figure below). There are two types of vehicles: cars (size 1x1) and trucks (size 2x1). Vehicles can be facing N/S (up/down) or E/W (left/right). If they are facing N/S, they can move only up or down (from one row to the next); if they are facing E/W, they can move only left or right (from one column to the next). Vehicles can move any number of squares, as long as no other vehicles occupy squares along the way.



The goal of the game is always the same: to get car C0 to square number 8. This may involve moving other vehicles out of the way which, in turn, may involve moving other vehicles, etc. For example, in the right-hand side of the figure above, cars C0 and C1 are facing E/W, and truck T0 is facing N/S. In the initial state, T0 can move up one square, C0 can move either left or right one square, and C1 can move left one, two, or three squares. In this simple example, the shortest of many possible solutions is to move C1 left one square, move T0 down one square, and them move C0 right two squares.

You are to write state descriptions and operator representations for this game. For this problem, use one operator for moving a car and a separate operator for moving a truck. Use the PDDL representation language (see http://www.cs.yale.edu/homes/dvm). Explain briefly your major design decisions (e.g., how did you represent vehicle occupancy and the adjacency relations between squares).

Create several simple example problems and run them using the TPOP planner (available from http://www.cs.cmu.edu/~reids/planning/assignments.html). Untar the planner and read the README file in the tpop directory for instructions on how to make the code and run the planner. If you have difficulties, please contact Hakan Younes (lorens@cs). Try running at least the following Rush Hour problems (you may limit all these problems to search at most 25000 generated nodes):

- C0 faces E/W and starts on square 7, C1 faces N/S and starts on square 8, T0 faces E/W and starts on squares 11 and 12.
- C0 faces E/W and starts on square 7, C1 faces N/S and starts on square 8, T0 faces E/W and starts on squares 11 and 12, T1 faces E/W and starts on squares 3 and 4.
- C0 faces E/W and starts on square 5, C1 faces N/S and starts on square 7, T0 faces E/W and starts on squares 3 and 4, T1 faces N/S and starts on squares 2 and 6.
- C0 faces E/W and starts on square 6, C1 faces E/W and starts on square 12, C2 faces E/W and starts on square 3, T0 faces N/S and starts on squares 4 and 8, T1 faces N/S and starts on squares 11 and 15.

Please provide a pointer to the files containing the representations and problems you created.

Problem 2 (35%): "Rush Hour, The Sequel"

The purpose of this problem is to investigate how changes to the state and operator representations affects planning efficiency. You are to make several changes to the Rush Hour domain representation created for Problem 1 and compare the effects. At the least, investigate the effects of:

- Changes in the ordering of preconditions
- Use of a single (conditional) operator for moving both trucks and cars
- Use of a single operator for moving multiple squares at a time (one, two, or three)

For each change, run the Rush Hour problems you created in Problem 1 and compare results. Provide quantitative results showing the effects each change has, and discuss plausible reasons for why the changes you made affect efficiency in the way that you observe. You can use the tracing mechanism of TPOP (verbosity level > 1) to see how the planner is searching. In addition to the graphs and write-up, provide a pointer to the representations used for this problem.

Problem 3 (30%): “Rush Hour, III”

In this problem, you will be investigating how various search strategies affects planner performance. TPOP has various heuristics that it can use for choosing which plan to investigate next and for choosing which open condition to investigate for that plan. TPOP can also operate with “ground actions” (no variables – all actions are fully instantiated at run time).

Using the representation of your choice from Problems 1 and 2, systematically investigate the various options available in TPOP. For each of the classes of options (plan choice heuristic, condition choice heuristic, ground vs. va riablized actions) provide quantitative results showing the different effects, and also discuss plausible reasons for why the particular options have the effects that you observe. NOTE: Many of the results you observe will be counter-intuitive. It may take a bit of “detective work” to analyze the search traces to understand what is happening. NOTE: when using many of these heuristics, TPOP needs to create grounded actions. It turns out that instantiation is a *lot* more efficient if you supply type information with all the objects and action parameters.

Try solving problems from the gripper domain (tpop/examples) using different heuristics. Find a pair of heuristics such that one improves performance in the gripper domain over the other, but performs more poorly in the Rush Hour domain. Describe a difference in the two domains that could plausibly account for the difference in performance of the heuristics.

For extra credit: Use the same Rush Hour domain that you implemented for TPOP and run it with Prodigy4.0 (available at http://www.cs.cmu.edu/~prodigy). Study the effect of representation variations of the domain, including control rules and different heuristics. Propose a heuristic that, based on your investigation, is likely to work well in the Rush Hour domain. Provide a rationale for why you believe the heuristic should be effective.

Problem 4 (15%): Computational Complexity

Computational complexity of search algorithms is $O(cb^n)$, where "c" is the per-node cost of the algorithm (which may, or may not, be constant), "b" is the average branching factor at a node, and "n" is the search depth. While the performance of a search algorithm, on average, may be significantly better than the complexity would suggest, complexity analysis often provides insight into how the algorithm will perform.

You are to analyze the computational complexity of the following algorithm (POP is a simplified version of UCPOP, restricted to STRIPS-style operator representations), in terms of the formula given above. In particular, provide an analysis based on the number of actions in the plan, average number of variables in an operator representation, and average number of preconditions of an operator. Pay particular attention to the per-node cost of the POP algorithm. Also discuss whether your analysis suggests that using ground (fully instantiated) actions is likely to help, or hurt, search performance for this algorithm.

```
POP(initial-state, goals)
    plan = <A={Start, Finish}, O={Start < Finish}, B={}, L={}>
    agenda = {(goals, Finish)}
    Repeat until agenda is empty
        Select (and remove) an open condition (q, ac) from agenda
        If q is a conjunction, then add each conjunct to agenda
        Else if q is a literal and $ap \xrightarrow{\sim q} ac$ exists in L, then Fail
        Else choose ap (either a new or existing action) with an effect r that unifies with q
            Add {$ap \xrightarrow{q} ac$ } to L
            Add MGU(q, r, B) to B
            Add {(ap < ac), (ap < Finish), (Start < ap)} to O
            If ap is new, add preconditions to agenda and any variable constraints to B
        For each causal link $a_i \xrightarrow{p} a_j$ and each at action which threatens the link,
        choose a resolution mechanism
            Promotion: Add (aj < at) to O
            Demotion : Add (at < ai) to O
        Determine consistency of plan and fail if plan is inconsistent
```