# Classical Planning with Avoid Conditions: Technical Appendix

**Marcel Steinmetz**[1], **Jörg Hoffmann**[1], **Alisa Kovtunova**[2], **Stefan Borgwardt**[2]
[1] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2] Institute of Theoretical Computer Science, Technische Universität Dresden, Germany
steinmetz@cs.uni-saarland.de, hoffmann@cs.uni-saarland.de, alisa.kovtunova@tu-dresden.de,
stefan.borgwardt@tu-dresden.de

## Proofs: $\varphi$-Traps

Some of our claims use *Progress*$(\phi, a)$ for general fact formulae $\phi$. In our proofs, we use the following *semantic* definition:

$$[\mathit{Progress}(\phi, a)] := \{s[\![a]\!] \mid s \in \mathcal{S}, \mathit{pre}_a \subseteq s, s \models \phi\} \quad (1)$$

We write $s \models \mathit{Progress}(\phi, a)$ if $s \in [\mathit{Progress}(\phi, a)]$. Progression can be defined *syntactically* along the lines of Rintanen's (2008) *regression* definition.

### Proof of Theorem 2

Let $\Psi$ be some DNF formula of facts without negation. We need to show that the set of states $[\Psi]$ represented by $\Psi$ is a $\varphi$-trap if every element $\psi \in \Psi$ satisfies

**(t$^\varphi$1)** $\forall s \in \mathcal{S}$, if $s \models \psi \wedge \mathcal{G}$ then $s \models \varphi$; and

**(t$^\varphi$2)** $\forall s \in \mathcal{S}$ and $\forall a \in \mathcal{A}$, if $s \models \mathit{Progress}(\psi \wedge \neg\varphi, a)$ then $s \models (\Psi \vee \varphi)$.

Assume that both conditions are satisfied for all elements of $\Psi$. Let $s \in [\Psi]$ be any state that satisfies $\Psi$, and let $\psi \in \Psi$ be some conjunction for which $s \models \psi$. If $s$ is a goal state, then $s \models \psi \wedge \mathcal{G}$, i.e., $s \models \varphi$ as per **(t$^\varphi$1)**. Therefore, $[\Psi]$ must satisfy **(T$^\varphi$1)**. If $s \models \varphi$, $s$ cannot possibly violate **(T$^\varphi$2)**. Assume that $s \not\models \varphi$, i.e., $s \models \neg\varphi$. Let $a$ be any action that is applicable in $s$. Plugging $\phi = \psi \wedge \neg\varphi$ into (1) gives $s[\![a]\!] \models \mathit{Progress}(\psi \wedge \neg\varphi, a)$. As per assumption **(t$^\varphi$2)**, hence $s[\![a]\!] \in [\Psi]$ or $s[\![a]\!] \models \varphi$. In either case, **(T$^\varphi$2)** is satisfied.

### (t$^\varphi$2a) and (t$^\varphi$2b) are not necessary for (t$^\varphi$2) to hold

Consider two binary variables $x$ and $y$, an action $a$ with precondition $\langle x, 0\rangle$ and effect $\langle x, 1\rangle$, $\psi = \langle x, 0\rangle$, and $\Phi = (\langle x, 0\rangle \wedge \langle y, 1\rangle) \vee (\langle x, 1\rangle \wedge \langle y, 0\rangle)$. **(t$^\varphi$2)** is satisfied since $[\mathit{Progress}(\psi \wedge \neg\Phi, a)] = \{\{\langle x, 1\rangle, \langle y, 0\rangle\}\}$, which satisfies $\Phi$. In contrast, **(t$^\varphi$2a)** and **(t$^\varphi$2b)** are not satisfied since neither $\langle x, 0\rangle$ implies $\Phi$, nor does $\langle x, 1\rangle$ imply $\Phi$.

### Proof of Theorem 4

We first show that the $\varphi$-trap refinement procedures terminates with a $\varphi$-trap $\hat{\Psi}$, if the set of states $\hat{S}$ used for the refinement satisfies

**(i)** $\Psi \vee \bigvee_{s \in \hat{S}} s$ is a $\varphi$-trap, and

**(ii)** $\hat{S}$ does not contain a goal state.

Condition (ii) is necessary to guarantee that such variable as in line 2 of Algorithm 2 exists. By adding the corresponding fact to $\psi_s$, $\psi_s \wedge \mathcal{G}$ can no longer be satisfied, i. e., every $\psi_s$ satisfies **(t$^\varphi$1)**. The second loop only adds facts to the conjunctions. These modifications cannot make **(t$^\varphi$1)** become false again. Hence, all $\psi_s$ satisfy **(t$^\varphi$1)** after termination.

Note that $(\Psi \vee \bigvee_{s \in \hat{S}} s)$ implies $\hat{\Psi}$ during the entire refinement call, because each $s \in \hat{S}$ implies $\psi_s$ by construction. Hence, if some $\psi_s$ and action $a \in \mathcal{A}$ exist for which the condition in line 4 is satisfied, it must hold that $\psi_s \subset s$ as per the assumption (i) above. A variable as required in line 5 exists. The same argument shows that the loop must terminate eventually, namely when $\psi_s = s$ for all $s \in \hat{S}$ at the latest. The loop termination condition ensures that all $\psi_s$ satisfy **(t$^\varphi$2a)** and **(t$^\varphi$2b)** for all $a \in \mathcal{A}$, i. e., all $\psi_s$ satisfy **(t$^\varphi$2)**.

The previous elements in $\psi \in \Psi$ are not affected. As per assumption (i), they (still) must satisfy **(t$^\varphi$1)**. Exchanging $s$ by $\psi_s$ in (i) cannot make **(t$^\varphi$2)** become false either, as $\Phi$ remains part of $\hat{\Psi}$, and membership in $(\Psi \vee \bigvee_{s \in \hat{S}})$ implies membership in $\hat{\Psi}$. In conclusion, every element in $\hat{\Psi}$ satisfies **(t$^\varphi$1)** and **(t$^\varphi$2)**. By Theorem 2, $\hat{\Psi}$ constitutes a $\varphi$-trap.

We finally need to show that every $\hat{S}$ identified during search satisfies (i) and (ii). Steinmetz and Hoffmann's (2017) identification of $\hat{S}$ is left unchanged, i.e., search calls a trap refinement once states $\hat{S}$ have been explored, whose successors leaving $\hat{S}$ were pruned due to the current $\Psi$. Since search terminates upon finding a (non-pruned) goal state, all goal states in $\hat{S}$ must satisfy $\Psi$. Given that $\Psi$ is a $\varphi$-trap — which holds initially, and is guaranteed by the refinements, as shown above — all goal states in $\hat{S}$ must hence satisfy $\varphi$. With the constraints on the successors, (i) is hence guaranteed by the identification method. $\hat{S}$ satisfies (ii) because every goal state in $\hat{S}$ must be identified by $\Psi$, and states that satisfied $\Psi$ are not explored by search, i. e., they cannot appear in $\hat{S}$.

## The REDONE Benchmarks

**CaveDiving** The goal requires to hire every initially available diver eventually. There is an (acyclic) preclude relation-

| | | COVERAGE | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Compil. | | | | $\varphi$-Prediction | | | | | | | | | |
| | | $\neg\varphi$ | LTL | $\mathcal{X}$ | prune-$\varphi$ | k-trap | | Ŝ-trap | | aOri | | aInt | | aDet | |
| Domain | # | | | | | 2 | 3 | | $k2$ | 25k | 100k | 25k | 100k | 25k | 100k |
| Satisficing | | | | | | | | | | | | | | | |
| CaveDiving-REDONE | 20 | 0 | **7** | **7** | **7** | **7** | **7** | 6 | 6 | **7** | **7** | **7** | **7** | 0 | 0 |
| Fridge-REDONE | 24 | 1 | 6 | 20 | **21** | **21** | 11 | **21** | **21** | **21** | **21** | **21** | **21** | 13 | 11 |
| Miconic-REDONE | 178 | 0 | 25 | 20 | 117 | 117 | 117 | **120** | **120** | 117 | 117 | 111 | 111 | 72 | 67 |
| Nurikabe-REDONE | 20 | 0 | 2 | **12** | 11 | 11 | 8 | 11 | 11 | 9 | 9 | 7 | 6 | 4 | 0 |
| Openstacks-REDONE | 60 | 0 | 4 | 12 | **13** | **13** | **13** | **13** | **13** | **13** | **13** | **13** | **13** | 8 | 7 |
| Trucks-REDONE | 30 | 0 | 7 | 8 | **18** | **18** | **18** | **18** | **18** | **18** | **18** | **18** | **18** | 15 | 15 |
| Driverlog-ROAD | 21 | 8 | 4 | 9 | **11** | **11** | **11** | **11** | **11** | **11** | **11** | **11** | **11** | 6 | 5 |
| Rovers-ROAD | 64 | 4 | 2 | 7 | 12 | 14 | **17** | 12 | 14 | 12 | 12 | 12 | 12 | 12 | 12 |
| TPP-ROAD | 40 | 4 | 4 | 8 | 9 | 9 | 7 | **10** | **10** | 9 | 8 | **10** | 9 | 9 | 8 |
| Transport-ROAD | 116 | 26 | 32 | 49 | 52 | 58 | **64** | 47 | 55 | 50 | 50 | 52 | 52 | 23 | 14 |
| $\sum$ | 573 | 43 | 93 | 152 | 271 | **279** | 273 | 269 | **279** | 267 | 266 | 262 | 260 | 162 | 139 |
| Optimal | | | | | | | | | | | | | | | |
| CaveDiving-REDONE | 20 | 0 | **7** | | **7** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | **7** | 0 | 0 |
| Fridge-REDONE | 24 | 1 | 6 | | **10** | **10** | **10** | **10** | **10** | **10** | **10** | **10** | **10** | 2 | 2 |
| Miconic-REDONE | 178 | 0 | 25 | | 68 | 68 | 68 | **70** | **70** | 62 | 61 | 62 | 62 | 28 | 23 |
| Nurikabe-REDONE | 20 | 0 | 2 | | **10** | **10** | 8 | **10** | **10** | 9 | 9 | 7 | 6 | 3 | 0 |
| Openstacks-REDONE | 60 | 0 | 4 | | **25** | **25** | **25** | **25** | **25** | **25** | 24 | **25** | **25** | 20 | 19 |
| Trucks-REDONE | 30 | 0 | 10 | | **11** | **11** | **11** | **11** | **11** | **11** | **11** | **11** | **11** | 8 | 8 |
| Driverlog-ROAD | 21 | 7 | 12 | | **13** | **13** | **13** | **13** | **13** | **13** | **13** | **13** | **13** | 8 | 7 |
| Rovers-ROAD | 64 | **3** | **3** | | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** | **3** |
| TPP-ROAD | 40 | **0** | **0** | | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| Transport-ROAD | 116 | 23 | 41 | | 45 | **49** | **49** | 45 | **49** | 45 | 45 | 46 | 46 | 16 | 7 |
| $\sum$ | 573 | 34 | 110 | | 192 | 196 | 194 | 194 | **198** | 185 | 183 | 184 | 183 | 88 | 69 |
| Unsolvability | | | | | | | | | | | | | | | |
| CaveDiving-REDONE | 20 | 0 | 4 | 7 | 7 | **20** | 16 | 7 | **20** | 7 | 7 | 7 | 7 | 16 | 16 |
| Miconic-REDONE | 28 | 0 | 0 | 0 | 18 | **19** | 18 | 16 | 16 | 18 | 18 | 16 | 16 | 16 | 16 |
| Driverlog-ROAD | 22 | 8 | 4 | 8 | 8 | 9 | **12** | 5 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| Rovers-ROAD | 64 | 0 | 0 | 0 | 0 | 16 | **22** | 14 | 21 | 0 | 0 | 2 | 2 | 1 | 1 |
| TPP-ROAD | 40 | 4 | 0 | 4 | 4 | 8 | 8 | 6 | **10** | 4 | 4 | 5 | 5 | 4 | 4 |
| Transport-ROAD | 116 | 25 | 32 | 46 | 46 | 53 | 61 | 59 | **64** | 46 | 45 | 48 | 46 | 55 | 56 |
| $\sum$ | 290 | 37 | 40 | 65 | 83 | 125 | 137 | 107 | **139** | 83 | 82 | 86 | 84 | 100 | 101 |

| | | SEARCH REDUCTION FACTORS (left: geometric mean, right: max) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\varphi$-Prediction vs. prune-$\varphi$ | | | | | | | | | |
| | | k-trap | | Ŝ-k-trap | | aOri | | aInt | | aDet | |
| Domain | # | $k{=}3$ | | $k{=}2$ | | 100k | | 100k | | 100k | |
| Satisficing | | | | | | | | | | | |
| CaveDiving-REDONE | 20 | 1.0 | 1.0 | 1.4 | 1.5 | 1.6 | 3.1 | 1.0 | 1.1 | | |
| Fridge-REDONE | 24 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Miconic-REDONE | 178 | 1.6 | 415 | 2.5 | 415 | 1 | 1 | 1.3 | 145 | 1.3 | 145 |
| Nurikabe-REDONE | 20 | 1.4 | 3.7 | 2.1 | 86.1 | 1 | 1 | 1.0 | 1.1 | | |
| Openstacks-REDONE | 60 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Trucks-REDONE | 30 | 1 | 1 | 1.2 | 1.7 | 1.1 | 2.2 | 1.0 | 1.2 | 1.0 | 1.1 |
| Driverlog-ROAD | 21 | 1.4 | 23.6 | 1.0 | 1.3 | 1 | 1 | 1.0 | 1.0 | 1 | 1 |
| Rovers-ROAD | 64 | 4.4 | 310.3 | 3.8 | 310.3 | 1 | 1 | 1.4 | 75.0 | 1 | 1 |
| TPP-ROAD | 40 | 3.0 | 258.2 | 2.4 | 258.2 | 1 | 1 | 1.3 | 9.9 | 1.9 | 258.2 |
| Transport-ROAD | 116 | 5.5 | 76.0K | 3.5 | 76.0K | 1 | 1 | 1.3 | 6.2 | 1.4 | 412.5 |
| $\sum$ | 573 | 2.0 | 76.0K | 2.2 | 76.0K | 1.0 | 3.1 | 1.2 | 145 | 1.2 | 412.5 |
| Optimal | | | | | | | | | | | |
| CaveDiving-REDONE | 20 | 1.0 | 1.0 | 1.0 | 1.0 | 1.3 | 2.0 | 1.0 | 1.0 | | |
| Fridge-REDONE | 24 | 1.0 | 1.0 | 1 | 1 | 1 | 1 | 1.0 | 1.0 | 1 | 1 |
| Miconic-REDONE | 178 | 1.7 | 415 | 1.9 | 415 | 1 | 1 | 1.3 | 145 | 1.4 | 145 |
| Nurikabe-REDONE | 20 | 1.4 | 1.9 | 1.4 | 2.1 | 1 | 1 | 1.0 | 1.3 | | |
| Openstacks-REDONE | 60 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Trucks-REDONE | 30 | 1 | 1 | 1.1 | 1.2 | 1.0 | 1.1 | 1.0 | 1.1 | 1.0 | 1.0 |
| Driverlog-ROAD | 21 | 1.7 | 9.0 | 1.1 | 1.5 | 1 | 1 | 1.0 | 1.0 | 1.1 | 1.5 |
| Rovers-ROAD | 64 | 1.5 | 2.5 | 1.5 | 2.5 | 1 | 1 | 1.2 | 1.4 | 1 | 1 |
| TPP-ROAD | 40 | | | | | | | | | | |
| Transport-ROAD | 116 | 2.7 | 1.4K | 2.0 | 1.4K | 1 | 1 | 1.2 | 3.2 | 1.2 | 19.1 |
| $\sum$ | 573 | 1.6 | 1.4K | 1.5 | 1.4K | 1.0 | 2.0 | 1.2 | 145 | 1.2 | 145 |
| Unsolvability | | | | | | | | | | | |
| CaveDiving-REDONE | 20 | 17.4K | 29.5K | 17.4K | 29.5K | 1.1 | 1.2 | 17.4K | 29.5K | 17.4K | 29.5K |
| Miconic-REDONE | 28 | 1.6 | 1.7 | 10.0 | 15.4 | 1 | 1 | 15.0 | 46.6K | 17.3 | 46.6K |
| Driverlog-ROAD | 22 | 0.45M | 36.4M | 0.45M | 36.4M | 1 | 1 | 52.2K | 36.4M | 0.45M | 36.4M |
| Rovers-ROAD | 64 | | | | | | | | | | |
| TPP-ROAD | 40 | 315.7 | 0.45M | 3.2K | 0.45M | 1 | 1 | 0.10M | 0.45M | 0.10M | 0.45M |
| Transport-ROAD | 116 | 163.1 | 28.7M | 65.7 | 28.7M | 1 | 1 | 35.0 | 46.8K | 913.4 | 28.7M |
| $\sum$ | 290 | 196.4 | 36.4M | 215.9 | 36.4M | 1.0 | 1.2 | 154.4 | 36.4M | 1.3K | 36.4M |

Table 1: Left half: coverage results, best in **bold**. Results for the compilations are shown for the base configurations only. The configuration names are described in the text. Right half: ratio of states visited by prune-$\varphi$ versus states visited with a $\varphi$-predictor on top ($K$ for thousand, $M$ for million). Larger values indicate more pruning. For each method per-domain geometric mean (left) and maximum values (right) are shown. Values between different configurations are not directly comparable.

ship between the divers, according to which hiring certain divers will no longer be possible depending on which divers have been hired before. Our avoid condition reformulation models this relation explicitly by ensuring that the diver that has been hired last is not precluded by any diver that has already been hired. To generate the unsolvable instances, we extended the standard IPC instances by adding a cycle to the preclude relationships.

**Fridge** Fridges must be repaired by swapping compressors. The action conditions ensure that (1) compressors cannot be removed if not all screws have been unfastened; (2) a new compressor cannot be attached to a fridge if the old one has not been removed; (3) compressors cannot be attached if some of their screws are fastened; and (4) a fridge can only be started if all screws of the new compressor are fastened. All four conditions translate directly into $\varphi$, satisfied if (1/3) a compressor is moved while some screw is still fastened; (2) a fridge contains multiple compressors; and (4) a fridge is running whose compressor has some unfastened screws.

**Miconic** An elevator benchmark with complex constraints on the passengers. The elevator can move up, down, and stop at the current floor. Stopping at a floor departs all passengers in the elevator whose destination is that floor, and boards all passengers which are waiting at that floor. Our reformulation allows to board and depart each passenger individually. The avoid condition ensures that the elevator does not move before having boarded/departed all passenger as just described. Additionally, the avoid condition takes the role of enforcing the passenger constraints, removing action preconditions accordingly: (1) the elevator may only move up (down) if no passenger is boarded that wants to go down (up); (2) passengers of conflict class $A$ must not be boarded at the same time as passengers of the conflict class $B$; (3) some passengers must never be alone in the elevator; (4) the elevator must not stop at a floor if a passenger is boarded which should not have access to this floor; (5) some passengers have a "non-stop" requirement that once boarded disallow the elevator to stop at any floor other than the passenger's destination; (6) VIP passengers must be served before all others. To encode these constraints into the avoid condition, we added additional facts representing whether the elevator has stopped, moved up, or moved down.

We generated additional Miconic instances of the following kind. We split $n$ passengers and $m$ floors into half. One half of the passengers is assigned to conflict class $A$, and the other to conflict class $B$. The class $A$ passengers are placed in the bottom half of the floors, the $B$ passengers at the upper part. The goal is to swap their places. To not satisfy $\varphi$ eventually, each stop of the elevator must flip the class of the passengers in the elevator (departing $A$ and board $B$, or vice versa depart $B$ and board $A$). An instance is made unsolvable by introducing an additional $A$ passenger, whose initial position and goal make this flip impossible. We generated 28 solvable and 28 unsolvable instances of this form, scaling passengers $n$ from 12 to 36 and floors $m$ from 16 to 28 with increments of 4.

**Nurikabe** Nurikabe distributes numbers on a 2D grid. The goal is to find for every numbered cell, a contiguous region surrounding this cell that contains exactly so many cells as given by the number. Different regions must be disconnected. The latter constraint can be trivially encoded as an avoid condition: adjacent cells must not be assigned to different regions.

**Openstacks** One needs to create products to serve orders. A product must not be made until all orders are started that include this product. Vice versa, an order must not be shipped before all its products were made. Both constraints are naturally expressed as an avoid condition.

**Trucks** Each truck has a limited storage area, with positions ordered from front to back. A position can only be accessed if no position closer to the front is occupied. We add facts for each truck that represent which of the truck's storage area positions has been accessed last. The constraint is replaced by the avoid condition that checks whether some position in front of the one accessed last is occupied.

## Extended Results Table

See Table 1.

## References

Rintanen, J. 2008. Regression for Classical and Nondeterministic Planning. In Ghallab, M., ed., *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*, 568–572. Patras, Greece: Wiley.

Steinmetz, M.; and Hoffmann, J. 2017. Search and Learn: On Dead-End Detectors, the Traps they Set, and Trap Learning. In Sierra, C., ed., *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*, 4398–4404. AAAI Press/IJCAI.