Lazy Evaluation of Negative Preconditions in Planning Domains (Extended Abstract)

Santiago Franco[1], **Jamie O. Roberts**[2], **Sara Bernardini**[1]
[1]Department of Computer Science, Royal Holloway University of London
[2]School of Engineering, Institute for Integrated Micro and Nano Systems, The University of Edinburgh
santiago.francoaixela@rhul.ac.uk, sara.bernardini@rhul.ac.uk, jamie.owen.roberts@gmail.com

Introduction

Fast Downward (*FD*) (Helmert 2006) is one of the most popular planners currently in use. FD transforms PDDL2.2 tasks (Edelkamp and Hoffmann 2004) (hereinafter, PDDL) into Finite Domain Representation (*FDR*) tasks (Helmert 2009). FDR is a grounded representation that uses multi-valued state variables, i.e. each variable has a finite domain where each value corresponds to an atom that is mutually exclusive with the rest of ground atoms in the domain (plus 'null').

An FDR state is composed of a finite set of variables with their corresponding values, which can change as a result of applying an action. A variable-value assignment indicates that the value assigned to the variable is true, while all other ground atoms in the variable's domain are false. Given a set of mutually exclusive atoms of size D, the memory cost in bits of its corresponding FDR variable is $\lceil log_2(D) \rceil$. The equivalent set of binary variables, representing whether a ground atom is true or false, costs D bits. As such, for those problems characterized by a large set of mutually exclusive ground atoms, FDR is the obvious choice.

However, FDR incurs a significant disadvantage when operators present negative preconditions. As FDR does not have a direct way to represent a ground negative literal, when negative preconditions are translated into FDR, FD instantiates all possible combinations of positive ground atoms that are logically equivalent to the negative preconditions and, then, for each combination, creates a new operator that is identical to the original operator except for having an extra precondition corresponding to the combination.

If a domain contains operators with negative preconditions affecting several state variables, an exponential growth in the number of operators can easily occur. This effectively excludes entire classes of problems from a scalable formulation in FDR, such as multi-agent navigation problems, which are vital in many fields. Negative preconditions are needed to verify the occupancy status of different cells in a grid.

To overcome this limitation, we present a technique that avoids the multiplication of operators by introducing the notion of *negative facts*, which are evaluated *on the fly* during search when required. We extend FD to incorporate our approach and show the benefits it offers by presenting a broad evaluation of domains that feature negative preconditions.

Lazy Evaluation of Negative Preconditions

Given a PDDL operator o with m negative preconditions affecting m different state variables and assuming that all of them have a domain of size n, the number of generated operators is the size of the Cartesian product among them, i.e. $(n-1)^m$. Our goal is to avoid this combinatorial explosion in the number of operators by skipping the *multiply-out* operation by postponing the evaluation of the operators' negative preconditions until the search phase, which is the last phase taking place after translation and knowledge compilation. We call our evaluation of negative preconditions *lazy* because we perform it only when we have verified that all the positive preconditions hold, which means that the evaluation of negative preconditions cannot be postponed further.

We propose a Three-Phased Precondition approach (*TPP-FD*) that allows us to avoid the *multiply-out* operation and check negative preconditions on the fly. The first phase of TPP-FD is implemented within the modified task generation module. Given a PDDL task Π, an operator $o \in O$ and the synthesized state variables $\mathcal{V}^{\text{MFDR}}$, the algorithm translates o into the modified FDR operator o^{MFDR}. The only difference with respect to the FD translation is the omission of the *multiply-out* method and the creation of negative preconditions for o^{MFDR}. The second phase of TPP-FD creates the *Modified-Successor-Generator* data structure. The algorithm functions exactly like FD's corresponding one, but disregards operators' negative preconditions in tree construction. The last phase of TPP-FD is an updated version of A* where the operators' negative preconditions are checked on the fly. Candidate operators are provided by the modified successor generator but are rejected if any of the negative preconditions make the grounded operator inapplicable.

Alternative Approaches

TPP-FD is useful in domains where there are operators with negative preconditions that affect one or more multi-valued variables, e.g. multi-agent grid-based problems. Most IPC domains either avoid negative preconditions, use bookmarking predicates or keep the size of the problems small enough to facilitate grounding. None of the IPC problems present a 3D grid-based representation, despite the fact that grids are a model frequently used in several areas, e.g. robotics.

We use a multi-agent grid-based demonstration domain called *Drone*, where drones occupy the cells of a Cartesian 3D grid. Each cell is defined by its X, Y and Z coordinates. Since a drone can only occupy one cell in the grid, the position of each drone can be represented as an FDR variable $v \in \mathcal{V}$ whose domain is the size of grid. A randomized unique goal position is chosen for each drone in the problem. All drones start outside the grid and can enter it by using the 'lift-off' action. Each drone can fly to adjacent, non-diagonal cells as long as it is unoccupied[1].

We briefly discuss three approaches to model an unoccupied cell: *Bookmarked*, *Positive Normal Form* and TPP-FD. *Bookmarked* substitutes negative precondition checks per (drone, cell) pair by a predicate per cell. The predicate indicates if a cell is free, e.g (free ?x ?y ?z). PNF would simply transform each negative check, e.g. (not (at ?Drone ?x ?y ?z)) into a single predicate, e.g. (not-drone ?Drone ?x ?y ?z). PNF is easy to automate but worse than the Bookmarked approach as it requires one predicate per (drone,cell) pair. No known automated Bookmarking method exists. The number of normalized grounded fly operators is $O = N \times (6 \times (GS^3 - GS^2) + 1)$ for both TPP-FD and STRIPS-FD, where N is the number of drones and GS the length of a regular 3D grid. Unmodified FDR-FD has exponentially more operators: $O' = O \times (GS^3 + 1)^{N-1}$. Bookmarked trades-off the exponential growth of operators as a function of the number of drones for polynomial growth in memory usage per state as a function of the number of cells, $AdditionalFactsPerState = GS^3$. PNF would require $AdditionalFactsPerState = N * GS^3$. Finally, TPP-FD trades-off using extra memory for time, by delaying the negative preconditions checks to FD's online search phase.

Experiments

We generate 100 randomized problems per grid, using an Intel Xeon E5-2640 cluster at 2.60GHz, with 6 GB memory and 1,800 seconds per problem. In the *Drone* domain, we employ uniform 3D grids from $5 \times 5 \times 5$ to $17 \times 17 \times 17$, with the number of drones equal to the grid size (GS).

We compare *STRIPS-FD*, *FDR-FD*, and *TPP-FD* running A* with iPDB. *STRIPS-FD* uses binary state variables hence preventing the negative preconditions' combinatorial explosion. *FDR-FD* represents standard FD. *Ng* and *Bm* stand for the Negation and Bookmarked domain variants. Both *TPP-FD* and *FDR-FD* perform equally for *Bm*.

Table 1 shows the coverage for the Drone domain. TPP-FD scales up much better when dealing with negative preconditions. FDR-FD runs out of memory while grounding any of the *Ng* problems. STRIPS-FD generates thousands of variables, increasing the complexity of iPDB's incremental pattern selection, which in turn deteriorates FD's performance when combined with the high memory usage.

Table 2 shows memory usage per state. TPP-FD and FDR-FD are equivalent memory-wise. *Ng* is orders of magnitude more efficient than using bookmarking predicates. However, memory reduction would be linear for domains needing bookmarking predicates, e.g. Traveling Salesman Problems.

	TPP-FD		STRIPS-FD	
GS	Ng	Bm	Ng	Bm
5	**100**	**100**	3	3
6	**100**	99	0	0
7	**99**	97	0	0
8	**97**	**97**	0	0
9	94	**96**	0	0
10	94	**95**	0	0
11	**95**	**95**	0	0
12	87	**92**	0	0
13	**87**	22	0	0
14	**91**	0	0	0
15	**84**	0	0	0
16	**90**	0	0	0
17	**90**	0	0	0
Sum	**1308**	893	103	99

Table 1: Coverage for Drone Domains.

	TPP-FD		STRIPS-FD	
GS	Ng	Bm	Ng	Bm
4	**4**	12	32	44
5	**8**	20	80	96
6	**8**	36	164	328
7	**12**	52	304	344
8	**12**	76	516	580
9	**12**	104	824	916
10	**16**	140	1,252	1,380

Table 2: Memory usage per state in bytes, *Drone* domains.

Conclusions

This extended abstract presents *TPP-FD*, a method addressing FD's exponential increase in grounded operators due to negative preconditions. It bypasses the traditional *multiply-out* step in PDDL-to-FDR translation by directly checking negative preconditions during search. We showed that TPP-FD can significantly outperform existing alternatives, mainly the domain-specific Bookmarked and automated PNF approaches. TPP-FD integrates smoothly with current systems, activating only when FDR variables with negative preconditions exist. This could extend traditional planning to complex multi-agent grid-based navigation and logistics problems. Axioms, heuristics and automated bookmarking predicate removal (when beneficial) are future work.

References

Edelkamp, S.; and Hoffmann, J. 2004. Pddl 2.2: The language for the classical part of the 4th international planning competition, Albert Ludwigs Universität Institüt fur Informatik. Technical report, Germany, Technical Report.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5): 503 – 535. Advances in Automated Plan Generation.

[1]https://github.com/francos3/DroneVariantsPDDLExamples