Where Ignoring Delete Lists Works: Local Search Topology in Planning Benchmarks

Jörg Hoffmann

Institute for Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee, Geb. 52
79110 Freiburg, Germany
hoffmann@informatik.uni-freiburg.de

Technical Report No. 185

March 20, 2003

Abstract

During the last five years, the planning community has seen vast progress in terms of the sizes of benchmark examples that domain-independent planners can tackle successfully. The key technique behind this progress is the use of heuristic functions based on relaxing the planning task at hand, where the relaxation is to assume that all delete lists are empty. The success of such methods in many of the current benchmarks suggests that in those task's state spaces relaxed goal distances yield a heuristic function of high quality.

Investigating a large range of planning domains, I prove that the latter is indeed the case. I identify several key characteristics of local search topology under relaxed goal distances, concerning the non-existence of unrecognized dead ends, as well as constant upper bounds on the difficulty of escaping local minima and benches. These characteristics occur in the majority of the current benchmarks. This explains the recent success of heuristic planners. Specifically, it follows that FF's search algorithm, using an idealized heuristic function, is a polynomial solving mechanism in (at least) eleven commonly used benchmark domains. The proofs shed light on what the structural reasons are behind the topological phenomena, giving hints on how these phenomena might be automatically recognizable. I discuss the consequences of my findings on ways of benchmarking in planning.

Contents

1 Introduction

During the last five years, one of the strongest trends in the planning community has been that towards heuristic planners, more specifically towards the use of heuristic distance (e.g., goal distance) estimation functions. The best runtime results, progressing far beyond the sizes of benchmark examples that previous domain-independent planners could tackle successfully, have been achieved based upon a technique phrased "ignoring delete lists". There, the heuristic function is derived by considering a relaxation of the planning task at hand, where the relaxation is to assume that all delete lists (i.e. the negative effects of the available planning operators) are empty. During search, may it be forward or backward, state space or plan space, the heuristic value of a search state in this framework is (an estimate of) the difficulty of extending the state to a solution using the relaxed operators. In this article, I offer an explanation of the success of such methods: as it turns out most of the current planning benchmarks share a kind of structure that makes relaxed planning a very informative heuristic.

Extending a search state to a solution using the real operators is at least as hard as it is when using the relaxed operators so optimal relaxed solutions can, in principle, be used to derive admissible heuristic functions. However, as was first proved by Bylander [13], deciding bounded plan existence is NP-hard even when there are no delete lists. Thus there is not much hope to find optimal relaxed plans (i.e., optimal relaxed solution-extensions of search states) fast. Instead, one can *approximate* the length of an optimal relaxed plan to a search state. Techniques of this kind were first, independently, proposed by McDermott [52] and by Bonet et al [11], who developed the planners Unpop [52, 54], and HSP1 [11, 7]. Both these planners perform forward state space search guided by an approximation of relaxed goal distance; Unpop approximates that distance by backchaining from the goals, HSP1 approximates that distance by a forward value iteration technique. In the AIPS-1998 planning competition [53], HSP1 compared well with four other competitors. This inspired the development of HSP-r and HSP2 [8, 10, 9], GRT [61, 62], AltAlt [56, 65], as well as FF [38, 46, 40]. In comparison to HSP1, HSP-r avoids heuristic re-computations by changing the search direction; HSP2 implements the various HSP versions in a configurable hybrid system; GRT avoids heuristic re-computations by changing the heuristic direction (the direction in which relaxed plans are computed); AltAlt uses a planning graph to extract heuristic values; and FF uses a modified technique for approximating relaxed plan length, as well as new pruning and search techniques. The results reported for FF [39] inspired the development of Mips [20] and STAN4 [24], which integrate elaborated variations of FF's relaxed plan length estimation technique into hybrid systems. In the AIPS-2000 planning competition [2], the heuristic planners dramatically outperformed the other approaches runtime-wise, scaling up to benchmark examples far beyond reach of previous, e.g., Graphplan-based [5, 6], systems. Out of the 14 competing domain-independent planners, FF was the only one winning a 1st prize, while three of the four 2nd prize winners (HSP2, STAN4, and Mips) were out of the same planner family. This caused the trend towards heuristic planners to still increase. Various researchers extended relaxed plan distance estimation techniques to temporal and numerical settings [16, 42, 19]. Others adapted them for use in partial order plan-space search [57, 67], developed variations of them to provide new means of heuristic guidance [58, 64], or modified them to take exclusion relations in a planning graph into account [29]. In the AIPS-2002 planning competition [25], out of 11 domain-independent competing systems, 7 were using relaxed plan distance estimations in one or the other form. All three price-winning domain-independent planners (LPG [29, 30], Mips [19], and VHPOP [67]) were from this group. The 1st prize winner LPG uses, amongst other heuristics, a relaxed planning technique to estimate the difficulty of (sub-)goal achievement in a planning graph.

The success of relaxed plan based heuristic methods in many of the current planning benchmarks suggests that, in those tasks, relaxed plans deliver highly informative heuristics. This intuition was first suported in 2001 by empirical observations made by the author in small random example benchmarks [41]. In the article at hand, I *prove* that relaxed plans yield a high quality heuristic function in most of the commonly used benchmark domains. The investigation takes place in the setting of forward state space search as used by, e.g., Unpop, HSP, Mips, and FF. I idealize matters in that I consider the heuristic value given by the *optimal* relaxed plan length to each search state. For a search state s, the optimal relaxed plan length to s is denoted with $h^+(s)$. As described above, computing the h^+ function is NP-hard, and the current state-of-the-art heuristic planners use various approximations of h^+ to guide search. Now, it is common knowledge that the behavior of heuristic search methods (may they be global or local, i.e., with or without backtracking mechanisms) depends crucially on the quality of the underlying heuristic function. This has, e.g., been studied in the SAT community, e.g. by Frank et al. [26]. In their work, Frank et al. (empirically) investigate properties of the *local search topology*, i.e., of topological properties like the sizes of local minima etc., in SAT instances under a standard heuristic function. I adapt Frank et al.'s definitions to AI planning. I investigate a range of 25 commonly used STRIPS and ADL benchmark domains including all competition examples, and, looking at the individual domains in turn, I identify several key characteristics of the topology of the respective search spaces under h^+. The characteristics are the following.

1. In 20 of the benchmark domains, there are no unrecognized dead ends, i.e., no states from which the goal is unreachable but for which there is a relaxed plan.

2. In 16 of the above 20 benchmark domains, one can always escape local minima (regions where all neighbors have a higher heuristic value) within a number of steps that is constant across all instances of the domain (in fact, in 13 of these domains there are no local minima at all).

3. In 11 of the above 16 benchmark domains, one can always escape benches (regions where all states have the same heuristic value) within a number of steps that is constant across all instances of the domain.

These results serve to explain the excellent performance of state-of-the-art heuristic planners in the current planning benchmarks. The results are idealized in that they consider h^+, not the approximations used in practice, and in that they consider the search space faced by a forward state space search only. Nevertheless it seems likely that all relaxed plan based heuristic approaches benefit from the quality of h^+, or, in the negative cases, suffer from its lack of quality (some more on this in the discussion, Section 12). Most specifically, the results prove that FF's search algorithm is a polynomial solving mechanism in 11 of the investigated 25 domains, under the idealizing assumption that FF's approximative heuristic function identifies the real h^+ distances. FF's search algorithm tries to escape local minima or benches by means of a breadth-first search. If local minima and benches can always be escaped from within a constant number of steps – as is the case in 11 of the domains – then the effort spent in the search is polynomially bounded. This holds true, e.g., in the classical *Logistics* domain.

Beside the "positive" results proving characteristic qualities of the h^+ function, the investigation also provides counter-examples in the negative cases. The results divide the current benchmark domains into a taxonomy of classes which differ in terms of the topological behavior with respect to h^+. The proof methods shed light on which structural properties – at the level of the planning tasks' definitions – are responsible for the topological phenomena. This knowledge gives hints as to how the topological phenomena might be automatically recognizable. Indeed, many of the structural properties I identify are of a syntactical nature. Seeing as it turns out a lot of the current planning benchmarks are quite similar in structure, particularly in a structure that favors exploration by relaxed plan based heuristic mechanisms, the results raise certain doubts about the adequacy of the current planning benchmarks for evaluating planning research. I address this issue, and other matters, in the discussion.

The paper is organized as follows. I start by giving a formal framework for STRIPS and ADL domains (which is needed to enable proofs over planning domains rather than single tasks) in Section 2. I then overview the investigated domains in Section 3. Section 4 contains the definitions of h^+, and of local search topology. In Section 5, I present the key lemmata underlying most of the proofs in the single domains. These proofs are given in the subsequent sections for each domain in turn, where domains with similar topological properties – and thus proofs – form blocks (Sections 6 to 10) to improve readability. Section 11 provides an overview of the resulting planning domain taxonomy, and Section 12 concludes the article by discussing various issues such as contribution and future work. Formal definitions of all domains are given in Appendix A.

Please note that this, rather lengthy, technical report is mainly intended as a source of details for people who have read the shorter journal article [45], or the conference paper [43]. The latter papers are likely much better suited to get a general understanding of the investigation at hand, so if the reader has not yet read these papers, she/he is strongly encouraged to do so.

2 Planning Framework

To enable proofs to properties of planning *domains*, rather than single tasks, we need a formal definition of what a planning domain *is*. The only other work the author is aware of that uses a formal notion of planning domains is recent work by Malte Helmert [36, 37]. There, the semantics of different transportation and construction domains are formalized in order to investigate their computational complexity. In difference to that, the definitions here are strictly syntax-oriented – after all, the h^+ function we want to consider is extracted directly from the syntax of the planning task.

The subsequent definitions formalize, in a straightforward manner, the way domains are usually handled in the community. I start with the STRIPS [21] language, then explain how it is extended to ADL [59]. When talking about sets, I always mean *finite* sets unless explicitly said otherwise.

2.1 Predicates, Facts, and Groundings

The base entity in both STRIPS and ADL are facts, i.e., predicates whose arguments are constants or variables. We take a strictly syntactical point of view, talking about strings which are sequences of symbols over an alphabet $\Sigma \supseteq \{\mathrm{a}, \dots, \mathrm{z}, \mathrm{A}, \dots, \mathrm{Z}, 0, \ \dots, \ 9, \text{-}\}$. *Predicates* are pairs (prd, ar), $\mathrm{prd} \in \Sigma^+$ being a string – the name of the predicate – and $ar \in \mathbb{N}$ being the arity of the predicate. *Constants* are strings, *variables* are abstract entities different from the constants. Given a predicate (prd, ar), a *fact* is a tuple $(\mathrm{prd}, \mathrm{arg}_1, \dots, \mathrm{arg}_{ar})$, where all *arguments* arg_i are either constants or variables. We say that the fact is *based on* (prd, ar). One important concept is that of grounding. A fact is *grounded in variable v with a constant c* by replacing all occurrences of v with c. If all arguments of a fact are constants, then the fact is a *ground fact*, or *proposition*. The (infinite) set of all ground facts is denoted as F.

Where that makes sense, we allow subscripts to indicate the predicates, variables, and/or constants which can be used within a construct. An example is the set of ground facts F. Given a set of predicates $\mathcal{P}$, a set of variables V, and/or a set of constants C, the ground facts that can be built when restricting oneself to the respective sets are denoted by $F^{\mathcal{P}}$ (all ground facts using only predicates in $\mathcal{P}$), $F^{\mathcal{P},C}$ (all ground facts using only predicates in $\mathcal{P}$ and only constants in C), or $F^{\mathcal{P},V,C}$ (all ground facts using only predicates in $\mathcal{P}$, only variables in V, and only constants in C), etc. We say that a construct is *based on* $\mathcal{P}$, V, and/or C.

2.2 States, Actions, and Tasks

A grounded STRIPS task representation consists of two states (the initial state and the goal state) as well as a set of actions. A *state* is a set of ground facts $s \subseteq F$ – the propositions that are true in the state. An *action* is a triple $a = (pre(a), add(a), del(a))$ of sets of ground facts: the *precondition* $pre(a) \subseteq F$, the *add list* $add(a) \subseteq F$, and the *delete list* $del(a) \subseteq F$ of the action. The (infinite) set of all STRIPS actions is denoted as $\mathcal{A}$. A *STRIPS task* is a triple (A, I, G) of *action set* $A \subseteq \mathcal{A}$, *initial state* $I \subseteq F$, and *goal state* $G \subseteq F$. The semantics will be given below. Let us first define in which way tasks are instances of domains.

2.3 Operators, Instances, and Domains

A domain is essentially a (possibly infinite) set of planning tasks. This set is defined in terms of the predicates used, a set of operators (functions into the set of actions), and a (possibly infinite) set of instances. Applying the operators to sets of constants given by the instances yields the tasks that belong to the domain. The definitions are as follows. A *STRIPS operator with p parameters* is a tuple $o = (V(o), pre(o), add(o), del(o))$, where $V(o) = (v_1, \ldots, v_p)$ is a p-tuple of variables, the *operator parameters* (we use $V(o)$ also to denote the set $\{v_1, \ldots, v_p\}$ of these parameters), $pre(o) \subseteq F^{V(o),\emptyset}$, $add(o) \subseteq F^{V(o),\emptyset}$, and $del(o) \subseteq F^{V(o),\emptyset}$ are sets of facts. The only variables are operator parameters, and the operator is not allowed to use constants.[1] We identify o with a function $o : (\Sigma^+)^p \mapsto \mathcal{A}$ from the set of all p-tuples of constants to the set of all STRIPS actions. The image of a constant tuple under o is the result of grounding (the facts in) $pre(o)$, $add(o)$, and $del(o)$ in the operator parameters with the respective constants. Let us focus on the instances. A *STRIPS instance* is a triple (C, I, G) of *constants* $C \subseteq \Sigma^+$, initial state $I \subseteq F^C$, and goal state $G \subseteq F^C$ (the states use only the available constants). Domains, finally, are these. A *STRIPS domain* is a triple $\mathcal{D} = (\mathcal{P}, \mathcal{O}, \mathcal{I})$: a set of predicates $\mathcal{P}$, a set $\mathcal{O}$ of STRIPS operators based on $\mathcal{P}$, and a possibly infinite set $\mathcal{I}$ of STRIPS instances based on $\mathcal{P}$. For $(C, I, G) \in \mathcal{I}$, we say that (C, I, G) is an *instance of* $\mathcal{D}$.

Instances (C, I, G) of a domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ are directly associated with (ground) tasks (A, I, G), namely by applying all operators in $\mathcal{O}$ with p parameters to all p-tuples of constants in C to obtain the action set A. (this is the process of *grounding* which is done as a pre-process in many planning systems such as, e.g., HSP, Mips, and FF).

2.4 Plans

Let us consider the semantics of action applications. These are defined via the *Result* function, mapping states and action sequences to states. An action a is *applicable* to a state s, if $pre(a) \subseteq s$. In that case, applying the (sequence consisting solely of the) action yields the resulting state $Result(s, \langle a \rangle) := s \cup add(a) \setminus del(a)$. If the action is not applicable in s, the result of applying it is undefined. The result of applying sequences of arbitrarily many actions is defined recursively by $Result(s, \langle\rangle) := s$, and $Result(s, \langle a_1, \ldots, a_n \rangle) := Result(Result(s, \langle a_1, \ldots, a_{n-1} \rangle), \langle a_n \rangle)$.

The semantics of tasks are as follows. Given a STRIPS task (A, I, G), a *plan*, or *solution*, for (A, I, G) is a sequence $P \in A^*$ such that $G \subseteq Result(I, P)$: a sequence of actions that maps the initial state into a state that fulfills all goals. The task is *solvable* if there is at least one plan for it. A plan is *optimal* if there is

[1] The latter restriction is made for notational simplicity only.

no plan that contains less actions – the shorter the plan, the better it is considered to be. Note that, as announced in the introduction, the definition stays within the forward state space search framework where plans are simple sequences of actions.

2.5 Extension to ADL

In addition to what is possible in STRIPS, there can be complex conditions and conditional effects in ADL, i.e., conditions that are logic formulae and effects that depend on the state of execution. More precisely, the logic formulae are first-order formulae without equality, using the set of predicate symbols of the respective domain (in particular there are no function symbols). The (infinite) set of all formulae is denoted as $\mathcal{L}$, the (infinite) set of all *closed* formulae – the formulae without free variables – is denoted as $\mathcal{L}_0$. We allow subscripts to indicate the used predicates etc. When a formula is based on a set C of constants, the quantifiers $\forall x$ or $\exists x$ are associated with C, i.e., of the form $\forall x \in C$ respectively $\exists x \in C$. Formulae based on constant sets are used in ground tasks, where the range of the quantifiers must be fixed to the respective constants in order to validate the logical semantics. Many definitions for STRIPS, like those of facts and states, remain unmodified in ADL. The other definitions are as follows.

In ADL actions and tasks, we fix a set of constants. This set of constants will be given by the respective instance. The definitions are as follows. Given a set of constants C. An *ADL action* is a pair $a = (pre(a), E(a))$ where $pre(a) \in \mathcal{L}_0^C$ is a closed formula and $E(a)$ is a set of ground effects; *ground effects* are triples $e = (con(e), add(e), del(e))$ where $con(e) \in \mathcal{L}_0^C$, the *effect condition*, is a closed formula and $add(e) \subseteq F^C$ as well as $del(e) \subseteq F^C$ are sets of ground facts (the add- respectively delete-list). The (infinite) set of all ADL actions is denoted as $\mathcal{A}_{adl}$. ADL tasks are triples (A, I, G) of action set $A \subseteq \mathcal{A}_{adl}^C$, initial state $I \subseteq F^C$, and *goal condition* $G \in \mathcal{L}_0^C$. Roughly, the semantics will be the same as before except that effects are only applied if their conditions hold true, and that a solution plan must reach a state where the goal condition is fulfilled.

Let us extend the definition of domains. An *ADL operator with p parameters* is a triple $o = (V(o), pre(o), E(o))$, where $V(o) = (v_1, \ldots, v_p)$ are (as before) the parameters, $pre(o) \in \mathcal{L}^{V(o),\emptyset}$ is the precondition, and $E(o)$ is a set of effects. Effects are tuples $e = (V(e), con(e), add(e), del(e))$, where $V(e)$ is a tuple of variables, the *effect parameters* (also used to denote the set of these parameters), with $V(e) \cap V(o) = \emptyset$; $con(e) \in \mathcal{L}^{V(o)\cup V(e),\emptyset}$ is a formula; and $add(e) \subseteq F^{V(o)\cup V(e),\emptyset}$ and $del(e) \subseteq F^{V(o)\cup V(e),\emptyset}$ are sets of facts. Like in STRIPS, the only variables are the parameters, and no constants are used. Given a set of constants C, we identify o with a function $o^C : (\Sigma^+)^p \mapsto \mathcal{A}_{adl}^C$ from p-tuples of constants to ADL actions. The image of a constant tuple under o^C is the result of grounding the operator in its parameters with the respective constants; inserting for each effect with p' effect parameters and each p'-tuple in C one copy of the effect grounded in its parameters with the respective constants; associating the quantifiers in all formulae with the set C. An *ADL instance* is a triple (C, I, G) like in STRIPS, except that $G \in \mathcal{L}_0^C$ is a formula. Domains are triples $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ like in the STRIPS case; for syntactical convenience, we allow the set of operators $\mathcal{O}$ to comprise both ADL-and STRIPS-operators (based on $\mathcal{P}$). Instances of domains are associated with tasks by, as before, applying the operators to the respective tuples of constants.

We define the semantics of actions by overloading the *Result* function. The $\models$ relation between states and formulae is defined in the standard inductive manner. The result of applying a sequence consisting solely of one action $a = (pre(a), E(a))$ to a state s is defined as $Result(s, \langle a \rangle) := s \cup A(s,a) \setminus D(s,a)$, where $A(s,a)$ $(D(s,a))$ is the union of the add (delete) lists of the effects $e = (con(e), add(e), del(e)) \in E(a)$ whose effect condition is fulfilled in s, i.e., for which $s \models con(e)$ holds. Prerequisite for applicability is fulfillment of the precondition, $s \models pre(a)$. If an action is not applicable to a state, then the result of applying it is undefined. The result of applying an action sequence is defined exactly as before. A plan for a task (A, I, G) is a sequence $P \in A^*$ of actions for which $Result(I, P) \models G$ holds.

Two remarks befor we proceed. First, ignoring delete lists simplifies a task only if all formulae are negation-free. In STRIPS domains this is the case by definition. Not so in ADL domains. Within a domain, any task can be polynomially transformed (normalized) to have that property: compute the negation normal form to all formulae (negations only in front of facts), then introduce for each negated fact $\neg(\text{prd}, c_1, \dots, c_{ar})$ a new fact $(\text{not-prd}, c_1, \dots, c_{ar})$ and make sure it is true in a state iff $(\text{prd}, c_1, \dots, c_{ar})$ is false. This process has first been proposed by Gazen and Knoblock [27], and is used as a pre-process in FF [46] (which is the only heuristic planner the author is aware of that can deal with full scale ADL). In the investigation at hand, the domains are defined such that formulae are negation-free from begin with, i.e., we consider the normalized versions of the domains.

Second remark, a frequent phenomenon in benchmark planning tasks is the occurence of *static* predicates. These are predicates which are not affected by any effect of the available operators. Typical examples are type information, or the connectivity between locations in a transportation task. One can get rid of such static predicates by simple pre-processing techniques [49] such as used in, e.g., FF [46]. In STRIPS, propositions that are statically true can be skipped from action preconditions or the goal; if a proposition is statically false then all actions can be removed whose preconditions contain the proposition. More generally, i.e., in ADL, one can replace a static proposition with its truth value in any condition formula, and propagate the changes through the formula. If the formula simplifies to false then the respective action/action effect can be removed. In the investigation at hand, i.e., in the proofs considering single domains, I often assume that static predicates have been removed. This technique does not affect the topology results, and it simplifies the formulation of sufficient criteria that generalize over individual domains.

3 Investigated Benchmark Domains

As said before, the investigation focusses on a total of 25 commonly used STRIPS and ADL benchmark domains. These include all (STRIPS and ADL versions of) the competition examples, plus 7 more domains used in the literature. The domain collection is not exhaustive – other domains have appeared in the literature. To the best of the author's knowledge, however, the collection *does* cover the large majority of the (STRIPS and ADL) example problems that have been used for evaluating AI planning systems during the last 8 years. Brief descriptions of the domains are given as starters into the respective subsections investigating the domains' topological properties. Full formal definitions of all domains can be looked up in Appendix A. Note that, for formally defining a domain, one must amongst other things decide what exactly the instances are. For almost none of the investigated domains is there such a definition in the literature. Naturally, the approach I have taken is to abstract from the known example suits. In most cases the abstraction is obvious, in the less obvious cases the respective subsection of Appendix A includes some explanatory remarks.

In the section at hand I provide an overview over the investigated domain collection. To give the overview some structure, domains with related semantics are grouped together: the groups are those of transportation domains, of construction domains, and of domains that belong into neither of the former two groups.

3.1 Transportation Domains

In many planning benchmark domains, there are a number of locations, a number of objects that must be transported, and a number of vehicles which are the means of transportation. Operators mostly either move a vehicle, or load (unload) an object onto (from) a vehicle. The domain collection we look at contains 12 variations of this theme. The domains differ in terms of various constraints. One remarkable difference is that, in many domains, vehicles can move instantaneously between any two locations, while in other domains the movable links between locations form arbitrary graphs. We refer to such graphs as *road maps*. In transportation domains without road maps, movements between locations are instantaneous.

1. *Briefcaseworld*. A briefcase moves, by conditional effects, all objects along that are inside it. Used in the literature, e.g. [60, 50].

2. *Driverlog*. Trucks need drivers on board in order to move. Trucks and drivers move along different arbitrary road maps, the only constraint being that the movement links are bi-directional – if one can move from x to y one can also move back. Used in the AIPS-2002 competition.

3. *Ferry*. A single ferry transports cars one at a time. Used in the literature, e.g. [23].

4. *Grid*. A robot transports keys on a grid-like road map where positions can be locked and must be opened with keys of matching shapes. Used in the AIPS-1998 competition.

5. *Gripper*. A robot with two gripper hands transports a number of balls (one at a time in each hand) from one room to another. Used in the AIPS-1998 competition.

6. *Logistics*. The classical transportation domain, where trucks/airplanes transport packages within/between cities. Used in the AIPS-1998 and AIPS-2000 competitions.
7. *Miconic-STRIPS*. An elevator transports passengers, using explicit actions to board/deboard passengers. Used in the AIPS-2000 competition.
8. *Miconic-SIMPLE*. Version of *Miconic-STRIPS* where passengers board/deboard "themselves" by conditional effects of the action that stops the elevator at a floor. Used in the AIPS-2000 competition.
9. *Miconic-ADL*. Version of *Miconic-SIMPLE* where various constraints must be obeyed (e.g., VIPs first). Used in the AIPS-2000 competition.
10. *Mprime*. Trucks use non-replenishable fuel items, and fuel can be transferred between locations. The movable links between locations form an arbitrary road map. Used in the AIPS-1998 competition.
11. *Mystery*. Like *Mprime* but without the possibility to transfer fuel. Used in the AIPS-1998 competition.
12. *Zenotravel*. Airplanes use fuel items which can be replenished one by one using a refuel operator. Used in the AIPS-2002 competition.

3.2 Construction Domains

The construction domains are generally not as closely related as the transportation domains above. What they (i.e., the construction domains) have in common, roughly, is that a complex object must be built out of its individual parts. There are 6 such domains in the collection we look at.

1. *Assembly*. A complex object must be assembled together out of its parts, obeying various ordering constraints. The parts themselves might need to be assembled beforehand. Used in the AIPS-1998 competition.
2. *Blocksworld-arm*. The classical construction domain, where blocks are picked up/put down or stacked onto/unstacked from each other by means of a robot arm. The goal is to re-arrange the blocks in a given stacking order. Used in the AIPS-2000 competition.
3. *Blocksworld-no-arm*. Like above, but blocks are moved around directly from a block to a block / from the table to a block / from a block to the table.
4. *Depots*. Combination of *Blocksworld-arm* and *Logistics*. Objects must be transported between locations before they can be stacked onto each other. Used in the AIPS-2002 competition.
5. *Freecell*. STRIPS encoding of the solitaire card game that comes with Microsoft Windows. The "complex object" to be built is formed by the goal stacks of cards (that have successfully been put away). Used in the AIPS-2000 competition.

6. *Hanoi*. STRIPS encoding of the classical Towers of Hanoi problem. Used in the literature, e.g. [48].

3.3 Other Domains

There are 7 domains in the collection whose semantics do not exactly fit into either of the above groups.

1. *Fridge*. For a number of fridges, the broken compressors must be replaced, involving several working steps (like unfastening all screws that hold the old compressors). Used in the literature, e.g. [48, 46].

2. *Movie*. In order to watch a movie, one must buy snacks, set the counter on the videorecorder to zero, and rewind the tape. Used in the AIPS-1998 competition.

3. *Rovers*. A number of rovers must navigate along a road map, take soil/rock samples as well as images, and communicate the resulting data to a lander. Used in the AIPS-2002 competition.

4. *Satellite*. A number of satellites must take images (of phenomena in space), using appropriate instruments. Used in the AIPS-2002 competition.

5. *Schedule*. A number of objects must be processed (painted, e.g.) on a number of machines. Used in the AIPS-2000 competition.

6. *Simple-Tsp*. A trivial STRIPS version of the TSP problem, where the move operator can be applied between any two locations. Used in the literature, e.g. [23].

7. *Tireworld*. A number of flat tires must be replaced, applying various working steps. Used in the literature, e.g. [5, 10].

4 Local Search Topology

This section defines the topological concepts which the subsequent investigation will be about. We start with the basic concepts of the forward state space and optimal relaxed distances, then proceed to dead end states and general topological phenomena. Though most of the definitions would make sense for unsolvable tasks, too, we restrict ourselves to solvable tasks from now on. After all, the main focus of the investigation is to explain why heuristic planners can *find plans* fast in many benchmark examples.

4.1 State Spaces, and $\mathbf{h}^+$

The search space we consider is, as said before, the space of all states that are reachable from the initial state: the so-called *state space*.

Definition 1 *Given a solvable task* (A, I, G)*. The set* S *of* reachable states *is*

$$S := \{s \mid \exists\, P \in A^* : Result(I, P) = s\}$$

The state space to (A, I, G) *is the directed graph* (S, E) *where the nodes are the reachable states, and the edge set is*

$$E := \{(s, s') \mid s, s' \in S,\ \exists\, a \in A : Result(s, \langle a \rangle) = s'\}$$

When searching the state space, one wants to focus on the "good" states. A state is the better the closer it is to the goal.

Definition 2 *Given a solvable task* (A, I, G)*, with state space* (S, E)*. The* goal distance *of* $s \in S$ *is*

$$gd(s) := min\{n \mid P = \langle a_1, \ldots, a_n \rangle \in A^*, P \text{ is plan for } (A, s, G)\}$$

where the minimum over an empty set is ∞*.*

A state s has zero goal distance $gd(s) = 0$ if and only if s fulfills the goal condition. We refer to such states as *goal states.* If, starting from a state s, the goal can not be reached anymore then the set in the above definition is empty, resulting in $gd(s) = \infty$. We will discuss such "dead end" states below in Section 4.2.

Computing the goal distance of a state is of course as hard as planning itself. So one needs a function that approximates gd: a heuristic. While the article focusses only on one specific heuristic, h^+ namely, the subsequent definitions do not depend on the particular heuristic function used. The formulations below are thus somewhat more general than strictly necessary for the investigation at hand.

Definition 3 *Given a solvable task* (A, I, G)*, with state space* (S, E)*. A* heuristic *is a function* $h : S \mapsto \mathbb{N} \cup \{\infty\}$*, such that* $h(s) = 0 \Leftrightarrow gd(s) = 0$*.*

The restrictions that we do make are that h is a function into the natural numbers, and returns 0 only for goal states. These assumptions are necessary to ensure the definitions below make sense. As an exception, h can return ∞ to indicate that the state at hand might be a dead end. This will be discussed below.

Algorithmically, the main problem is, how can one come up with a domain-independent heuristic function? The answer that has independently been proposed by McDermott [52] and Bonet et al. [11] is to relax the planning task by ignoring the delete lists.

Definition 4 *Given a STRIPS action* $a = (pre(a), add(a), del(a))$*, the* relaxation a^+ of a *is*

$$a^+ := (pre(a), add(a), \emptyset).$$

Given an ADL action $a = (pre(a), E(a))$*, the* relaxation a^+ of a *is defined as*

$$a^+ := (pre(a), \{(con(e), add(e), \emptyset) \mid e = (con(e), add(e), del(e)) \in E(a)\}).$$

For a set A *of actions, the* relaxation A^+ of A *is* $A^+ := \{a^+ \mid a \in A\}$*. The* relaxation of a planning task (A, I, G) *is* (A^+, I, G)*. An action sequence* $P = \langle a_1, \ldots, a_n \rangle \in A^*$ *is a* relaxed plan for (A, I, G) *if* $\langle a_1^+, \ldots, a_n^+ \rangle$ *is a plan for* (A^+, I, G)*.*

Definition 4 stands at the very center of the investigation, and its terminology will be used throughout the article which is why I introduce the rather many abbreviations.

As a measure of goal distance for a search state s, one is interested in how many relaxed actions one at least needs to execute in s in order to achieve the goal, i.e., one wants to know the length of an optimal relaxed plan for s.

Definition 5 *Given a solvable task* (A, I, G)*, with state space* (S, E)*. The* relaxed goal distance *of* $s \in S$ *is*

$$h^+(s) := min\{n \mid P = \langle a_1, \ldots, a_n \rangle \in A^*, P \text{ is relaxed plan for } (A, s, G)\}$$

where the minimum over an empty set is ∞*.*

As has been said before, computing h^+ is NP-hard, and the current state-of-the-art heuristic planners approximate h^+ in various ways in order to obtain search control knowledge.

4.2 Dead Ends

One phenomenon that is clearly relevant for the performance of heuristic state space search is that of *dead ends.* These are states from which the goal is unreachable.

Definition 6 *Given a solvable task* (A, I, G)*, with state space* (S, E)*. A state* $s \in S$ *is a* dead end *if* $gd(s) = \infty$*.*

As formalized in Definition 3, a heuristic function h can return $h(s) = \infty$ for a state s. Taking this as an indication that s is a dead end, the obvious idea for any forward search algorithm is to remove s from the search space. This technique is only adequate if s is really a dead end, i.e., if h is completeness-preserving in the following sense.

Definition 7 *Given a solvable task* (A, I, G)*, with state space* (S, E) *and heuristic* h*. The heuristic is* completeness preserving*, if* $h(s) = \infty \Rightarrow gd(s) = \infty$ *for all* $s \in S$*. With a completeness-preserving heuristic, a dead end state* s *is* recognized *if* $h(s) = \infty$ *and* unrecognized *otherwise.*

Note that h^+ is completeness preserving: if a task can not be solved even when ignoring the delete lists, then the task is unsolvable. From now on we assume that the heuristic we look at is completeness preserving. With respect to dead ends, any planning state space falls into one of the following four classes.

Definition 8 *Given a solvable task* (A, I, G)*, with state space* (S, E) *and completeness-preserving heuristic* h*. The state space is*

1. undirected*, if* $\forall (s, s') \in E : (s', s) \in E$
2. harmless*, if* $\exists (s, s') \in E : (s', s) \notin E$*, and* $\forall s \in S : gd(s) < \infty$

3. recognized, *if* $\exists s \in S : gd(s) = \infty$, *and* $\forall s \in S : gd(s) = \infty \Rightarrow h(s) = \infty$

4. unrecognized, *if* $\exists s \in S : gd(s) = \infty \wedge h(s) < \infty$

In the first class, there can be no dead ends because everything can be undone; in the second class, some things can not be undone, but those single-directed state transitions do not do any harm; in the third class, there are dead end states but all of them are recognized by the heuristic function. The only critical case for heuristic search is class four, where a search algorithm can run into a dead end without noticing it. This is particularly relevant if the unrecognized dead end can be arbitrarily deep in the sense that arbitrarily many other unrecognized dead end states can be reached from it.

Definition 9 *Given a solvable task* (A, I, G), *with state space* (S, E) *and completeness-preserving heuristic* h. *Let* s *be an unrecognized dead end. The* depth *of* s *is the number of states* s' *such that* s' *is an unrecognized dead end, and* s' *is reachable from* s *by a path that moves only through unrecognized dead ends.*

The subsequent investigation will, for each of the 25 benchmark domains looked at, determine exactly in which of the above four dead end classes the instances of the domain belong. For the domains where it turns out that there can be unrecognized dead ends – which as said before are five domains in total – I will construct examples showing that the unrecognized dead ends can be arbitrarily deep.

4.3 Plateaus

Let us now get into the definitions of general topological phenomena, i.e., of relevant properties of the search space surface. I adapt the definitions given for SAT by Frank et al. [26]. The difference between the SAT framework there and the planning formalism here is the directedness of state transitions. In the SAT search spaces considered by Frank et al, all state transitions can be traversed in both directions. As we have seen above already, single-directed state transitions can have an important impact on the search space topology, enabling for example the existence of dead ends.

Taking directed edges into consideration, the definitions are slightly more involved than the original ones by Frank et al., but move along the same lines of argumentation. The definitions are similar to what I used in my earlier empirical investigation [41]. The differences lie in a few simplifications made to focus the definitions on the issues relevant in the theoretical investigation at hand (some more on this below).

The base entity in the state space topology are what Frank et al. name *plateaus*. These are regions that are equivalent under reachability aspects, and look the same from the point of view of the heuristic function.

Definition 10 *Given a solvable task* (A, I, G), *with state space* (S, E) *and completeness-preserving heuristic* h. *For* $l \in \mathbb{N} \cup \{\infty\}$, *a* plateau P of level l

is a maximal subset of S for which the induced subgraph in (S, E) is strongly connected, and $h(s) = l$ for each $s \in P$.

The difference to the undirected case is that I require the states on the plateau to be *strongly* connected — with undirected state transitions this is trivially fulfilled by any set of connected states. Similar to Frank et al., I distinguish between plateaus in terms of the possibilities of leaving them, respectively their heuristic level.

Definition 11 *Given a solvable task (A, I, G), with state space (S, E) and completeness-preserving heuristic h. Let $P \subseteq S$ be a plateau. An* exit *of P is a state s that is reachable from P, such that there exists a state s', $(s, s') \in E$, and $h(s') < h(s)$.*

Exits are states from which one can leave a plateau, or more precisely the heuristic level on which the plateau lies. Note that we allow exits to not lie on the plateau P itself, namely when the path to the exit leaves P. Allowing exits of P to not lie on P itself is the main difference to the definitions given by Frank et al.

In what follows, we need the notion of *flat paths*, to capture when an exit state can be reached without temporarily worsening the heuristic value. Flat paths are paths in (S, E) where all states on the path have the same h value. Based on the behavior with respect to exits, Frank et al. distinguish between four different classes of plateaus: local minima (no exits lie on the plateau), benches (at least one exit lies on the plateau), contours (all states on the plateau are exits), and global minima (plateaus corresponding to a solution). The definition for directed state spaces is the following.

Definition 12 *Given a solvable task (A, I, G), with state space (S, E) and completeness-preserving heuristic h.*

- *(a) A* recognized dead end *is a plateau P of level $h = \infty$.*
- *(b) A* local minimum *is a plateau P of level $0 < h < \infty$ from which no exit is reachable on a flat path.*
- *(c) A* bench *is a plateau P of level $0 < h < \infty$ such that at least one exit is reachable from P on a flat path, and at least one state on P is not an exit.*
- *(d) A* contour *is a plateau P of level $0 < h < \infty$ that consists entirely of exits.*
- *(e) A* global minimum *is a plateau P of level 0.*

Each plateau belongs to exactly one of these classes. In solvable tasks, global minima are exactly the plateaus of level 0. Intuitively, the roles that the different kinds of plateaus play for heuristic search are the following: recognized dead ends can be ignored with a completeness-preserving heuristic function; local minima are difficult because all neighbors look worse so it is not clear in which direction to move next; benches are potentially easier because one can step off them in the direction towards the goal; from contours, one can step off immediately.

The main difficulty for a heuristic search is how to deal with the local minima and the benches. In both cases, when facing a search state s, a search algorithm must find a path to a state s' that is an exit, i.e., that has a neighbor with better heuristic value. How difficult it is to find such an exit can be assessed by a variety of different parameters. The size or diameter of the local minimum/the bench, and the number of exit states reachable, to name some important ones. In difference to my earlier empirical investigation [41], I do not formally define these parameters here. In the planning benchmarks, there are no (or very few) provable observations to be made about these parameters anyway. What one *can* frequently observe are properties of the *distance* to the nearest exit state.

The distance $dist(s, s')$ between any two states $s, s' \in S$ is the usual graph distance: the length of a shortest path from s to s' in (S, E), or ∞ if there is no such path. The exit distance of a search state s is defined as follows.

Definition 13 *Given a solvable task (A, I, G), with state space (S, E) and completeness-preserving heuristic h. Given a state $s \in S$. The* exit distance *of s is*

$$ed(s) := min\{dist(s, s') \mid h^+(s') = h^+(s), \exists s'' : (s', s'') \in E, h^+(s'') < h^+(s')\}$$

where the minimum over an empty set is ∞. The maximal local minimum exit distance *of (S, E) is*

$$mlmed(S, E) := max\{ed(s) \mid s \in S \text{ lies on a local minimum}\}$$

where the maximum over an empty set is 0. The maximal bench exit distance *of (S, E) is*

$$mbed(S, E) := max\{ed(s) \mid s \in S \text{ lies on a bench}\}$$

where the maximum over an empty set is 0.

As one would expect, the exit distance of a search state simply is the distance to the nearest exit state. Local minima and benches alike can be escaped from by finding an exit. The maximal difficulty of doing this is captured by the $mlmed(S, E)$ and $mbed(S, E)$ values: search needs to look at most that many steps ahead in order to find the desired state. We will find that, in the majority of the current planning benchmark domains, under h^+ there are constant upper bounds on $mlmed(S, E)$ and/or $mbed(S, E)$ that are valid across *all* instances of the domain. This is clearly a property that is likely to enable good scaling performance of any search algorithm that uses (an approximation of) h^+ as its heuristic function. In particular, FF's search algorithm is essentially an iteration of breadth-first searches for a state with better heuristic value [46]. With constant upper bounds on $mlmed(S, E)$ and $mbed(S, E)$ within a domain, this strategy terminates in polynomial time.

The following is an implication that is relevant for the subsequent investigation.

Proposition 1 *Given a solvable task (A, I, G), with state space (S, E) and a completeness-preserving heuristic h. If there exists an unrecognized dead end $s \in S$ then $mlmed(S, E) = \infty$.*

Proof: Let s be an unrecognized dead end, with minimal evaluation $h(s) > 0$. All states s' that are reachable from s are dead ends, and have therefore $h(s') \geq h(s)$. Thus the plateau on which s lies is a local minimum – no exits are reachable, in particular not on flat paths. This also shows that $ed(s) = \infty$. ∎

Proposition 1 tells us that, if there are unrecognized dead ends possible in a domain, then there can not be an upper bound on the maximal local minimum exit distance. The reason is that the unrecognized dead end with minimal heuristic level lies on a local minimum from which no exit states are reachable at all. In consequence, once an example unrecognized dead end is constructed in a domain, one does not need to verify the properties of the maximal local minimum exit distance anymore.

5 A Structural Core

In many of the investigated domains, similar patterns of structure — at the level of the planning task *specifications* — cause the characteristic qualities of h^+. These patterns of structure thus form the core of the theory behind the high quality of h^+. I now capture this structural core in the form of a number of definitions and lemmata. The lemmata formulate sufficient criteria implying that (the state space of) a planning task has certain topological properties. Proofs for domains will, where possible, proceed by applying the lemmata to arbitrary instances. For the sake of simplicity, I give the theory for STRIPS tasks only. The proofs for ADL tasks are along the same lines of argumentation.

5.1 Dead Ends

The first half of the structural core consists of criteria sufficient for the non-existence of dead ends. Our starting point is a reformulated version of results published by Koehler and Hoffmann [48]. We need the notion of *inconsistency*, which is defined as follows. Two facts are inconsistent if there is no reachable state that contains both of them. A set of facts F is inconsistent with another set of facts F' if each fact in F is inconsistent with at least one fact in F'.

If to each action there is an inverse action that undoes the action's effects, then the corresponding state space is undirected.

Definition 14 *Given a STRIPS planning task (A, I, G). An action $a \in A$ is* invertible*, if*

1. *$add(a)$ is inconsistent wth $pre(a)$,*
2. *$del(a) \subseteq pre(a)$, and*

3. there is an action $\overline{a} \in A$ such that

(a) $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$,

(b) $add(\overline{a}) = del(a)$, and

(c) $del(\overline{a}) = add(a)$.

For any state s and applicable action a, $\overline{a}$ is applicable in $Result(s, a)$ due to part 3(a) of Definition 14. Parts 1 and 2 of that definition make sure that a's effects do in fact appear, and parts 3(b) and (c) make sure that $\overline{a}$ undoes exactly those effects.

Lemma 1 *Given a STRIPS planning task (A, I, G). If all actions $a \in A$ are invertible, then the state space to the task is undirected.*

Proof:

Say we have a reachable state $s \in S$, and an action a that is applicable, $pre(a) \subseteq s$. We need to show that there is an action $\overline{a}$ such that $Result(Result(s, \langle a \rangle), \langle \overline{a} \rangle) = s$. By prerequisite, we can choose an action $\overline{a}$ that inverts a in the sense of Definition 14. Applying $\overline{a}$ in $Result(s, \langle a \rangle) =: s'$ (note here that $pre(\overline{a}) \subseteq s'$ because $s' \supseteq (pre(a) \cup add(a)) \setminus del(a))$ leads us back to s:

$$
\begin{aligned}
Result(s', \langle \overline{a} \rangle) &= (s' \cup add(\overline{a})) \setminus del(\overline{a}) \\
&= (((s \cup add(a)) \setminus del(a)) \cup del(a)) \setminus add(a) \\
&= (s \cup add(a)) \setminus add(a) \quad \text{as } del(a) \subseteq pre(a) \subseteq s \\
&= s \quad \text{as } add(a) \cap s = \emptyset
\end{aligned}
$$

For the last equation, $add(a) \cap s = \emptyset$ holds because $add(a)$ is inconsistent wth $pre(a) \subseteq s$. ■

All prerequisites of Lemma 1 are purely syntactical except inconsistency. Deciding inconsistency is as hard as planning itself.[2] There are however several polynomial time (incomplete but sufficient) approximation techniques for inconsistency in the literature [6, 22, 28, 63].

The next is a new criterion that is weaker, and implies only the non-existence of dead ends in the solvable case; that is, if the criterion holds then the state space of the task, if it is solvable, is at most harmless (either undirected or harmless).

For the non-existence of dead ends, it is not necessary that for each action a the inverse action $\overline{a}$ undoes *exactly* a's effects. It is already enough if $\overline{a}$ achieves (at least) all facts that have been true before. Actions for which such a weak form of inverse action exists are called *at least invertible.*

Definition 15 *Given a STRIPS planning task (A, I, G). An action $a \in A$ is* at least invertible, *if there is an action $\overline{a} \in A$ such that*

[2] For an arbitrary task, introduce two new propositions, one being permanently true and one which can only be made true if the original goal is achieved beforehand; the new propositions are inconsistent iff the original task is unsolvable.

1. $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$*,*

2. $add(\overline{a}) \supseteq del(a)$*, and*

3. $del(\overline{a})$ *is inconsistent with* $pre(a)$*.*

The first two conditions of the definition are obvious. As for the last point, note that we do not refer to a's add list at all. Inconsistency of $\overline{a}$'s deletes with a's preconditions implies that $\overline{a}$ deletes only facts that have not been true in s (but might have been made true by a's add list). Note also that Definition 14 is stronger than Definition 15. If $del(\overline{a}) = add(a)$, and $add(a)$ is inconsistent with $pre(a)$, then, of course, $del(\overline{a})$ is inconsistent with $pre(a)$.

Another situation when an action cannot lead into dead ends is the following: the action's delete effects are irrelevant — they are not important for anything in the task but the action itself; and the action's add effects are static, that is, not deleted by any action. By the latter property applying the action once is enough, and by the former property applying the action does not rule out any possibilities except applying it a second time.

Definition 16 *Given a STRIPS planning task* (A, I, G)*. An action* $a \in A$ has irrelevant delete effects*, if*

$$del(a) \cap (G \cup \bigcup_{a \neq a' \in A} pre(a')) = \emptyset$$

An action has static add effects*, if*

$$add(a) \cap \bigcup_{a' \in A} del(a') = \emptyset$$

If all actions in a solvable task are either at least invertible or have irrelevant delete effects and static add effects, then the state space is at most harmless.

Lemma 2 *Given a solvable STRIPS planning task* (A, I, G)*. If it holds for all actions* $a \in A$ *that either*

1. a *is at least invertible, or*

2. a *has irrelevant delete effects and static add effects,*

then there are no dead ends in the state space to the task, i.e., $gd(s) < \infty$ *for all* $s \in S$*.*

Proof: To any reachable state $s = Result(I, \langle a_1, \ldots, a_n \rangle) \in S$, we need to identify a solution P for (A, s, G). Let $\langle p_1, \ldots, p_m \rangle \in A^*$ be a solution for (A, I, G) (which exists as (A, I, G) is solvable by prerequisite). We construct P as follows.

$M := \emptyset$
for $i := n \ldots 1$ **do**
    if a_i is at least invertible by $\overline{a_i}$ **then**

if $\overline{a_i} \notin M$ apply $\overline{a_i}$ **endif**
else $M := M \cup \{a_i\}$
endif
endfor
for $i := 1 \dots m$ **do**
if $p_i \notin M$ **then** apply p_i **endif**
endfor

M serves as a kind of memory set for the actions that could not be inverted. We need to prove that the preconditions of all applied actions are fulfilled in the state where they are applied, and that the goals are true upon termination.

Let us start with the first loop. We denote by $s_i := result(I, \langle a_1, \dots, a_i \rangle)$ the state after executing the ith action on the path to s, and by s'_i the state before the first loop starts with value i. We prove (for use below) the stronger property

$$s'_i \supseteq (s_i \cap (G \cup \bigcup_{a \in A \setminus M_i} pre(a))) \cup \bigcup_{a \in M_i} add(a)$$

M_i here denotes the current state of the set. Applicability of actions is part of the proof. We proceed by backward induction over i. If $i = n$, we got $s'_i = s_i$ and $M_i = \emptyset$, so the equation is trivially true. Now say the equation is true for $i \geq 1$. We prove that the equation holds for $i - 1$. If a_i is not at least invertible, then no action is applied, $s'_{i-1} = s'_i$, and $M_{i-1} = M_i \cup \{a_i\}$. Concerning the left hand side of the equation, we observe that a_i does by prerequisite not delete any fact from $G \cup \bigcup_{a \in A \setminus M_{i-1}} pre(a)$ (M_{i-1} contains a_i), so all relevant facts from s_{i-1} have already been true in s'_i; concerning the right hand side of the equation, we observe that the facts in $add(a_i)$ are never deleted by prerequisite, so $\bigcup_{a \in M_{i-1}} add(a)$ is contained in s'_i. Now say that a_i is at least invertible by $\overline{a_i}$. We got $M_{i-1} = M_i$. Say $\overline{a_i}$ is applied. It is applicable because its preconditions are contained in s_i, and it is not an element of M_i. For the resulting state s'_{i-1}, all facts that a_i has deleted from s_{i-1} are added, and only facts are deleted that have not been true in s_{i-1} anyway; also, none of the add effects of actions in M_i is deleted, so the equation is fulfilled. Finally, if $\overline{a_i}$ is not applied, then its add effects are already contained in s'_i, subsuming the delete effects of a_i.

Inserting $i = 0$ in the equation we have just proven, we get

$$s_0 \supseteq (I \cap (G \cup \bigcup_{a \in A \setminus M_0} pre(a))) \cup \bigcup_{a \in M_0} add(a)$$

s_0 is the state from which the second loop is started. So we start a solution plan, excluding the actions in a set M_0, from a state including all initial facts that are contained in the goal or in the precondition of any action not in M_0. As the state additionally contains all add effects of all actions in M_0, and those add effects are not deleted by any action, it is clear that we can simply skip the actions in M_0 and achieve the goal. ■

Like it is the case for Lemma 1, all prerequisites of Lemma 2 are purely syntactical except inconsistency.[3]

[3] Note that Lemma 2 subsumes Lemma 1 in the sense that all tasks that fulfill the prerequisites

5.2 Local Minima

The second half of the structural core behind my proofs consists of a yet weaker notion of invertibility (invertibility in the relaxation), and of an important kind of relationship between the role of an action in the real task and its role in the relaxed task. Combining these definitions with the notion of irrelevant delete effects yields a criterion that is sufficient for the non-existence of local minima under h^+ (or, equivalently, for 0 being an upper bound on the maximal local minimum exit distance). The criterion can be directly applied in 7 of the 25 investigated domains, and can be applied with slight modifications in 3 more domains.

We start with the following simple technical observation.

Lemma 3 *Given a STRIPS task (A, I, G), a state s, and an action a that is applicable in s yielding the state s'. If P^+ is a relaxed plan for (A, s, G) that contains a, and $\overline{a} \in A$ is an action that is applicable in s' and for which $add(\overline{a}) \supseteq del(a)$ holds, then a relaxed plan for (A, s', G) can be constructed by removing a from P^+, and adding $\langle \overline{a} \rangle$ as a prefix.*

Proof: Without loss of generality, a starts the relaxed plan P^+ (a is applicable so one can execute its relaxed version immediately). After executing the relaxed version of $\overline{a}$ in s', all propositions are true that were true in s, plus the propositions that were made true by a. Thus, P^+ without a is a relaxed plan from there. ■

The importance of Lemma 3 is that (with the Lemma's terminology) there is a relaxed plan starting in s' that *has the same length* as the one for s. So if the relaxed plan for s was optimal, we can conclude that $h^+(s') \leq h^+(s)$ holds.

Examples of actions that qualify for the role of $\overline{a}$ in Lemma 3 are the inverse, and at least inverse, actions $\overline{a}$ in Definitions 14 and 15. These actions are always applicable in the state where a has been executed. Moreover, the add effects of an inverse action are exactly a's delete effects, and the add effects of an at least inverse action are a superset of a's delete effects. To fulfill the prerequisites of Lemma 3 the following weaker requirement is already sufficient.

Definition 17 *Given a STRIPS task (A, I, G). An action $a \in A$ is* at least relaxed invertible*, if there is an action $\overline{a} \in A$ such that $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$, and $add(\overline{a}) \supseteq del(a)$.*

The delete effects need not be mentioned in the above definition because $\overline{a}$, in the context of Lemma 3, is only used within the relaxation anyway. As indicated above, Definitions 14 and 15 are stronger than Definition 17, i.e., both invertible and at least invertible actions are also at least relaxed invertible.

Apart from (at least relaxed) invertible actions, the key property behind the lack of local minima under h^+ is, in most of the investigated domains, that every action that is good for solving the real task is also good for solving the relaxed task.

---

of the latter theorem also fulfill the prerequisites of the former.

Definition 18 *Given a task (A, I, G). An action $a \in A$* is respected by the relaxation *if, for any reachable state $s \in S$ such that a starts an optimal plan for (A, s, G), there is an optimal relaxed plan for (A, s, G) that also starts with a.*

Note that this definition is not restricted to STRIPS tasks, as the same terminology will be used in ADL tasks. As a simple example for an action that is respected by the relaxation, consider loading (unloading) a car onto (from) the ferry in *Ferry*: the only means of transporting a car is by using the ferry, so *all* plans, real or relaxed, must use the respective action if the car is not already at its goal position. We will say more on that topic below.

We now combine actions that are respected by the relaxation (Definition 18), the existence of actions that at least invert an action's effects in the relaxation (Definition 17), and irrelevant delete effects (Definition 16), to obtain the main tool for identifying planning tasks where h^+ does not yield any local minima.

Lemma 4 *Given a solvable STRIPS task (A, I, G), such that the state space (S, E) does not contain unrecognized dead ends. If each action $a \in A$*

1. *is respected by the relaxation, and*
2. *is at least relaxed invertible or has irrelevant delete effects,*

then there are no local minima in (S, E) under evaluation with h^+.

Proof: The states with $gd(s) = \infty$ are not on local minima by prerequisite, with $h^+(s) = \infty$. We will prove that, in each reachable state s with $gd(s) \neq \infty$, if an action a starts an optimal plan for (A, s, G) then $h^+(Result(s, \langle a \rangle)) \leq h^+(s)$. This proves the lemma: iterating the argument, we obtain a path to a goal state on which the value of h^+ decreases monotonically.

Let s be a reachable state with $gd(s) \neq \infty$. Let a be an action that starts an optimal plan for (A, s, G). We denote $s' := Result(s, \langle a \rangle)$. The action is respected by the relaxation so there is an optimal relaxed plan P^+ for (A, s, G) that starts with a. First case, removing a from P^+ yields a relaxed plan for (A, s', G). Then $h^+(s') < h^+(s)$ follows and we are finished. This is the case, in particular, if a has irrelevant delete effects: the propositions that a deletes are not needed by any other action nor by the goal, so P^+ without a achieves the goal starting from s' (where a has already been applied). Second case, assume removing a from P^+ does *not* yield a relaxed plan for s'. Then, with what was said before, a does not have irrelevant delete effects and must thus be at least relaxed invertible: there is an action $\overline{a} \in A$ with $pre(\overline{a}) \subseteq (pre(a) \cup add(a)) \setminus del(a)$ and $add(\overline{a}) \supseteq del(a)$. Applying Lemma 3 finishes the argument: the action $\overline{a}$ is guaranteed to be applicable in s', and re-achieves a's delete effects, so we can replace a in P^+ with $\overline{a}$ to obtain a relaxed plan for (A, s', G), yielding $h^+(s') \leq h^+(s)$. ■

As the argument in the proof only applies to solvable states, we must postulate that there are no or only recognized dead ends, like when the task at hand is undirected or harmless. Note that the second prerequisite of Lemma 4 is already fulfilled by planning tasks qualifying for the undirectedness or harmlessness criteria given by Lemmata 1 and 2. The missing bit that implies the non-existence of local minima under h^+ is then the first prerequisite, all actions being respected by the relaxation. The reason why this half of the lemma's prerequisites holds true in many of the benchmark domains is that, intuitively, if an action a is needed for an optimal solution, then this action either

- achieves some proposition that is needed by *all* solutions from the current state, no matter if relaxed or not, and that can not be achieved in a better way (there is no action achieving strictly more propositions than a); or
- achieves some proposition representing one way to get closer to the goals, where no other way is better, relaxed or not.

A typical (with respect to the benchmark collection) example for actions of the first kind are the *Ferry* actions mentioned above, loading (unloading) a car onto (from) the ferry — one has no other choice. A typical example for an action of the second kind is, in *Gripper*, picking up a ball that must still be transported, when both hands are empty: it does not make a difference which hand is used.

5.3 Exit Distances

Let us focus on maximal local minimum/bench exit distances. I do not have a general sufficient criterion implying upper bounds in either case (except the special case above where there are no local minima at all and thus 0 is an upper bound on the maximal local minimum exit distance). I do however have a proof technique that works in all the 8 domains where an upper bound on the maximal bench exit distance will be proved (in the absence of local minima). The proof technique works as follows. In planning tasks that fulfill the requirements for Lemma 4, optimal solution paths are flat in the sense that h^+ does not increase when one moves along them. So one can move along an optimal solution path from any reachable state s, and identify at which point, at the latest, a strictly shorter relaxed plan than the optimal one for s can be constructed. Relaxed plans here will be constructed along the path according to the proof of Lemma 4: one either replaces an applied action a with its inverse counterpart $\overline{a}$ (if its delete effects are still needed), or removes a completely (if its delete effects are irrelevant upon application). In the latter case, one is finished.

Proposition 2 *Given a planning task (A, I, G), a reachable state s, and an action a that starts an optimal relaxed plan P^+ for (A, s, G). If removing a from P^+ yields a relaxed plan for $(A, Result(s, a), G)$, then $h^+(Result(s, a)) < h^+(s)$, i.e., s is an exit state under h^+.*

Proof: Obvious. ∎

Note that this proposition corresponds to the first case in the proof of Lemma 4. Note also that the proposition's prerequisites are fulfilled if the action a is respected by the relaxation and has irrelevant delete effects.

As said before, we now focus on each single domain in turn, grouping domains with similar topological properties together in sections. The order of the sections and domains is such that the topological properties become gradually more complex, and the proofs become gradually more difficult.

6 No Local Minima and Bounded Maximal Bench Exit Distance

We first consider a number of domains where there are no local minima at all under h^+ (so by Proposition 1 there can neither be unrecognized dead ends), and the maximal bench exit distance is bounded by a constant. The domains are *Simple-Tsp*, *Movie*, *Gripper*, *Ferry*, *Logistics*, *Miconic-STRIPS*, *Miconic-Simple*, and *Tireworld*. We focus on them in that order. For each domain I give a brief description of the semantics, then consider the dead end class of the domain's instances, prove the non-existence of local minima, and determine an upper bound on the maximal bench exit distance.

6.1 Simple-Tsp

In *Simple-Tsp*, there is a single operator to move between locations. The precondition states that the locations must be different (which condition I have inserted to avoid absurd actions), and as an effect the destination location is visited. The instances specify a number of locations that must be visited, starting in one of them. See the domain details in Appendix A.23.

All instances of *Simple-Tsp* are solvable. We consider the tasks that result when removing static predicates (this is necessary in order to directly apply the sufficient criteria presented earlier). Removing static predicates in a *Simple-Tsp* task yields only actions that move between different locations, getting rid of the non-equality precondition.

Corollary 1 *Given an instance (C, I, G) of the Simple-Tsp domain, the corresponding state space is harmless.*

Proof: Obviously, the state space is not undirected, as visiting a location can not be undone. Furthermore, the actions are at least invertible in the sense of Definition 15: move(x, y) is at least inverted by move(y, x). With Lemma 2, we are finished. ■

Theorem 1 *Given an instance (C, I, G) of the Simple-Tsp domain, the corresponding state space does not contain local minima under h^+.*

Proof: As all actions are at least invertible, with Lemma 4 it suffices to show that all actions are respected by the relaxation. If a move(x, y) starts an optimal plan, then y has not yet been visited, so one optimal relaxed plan starts with the same action. ■

Let us introduce an abbreviation. Henceforth, if s is a reachable state in a task (A, I, G), and P^+ is a relaxed plan for (A, s, G), then we also denote this by saying *P^+ is a relaxed plan for s*.

Theorem 2 *Given an instance (C, I, G) of the Simple-Tsp domain, the maximal bench exit distance in the corresponding state space is 0 under h^+.*

Proof: By the argumentation in the proof of Theorem 1, all actions are respected by the relaxation. In any non-goal state s, an optimal plan starts by visiting one yet unvisited location, move(x, y). Let P^+ be an optimal relaxed plan for s that starts with a. We can assume that a is the only moving action in P^+ that starts in x: else, the actions of the form move(x, y') can be replaced by move(y, y'). Thus, removing a from P^+ yields a relaxed plan for $Result(s, \langle a \rangle)$, and s is an exit by Proposition 2. ∎

I remark that, trivially, in *Simple-Tsp* every optimal plan is also an optimal relaxed plan, implying $h^+ \equiv gd$, from which the two above claims follow immediately. I have chosen the formulation above to demonstrate that *Simple-Tsp* exhibits similar structural patterns like the other domains where I prove the non-existence of local minima and an upper bound on the maximal bench exit distance.

6.2 Movie

In *Movie*, there are seven different operators: one to rewind the tape, whose delete effect is that the counter is no longer at zero; one to reset the counter, whose only effect is that the counter is at zero; five to get different kinds of snacks, whose only effect is that one has the respective snack. Instances differ only in terms of the number of items that there are of each sort of snacks. The goal is always to have one snack of each sort, to have the tape rewound, and to have the counter at zero. See the domain details in Appendix A.17.

All instances of *Movie* are solvable.

Corollary 2 *Given an instance (C, I, G) of the Movie domain, the corresponding state space is harmless.*

Proof: The state space is not undirected because getting some snack, for example, can not be undone. All actions produced by one of the getting snacks operators have irrelevant delete effects (none at all) and static add effects in the sense of Definition 16. The rewind-movie() and reset-counter() actions are both at least invertible by reset-counter() in the sense of Definition 15. By applying Lemma 2, we are finished. ∎

Theorem 3 *Given an instance (C, I, G) of the Movie domain, the corresponding state space does not contain local minima under h^+.*

Proof: All actions produced by one of the getting snacks operators, and the reset-counter() action, have irrelevant delete effects (none at all), and the rewind-movie() action is at least relaxed invertible. With Lemma 4 it suffices to show that all actions are respected by the relaxation. This is trivial: if a snack is not yet there, one must get it; if the counter is not at zero, one must reset it; if the tape is not yet rewound, one must rewind it. ∎

Theorem 4 *Given an instance (C, I, G) of the Movie domain, the maximal bench exit distance in the corresponding state space is* 1 *under h^+.*

Proof: By the argumentation in the proof of Theorem 3, all actions are respected by the relaxation, and h^+ never increases on optimal solution paths. All actions except rewind-movie() have irrelevant delete effects, so states s where they start an optimal plan are exits by Proposition 2. Assume the only action starting an optimal plan for the state s is rewind-movie(), meaning that one has all snacks already. Then, applying rewind-movie() to s yields a successor state s' where reset-counter() starts an optimal plan (in fact, reset-counter() achieves the goal from s'), so an upper bound on the exit distance is 1. In the state s where one has got all snacks and the counter is at zero, but the movie is not rewound, no action except rewind-movie() can change s at all, and $h^+(s) = h^+(s') = 1$, so then the exit distance is exactly 1. ∎

6.3 Gripper

In *Gripper*, there is the operator move(x, y) to move between locations, the operator pick(x, y, z) to pick up a ball (at a location with a hand), and the operator drop(x, y, z) to drop a ball (at a location from a hand). There are always exactly two locations, room a and room b, and a number of balls shall be transported from a into b by using two gripper hands l and r that can each carry one ball at a time (we use the notation for instances as given in Appendix A.11). Instances differ only in terms of the number of balls. The moves are constrained to be between different rooms, and all operator parameters are constrained to be of the appropriate types, i.e., for example, in a move both parameters must fulfill a (room, x) predicate.

All instances of *Gripper* are solvable. Removing static predicates gets rid of the type information and of actions that try to move within the same room.

Corollary 3 *Given an instance (C, I, G) of the Gripper domain, the corresponding state space is undirected.*

Proof: After transformation with respect to static predicates, all actions are invertible in the sense of Definition 14: pick(x, y, z) is inverted by drop(x, y, z) and vice versa, and move(x, y) is inverted by move(y, x). By applying Lemma 1, we are finished. ∎

Theorem 5 *Given an instance (C, I, G) of the Gripper domain, the corresponding state space does not contain local minima under h^+.*

Proof: All actions are invertible, so with Lemma 4 it suffices to show that all actions are respected by the relaxation. If the optimal starting action in a reachable state s is a pick(x, y, z) or a drop(x, y, z), this holds because then the ball x is not yet in room b, and must be transported. If the optimal starting action is a move(x, y), then there are either balls left to pick in $y = a$, or balls left to drop in $y = b$, so a relaxed plan from s has no choice but applying the respective move action. ■

Theorem 6 *Given an instance (C, I, G) of the Gripper domain, the maximal bench exit distance in the corresponding state space is at most* 1 *under h^+.*

Proof: By the argumentation in the proof of Theorem 5, all actions are respected by the relaxation; the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. Assume we have a reachable state s and an optimal starting action *act* (we use that name to avoid confusion with the *room* a used in the instance), yielding the successor state s'. Assume also that *act* starts the optimal relaxed plan P^+ for s. If $act = \text{drop}(x, y, z)$ then $y = b$, and removing *act* from P^+ yields a relaxed plan for s' as x is in its goal position and (carry, x, z) is not needed anymore; thus in that case s is an exit. If $act = \text{pick}(x, y, z)$ then $y = a$, and P^+ also contains the actions move(a, b) and drop(x, b, z). A relaxed plan for s' can be constructed by removing *act* from P^+ and placing the two other actions up front: that way, (free, z) is re-achieved before any other action is applied; the deletion of (at, x, a) does not matter as x is picked up only once due to the optimality of P^+. So in that case, s is also an exit. Finally, consider the case where $act = \text{move}(x, y)$. Then, the successor state s' falls into one of the two above cases, so an upper bound on the exit distance is 1. ■

If, in the situation described at the end of the proof to Theorem 6, *pick*s or *drop*s are left to do in room x, then the relaxed plan from s' must contain the inverse action move(y, x), so then the exit distance is exactly 1.

6.4 Ferry

Ferry is very closely related to *Gripper*. There are three operators, one to sail a ferry between two locations, one to board a car onto the ferry, and one to debark a car from the ferry. The ferry can transport only one car at a time, which is realized by an (empty-ferry) predicate. The goal is to have a subset of the cars at their goal locations. The sailing actions are constrained to be between different locations, and all operator parameters are constrained to be of the appropriate types, i.e., for example, in a debark(x, y) action x must fulfill a (car, x) predicate while y must fulfill a $(\text{location}, y)$ predicate. See the domain details in Appendix A.7.

All instances of *Ferry* are solvable. Removing static predicates gets rid of the type information and of actions that try to move within the same location.

Corollary 4 *Given an instance (C, I, G) of the Ferry domain, the corresponding state space is undirected.*

Proof: After transformation with respect to static predicates, all actions are invertible in the sense of Definition 14: board(x, y) is inverted by debark(x, y) and vice versa, and sail(x, y) is inverted by sail(y, x). By applying Lemma 1, we are finished. ∎

Theorem 7 *Given an instance (C, I, G) of the Ferry domain, the corresponding state space does not contain local minima under h^+.*

Proof: All actions are invertible, so with Lemma 4 it suffices to show that all actions are respected by the relaxation. If the optimal starting action in a reachable state s is a board(x, y) or a debark(x, y), this holds because then the car x is not yet at its goal location, and must be transported. If the optimal starting action is a sail(x, y), then at location y there is either a car to board, or a car to debark, so a relaxed plan from s will have to visit that location. When this is done does not matter in the relaxation, as the (at-ferry$, y$) proposition remains true throughout relaxed plan execution. So one can start an optimal relaxed plan from s by sailing the ferry to y. ∎

Theorem 8 *Given an instance (C, I, G) of the Ferry domain, the maximal bench exit distance in the corresponding state space is at most* 1 *under h^+.*

Proof: By the argumentation in the proof of Theorem 7, all actions are respected by the relaxation; the instance fulfills the prerequisites of Lemma 4, so h^+ never increases on optimal solution paths. Assume we have a reachable state s and an optimal starting action a, yielding the successor state s'. Assume also that a starts the optimal relaxed plan P^+ for s. If $a =$ debark(x, y) then either y is the goal location of x, or x has no goal location, so removing a from P^+ yields a relaxed plan for s' as (on$, x$) is not needed anymore; thus in that case s is an exit. If $a =$ board(x, y) then x must be transported, and P^+ also contains the actions sail(y, y') and debark(x, y'), where y' is the goal location of x; a relaxed plan for s' can be constructed by removing a from P^+ and placing the two other actions up front: that way, (empty-ferry) is re-achieved before any other action is applied; the deletion of (at$, x, y$) does not matter as x is boarded only once due to the optimality of P^+. So in that case, s is also an exit. Finally, consider the case where the optimal starting action is $a =$ sail(x, y). Then, the successor state s' falls into one of the two above cases — some car is boarded or debarked — so an upper bound on the exit distance is 1. ∎

If, in the situation described at the end of the proof to Theorem 8, a car is left at x which must be boarded later, or some other car must still be transported to x, then the relaxed plan from s' must contain the inverse action sail(y, x), so then the exit distance is exactly 1.

6.5 Logistics

In *Logistics*, there are six operators to drive a truck between two locations within a city, to fly an airplane between two airports, to load (unload) an object onto (from) a truck at a location, and to load (unload) an object onto (from) an airplane at an airport. There is always at least one city, and each city has a non-zero number of locations one of which is an airport. There is an arbitrary number of airplanes (which are located at airports) and of objects. The goal is to have a subset of the objects at their goal locations. The moves are constrained to be between different locations, and all operator parameters are constrained to be of the appropriate types. See the domain details in Appendix A.13.

An instance of *Logistics* can be unsolvable when there is an object that must be transported, but there is no adequate vehicle for doing so. Removing static predicates gets rid of the type information and of actions that try to move within the same location.

Corollary 5 *Given an instance (C, I, G) of the Logistics domain, the corresponding state space is undirected.*

Proof: After transformation with respect to static predicates, all actions are invertible in the sense of Definition 14: load-truck(x, y, z) actions are undone by unload-truck(x, y, z), symmetrically for other loading or unloading actions; drive-truck(x, y, z, v) is undone by drive-truck(x, z, y, v), symmetrically for fly-airplane actions. By applying Lemma 1, we are finished. ■

Theorem 9 *Given a solvable instance (C, I, G) of the Logistics domain, the corresponding state space does not contain local minima under h^+.*

Proof: As all actions are invertible, with solvability (implying the non-existence of dead ends) and Lemma 4 it suffices to show that all actions are respected by the relaxation. If the optimal starting action a in a reachable state s is any loading or unloading action, then a is respected by the relaxation because in that case the respective package is not yet at its goal location, so it must be transported, and it is best — for relaxed as well as for real plans — to do that when an adequate vehicle is already at the respective location. If the optimal starting action a is a driving or flying action to some location z, then some package must either be loaded or unloaded at z, so a relaxed plan from s has no choice but applying some action moving a transportation vehicle (of a's kind) there. All vehicles are equally good, except when there is a clever choice, i.e., a vehicle that already carries packages to be unloaded at z; but then, a will move one of those vehicles just like an optimal relaxed plan will, and all such vehicles are equally good in the relaxation. ■

Theorem 10 *Given a solvable instance (C, I, G) of the Logistics domain, the maximal bench exit distance in the corresponding state space is at most* 1 *under h^+.*

Proof: By the argumentation in the proof of Theorem 9, all actions are respected by the relaxation; the instance fulfills the prerequisites of Lemma 4, so h^+ monotonically decreases on optimal solution paths. Assume we have a reachable state s and an optimal starting action a, yielding the successor state s'. Assume also that a starts the optimal relaxed plan P^+ for s. If a is a loading (unloading) action, its only delete is the at (in) relation of the transported object; as the object is loaded from the respective location (unloaded from the respective vehicle) only once in P^+, removing a from P^+ yields a relaxed plan for s', so s is an exit. Otherwise, consider the case where a drives or flies some vehicle x from y to z. Then s' is an exit: no optimal plan moves the same vehicle twice in a row — all locations are immediately accessible, if accessible at all — so one optimal plan for s' starts by loading (unloading) some package to (from) x. Thus an upper bound on the exit distance is 1. ■

If, for example, in the situation described at the end of the proof to Theorem 10, x is a truck, y is the airport, and the package must be transported to a different city, then the relaxed plan from s' must include the action driving x back from z to y, so then the exit distance is exactly 1.

6.6 Miconic-STRIPS

In *Miconic-STRIPS*, there are four operators to move the lift up from one floor to another floor, to move the lift down from one floor to another floor, to board a passenger at a floor (the precondition being that the floor is the passenger's origin, and the only effect being that the passenger is boarded), and to let a passenger depart at a floor (the preconditions being that the floor is the passenger's destination and that the passenger is boarded, and the effects being that the passenger is served but no longer boarded). There is always at least one floor, and an arbitrary number of passengers each of which is given an origin and a destination floor. The goal is to serve all passengers. Lift moves are constrained so that the floors stand in the appropriate relation to each other (i.e., one can only move up from x to y if y is indeed above x, and vice versa), and all operator parameters are constrained to be of the appropriate types. See the domain details in Appendix A.16.

All instances of *Miconic-STRIPS* are solvable. Removing static predicates gets rid of the type information, of actions that try to move between inappropriate (in particular identical) locations, and of actions that board or depart a passenger at a location other than its origin respectively destination.

Corollary 6 *Given an instance (C, I, G) of the Miconic-STRIPS domain, the corresponding state space is at most harmless.*

Proof: The lift move actions are invertible in the sense of Definition 14, the boarding actions are at least invertible in the sense of Definition 15, and the departing actions have irrelevant delete effects and static add effects in the sense of Definition 16. With Lemma 2, we are finished. ■

Because boarding and departing can not be undone, the state space is not undirected if there is at least one passenger.

Theorem 11 *Given an instance (C, I, G) of the Miconic-STRIPS domain, the corresponding state space does not contain local minima under h^+.*

Proof: All actions are either at least invertible or have irrelevant delete effects. Therefore, with Lemma 4 it suffices to show that all actions are respected by the relaxation. For boarding and departing actions, this is obvious. For up(x, y) and down(x, y) actions, if an optimal plan starts with such an action, then at least one passenger must board or depart at floor y, so any relaxed plan will have to visit that floor. When this is done does not matter in the relaxation — the (lift-at, y) proposition remains true throughout relaxed plan execution — so one can start an optimal relaxed plan by moving the lift to y. ∎

Theorem 12 *Given an instance (C, I, G) of the Miconic-STRIPS domain, the maximal bench exit distance in the corresponding state space is at most 1 under h^+.*

Proof: By the argumentation in the proof of Theorem 11, all actions are respected by the relaxation; the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. Assume we have a reachable state s and an optimal starting action a, yielding the successor state s'. The boarding and departing actions have irrelevant delete effects (boarding has no delete effects at all), so if a is one of those actions, then s is an exit by Proposition 2. In the other case, if a is an up or down action, then s' falls into the first case because no optimal plan moves the lift twice in a row (all floors are directly accessible from each other). So an upper bound on the exit distance is 1. ∎

If, for example, in the situation described at the end of the proof of Theorem 12, there are two floors, the lift is in the bottom floor in s, and the single passenger wants to travel from the top to the bottom floor, then the relaxed plan from s' must include the action moving the lift back from the top to the bottom floor, so then the exit distance is exactly 1.

6.7 Miconic-SIMPLE

The *Miconic-SIMPLE* domain is semantically rather similar to the *Miconic-STRIPS* domain we have discussed above. The difference is that boarding and departing passengers is not done by separate actions, but by conditional effects that can appear when the lift *stops*. Once the lift is stopped, all boarded passengers that have their destination at the respective floor depart, and all passengers still waiting at the respective floor are boarded. The former conditional effects add that the respective passengers are served, and delete that they are boarded. The latter conditional effects require that the respective passengers are not served (on top of having their origin at the respective floor), and add that they are boarded. The lift move operators are identical to those in *Miconic-STRIPS*. The instances are also the same. All operator and effect parameters are constrained to be of the appropriate types. See the domain details in Appendix A.15.

All instances of *Miconic-SIMPLE* are solvable. Removing static predicates gets rid of the type information, of actions that try to move between identical locations, and of conditional effects that board or depart a passenger at a location other than its origin respectively destination. My sufficient criteria for the non-existence of dead ends are not directly applicable to *Miconic-SIMPLE* instances, so I insert a short proof.

Proposition 3 *Given an instance (C, I, G) of the Miconic-SIMPLE domain, the corresponding state space is at most harmless.*

Proof: The floors are all directly accessible from each other. The deletes of the stopping effects are all irrelevant in the sense that they are not needed anymore once those effects have occured. One can thus, from any reachable state s, construct a plan by moving the lift back to the floor where it was in the initial state, and thereafter executing a plan for the task. ∎

If there is at least one passenger, the instance is not undirected, as the effects of the stop actions can not be undone. Let us focus on local minima and the maximal exit distance.

Theorem 13 *Given an instance (C, I, G) of the Miconic-SIMPLE domain, the corresponding state space does not contain local minima under h^+.*

Proof: We show that all actions are respected by the relaxation. This suffices with the following slight extension of Lemma 4. The up(x, y) and down(x, y) actions are invertible in the sense of Definition 14. The delete effects of the stop(x) actions are conditional and thus are not irrelevant in the strict syntactical sense of Definition 16. They do, however, have the same semantics — the only delete effects are that a passenger reaching the destination is no longer boarded and no longer not-served. These propositions are irrelevant upon execution of the action. If the actions are respected by the relaxation, one can thus construct, from any state, a monotonically decreasing path to the goal in parallel to the proof of Lemma 4.

The up(x, y) and down(x, y) actions are respected by the relaxation due to the same reasons as given for *Miconic-STRIPS* in the proof of Theorem 11. Assume $a =$ stop(x) starts an optimal plan for some reachable state s. Then a also starts an optimal relaxed plan for s. We prove that by contradiction. Assume that no optimal relaxed plan for s starts with stop(x), although the lift is already located at x. Then the following must hold. First, there is a different floor x' where another passenger with destination x is waiting; second, there is no passenger for x' waiting at x. But in this situation, executing the stop at x' first would also yield a shorter real solution plan. ∎

Theorem 14 *Given an instance (C, I, G) of the Miconic-SIMPLE domain, the maximal bench exit distance in the corresponding state space is at most* 1 *under h^+.*

Proof: By the argumentation in the proof of Theorem 13 all actions are respected by the relaxation, h^+ never increases on optimal solution paths, and the stop(x) actions have delete effects that are irrelevant once they have appeared. Assume we have a reachable state s and an optimal starting action a, yielding the successor state s'. If a is a stop(x) action, then s is an exit by Proposition 2. In the other case, if a is an up or down action, then s' falls into the first case because no optimal plan moves the lift twice in a row (all floors are directly accessible from each other). So an upper bound on the exit distance is 1. ∎

In the same situation that is described for the *Miconic-STRIPS* domain in Section 6.6 below Theorem 12, the exit distance is exactly 1.

6.8 Tireworld

In *Tireworld*, one must replace a number of flat tires. This involves a collection of objects that must be used in the appropriate working steps. The details are in Appendix A.24. Briefly summarized for the proofs below, the situation is as follows. There is a boot that can be either opened or closed; it is initially closed and shall be so in the end. There are a pump, a wrench, and a jack which can be fetched or put away (from/into the boot); they are initially in the boot and shall be put back in the end. The spare wheels are initially not inflated, and can be inflated using the pump (the add effect is that the wheel is inflated, the delete effect is that it is no longer not-inflated). Each hub is fastened with nuts; these can be loosened or tightened, using the wrench, while the respective hub is on ground. The jack can be used to either jack up or jack down a hub. Once a hub is jacked up, one can undo the (loose) nuts, or do them up; if the nuts are undone, one can remove the respective wheel, or put on one. An optimal solution plan is this: open the boot; fetch the tools; inflate all spare wheels; loosen all nuts; in turn jack up each hub, undo the nuts, remove the flat wheel, put on the spare wheel, do up the nuts, and jack the hub down again; tighten all nuts; put away the tools; and close the boot.

All instances of *Tireworld* are solvable.

Corollary 7 *Given an instance (C, I, G) of the Tireworld domain, the corresponding state space is harmless.*

Proof: The state space is not undirected because inflating a wheel can not be undone. Transforming with respect to static predicates makes sure that actions are type correct i.e., put-on-wheel(x, y) actions for example are only there for wheels x and hubs y. After transformation, all actions except inflate(x) are invertible in the sense of Definition 14 (for example, open-boot() and close-boot() invert each other). As inflate(x) actions have irrelevant delete effects and static add effects in the sense of Definition 16, we are finished with Lemma 2. ∎

Before I start with the proofs, one technical remark on the notation: in the formal definition of the domain, see Appendix A.24, some operators have a parameter — the tool used — that must always be grounded with the same constant — the one representing the respective tool. These parameters serve to replace the domain constants that are used in the original formulation. I omit these parameters here to improve readability (as an example, I write inflate($x \ldots$) instead of inflate(x, y) where y must always be the pump for the action to be applicable).

Theorem 15 *Given an instance (C, I, G) of the Tireworld domain, the corresponding state space does not contain local minima under h^+.*

Proof: The inflate($x \ldots$) actions have irrelevant delete effects, and all other actions are invertible, so with Lemma 4 it suffices to show that all actions are respected by the relaxation. This is obvious: if any action a starts an optimal plan from a reachable state, then that action adds a proposition that must be achieved, and that can only be achieved by a. As an example, consider the actions that open/close the boot, or loosen/tighten nuts. If those actions are needed to achieve the goal, they can not be avoided by the relaxation: as for opening (closing) the boot, this is necessary to fetch tools (achieve the respective goal); as for loosening (tightening) the nuts, this is necessary for removing the respective flat wheel (achieving the respective goal). ∎

The proof regarding the maximal bench exit distance is simple but a bit difficult to formulate in detail, due to the large number of different working steps. For notation details, consider Appendix A.24 (flat wheels are w_i, spare wheels are s_i, hubs are h_i, and nuts are n_i). The maximal bench exit distance in *Tireworld* is, in difference to the other domains we have seen so far, where the constant is either 0 or 1, bounded by 6. Roughly, this is the "depth" of the working process, i.e., the maximum number of working steps we need to take before we can apply an action that does not need to be undone later on.

Theorem 16 *Given an instance (C, I, G) of the Tireworld domain, the maximal bench exit distance in the corresponding state space is at most* 6 *under h^+.*

Proof: By the argumentation in the proof of Theorem 15, all actions are respected by the relaxation; the instance fulfills the requirements for Lemma 4, so h^+ never increases on optimal solution paths. It now suffices to show that a state s' with a better evaluated neighbor can be reached on an optimal solution path from any reachable state s in at most 6 steps. We will use the fact that a state s' is an exit if there is an action a starting an optimal plan for (A, s', G), such that removing a from an optimal relaxed plan P^+ for (A, s', G) yields a relaxed plan for $(A, Result(s', \langle a \rangle), G)$, c.f. Proposition 2.

Consider the actions close-boot(), put-away(x), inflate($x \ldots$), tighten($x, y \ldots$), do-up($x, y \ldots$), remove-wheel(x, y), and put-on-wheel(x, y). For all these actions it holds that when they start an optimal plan for a state, then the propositions that they delete are not needed anymore after applying them, neither by a real nor by a relaxed plan: a do-up($x, y \ldots$) action, for example, deletes (have, x) and (not-fastened, y) for nuts x and hub y; if do-up($x, y \ldots$) starts an optimal plan, then this means that the spare wheel has been put on the hub y already and that the nuts belong to the respective hub, so neither of both propositions will be needed for achieving the goals. Our proof thus comes down to identifying in at most how many steps on an optimal path a state s' can be reached from which one of the above actions starts an optimal plan. We distinguish the following cases.

1. There is a spare wheel that is not inflated, $(\text{not-inflated}, s_i) \in s$ for some s_i. The wheel must be inflated, for which one must at most open the boot and fetch the pump, yielding the distance 2.

2. The above case 1 does not hold, and there is a flat wheel that is still on, or a spare wheel that is on the wrong hub, i.e., $(\text{on}, w, h_j) \in s, (\text{on}, w, h_j) \notin G$ for some w and h_j. We identify an optimal path to a state where remove-wheel(w, h_j) can be applied (and thus starts an optimal plan). First, one must at most open the boot and fetch the wrench (maximal 2 steps). If nuts are tightened to h_j and h_j is jacked up, then h_j must be jacked down; if some other hub with a flat or wrong wheel on is already jacked up, one must at most jack down that hub (which is necessary anyway in order to loosen the nuts on the respective hub); if a hub with a good wheel is jacked up, one can jack down that hub, yielding a better state; else, the jack must at most be fetched from the boot (maximal 1 more step in all four cases). Loosening the nuts tightened to h_j, jacking up h_j, and undoing the respective nuts takes at most another 3 steps, so our path is at most 6 steps long. Note that all steps on the path are unavoidable implying that the path is an optimal solution path from s.

3. The above cases do not hold, and there is a hub that is free, $(\text{free}, h_i) \in s$ for some h_i. One must put on the respective spare. If the spare is in the boot, then fetching it yields a state with better h^+. Else, one has the spare already (otherwise case 2 would hold). One must at most perform the same steps as in case 2 (note that nuts can be done up and tightened to h_i even if there is no wheel on it), yielding again an unavoidable path of maximal length 6.

4. The above cases do not hold, and some nuts are still not done up to the correct hub, $(\text{loose}, n_i, h_i) \notin s$ for some n_i and h_i with $(\text{tight}, n_i, h_i) \in G$. If the nuts are in the boot, fetching them yields a better state. If some hub is jacked up, we can assume that either the correct nuts are fastened to that hub, or that the hub is h_i. Therefore, if some other nuts are fastened to h_i, maximal 5 unavoidable steps similar as above lead to a state where those other nuts can be undone, yielding a better state; otherwise maximal 4 unavoidable steps lead to a state where do-up$(n_i, h_i \dots)$ can be applied.

5. The above cases do not hold, and a hub is jacked up, $(\text{not-on-ground}, h_i) \in s$ for some h_i. The hub can immediately be jacked down, improving the h^+ value.

6. The above cases do not hold, and some nuts are still loose, $(\text{loose}, n_i, h_i) \in s$ for some n_i and h_i. The nuts must be tightened which involves at most opening the boot and fetching the wrench.

7. If none of the above cases holds, then there is either a tool or a flat wheel that can be put away, or one can close the boot.

The cases are exhaustive, and in none of them does one need more than 6 steps in order to reach an exit. ■

The maximal bench exit distance is in fact exactly 6 for all *Tireworld* instances. Consider the state where all hubs except one have already been processed completely (spare wheel inflated and on, nuts tight, flat wheel in boot), the boot is closed, the wrench and the jack are in the boot, the last spare wheel is already inflated, and nothing has yet been done about the last flat wheel. We are in the second case of the list given in the above proof of Theorem 16, and we require the whole sequence of opening the boot, fetching the wrench and the jack, loosing the nuts, jacking up the hub, and undoing the nuts in order to get to a state where the flat wheel can be removed. All the actions in the sequence must be inverted later on, and there is no other way of improving the h^+ value, so there the exit distance is exactly 6.

7 No Local Minima

The second group of domains we consider have no local minima under h^+ (so by Proposition 1 neither have unrecognized dead ends), but an unbounded maximal bench exit distance. The domains are *Briefcaseworld*, *Fridge*, *Blocksworld-no-arm*, *Grid*, and *Hanoi*. We focus on them in that order. For each domain I give a brief description of the semantics, then consider the dead end class of the domain's instances, prove the non-existence of local minima, and give an example showing that there is no upper bound on the maximal bench exit distance. For *Fridge*, I also show that the maximal bench exit distance grows in the number of screws in an instance, but not in the number of fridges.

7.1 Briefcaseworld

In *Briefcaseworld*, a number of portables must be transported, where the transportation is done via conditional effects of the move actions. There are three operators: putting in a portable at a location can be done if the portable and the briefcase are at the respective location, and the portable is not yet inside; taking a portable out can be done if it is inside; a move can be applied between two locations, and achieves, beside the is-at relation for the briefcase, the respective at relations for all portables that are inside (i.e., the portables inside are moved along by conditional effects). The goal is to have a subset of the portables, and the briefcase, at their goal locations. The move actions are constrained to be between different locations, and all operator parameters are constrained to be of the appropriate types. See the domain details in Appendix A.4.

All instances of *Briefcaseworld* are solvable. Removing static predicates gets rid of the type information and of actions that try to move within the same location.

Proposition 4 *Given an instance (C, I, G) of the Briefcaseworld domain. The corresponding state space is undirected.*

Proof: Though none of the actions is invertible in the strict syntactical sense of Definition 14, to each reachable state and applicable action a yielding the successor state s' there is an action $\overline{a}$ leading back to s from s'. If a is a move(x,y), it can be undone by move(y,x). If a is a put-in(x,y), it can be undone by take-out(x). If a is a take-out(x), it can be undone by put-in(x,y) where y is the current location of the briefcase and x (with x being inside prior to the application of take-out(x), it is at the same location as the briefcase in s). ■

Theorem 17 *Given an instance (C, I, G) of the Briefcaseworld domain, the corresponding state space does not contain local minima under h^+.*

Proof: I prove that the heuristic value does not increase on optimal solution paths. The put-in(x,y) action is respected by the relaxation, because if such an action starts an optimal plan, then x is not at its goal location, so it must be put in. Also, as seen in the proof to Proposition 4, the action can be inverted. Now assume we got a reachable state s, and an optimal plan for (A, s, G) starting with a take-out(x) action. Assume P^+ is an optimal relaxed plan for (A, s, G). Then P^+ is also an optimal relaxed plan for (A, s', G), where s' is the state that results from applying take-out(x) to s: x is already at its goal location, so P^+ does not do anything with it (x can be kept inside the briefcase in the relaxation, as moving away does not delete its current location there). Finally, assume we got a reachable state s and an optimal plan for (A, s, G) starting with a move(x,y) action. Assume P^+ is an optimal relaxed plan for (A, s, G), and s' is the state that results from applying move(x,y) to s. If P^+ starts with move(x,y), then we can replace move(x,y) in P^+ with move(y,x) to form a relaxed plan for (A, s', G), so we are finished then. In the other case, P^+ starts with some other move action move(x,z) (because moving away starts an optimal real plan, no portables must be put in at the current location). We construct a relaxed plan for (A, s', G) from P^+ by: replacing move(x,y), if it is contained in P^+, with move(z,y) (so the at relations that were achieved by move(x,y) in P^+ are now achieved by move(z,y); note that (is-at$,z$) is achieved by move(x,z)); and replacing all other move(x,y'), $y' \neq y$, actions in P^+ with move(y,y') (so the respective at relations are achieved; note that (is-at$,y$) is fulfilled in s'). ■

The shortest path to an exit can be arbitrarily long in the *Briefcaseworld* domain because taking out a portable is not necessary in a relaxed plan. Consider the following instances. There are two locations l_1 and l_2 and n portables $p_1, \ldots, p_n$. The briefcase is initially at l_1 and shall be at l_1 for the goal. The portable p_1 must be transported from l_2 to l_1, and the portables $p_2, \ldots, p_n$ must be transported from l_1 to l_2. An optimal plan puts in all of $p_2, \ldots, p_n$, moves to l_2, and puts in p_1; it then takes out all of $p_2, \ldots, p_n$, and moves back to l_1. Consider the state s where the briefcase is in l_2, and all portables are inside. $h^+(s) = 1$, as moving to l_1 does the job in the relaxation. However, the nearest state s' with $h^+(s') < h^+(s)$ is the nearest goal state, which is n steps away.

7.2 Fridge

In *Fridge*, one must replace the broken compressor in a fridge. To do this, one must remove the compressor; this involves unfastening the screws that hold the compressor, which in turn involves first switching the fridge off. The goal is to have the new compressor attached to the fridge, all screws fastened, and the fridge switched back on. The origin of this domain is a STRIPS formulation. My adaption allows for an arbitrary number of fridges and screws, where each compressor is fastened by the same (arbitrary, at least one) number of screws. The adaption involves introducing a complex precondition: a compressor can only be removed if *all* screws are unfastened.

The discussion is probably easier to understand after a look at the formal definition of the domain in Appendix A.9. A brief summary is this. There are six operators. One can stop (start) a fridge, which exchanges the respective (fridge-on, x) with the respective (not-fridge-on, x) proposition (vice versa). One can unfasten (fasten) a screw from (to) a compressor attached to a fridge, which exchanges the respective (screwed, x) with the respective (not-screwed, x) proposition (vice versa). To do that, the fridge needs to be turned off, the compressor needs to be attached, and the screw must fit the compressor (each screw fits only the compressor it is originally fastened to, and that compressor's replacement). Finally, one can remove (attach) a compressor from (to) a fridge, which exchanges the respective (attached, x, y) proposition with the respective (free, x) and (free, y) propositions for the fridge and the compressor. Removing a compressor requires that the fridge is turned off, and that none of the screws that fit the compressor are fastened. Attaching a compressor requires that the fridge is turned off, and that the compressor fits the fridge (for each fridge, only the initial compressor and its replacement fit). All operator parameters are constrained to be of the appropriate types.

All instances of *Fridge* are solvable (by in turn stopping a fridge, unfastening the screws, exchanging the compressors, fastening the screws, and starting the fridge again).

Proposition 5 *Given an instance (C, I, G) of the Fridge domain, the corresponding state space is undirected.*

Proof: After removing static predicates, the stop-fridge(x) and start-fridge(x) actions as well as the fasten(x, y, z) and unfasten(x, y, z) actions are invertible in the sense of Definition 14 (by each other, respectively). For remove-compressor(x, y) and attach-compressor(x, y), matters are a bit more complicated. While they obviously undo each other's effects, applicability is slightly less obvious. Assume a compressor c has just been removed from a fridge *f* in a reachable state s. Then, c fits f — otherwise it couldn't have been attached in s — so one can attach c back onto *f*. Now assume a compressor c has just been attached to a fridge *f* in a reachable state s. The question is, are the screws all unscrewed? If not, then they must have been fastened (and not unfastened afterwards) to a matching compressor that is already attached to a fridge. However, both matching compressors can be attached only to the same fridge *f*, and *f* has been free in s. ■

Theorem 18 *Given an instance (C, I, G) of the Fridge domain, the corresponding state space does not contain local minima under h^+.*

Proof: We show that all actions are respected by the relaxation. With the argumentation given above in the proof to Proposition 5, this suffices by a slight extension of Lemma 4, allowing for the inverse relation between remove-compressor(x, y) and attach-compressor(x, y). This does not syntactically follow Definition 14, but has the same consequences semantically.

It is obvious that start-fridge(x) and attach-compressor(x, y) actions are respected by the relaxation: such actions can only start an optimal plan if they add the respective goal, which can not be added by any other action. So those actions are then contained in any optimal relaxed plan P^+, and can be used to start P^+ because they are applicable. If fasten(x, y, z) starts an optimal plan, then this means that y is the new compressor for z, so that action is also part of an optimal relaxed plan (the relaxed plan could also achieve the goal (screwed, x) by fastening x to the broken compressor for z, but this involves more steps). If stop-fridge(x) starts an optimal plan, then this means that either the broken compressor is attached to x, or that one of the screws is still unfastened. In both cases a relaxed plan must also stop x in order to apply the respective actions: the first case is obvious, the second case follows because the screws can only be fastened to a matching compressor, which in turn can only be attached to x. If unfasten(x, y, z) starts an optimal plan, then y is the broken compressor for z, so unfastening x prior to removing y from z can not be avoided in a relaxed plan. Similarly, remove-compressor(x, y) starting an optimal plan means that x is the broken compressor for y, removing which can not be avoided. ■

The maximal bench exit distance in *Fridge* is not bounded. Consider the instances with a single fridge and n screws. From the initial state, an optimal relaxed plan stops the fridge, unfastens all screws, removes the broken compressor, and attaches the new one, so $h^+(I) = n + 3$. The nearest better state is the one where the broken compressor has been removed already: in all states where that has not been done, one must in the relaxed plan always do the removing and attaching action, either start or stop the fridge, and either fasten or unfasten each screw, depending on the current state of the fridge or screw (if a screw is unfastened, it needs to be fastened for achieving the respective goal, and if a screw is fastened, it needs to be unfastened in order to remove the broken compressor; similarly for the fridge). So h^+ is constantly $h^+(s) = n + 3$ for all states s where the broken compressor is attached to the fridge. Getting from the initial state to a state where the broken compressor can be removed requires $n + 1$ steps, which is a lower bound on the maximal bench exit distance. In fact, it is also an upper bound.

Theorem 19 *Given an instance (C, I, G) of the Fridge domain, with n screws per fridge. Then the maximal bench exit distance in the corresponding state space is at most $n + 1$.*

Proof: We distinguish the following cases.

- From a state where at least one old compressor c is attached to a fridge *f* one can get to a state with lower h^+ value by at most stopping *f*, unfastening all screws that are fastened to c ($\leq n$ steps), and removing c: in the resulting state, the remove-compressor(c,*f*) action can be left out of the relaxed plan.
- From a state where no compressor is attached at all, one can get to a better state by attaching a new compressor c to the fridge *f* it fits, first stopping *f* if it is turned on: in the resulting state, one can leave attach-compressor(c,*f*) out of the relaxed plan.
- Say none of the above cases holds, but we are not in a goal state. Then there is a new compressor c that is attached to a fridge *f*, such that at least one screw s fitting c is not yet fastened. One gets to a better state by stopping *f* if it is on, and fastening s: in the resulting state, one can leave fasten(s, c,*f*) out of the relaxed plan.

■

As per Theorem 19, the maximal bench exit distance in *Fridge* grows in the number of screws, but *not* in the number of fridges.

7.3 Blocksworld-no-arm

In *Blocksworld-no-arm*, there are three operators: move a block x from the table to a block y (which exchanges (clear, y) and (on-table, x) with (on, x, y)); move a block x from a block y to the table (which does the above exchange the other way round); finally, move a block x from a block y to a block z, which exchanges (on, x, z) and (clear, y) with (on, x, y) and (clear, z). The initial state of an instance specifies the initial positions of the blocks, the goal state specifies a (consistent, i.e., cycle-free) set of (on, x, y) propositions. In the operators that move a block onto another block, it is required that the two blocks are different (which does not follow from the other preconditions, so must be postulated to avoid absurd actions). See the domain details in Appendix A.3.

All instances of *Blocksworld-no-arm* are solvable (by moving all blocks to the table and thereafter building the goal stacks from bottom to top). Removing static predicates gets rid of actions that try to move a block onto itself.

Corollary 8 *Given an instance (C, I, G) of the Blocksworld-no-arm domain, the corresponding state space is undirected.*

Proof:

After transformation with respect to static predicates, all actions are invertible in the sense of Definition 14: move-table-to-block(x, y) is inverted by move-block-to-table(x, y) and vice versa, and move-block-to-block(x, y, z) is inverted by move-block-to-block(x, z, y). By applying Lemma 1, we are finished.

■

Proving the non-existence of local minima under h^+ is more complicated than in the domains we have seen so far.

Theorem 20 *Given an instance (C, I, G) of the Blocksworld-no-arm domain, the corresponding state space does not contain local minima under h^+.*

Proof: We prove that there is always at least one optimal solution path on which the heuristic value does not increase. Let $s \in S$ be a reachable state with $gd(s) > 0$. We distinguish the following two cases:

1. There is an optimal plan for (A, s, G) starting with an action a that puts some block x into its goal position on block y.

2. No optimal plan for (A, s, G) starts with such an action.

In the first case, we will see that there is an optimal relaxed plan for (A, s, G) containing a. In the second case, we will identify an optimal plan P_s for (A, s, G) starting with an action a that moves a block x onto the table, and an optimal relaxed plan P_s^+ for (A, s, G) containing an action a' that moves x. In both cases, we prove that $h^+(Result(s, \langle a \rangle)) \leq h^+(s)$, which finishes the argumentation.

Assume there is an optimal plan for (A, s, G) starting with a, putting block x into its goal position on block y, $(\text{on}, x, y) \in G$, i.e., $a = \text{move-table-to-block}(x, y)$ or $a = \text{move-block-to-block}(x, z, y)$ for some third block z. Obviously, all relaxed plans for (A, s, G) must also at some point move x onto y. If this is done in a different way than by a, then x must first be moved to a different position, involving one more step. Moreover, moving x to a different position can achieve — apart from the (clear, z) proposition that is then also achieved by a — only an $(\text{on-table}, x)$ or (on, x, u) proposition. Those propositions are of no other use than for moving around x. It follows that all optimal relaxed plans for (A, s, G) use a for achieving (on, x, y). With invertibility, we are finished.

Now assume that there is no optimal plan for (A, s, G) that starts by putting some block into its goal position. We have a closer look at the situation. The blocks are arranged on the table in stacks — a bottom block directly on the table, a sequence of blocks transitively on that block, a clear top block. We say that a stack is a *good* stack, if it is consistent with the goal conditions, i.e., if $(\text{on}, x, y) \in G$ and x or y are in the stack, then x is located on y (implying that both blocks are in the stack). Obviously, optimal plans leave good stacks untouched — blocks in these stacks are either not mentioned in the goal, or already in their final position.

Focusing on the stacks in s that are *not* good, we observe that there is at least one non-good stack in s such that an optimal plan starts by putting the respective top block x to the table. This is because the optimal plan has no other choice but to move a top block of a bad stack (good stacks remain untouched), and because by prerequisite x is not put onto its goal position – moving x to the table can only be better than moving it onto some block where it does not need to be finally. Further, we observe that any relaxed plan for s will also move x – the optimal real plan moves x in order to access a block below it; a relaxed plan must do the same.

Say x is located on y, and $a := \text{move-block-to-table}(x, y)$. By what we observed above, an arbitrary optimal relaxed plan P_s^+ for (A, s, G) contains an action moving x away from y; let a' be the first such action. If $a' = \text{move-block-to-table}(x, y)$, we are finished with invertibility. Assume $a' =$ move-block-to-block(x, y, z). Denote by $s' := Result(s, \langle a \rangle)$ the result of applying a in s. We distinguish two cases:

1. $(\text{on}, x, y) \in G$: Construct an action sequence $P^+_{s'}$ by replacing a' in P^+_s with move-table-to-block(x, y). $P^+_{s'}$ is a relaxed plan for (A, s', G): no action before a' needs (on, x, y) as precondition (by prerequisite, a' is the first action moving x away from y); executing the relaxed version of move-block-to-block(x, y, z) in P^+_s achieves one more proposition, (on, x, z), but x is already in its goal position, so the optimal relaxed plan P^+_s moves x only once, thus (on, x, z) is not needed by any other action.

2. $(\text{on}, x, y) \notin G$: Construct an action sequence $P^+_{s'}$ by replacing a' in P^+_s with move-table-to-block(x, z). Now, $P^+_{s'}$ is a relaxed plan for (A, s', G) unless there is some action that needs (on, x, y) as precondition, which must be a move-block-to-block(x, y, u) (moving x to the table is useless after applying the relaxed version of a'). Such an action comes after a' in P^+_s (remember that a' is the first action moving x away from y), so it can be replaced by move-block-to-block(x, z, u) in $P^+_{s'}$. This replacement does not achieve (clear, y), but that proposition has already been true in s' anyway.

In both cases, we have a relaxed plan $P^+_{s'}$ for (A, s', G) with $|P^+_{s'}| = |P^+_s|$, which concludes the argumentation. ■

The maximal bench exit distance in *Blocksworld-no-arm* is not bounded. Consider an instance with $n + 1$ blocks $b_1, \ldots, b_{n+1}$, initial state $I = \{(\text{clear}, b_1), (\text{on}, b_1, b_2), \ldots, (\text{on}, b_{n-1}, b_n), (\text{on-table}, b_n), (\text{clear}, b_{n+1}), (\text{on-table}, b_{n+1})\}$ and goal state $G = \{(\text{on}, b_1, b_2), \ldots, (\text{on}, b_n, b_{n+1})\}$. Here, the stack $b_1, \ldots, b_n$ needs to be disassembled, and rebuilt on top of b_{n+1}. An optimal relaxed plan for I has length n, by moving $b_1 \ldots b_{n-1}$ to the table, and moving b_n on top of b_{n+1}. Now, $h^+(s') \geq n$ for all reachable states s' where b_n is not on top of b_{n+1}: b_n must be moved to b_{n+1}; to make this possible, b_{n-1} must be moved away from b_n if it is currently on top of it, and otherwise it must be moved to b_n for achieving the goal; the same argument can iteratively applied upwards to b_1. Now consider again the initial state. Reaching, from there, the nearest state s' where b_n is on top of b_{n+1} involves n steps moving $b_1 \ldots b_{n-1}$ away, and moving b_n onto b_{n+1}. Thus the maximal exit distance in the instance is at least n.

7.4 Grid

In *Grid*, a robot must move along places that are arranged in a grid-like reachability relation; the places can be locked and there are keys of different shapes to open them; the goal is to have some keys at their goal places. More explicitly, there are five operators: one can move from place x to place y, which requires (apart from the obvious preconditions) that x and y are connected and that y is open (not locked); one can pick up a key y at a place x, which requires that the arm is empty (one can only hold one key at a time), and has as effects that one holds the key, that the arm is no longer empty, and that the key is no longer at x; one can put a key y down at a place x, which inverts the effects of picking y up at x; one can abbreviate the two previous actions by doing a pickup-and-lose of keys y and z at a place x; to do this, one must hold y, which is directly exchanged for z, i.e., the effects are that one holds z and that y is located at x; finally, one can unlock a place y when one is at a place x that is connected to y, and holds a key z that has the same shape v as the locked place y; the add effect of that is that the place x is open, the delete effect is that the place is no longer locked. The instances specify the initial locations of all keys, locked places, and of the robot, as well as the shapes of the keys and the locked places; the goal specifies places for a subset of the keys. The robot always starts on an *open* place. This does make a significant difference: if the robot is allowed to start on a locked place, there can be local minima under h^+ while otherwise, as we prove below, there are none. Intuitively, it makes more sense to let the robot be located only in open places; this is also the case in the published benchmark examples. The detailed formal definition of the *Grid* domain is in Appendix A.10.

A *Grid* instance can be unsolvable if, for example, some important place is locked with a shape for which there is no matching key.

Proposition 6 *Given a solvable instance (C, I, G) of the Grid domain, the corresponding state space is at most harmless.*

Proof: As the robot can only *open* locks, one possible solution plan for any reachable state s is this: re-establish the initial situation as far as possible, i.e., put all keys that have been moved back to their initial places, and move back to the initial place of the robot (this is possible because to all these places there must be a path on which all places are open — otherwise the keys, respectively the robot, could not have been moved). Afterwards, execute an arbitrary solution plan for the initial state, omitting actions that try to unlock a place that is already open. ■

As unlocking a place can not be undone, a *Grid* instance is not undirected if there is at least one locked place that can be opened. Let us focus on local minima.

Theorem 21 *Given a solvable instance (C, I, G) of the Grid domain, the corresponding state space does not contain local minima under h^+.*

Proof: Let $s \in S$ be a reachable state with $gd(s) > 0$. We identify a flat path to an exit, i.e., a path on which h^+ does not increase.

Let p be the current position of the robot, $(\text{at-robot}, p) \in s$. Let P_s^+ be an optimal relaxed plan for (A, s, G) (with solvability and Proposition 6, there is at least one such P_s^+). We focus in the following on one action a in P_s^+ that makes use of a key k at place p'. If there is an unlock action in P_s^+, then we select the first such action $a = \text{unlock}(p', y, k, v)$. If there is no single unlock, then we select an arbitrary putdown action $a = \text{putdown}(p', k)$; without loss of generality, there is at least one such action because we can assume that the last action in P_s^+ is a putdown (remember that the only goals are to have some keys at their goal places). We distinguish between three cases:

1. The robot is holding k in s, $(\text{holding}, k) \in s$.

2. The arm is empty in s, $(\text{arm-empty}) \in s$.

3. The robot is holding $k' \neq k$ in s, $(\text{holding}, k') \in s, k' \neq k$.

In the first case, we identify a flat path to an exit state s'. In the second and third cases, we identify a flat path to a state s' where the robot is holding k, i.e., where the first case holds.

Assume the robot is holding key k. The action a we have selected is either the first unlock in P^+_s, or a putdown, using k. In both cases, a could be executed given the robot was at place p', and in both cases, P^+_s contains what we will call an *open path from p to p'* in the following: P^+_s contains, preceding a, a subsequence $\langle \text{move}(p = p_1, p_2), \ldots, \text{move}(p_{n-1}, p_n = p') \rangle$ of moving actions connecting p to p', such that $(\text{open}, p_i) \in s$ for $1 \leq i \leq n$; the moving actions are there because otherwise a could not be applied at the point in P^+_s where it appears; p_1 is open because the robot can only be at open places in all reachable states by definition (note that at this point we need the restriction that the robot is not allowed to start at locked places), and $p_2, \ldots, p_n$ are open because we selected a to be the *first* unlock in P^+_s. Let $s_i := \mathit{Result}(s, \langle \text{move}(p_1, p_2), \ldots, \text{move}(p_{i-1}, p_i) \rangle)$. Our flat path is the above open path, and our exit state s' is $s' := s_n$. It suffices to show that $h^+(s_i) \leq h^+(s)$ for $1 \leq i \leq n$, and that there is a state $s'', (s', s'') \in E, h^+(s'') < h^+(s)$. Consider s_2, resulting from s by executing $\text{move}(p, p_2)$. As that action is contained in P^+_s, Lemma 3 allows us to construct a relaxed plan $P^+_{s_2}$ for (A, s_2, G) by removing $\text{move}(p, p_2)$, and adding $\langle \text{move}(p_2, p) \rangle$ as a prefix. Therefore, $h^+(s_2) \leq h^+(s)$. Now, the next action $\text{move}(p_2, p_3)$ on our open path is still contained in $P^+_{s_2}$, so the same argument applies to s_3, and so forth. Let $P^+_{s'}$ be the thus constructed relaxed plan for $s' = s_n$. Our action a is applicable in s', and it is contained in $P^+_{s'}$. Let $s'' := \mathit{Result}(s', a)$. We construct a relaxed plan for s'' as follows. First, we remove a from $P^+_{s'}$. If $a = \text{unlock}(p', y, k, v)$, that does the job: the single delete is (locked, y), which is only needed for actions that unlock y. Now, the optimal relaxed plan P^+_s for (A, s, G) does of course not open y twice, and we did not add new unlock actions when constructing $P^+_{s'}$. If $a = \text{putdown}(p', k)$, the delete is $(\text{holding}, k)$. No unlock actions are contained in $P^+_{s'}$, but $(\text{holding}, k)$ might be needed by $\text{pickup-and-lose}(x, k, z)$ actions contained in $P^+_{s'}$. These can, however, be replaced by $\text{pickup}(x, z)$ actions when the relaxed plan starts out in s''. In both cases for the selection of a, the resulting relaxed plan for (A, s'', G) is one step shorter than that for (A, s', G), which concludes the argument.

We still need to deal with the cases where the arm is empty in s, or where the robot is holding some different key $k' \neq k$. In both cases, we can move along open paths in P^+_s to end up in a state s' where the robot holds k. Consider the case where $(\text{arm-empty}) \in s$. At the point in P^+_s where a is applied, $(\text{holding}, k)$ has been achieved. This was done via a $\text{pickup}(p', k)$ or a $\text{pickup-and-lose}(p', k'', k)$ action preceding a in P^+_s. In the first case, we stop, in the second case, we apply the same argument to k'', i.e., before $\text{pickup-and-lose}(p', k'', k)$ is applied in P^+_s there must be some action achieving $(\text{holding}, k'')$. At some point, a key must be picked up, as initially in s no key is held. So we end up with a list $k_1, \ldots, k_n = k$ of keys such that $\langle$pickup(p_1, k_1), pickup-and-lose(p_2, k_1, k_2), $\ldots$, pickup-and-lose$(p_n, k_{n-1}, k_n = k)\rangle$ is a subsequence of P_s^+ for certain places $p_1, \ldots, p_n$. P_s^+ contains open paths from p to all $p_i, 1 \leq i \leq n$ (remember that either our selected action a using k is the very first unlock in P_s^+, or there are no unlocks needed at all). We move along the path from p to p_1, execute pickup(p_1, k_1), and move back from p_1 to p. At all points on this path we can use, like above, Lemma 3 to construct a relaxed plan for the new state that has the same number of steps like P_s^+: putdown(p_1, k_1) inverts pickup(p_1, k_1), and the actions moving back from p_1 to p are all contained in the relaxed plan constructed when moving to p_1. Moving back to p constructs a relaxed plan that contains exactly the same move actions as P_s^+. We can thus in the same fashion move from p to p_i, $2 \leq i \leq n-1$, execute pickup-and-lose(p_i, k_{i-1}, k_i) (inverted by pickup-and-lose(p_i, k_i, k_{i-1})), and move back to p. We finally move from p to p_n and execute pickup-and-lose$(p_n, k_{n-1}, k_n = k)$ which brings us to the desired state s' where the robot is holding the key k.

Finally, consider the case where $(\text{holding}, k') \in s, k' \neq k$. Like in the above case, at the point in P_s^+ where a is applied, $(\text{holding}, k)$ has been achieved. So, like above, there is either a pickup(p', k) or a pickup-and-lose(p', k'', k) action preceding a in P_s^+, the same holding for k'' unless $k'' = k'$. Iterating the argument gives us a key list $k_1, \ldots, k_n = k$ such that either $\langle$pickup(p_1, k_1), pickup-and-lose(p_2, k_1, k_2), $\ldots$, pickup-and-lose$(p_n, k_{n-1}, k_n = k)\rangle$ or $\langle$pickup-and-lose$(p_1, k' = k_1, k_2)$, $\ldots$, pickup-and-lose$(p_n, k_{n-1}, k_n = k)\rangle$ is a subsequence of P_s^+. The first case can be reduced to the second case by replacing pickup(p_1, k_1) with pickup-and-lose(p_1, k', k_1) in P_s^+, and adding k' as the front member of the key list. Just like we did above, we can then move along the open paths from p to p_i, execute the adequate pickup-and-lose(p_i, k_{i-1}, k_i) action, and move back to p, finally ending up in the state s' where pickup-and-lose$(p_n, k_{n-1}, k_n = k)$ has been executed, and the robot is thus holding the correct key, which case we have treated earlier. ∎

I remark two things about the above argumentation. First, Theorem 21 also holds when there is no pickup-and-lose operator (in fact, the proof is a lot easier in this case). Second, the identified flat paths can be arbitrarily long and the maximal bench exit distance in *Grid* is thus *not* proved to be bounded by a constant. Consider the simple example where the robot and one key k have initially n moving steps distance, and k shall be put to the robot's starting place. The h^+ value for the initial state is $n+2$: walk over to the key, pick it up, and drop it at the starting place (remember that the at-robot propositions are not deleted). The nearest state with better h^+ value is the state where k has been picked up, which is $n+1$ steps away.

7.5 Hanoi

The *Hanoi* domain is a STRIPS encoding of the classical *Towers of Hanoi* problem. There are n discs $d_1, \ldots, d_n$, and three pegs p_1, p_2, and p_3. There is a single operator that moves a constant x from a constant y onto a constant z (note that the operator parameters can be grounded with discs as well as pegs). The preconditions of the move are that x is on y, x is clear, z is clear, and x *is smaller than* z; the effects are that x is on z and y is clear, while x is no longer on y and z is no longer clear. The semantics of *Towers of Hanoi* are encoded via the smaller relation. This relation holds in the obvious way between the discs, and all discs are smaller than the pegs (the pegs are not smaller than anything so they can not be moved). The instances differ in terms of the number n of discs that must be transferred from p_1 to p_3. See the domain details in Appendix A.12.

All instances of *Hanoi* are solvable.

Corollary 9 *Given an instance* (C, I, G) *of the Hanoi domain, the corresponding state space is undirected.*

Proof: After transformation with respect to static predicates, all actions are invertible in the sense of Definition 14: move(x, y, z) is inverted by move(x, z, y). Finished with Lemma 1. ■

I prove the non-existence of local minima by first giving an exact measure of what $h^+(s)$ is for any reachable state s: as it turns out, h^+ is always equal to the number of discs that are not yet in their goal position. This suffices because an optimal plan does not move discs that are already where they should be.

Lemma 5 *Given an instance* (C, I, G) *of the Hanoi domain, and a reachable state* $s \in S$. *Denote* p_3 *by* d_{n+1}, *and let* $sg(s)$ *be defined as follows:*

$$sg(s) := min\{m \mid 1 \leq m \leq n+1, (on, d_m, d_{m+1}), \ldots, (on, d_n, d_{n+1}) \in s\}$$

Then $h^+(s) = sg(s) - 1$ *holds.*

Proof: $sg(s)$ is the number of the smallest disc that is already in its goal position in s, with all discs below it also being in their goal position ($sg(s) = n+1$ if no disc, or only p_3, is already in its goal position). If $sg(s) = 1$, the goal is achieved and there is nothing to show. Else, we show by induction over i, $i \leq n$, that i relaxed steps can get us to a state s' where (on, d_1, d_2), ..., $(\text{on}, d_i, d_{i+1}) \in s'$. With that, the lemma follows: after $sg(s) - 1$ steps, (on, d_1, d_2), ..., $(\text{on}, d_{sg(s)-1}, d_{sg(s)}) \in s'$, which gives us $G \subseteq s'$ as $(\text{on}, d_{sg(s)}, d_{sg(s)+1})$, ..., $(\text{on}, d_n, d_{n+1} = p_3)$ are true in s already (also, at least $sg(s) - 1$ relaxed steps are necessary as that many $(\text{on}, d_i, d_{i+1})$ propositions from G are missing in s, and each action achieves at most one such proposition).

We prove the stronger result that, after i relaxed steps, we can get to a state s' where (on, d_1, d_2), ..., $(\text{on}, d_i, d_{i+1}) \in s'$, (clear, d_1), ..., $(\text{clear}, d_{i+1}) \in s'$, and $(\text{clear}, d) \in s'$ for all discs or pegs d such that $(\text{on}, d_j, d) \in s$ for some $j \leq i$.

Base case $i = 1$: $(\text{clear}, d_1) \in s$ simply because s is reachable and no disc can be placed onto the smallest disc d_1. Either d_1 is on d_2 in s, or d_1 is on some other disc or peg d. In the first case, we execute the relaxed step move$(d_1, d_2, z)^+$ for some z that is clear in s; in the second case, d_2 is clear (no disc other than d_1 can be on d_2), so we can execute move$(d_1, d, d_2)^+$. In both cases, (clear, d_1), (clear, d_2), and (on, d_1, d_2) are contained in the resulting state s', and also (clear, d) holds for the disc or peg d on which d_1 is in s.

Inductive case $i \longrightarrow i+1, i+1 \leq n$: let s' be the state reached after i relaxed steps, with the required properties. Either d_{i+1} is on d_{i+2} in s (remember that we denote $d_{n+1} := p_3$), or d_{i+1} is on some other disc or peg d. In both cases there is an action applicable to s' which yields, in its relaxed version, a state that satisfies the requirements. Consider the second case. We can apply $\text{move}(d_{i+1}, d, d_{i+2})^+$: $(\text{clear}, d_{i+1}) \in s'$ by induction hypothesis; if d_{i+2} was not clear, this would be due to a smaller disc d_j, $j < i+1$, being on d_{i+2} in s, so again $(\text{clear}, d_{i+2}) \in s'$ per induction hypothesis. The resulting state has d clear, and d_{i+1} on d_{i+2}, so it fulfills the requirements. Consider now the first case, where d_{i+1} is on d_{i+2} in s. Applying $\text{move}(d_{i+1}, d_{i+2}, z)^+$ for some z would obviously yield a state satisfying the requirements (the only proposition that must be achieved is (clear, d_{i+2})). Concerning the applicability of $\text{move}(d_{i+1}, d_{i+2}, z)^+$ in s', $(\text{clear}, d_{i+1}) \in s'$ per induction hypothesis. It remains to find a disc or peg z with $(\text{smaller}, d_{i+1}, z) \in s'$ and $(\text{clear}, z) \in s'$. Let p be a peg such that d_{i+1} is not transitively on p in s, and z be the topmost disc transitively on p in s such that $(\text{smaller}, d_{i+1}, z) \in s'$, or $z := p$ if there is no such disc. z is clear: if not, $(\text{on}, d_j, z) \in s$ would hold for some disc $j \leq i$ smaller than d_{i+1}, so $(\text{clear}, z) \in s'$ per induction hypothesis. ■

Theorem 22 *Given an instance (C, I, G) of the Hanoi domain, the corresponding state space does not contain local minima under h^+.*

Proof: Follows from Lemma 5 and the fact that an optimal plan does not move around the discs that are already in their goal position. In a reachable non-goal state s, with $sg(s) > 1$ being the number of the smallest disc that is already in its goal position in s (all discs below it also being in their goal position), the optimal plan from s moves around smaller discs until the next smaller disc is in its goal position. Lemma 5 proves that, before this is achieved, h^+ remains $sg(s) - 1$ constantly, and that, when this is achieved, h^+ goes down to $sg(s) - 2$. Iterating the argument, h^+ does not increase on the optimal solution path from s. ■

I remark that the length of the flat paths is not bounded by a constant. Quite differently, their length is exponential in the number of discs: from the initial state of an instance with n discs, the nearest state where the largest disc d_n is in its goal position is 2^{n-1} steps away.

8 Bounded Maximal Local Minimum Exit Distance and Bounded Maximal Bench Exit Distance

In the next group of domains, there *are* local minima under h^+, but with upper bounds on the maximal local minimum exit distance (so by Proposition 1 there be no unrecognized dead ends). The maximal bench exit distances are also bounded. The domains are *Zenotravel*, *Satellite*, and *Schedule*. We focus on them in that order. For each domain I give a brief description of the semantics, consider the dead end class of the domain's instances, and give an example of a local minimum. I then give a proof argument showing that, for each reachable state s, there is an upper bound c on the distance to a state s' with $h^+(s') < h^+(s)$. This entails that $c-1$ is an upper bound to both the maximal local minimum exit distance and the maximal bench exit distance.

8.1 Zenotravel

Zenotravel is a transportation domain variant where the vehicles (called aircrafts) use fuel units that can be replenished using a refuelling operator. There is a number of cities, a number of aircrafts, a number of persons, and a number of different possible fuel levels. The fuel levels encode natural numbers by a (next, x, y) predicate – (next,f,f') is true iff f' is the next higher fuel level than f. The zero fuel level is denoted with f_0. The task is to transport the persons from their initial locations to their goal locations. There are the following five operators. One can board/debark a person p onto/from an aircraft a at a city c; this has just the obvious preconditions and effects. One can fly an aircraft a from a city c to a different city c', decreasing a's fuel level from f to f'; as a precondition, a must be at c, f must be a's current fuel level, and f' must be the next lower fuel level. One can also zoom the aircraft; this is exactly the same as flying it, except that zooming uses more fuel – the aircraft's fuel level is decreased by two units.[4] Finally one can refuel an aircraft a at a city c from fuel level f to fuel level f'. The conditions are that a is at c, that f is a's current fuel level, and that f' is the next higher fuel level. Thus aircrafts can be refuelled at any city, and in steps of one unit. See the domain details in Appendix A.25.

A *Zenotravel* instance is solvable as soon as there is at least one aircraft and one non-zero fuel level: the aircraft can be refueled after each flying action, and can be used to transport all persons. Removing static predicates gets rid of type information, and of actions that try to move between identical locations. We assume in the following that static predicates have been removed. The sufficient criteria for the non-existence of dead ends are not directly applicable, so I insert a short proof.

Proposition 7 *Given a solvable instance (C, I, G) of the Zenotravel domain, the corresponding state space is at most harmless.*

Proof: Boarding and departing actions invert each other in the sense of Definition 14. A flying action from c to c' can be undone by refuelling once at c', flying back from c' to c, and refuelling another time. Similarly, a zooming action can be undone (requiring two refuelling actions in each city). So the effects of all actions can (indirectly) be undone, and no dead ends can arise. ∎

If there are at least two cities, the instance is not undirected as the effects of the moving actions can not directly be undone.

[4] As the reader will have noticed, this does not make much sense; according to personal communication with Derek Long it is a domain bug that has been overlooked when preparing IPC-3.

There are local minima in *Zenotravel* under h^+. An example instance is this. There are two cities c_1 and c_2, one aircraft a_1, two persons p_1 and p_2, as well as two fuel levels f_0 and f_1. The goal for p_1 is to be at c_1, the goal for p_2 is to be at c_2. The following reachable state s forms a local minimum (note that we ignore static predicates): $s = \{(\text{at}, a_1, c_2), (\text{in}, p_1, a_1), (\text{at}, p_2, c_1), (\text{fuel-level}, a_1, f_1)\}$. A shortest relaxed plan for s is to fly from c_2 to c_1, debark p_1 at c_1, board p_2 at c_1, and debark p_2 at c_2. Thus $h^+(s) = 4$. The applicable actions to s are to debark p_1 at c_2, and to fly from c_2 to c_1. When applying the former action, in the resulting state s' the relaxed plan is the same as before except that p_1 must be boarded at c_2. So $h^+(s') = 5$. When applying the latter action, in the resulting state s'' a shortest possible relaxed plan is to debark p_1 at c_1, board p_2 at c_1, refuel, fly from c_1 to c_2 and debark p_2 at c_2. So $h^+(s'') = 5$, showing that s forms a local minimum.

From the above local minimum state s, one can reach a state with lower h^+ value within three steps. This holds true for *all* states – in particular for all local minima states – that can arise in *Zenotravel*.

Theorem 23 *Given a solvable instance (C, I, G) of the Zenotravel domain, and a reachable state s, $h^+(s) > 0$. The distance from s to a state s' with $h^+(s') < h^+(s)$ is at most* 3.

Proof: Let P_s^+ be an optimal relaxed plan for (A, s, G) (with solvability and Proposition 7, there is at least one such P_s^+). We distinguish between three cases:

1. P_s^+ contains a boarding or departing action that is applicable in s.
2. P_s^+ contains a refuelling action that is applicable in s.
3. None of the above cases holds, i.e., the only actions in P_s^+ that are applicable in s are flying or zooming actions.

In the first case, the desired state s' can be reached with a single step. Let a be a boarding or departing action that is applicable in s. A relaxed plan for $s' = \mathit{Result}(s, \langle a \rangle)$ can be constructed by removing a from P_s^+: if a boards person p at city c then, as P_s^+ is optimal, p must be boarded in order to reach the goal, and needs not be at c any longer; if a debarks person p at city c, then c is p's goal city and p needs not be inside the respective aircraft any longer.

In the second case, too, the desired state s' can be reached with a single step. Let a be a refuelling action that is applicable in s. Again, a relaxed plan for $s' = \mathit{Result}(s, \langle a \rangle)$ can be constructed by removing a from P_s^+: the fuel level of the respective aircraft ac must be zero in s (otherwise no refuelling would be needed in the relaxation); one refuelling action suffices in the relaxation so all actions in P_s^+ flying ac make use of fuel level one, which is given in $\mathit{Result}(s, \langle a \rangle)$ (there can't be any actions in P_s^+ zooming ac).

In the third case, the desired state s' can be reached with three steps. Say P_s^+ starts with an action a flying aircraft ac from city c to city c' (if P_s^+ starts with a zooming action then one can easily replace that with the corresponding flying action, and, if necessary, increment the fuel levels used by other actions in

P_s^+ moving ac). We fly ac from c to c', then we refuel ac once. For the resulting state s'' we can construct a relaxed plan $P_{s''}^+$ by replacing a with fly(ac, c', c) in P_s^+: after the relaxed execution of fly(ac, c', c) in s'' the same at-relations and fuel-levels are given as after execution of fly(ac, c, c') in s. Now, as P_s^+ is optimal it either boards some person onto ac at city c', or debarks some person from ac there. Similar to the first case above, one can apply the respective action a' in s'', and skip it from $P_{s''}^+$ to obtain a relaxed plan for $s' = \mathit{Result}(s'', \langle a' \rangle)$. This concludes the argument. ■

8.2 Satellite

In *Satellite*, satellites need to take images in different directions, in certain modes, using appropriate instruments. There is a number of satellites, a number of directions, a number of instruments, and a number of modes. There are the following five operators. One can turn a satellite s from a direction d to a different direction d'; the obvious preconditions and effects are that s must be pointing into d to execute the action, and points into d' upon execution. One can switch on an instrument i on board a satellite s, if s has power available; in effect, i has power, i is no longer calibrated, and s has no more power available. One can switch off an instrument i on board a satellite s, if i has power; in effect s has power available but i not anymore. One can calibrate an instrument i on board a satellite s in direction d, if d is a calibration target for i, s points at d, and i has power; the only effect is that i is calibrated. Finally, one can take an image with an instrument i on board a satellite s in direction i and mode m; the precondition is that i is calibrated, that i supports the mode m, that i has power, and that s points to d. The only effect is that one has the image data for d in mode m.

As indicated above, the goal in *Satellite* is to have the image data for a number of direction/mode pairs; also, satellites can have a goal requirement to point in a certain direction. The initial states are such that each satellite points in exactly one direction, each instrument is on board of exactly one satellite, each satellite (but no instrument) has power available, and no instrument is calibrated. The first two are purely physical restrictions, the third restriction is just one way of saying that each satellite has the power to run one instrument at a time. The last restriction is just one way of saying that each calibrated instrument has at least one calibration target. The restriction seems natural, and holds in the IPC-3 instances. It implies the non-existence of dead ends, as we will see below. Note that, without the restriction, there can be unrecognized dead ends: just imagine an instrument that is calibrated, that has no calibration target, that is switched off, and that is needed for taking an image; there is no plan as switching the instrument on deletes its calibration, but there is a relaxed plan as this ignores the aforementioned delete effect. In what follows we use the domain notations introduced in Appendix A.22, which should be self-explanatory.

A *Satellite* instance is solvable iff for each goal (have-image, d, m) there is an instrument i such that i has a calibration target, and supports m. The implication from left to right is obvious. From right to left, for each such goal one can switch i on (possibly switching some other instrument off first), turn the satellite s on which i is located towards i's calibration target, calibrate i, turn s towards d, and take the image (note that there is no problem turning a satellite into an arbitrary direction). Removing static predicates gets rid of type information, and of actions that try to turn satellites between identical directions. We assume in the following that static predicates have been removed. The sufficient criteria for the non-existence of dead ends are not directly applicable, so I insert a short proof.

Proposition 8 *Given a solvable instance* (C, I, G) *of the Satellite domain, the corresponding state space is at most harmless.*

Proof: Actions turning a satellite s from d to d' are inverted, in the sense of Definition 14, by turning the satellite from d' to d. Switching an instrument on can be undone by switching it off, beforehand calibrating the instrument if it was calibrated before (maybe turning the respective satellite into the calibration direction first, and back afterwards; note that the instrument must have at least one calibration target or else it could not have been calibrated in the first place). Switching an instrument off can be undone by switching it on, and calibrating it if it was calibrated before (maybe inserting actions turning the satellite). The actions calibrating instruments and taking images have no delete effects so their effects need never be undone in order to avoid dead ends. ■

If there is at least one instrument i that has a calibration target, the instance is not undirected as, e.g., the effects of switching i on can not directly be undone when i is calibrated.

There are local minima in *Satellite* under h^+. An example instance is this. There are one satellite sa, two directions d_1 and d_2, two instruments i_1 and i_2, as well as two modes m_1 and m_2. Say i_1 supports m_1, i_2 supports m_2, and the goal is to have an image of direction d_1 in each mode. The calibration target for both instruments is d_2 (and of course both instruments are on board sa). Consider the reachable state s where sa points into d_1, both instruments are calibrated, and both instruments are switched off. A shortest relaxed plan P_s^+ for s is to switch i_1 and i_2 on, then take the two required images of d_1 (which works as the deletes on the calibration of the instruments are ignored). Thus $h^+(s) = 4$. The applicable actions are to switch i_1 on, to switch i_2 on, or to turn s_1 to d_2. When applying the latter action, in the resulting state s' a shortest possible relaxed plan is to turn s_1 back to d_1, switch i_1 and i_2 on, and take the images, so $h^+(s') = 5$. When applying one of the former actions, the h^+ value increases to 6 which shows that s forms a local minimum. Say we switch i_1 on in s. To obtain a relaxed plan for the resulting state s'', one must replace, in P_s^+, the action switching i_1 on with the three actions switching i_1 off (to make power available for i_2), turning to d_2 (i.e., to the calibration target), and re-calibrating i_1. Thus a shortest relaxed plan for s'' is to switch i_1 off, turn sa to d_2, calibrate i_1, switch i_2 on, and take the two images. If we switch i_2 on in s we get a symmterical situation, which concludes the argument.

From the above local minimum state s, one must apply five steps in order to reach a state with lower h^+ value (switch i_1 on, turn to d_2, calibrate i_1, turn to

d_1, take the image in mode m_1). So 5 is a lower bound on the maximal distance to a better state in *Satellite*. It is also an upper bound.

Theorem 24 *Given a solvable instance (C, I, G) of the Satellite domain, and a reachable state s, $h^+(s) > 0$. The distance from s to a state s' with $h^+(s') < h^+(s)$ is at most* 5.

Proof: Let P_s^+ be an optimal relaxed plan for (A, s, G) (with solvability and Proposition 7, there is at least one such P_s^+). We distinguish between four cases:

1. P_s^+ contains an action a taking an image, or calibrating an instrument, such that a is applicable in s.
2. P_s^+ contains an action a switching an instrument on, such that a is applicable in s.
3. P_s^+ contains an action a switching an instrument off, such that a is applicable in s.
4. Cases 1, 2, and 3 do not hold (implying that P_s^+ contains an action a applicable in s that turns a satellite).

In case 1, the desired state s' can be reached with a single step. A relaxed plan for $s' = Result(s, \langle a \rangle)$ can be constructed simply by removing a from P_s^+: neither taking an image nor calibrating an instrument have any delete effects. Finished along the lines of Proposition 2.

In case 2, the desired state s' can be reached within at most five steps. To see this, we distinguish a further two cases. Say action a switches instrument i located on satellite sa on.

- First case, i is not calibrated in s. Then P_s^+ contains an action a' calibrating i (otherwise there would be no point in switching i on); a' uses a calibration target d^c, and if d^c is different from sa's current orientation d then P_s^+ also contains an action a'' turning sa from d to d^c. We obtain a state s'' by executing a (and a'', if $d \neq d^c$); a relaxed plan $P_{s''}^+$ for s'' can be constructed by removing a from P_s^+ and inserting switch-off(i, sa) at the start (as well as replacing turn-to(sa, d, d^c) with turn-to(sa, d^c, d), if $d \neq d^c$). In s'', one can apply a', i.e., calibrate i, and thus (similar to case 1 above) get to a state s' for which a relaxed plan is given by removing a' from $P_{s''}^+$. The distance from s to s' is at most 2.
- Second case, i is calibrated in s. Then P_s^+ contains an action a' taking an image with i, in direction d^i. Say the current orientation of sa is d, and d^c is a calibration target of i (there is at least one such d^c or else i could not be calibrated in s). We execute the following action sequence to reach the state s'.
    - Apply a, i.e., switch-on(i, sa), yielding the state s_1. A relaxed plan $P_{s_1}^+$ for s_1 is given by removing a from P_s^+, and inserting switch-off(i, sa), turn-to(sa, d, d^c) (if $d \neq d^c$), as well as calibrate(sa, i, d^c). We get $|P_{s_1}^+| \leq |P_s^+| + 2$.

- If $d = d^c$ do nothing, i.e., set $s_2 = s_1$ and $P^+_{s_2} = P^+_{s_1}$. If $d \neq d^c$, obtain s_2 by executing turn-to(sa, d, d^c). A relaxed plan $P^+_{s_2}$ for s_2 is then given by replacing turn-to(sa, d, d^c) with turn-to(sa, d^c, d) in $P^+_{s_1}$. We get $|P^+_{s_2}| \leq |P^+_s| + 2$.
- Calibrate i, yielding the state s_3. A relaxed plan $P^+_{s_3}$ for s_3 is given by removing calibrate(sa, i, d^c) from $P^+_{s_2}$. We get $|P^+_{s_3}| \leq |P^+_s| + 1$.
- If $d^c = d^i$ do nothing, i.e., set $s_4 = s_3$ and $P^+_{s_4} = P^+_{s_3}$. If $d^c \neq d^i$, obtain s_4 by executing turn-to(sa, d^c, d^i). A relaxed plan $P^+_{s_4}$ for s_4 can be constructed as follows. First, remove turn-to(sa, d^c, d) from $P^+_{s_3}$. If $d^i \neq d$, then P^+_s (and thus $P^+_{s_3}$) contains the action turn-to(sa, d, d^i); replace that with turn-to(sa, d^i, d). We get $|P^+_{s_4}| \leq |P^+_s|$.
- In s_4 we can apply a', i.e., take a required image with i in d^i. This (similar to case 1 above) leads to a state s' where a relaxed plan $P^+_{s'}$ is given by removing a' from $P^+_{s_4}$, yielding $|P^+_{s'}| \leq |P^+_s| - 1$.

The distance from s to s' is at most 5.

Say we are in case 3 above, P^+_s contains an applicable action a that switches an instrument i off. Say i is on board a satellite sa whose current orientation is d. We distinguish two cases: either P^+_s contains an action a' calibrating i, or taking an image with i, in some direction d', or P^+_s contains no such actions. In the latter case, a relaxed plan for $s' = \mathit{Result}(s, \langle a \rangle)$ is given by removing a from P^+_s, and we are finished. In the former case, we get to the desired state s' in at most two steps. If $d \neq d'$, apply turn-to(sa, d, d'), updating P^+_s by replacing turn-to(sa, d, d') with turn-to(sa, d', d) (if $d \neq d'$ then P^+_s must contain turn-to(sa, d, d')). We can then apply a', i.e., calibrate i or take an image, yielding a state s' where we can remove a' from P^+_s to form a relaxed plan.

Say we are in none of the above 3 cases. Then the starting action a of P^+_s turns a satellite sa from d to d'. We show that P^+_s can not contain any actions taking images with sa in d, or calibrating instruments on sa in d. With that, it follows that applying a yields a state s' where a relaxed plan is given by removing a from P^+_s, and replacing all turn-to(sa, d, d'') actions in P^+_s with turn-to(sa, d', d'') (a's only delete effect is (pointing, sa, d) which is only necessary for taking images, or calibrating). Let us focus on actions a' in P^+_s that take images, or calibrate instruments, in d. From our assumption that we are neither in case 2 nor in case 3 it follows that all instruments required by such actions a' are already switched on in s (otherwise P^+_s would need to include actions switching them on, or making power available by switching other instruments off; some of these actions would be directly applicable). Thus if there was an action a' in P^+_s taking an image, or calibrating an instrument, with sa in direction d, then a' would be applicable in s, in contradiction to our assumption that we are not in case 1. This concludes our argumentation. ■

8.3 Schedule

In *Schedule*, a collection of parts must be processed with a number of working steps changing their shape, surface, or color; one can also drill holes of varying widths in varying orientations. The full domain description is rather lengthy, with nine large operators that make use of a simple form of conditional effects (for example, painting an object results in the object being painted in the new color, and not any longer being painted in any other color it was painted in before). Full details are in Appendix A.21. The details that are important for the proofs below are the following. Initially, any part can have an arbitrary shape, one of three surface conditions, an arbitrary color, and arbitrarily many holes in arbitrary widths and orientations. For the goal, a part can be required to be of cylindrical shape, and to have arbitrary other properties. A part p is made cylindrical, with a rough surface, by applying the do-lathe($p \ldots$) action.[5] Polished (smooth) surfaces are achieved via the do-polish($p \ldots$) (do-grind($p \ldots$)) action. A hole of width w and orientation o is made by do-drill-press($p, w, o \ldots$). Finally, painting p in a color c is done by do-immersion-paint($p, c \ldots$). All operators require that the respective machine is not busy, and that the part is not currently processed elsewhere. These propositions are achieved (for all machines and parts) by do-time-step(). There are other operators, but as we will see the above operators suffice for solving the tasks. The do-polish($p \ldots$) and do-drill-press($p, w, o \ldots$) actions require that the part is cold; do-drill-press($p, w, o \ldots$) also requires that there is not already a hole in p of width w and orientation o. Apart from some static propositions like type information or the existence of the appropriate drill, none of those six operators has other preconditions. All parts p are cold initially and the only action that makes them hot is do-roll($p \ldots$) (which is not needed to achieve the goal). All operators that can make a part polished or drill a hole (apart from the two operators above, there is one more for each of these two jobs) require that the part is cold. There is no way of making a part cold. This can lead into dead ends.

I prove that all instances are solvable, and that, if there is no real plan from a state, then neither is there a relaxed plan.

Theorem 25 *Given an instance (C, I, G) of the Schedule domain, the corresponding planning task is solvable, and the corresponding state space is at most recognized under h^+.*

Proof:

We must show solvability, and that $h^+(s) = \infty$ for all $s \in S$ with $gd(s) = \infty$. For the initial state I of the task, a solution plan can be constructed as follows. Look at each part p in turn. If $(\text{shape}, p, \text{cylindrical}) \in G$ or $(\text{surface-condition}, p, \text{rough}) \in G$, and the respective proposition is not true in I, then apply do-lathe($p \ldots$), and afterwards do-time-step() (applying do-lathe($p \ldots$) for (surface-condition, p, rough) deletes all other shapes than

[5] In our formal definition, see Appendix A.21, most operators have parameters, like the machine, that must always be grounded with the same constant. These parameters serve to replace the domain constants that are used in the original formulation. We omit these parameters here to improve readability.

(shape, p, cylindrical); this does not matter as being cylindrical is the only shape that is allowed to be required for the goal). If (surface-condition, p, polished) $\in G$ or (surface-condition, p, smooth) $\in G$, and the respective proposition is not currently true, then apply do-polish($p\ldots$) or do-grind($p\ldots$), respectively, and afterwards do-time-step() (note that p has temperature cold initially, so do-polish($p\ldots$) can be applied). Next step, all still required holes (has-hole, p, w, o) $\in G$ can be achieved by iteratively applying do-drill-press($p, w, o\ldots$) and do-time-step() (p is still cold). Finally, if (painted, p, c) $\in G$ for some c, and (painted, p, c) is not currently true, we can apply do-immersion-paint($p, c\ldots$), and are finished. After another application of do-time-step(), the same process can be iterated for the next part until all parts are processed.

The same algorithm to construct a solution plan can be applied for every reachable state $s \in S$ unless there is some part p with (surface-condition, p, polished) $\in G$ or (has-hole, p, w, o) $\in G$ for some w and o, such that (temperature, p, cold) $\notin s$: all other preconditions of the required actions are obviously fulfilled at the point where they are needed — except the temperature, required non-static propositions are only those concerning busy machines, scheduled objects, or p not having a certain hole. The first two are achieved by doing time steps, the third one is only required by do-drill-press($p, w, o\ldots$). If that precondition is not true, this means that there already is an appropriate hole, so the action does not need to be applied at all. It can be that a part p has (temperature, p, cold) $\notin s$ when, on the path from the initial state to s, do-roll($p\ldots$) has been applied at some point. As there is no way of re-achieving (temperature, p, cold), s is then a dead end, $gd(s) = \infty$, if p needs to be polished or get holes made to achieve the goal: all actions that can achieve (surface-condition, p, polished) or (has-hole, p, w, o) for some w and o need (temperature, p, cold) as a precondition, which is not contained in s, and is not added by any action. But then, (A^+, s, G) is also unsolvable, $h^+(s) = \infty$.

■

Note that the above proof also proves that there can be dead ends. Also, the *Schedule* instances are not undirected: in each instance there is at least one part p; the effects of do-roll(p) can not be undone; the precondition requires merely that the respective machine is not busy and that the part is not processed elsewhere, so do-roll(p) can be applied, for example, in the initial state.

There can be local minima in *Schedule* under h^+, namely when one applies the working steps for a part in the wrong order (like painting before changing the shape). An example is the following (the example is a little complicated in order to demonstrate the worst-case for the local minima exit distance). There are three parts p_0, p_1 and p_2, and one color c. The goal is to have p_1 and p_2 cylindrical, polished, and painted in c (there is no goal for p_0). We consider the reachable state s where both p_1 and p_2 are already polished and painted in c, but not yet cylindrical, and p_0 has just been processed with the lathe so that machine is busy. We have $h^+(s) = 2$: a relaxed plan consists of do-roll($p_1\ldots$) and do-roll($p_2\ldots$). If we apply one of these actions, however, we end up in a dead end, $h^+ = \infty$ – the respective part is then not cold any longer, so the polished surface can not be re-achieved (see the proof to Theorem 25). We *can* apply do-time-step() to

s to free the lathe ($\rightarrow h^+ = 2$), and then apply one of do-lathe($p_1 \ldots$) or do-lathe($p_2 \ldots$) (applying any actions that do not achieve one of the "cylindrical" goals necessarily leads into states with $h^+ \geq 2$). Say we apply do-lathe($p_1 \ldots$) (applying do-lathe($p_2 \ldots$) leads into a symmetrical situation). The h^+ value for the result state s'' is $h^+(s'') = 4$: a shortest possible relaxed plan for s'' comprises do-time-step(), do-polish($p_1 \ldots$), do-immersion-paint($p_1, c \ldots$), and do-lathe($p_2 \ldots$). To reach a state s' with $h^+(s') < 2$, one must apply an action sequence of length 4 to s'', e.g. do-time-step() ($\rightarrow h^+ = 3$), do-polish(p_1) ($\rightarrow h^+ = 3$), do-time-step() ($\rightarrow h^+ = 2$), do-immersion-paint($p_1, c \ldots$) ($\rightarrow h^+ = 1$). The distance from s to the better state s' is thus 6. This is an upper bound in general.

Theorem 26 *Given a solvable instance (C, I, G) of the Schedule domain, and a reachable state s, $0 < h^+(s) < \infty$. The distance from s to a state s' with $h^+(s') < h^+(s)$ is at most* 6.

Proof: Let P_s^+ be an optimal relaxed plan for (A, s, G) (by prerequisite, there is at least one such P_s^+). We distinguish between seven cases:

1. P_s^+ contains the action $a =$ do-time-step(), and a is applicable in s.
2. P_s^+ contains an action a of the form do-immersion-paint($p, c \ldots$), do-drill-press($p, w, o \ldots$), or do-polish($p \ldots$), such that a is applicable in s.
3. P_s^+ contains an action a of the form do-spray-paint($p, c \ldots$), such that a is applicable in s.
4. P_s^+ contains an action a of the form do-grind($p \ldots$), such that a is applicable in s.
5. P_s^+ contains an action a of the form do-punch($p, w, o \ldots$), such that a is applicable in s.
6. P_s^+ contains an action a of the form do-lathe($p \ldots$), such that a is applicable in s.
7. P_s^+ contains an action a of the form do-roll($p \ldots$), such that a is applicable in s.

In case 1, the desired state s' can be reached with a single step. A relaxed plan for $s' = \mathit{Result}(s, \langle a \rangle)$ is given by removing a from P_s^+: the only delete effects of $a =$ do-time-step(), (busy, m) for machines m and (scheduled, p) for parts p, are irrelevant in that they are not needed by any action nor the goal. Finished along the lines of Proposition 2.

In case 2, s' can be reached with two steps: first apply a, then apply do-time-step(). For the resulting state s', removing a from P_s^+ yields a relaxed plan. To see this, consider that the delete effects of either of the possible actions are either irrelevant ((not-objscheduled) in all cases, (not-has-hole, p, w, o) in the case of do-drill-press($p, w, o \ldots$)), or non-goal properties of p (another color in the case of do-immersion-paint($p, c \ldots$), another surface condition in the case of do-polish($p \ldots$)).

The (color or surface) property that a achieves is part of the goal as P_s^+ is optimal; all other (color or surface) properties are then irrelevant. Due to the execution of do-time-step(), s' contains at least the (not-busy, m) and (not-scheduled, p) facts that s contains.

In case 3, three steps suffice: apply do-time-step(), then replace do-spray-paint($p, c\ldots$) with do-immersion-paint($p, c\ldots$) in P_s^+ (the add effects of both actions are the same, and the precondition of do-immersion-paint($p, c\ldots$) is, trivially, fulfilled upon applying do-time-step()). In the resulting situation, like in case 2 above, two more action applications yield the desried state s'.

In case 4, s' can be reached with four steps: apply a and do-time-step(); if p was painted in color c in s, then also apply do-immersion-paint($p, c\ldots$), and another do-time-step(). In the resulting state s', removing a from P_s^+ yields a relaxed plan. All of a's delete effects have been re-achieved except, possibly, (not-objscheduled), which is irrelevant, and a different surface condition of p, which is irrelevant as p needs be smooth for the goal or else P_s^+ would not be optimal.

In case 5, three steps suffice: if p must be rough for the goal, then apply a and do-time-step(); else, apply the sequence do-time-step(), do-drill-press($p, w, o\ldots$), do-time-step(). In the first case, a does not delete any relevant facts. As for the second case, do-drill-press($p, w, o\ldots$) has, in general, no relevant delete effects, and adds the same facts as do-punch($p, w, o\ldots$) except (surface-condition, p, rough) which is irrelevant when it is not part of the goal. Altogether, in both cases a relaxed plan for the resulting state s' is thus given by removing a from P_s^+ (note that do-punch($p, w, o\ldots$) requires that p is cold so this precondition of do-drill-press($p, w, o\ldots$) is given).

In case 6, s' can be reached with five steps. We distinguish between a further two cases.

- First case, p is hot in s. Then p is already cylindrical (do-roll($p\ldots$) must have been applied earlier), so a = do-lathe($p\ldots$) must be part of the optimal relaxed plan P_s^+ in order to achieve the goal condition (surface-condition, p, rough). Apply a and do-time-step(); if p was painted in color c in s, also apply do-immersion-paint($p, c\ldots$) and another do-time-step(). In the resulting state s', all relevant delete effects of a have been re-achieved. The distance from s to s' is at most four.

- Second case, p is cold in s. If executing do-time-step() in s yields one of the above cases 1 to 5, then proceed accordingly (using maximally four more steps). Else, apply a and do-time-step(); if p was polished (smooth) in s then further apply do-polish($p\ldots$) (do-grind($p\ldots$)) and do-time-step(); if p was painted in color c in s then further apply do-immersion-paint($p, c\ldots$). In the resulting state s', removing a from P_s^+ yields a relaxed plan. To see this, consider that, in s', all relevant delete effects of a have been re-achieved, except (not-busy, immersion-painter) and (not-scheduled, p). Now, if P_s^+ used the immersion painter in any way then that action would have become applicable after executing do-time-step() in contradiction to our assumptions here (see above). Similarly, if P_s^+ used p in any other way but in do-lathe($p\ldots$) then that would have become applicable after executing do-time-step() (note that

p is cold in our case here; also note that P_s^+ will not use do-roll($p\ldots$) on top of do-lathe($p\ldots$) as the add effects of the latter contain those of the former except for the irrelevant (temperature, p, hot) and (not-has-hole, p, w, o) facts). The distance from s to s' is at most five.

One can transform case 7 to case 6 by applying do-time-step(), then replacing do-roll($p\ldots$) with do-lathe($p\ldots$) in P_s^+ – as noted above, do-lathe($p\ldots$) contains all add effects of do-roll($p\ldots$) except for the irrelevant ones. The distance from s to the desired state s' is at most six: one plus the worst-case distance in case 6. ■

9 No Unrecognized Dead Ends

In the next group of domains, the maximal local minimum exit distance is unbounded, but there are no unrecognized dead ends. The domains are *Blocksworld-arm*, *Depots*, *Driverlog*, and *Rovers*. For each domain I give a brief description of the semantics, consider the dead end class of the domain's instances, and give an example showing that there is no upper bound on the maximal local minimum exit distance. In *Driverlog* and *Rovers* I also give an example showing that there is no upper bound on the maximal bench exit distance (which is not as relevant when the maximal local minimum exit distance is unbounded anyway). In *Rovers* I additionally give a proof argument showing that the maximal local minimum and bench exit distances grow only in the diameter of the underlying road map.

9.1 Blocksworld-arm

The instances of *Blocksworld-arm* are the same as those of *Blocksworld-no-arm*. The difference is that blocks are moved via an arm, so that there are four operators to pickup a block, put a block down, stack a block onto some other block, or unstack a block from some other block. See the domain details in Appendix A.2.

Corollary 10 *Given an instance (C, I, G) of the Blocksworld-arm domain, the corresponding state space is undirected.*

Proof: All actions are invertible in the sense of Definition 14: pickup(x) and putdown(x) invert each other, the same for stack(x, y) and unstack(x, y). Finished with Lemma 1. ■

The maximal local minimum exit distance in *Blocksworld-arm* is not bounded. This can be seen with the same example setting that I used to show that the maximal bench exit distance in *Blocksworld-no-arm* is not bounded. There are $n + 1$ blocks $b_1, \ldots, b_{n+1}$, with initial state $I = \{(\text{clear}, b_1), (\text{on}, b_1, b_2), \ldots, (\text{on}, b_{n-1}, b_n), (\text{on-table}, b_n), (\text{clear}, b_{n+1}), (\text{on-table}, b_{n+1})\}$ and goal state $G = \{(\text{on}, b_1, b_2), \ldots, (\text{on}, b_n, b_{n+1})\}$. The stack $b_1, \ldots, b_n$ needs to be disassembled, and rebuilt on top of b_{n+1}. An optimal relaxed plan for I has length $n + 1$, by unstacking b_i from b_{i+1} for $i < n$, picking up b_n, and stacking b_n onto b_{n+1}.

The initial state lies on a local minimum: the applicable actions are to pick up b_{n+1}, or to unstack b_1 from b_2. Applying the former action leads into a state with h^+ value $n+2$ (in addition to what's been done before, b_{n+1} must be put back onto the table or onto a different block). Applying the latter action leads into a state I' with $h^+(I') = h^+(I) = n+1$ (in the relaxed plan, stack b_1 back onto b_2 instead of unstacking it). From I', the one applicable action – stack b_1 onto b_2 – leads back into I, and the other applicable action – put b_1 onto the table – leads into a state with h^+ value $n+2$ (one must include the action picking up b_1 into the relaxed plan). This proves that I forms a local minimum together with I'. Now, consider that $h^+(s') \geq n+1$ for all reachable states s' where b_n is not held by the robot arm: b_n must be picked up (or unstacked), and be stacked onto b_{n+1}; to make this possible, b_{n-1} must be unstacked from b_n if it is currently on top of it, and otherwise it must be stacked onto b_n for achieving the goal; the same argument can be iteratively applied upwards to b_1. Now consider again the initial state. Reaching, from there, the nearest state where the robot arm holds b_n involves $2n-1$ steps. One must move b_i away for $i < n$ (unstack it from b_{i+1} and put it on the table), and pick up b_n. This proves that the maximal local minimum exit distance in the instance is at least $2n-1$. Note that the h^+ value of the nearest state s where the robot arm holds b_n is $h^+(s) = 2n-1$: the best relaxed plan for s is to stack b_n onto b_{n+1}, and for all b_i with $i < n$ to pick b_i up and stack b_i onto b_{i+1}. This shows that, on the shortest path from I to a state s' with $h^+(s') < h^+(I)$, the h^+ value increases by at least $n-2$ (i.e., arbitrarily high values when generalizing over n).

9.2 Depots

The *Depots* domain is a kind of mixture between *Logistics* and *Blocksworld-arm*. There is a set of locations, a set of trucks, a set of pallets, a set of hoists, and a set of crates. The trucks can transport crates between locations, the hoists can be used to stack crates onto other crates, or onto pallets. In a little more detail, there are operators to move a truck between (different) locations, to load a crate that is held by a hoist onto a truck at a location, to unload a crate with a hoist from a truck at a location, to lift a crate with a hoist from a surface (a pallet or a crate) at a location, and to drop a crate that is held by a hoist onto a surface at a location. A hoist can hold only one crate at a time. The crates are initially arranged in arbitrary stacks, where the bottom crate in each stack is standing on a pallet. The goal is to arrange the crates in some other arbitrary stacks on (possibly) other pallets, which can involve transporting crates to other locations (pallets can not be moved). See the domain details in Appendix A.5.

An instance of *Depots* can be unsolvable when there is a crate that must be transported but there is no truck, or when crates need be arranged at a location where there is no hoist. Removing static predicates gets rid of operator type information and of actions that try to drive a truck within the same location. State spaces in *Depots* are undirected so there are no dead ends in solvable instances.

Proposition 9 *Given an instance (C, I, G) of the Depots domain, the corresponding state space is undirected.*

Proof: Actions that drive a truck are invertible in the sense of Definition 14 by driving it back. Actions that load a crate onto a truck and actions that unload a crate from a truck invert each other in that sense. Actions that drop a crate are inverted in that sense by the respective action lifting the crate. Actions that lift a crate c with a hoist h from a surface s at a location l are not inverted by the respective dropping action in the strict sense of Definition 14: all parts of the definition are fulfilled except that the dropping action requires s to be at l; in difference to that the lifting action requires c to be at l. So there is no immediate formal reason why the dropping action should be applicable directly after the lifting action. This follows from the domain semantics, however: if c is at l and c is on s, both of which are requirements of the lifting action, then s must be at l, too. This concludes the argument. ■

The maximal local minimum exit distance in *Depots* is not bounded. This can be seen with essentially the same example setting that we used above to show the same thing in *Blocksworld-arm*. All we need to do is translate the names a bit. There is one location with a single hoist, there are $n + 1$ pallets $p_1, \dots, p_{n+1}$, and there are $n + 1$ crates $c_1, \dots, c_{n+1}$. The crates are arranged such that, initially, c_n is on p_1, c_i is on c_{i+1} for $1 \leq i < n$, and c_{n+1} is on p_2. The goal is to have b_i on b_{i+1} for $1 \leq i \leq n$ (i.e., as before but including the requirement that b_n is on b_{n+1}). Obviously, the semantics of this example are exactly the same as what we have seen in the *Blocksworld-arm* example above. The crate stack $c_1, \dots, c_n$ needs to be disassembled, and rebuilt on top of c_{n+1}. Putting a block onto the table corresponds to dropping a crate onto a pallet, picking a block up from the table corresponds to lifting a crate from a pallet, stacking/unstacking a block onto/from a block is just dropping/lifting a crate onto/from a crate. The only difference lies in that all operators refer to the single location – which does not play a role – and that there is "limited space on the table", in form of the $n + 1$ pallets – which is all the space we will ever need. Thus, like we have seen it in *Blocksworld-arm*, the initial state lies on a local minimum because disassembling the stack $c_1, \dots, c_n$ results in an increasing h^+ value. The maximal local minimum exit distance in the instance is at least $2n - 1$, and on the shortest path from I to a state s' with $h^+(s') < h^+(I)$, the h^+ value increases by at least $n - 2$.

9.3 Driverlog

Driverlog is a variation of *Logistics* where drivers are needed for the trucks, and where drivers and trucks can move along arbitrary (bi-directional) road maps. The road maps for drivers and trucks can be different. There are operators to load/unload an object o onto/from a truck t at a location l, to board/disembark a driver d onto/from a truck t at a location l, to walk a driver from location l to location l', and to drive a truck from location l to location l' using driver d. The preconditions and effects of loading/unloading objects are the obvious ones.

A driver d can board a truck t at location l only if d and t are at l, and the truck is empty; in effect, d is driving t, d is no longer at l, and the truck is no longer empty. Disembarking a driver has the obvious inverse preconditions and effects. In order to walk a driver from l to l' there must be a *path* between l and l'. In order to drive a truck t from l to l' using driver d, d must be driving t, and there has to be a *link* from l to l'. Paths and links form arbitrary graphs over the locations, the only restriction being that they are undirected, i.e., if a truck or driver can move from l to l' then it can also move back. This restriction is imposed on the *Driverlog* instances as can be generated with the IPC-3 generator.[6] The restriction does make a significant structural difference: as we see below, in the presence of directed edges there can be unrecognized dead ends, while with the competition restriction the *Driverlog* state spaces are undirected. The detailed formal definition of the *Driverlog* domain is in Appendix A.6.

An instance of *Driverlog* can be unsolvable when either an object, a truck, or a driver can not reach their goal location due to a lack of movement links. Removing static predicates gets rid of operator type information and of the predicates encoding movement links (these become implicitly represented by the ground movement actions whose preconditions can become true). Due to the bi-directional movement links, the state spaces in *Driverlog* are undirected so there are no dead ends in solvable instances.

Corollary 11 *Given an instance (C, I, G) of the Driverlog domain, the corresponding state space is undirected.*

Proof: All actions are invertible in the sense of Definition 14: loading and unloading an object invert each other, boarding and disembarking a driver invert each other, and walking a driver/driving a truck from l to l' is inverted by walking/driving from l' to l. Finished with Lemma 1. ■

As said above, when allowing directed road maps there can be unrecognized dead ends. Consider the following example. There are two locations l_1 and l_2, and there is a path and a link from l_1 to l_2 but not back. There is one truck t, and there are two drivers d_1 and d_2. Both drivers and the truck are initially at l_1. The goal requires d_1 to be at l_1, and t to be at l_2. One can solve the instance by boarding d_2 onto t, and driving t to l_2. However, when, in the initial state, walking d_2 to l_2, one ends up in a dead end state. Let's call the state s. The problem in s is that the only possible driver left to take t to l_2 is d_1; but when doing that, there is no way for d_1 to get back to l_1. In the relaxation, however, we can solve s simply by boarding d_1 onto t, and driving t to l_2: d_1 is at the goal position already in s, and this fact is not deleted. Thus s is an unrecognized dead end. Note that one can easily make s arbitrarily deep, by adding additional locations linked to l_1 (but not to l_2), as well as objects to be transported within these additional locations. Once d_2 has walked to l_2 there will be no way to achieve the goal, but a relaxed plan, no matter what other actions t and d_1 can take.

[6]According to personal communication with Derek Long and Maria Fox.

Even when restricting road maps to be undirected, as the IPC-3 generator does, the maximal local minimum exit distance in *Driverlog* is unbounded. Consider the following example. There are the locations l^t and l^d, as well as $l_1, \ldots, l_n$ where $n \geq 2$. The road maps are the following. There are bi-directional paths and links between l_i and l_{i+1} for $1 \leq i < n$. There are paths and links between l^t and l_1, there are paths and links between l_n and l^d. Between l^t and l^d there are only links, i.e., between these locations only trucks can move. There is one truck t, and one driver d. The goal is to have t at l^t, and d at l^d. In the initial state, both t and d are at l^d. We have $h^+(I) = 2$: the shortest relaxed plan is to board d, and drive to l^t. Let us consider the successor states of I. If d walks to l_n, we get the h^+ value 3: d must walk back. If d boards t we get the state I', with $h^+(I') = h^+(I) = 2$ (the shortest relaxed plan is to disembark d, and drive t to l^t). From I', we either get back to I by disembarking d, or we drive t to l_n which yields h^+ value 3 (the shortest relaxed plan drives t back), or we drive t to l^t yielding the state I''. The only interesting case is the last one where we get $h^+(I'') = h^+(I') = h^+(I) = 2$ (the shortest relaxed plan drives t to l^d, and disembarks d). The new states reachable from I'' are those given by either driving t to l_1, or disembarking d at l^t. In the first case, the shortest relaxed plan in the resulting state is to drive t back to l^t, then drive to l^d and disembark d (driving over $l_2, \ldots, l_n$ to l^d then disembarking d and driving to l^t takes longer as $n > 1$). So there h^+ is 3. In the second case, when one disembarks d in I'', the h^+ value also rises to 3: the shortest relaxed plan is to board d, drive t to l^d, and disembark d – one could also walk d over $l_1, \ldots, l_n$ to l^d but this, too, would take at least 3 steps as $n \geq 2$. This altogether shows that I, I', and I'' together form a local minimum. Now, remember that we have $h^+(I) = 2$. But obviously the nearest state, from I, where $h^+(s) < 2$ is the one where t is at l^t, and d is at l_n. Let's call this state s. The distance from I to s is $n + 3$: one must board d, drive t to l^t, disembark d, and walk from l^t via $l_1, \ldots, l_{n-1}$ to l_n. This shows that the maximal local minimum exit distance in the example instance is at least $n + 2$. Note that, on all paths from I to s, the h^+ value rises to at least $\lfloor n/2 \rfloor$ – namely when d has delivered the truck to l^t, and reached the middle of the path $l_1, \ldots, l_n$. So in general one must increase the h^+ value by arbitrarily high values in order to reach a better state.

Note that the above example instance uses a road map where there is a movement link that can be taken by trucks, but not by drivers. While this is something that seems natural, and that does occur in the IPC-3 instances, it is an open question whether there is a bound on the maximal local minimum exit distance (or, actually, whether there are local minima at all) in the special case where a link for trucks implies a path for drivers. In any case, benches can be arbitrarily large, i.e., there is no upper bound on the maximal bench exit distance. Consider the example where there are n locations $l_1, \ldots, l_n$ connected by paths and links from l_i to l_{i+1} and vice versa for $1 \leq i < n$. Say there are a truck t and a driver d which are initially at l_1, and there is an object o that is initially at l_n and shall be transported to l_1. Consider the state s where d has boarded t in the initial state. We have $h^+(s) = n + 1$: the relaxed plan is to drive all the way to l_n, then load o at l_n and unload o at l_1. The h^+ value does not decrease until t has reached

l_n and loaded o: when, starting from s, one has driven the truck to l_i then the relaxed plan must include actions driving from l_i to l_n, as well as actions driving from l_i to l_1.

9.4 Rovers

In *Rovers*, a number of rovers must navigate through a road map of waypoints, take rock and soil samples as well as images, and communicate the data to a number of landers. The available operators are the following. One can navigate a rover r from one waypoint w to another waypoint w' – to do this, w' must be visible from w, and r must be able to traverse from w to w' (plus the obvious condition that r must be at w). One can sample soil/rock with a rover r at a waypoint w using a store s – to do so, r must be equipped for soil/rock analysis, s must be the store of r, s must be empty, and there must be soil/rock to sample at w; in effect one has the soil/rock analysis to w, the soil/rock sample is no longer at w, and s is no longer empty. One can empty a full store by dropping the store; the only effect is that the store is no longer full. One can calibrate a camera c on a rover r at a waypoint w using an objective o, and one can take an image of an objective o in mode m with a camera c on a rover r at a waypoint w. For both operators, r must be equipped for imaging, c must be on board r, and o must be visible from w. To calibrate the camera, o must be a calibration target for c. To take an image, c must support m, and c must be calibrated already. The only effect of calibrating is that c is calibrated, the effects of taking an image are that r has the image data for o in mode m, and that c is no longer calibrated. Finally, there are three operators with which a rover r at waypoint w can communicate data to a lander l at waypoint w'; for rock or soil the data is that about the waypoint w'' where the sample was taken, for images the data is that about the respective objective o and mode m. To enable the communication, w' must be visible from w, and of course r must have the data. The only effect is that the data is communicated. The goal is always to have a number of soil/rock/image data pieces communicated.

The definition of the *Rovers* instances is along the lines of what the IPC-3 generator can produce. In what follows, we use the formal notations for the domain as introduced in Appendix A.20. The notations should be self-explanatory, otherwise the reader is referred to the appendix.

As said, a *Rovers* instance features a road map of waypoints, some rovers, some landers, some stores, some cameras, some objectives, and some modes. Rovers and landers have initial positions on the waypoint map, each store and each camera is on board a rover. The instances are restricted in that the visibility and navigability between waypoints are bi-directional — if waypoint x is visible from waypoint y then the same holds true vice versa; if a rover can move from x to y then it can also move back. I remark that the restriction makes a significant structural difference: while with undirected movability (i.e., , wp-visible and can-traverse relations) the domain is free of unrecognized dead ends (see below), in the presence of directed movability there can be such dead ends. As an example, consider an instance where one must take an image at waypoint w^i, and can calibrate the camera on board rover r at waypoints w^c_1 or w^c_2. Assume that there is a fourth waypoint w, and that r can (only) move from w_1^c to w, from w_1^c to w^i, from w to w^i, and from w to w_2^c. When r starts at w_1^c, moving to w yields an unrecognized dead end state: the camera can now only be calibrated at w_2^c, but from w_2^c there is no way to w^i; however, in the relaxation the fact (at, w) is not deleted by navigate(r, w, w_2^c) and thus there is a relaxed plan.

The instances are also restricted in that no camera is initially calibrated. This is just one way of saying that each calibrated camera has at least one calibration target. Again, without the restriction there can be unrecognized dead ends. Just imagine a camera that is initially calibrated but for which there is no calibration target, and that must be used for taking two images. The relaxed plan ignores the fact that the calibration of the camera will be deleted after taking the first image. But for that reason there is no plan.

Removing static predicates gets rid of type information, and of static instance properties like which camera is on board which rover. Let us be precise about how *Rovers* behaves with respect to dead ends. First, observe that dead ends exist. Say there is a soil sample at waypoint w^s that must be analyzed, there are two rovers r_1 and r_2, and one lander l at waypoint w^l. Say further that both rovers can navigate to w^s, that r_1 can navigate to a point visible from w^l, and that r_2 can not navigate to such a point. If one analyzes the soil sample with r_2, one ends up in a dead end: the fact $(\text{at-soil-sample}, w^s)$ can not be re-achieved, and r_2 can not communicate the analysis result to the lander. Note that there isn't a relaxed plan in this case either. This holds true in general.

Lemma 6 *Given an instance (C, I, G) of the Rovers domain, and a reachable state s. Then there is a plan for (A, s, G) if and only if there is a relaxed plan for (A, s, G).*

Proof: Let us identify exactly when there is a plan for the state s. We need a few notations. For a rover r, by $W(r)$ we denote all waypoints that r can navigate to, given the wp-visible and can-traverse relations and r's position in I. By $VW(r)$ we denote all waypoints that can become visible for the rover, i.e., all waypoints w' such that there is a waypoint $w \in W(r)$ with $(\text{wp-visible}, w, w') \in I$; similarly, by $VO(r)$ we denote all objectives that can become visible for the rover, i.e., all objectives o such that there is a waypoint $w \in W(r)$ with $(\text{ob-visible}, w, o) \in I$ (note that the reachability relations do not change over time). By w^l we denote the waypoint where a lander l is located. There is a plan for s iff:

1. For each goal fact $(\text{communicated-soil-data}, w)$ there is a rover r and a lander l such that $w^l \in VW(r)$, and either $(\text{have-soil-analysis}, r, w) \in s$, or $(\text{at-soil-sample}, w) \in s$, r is equipped for soil analysis, and $w \in W(r)$.

2. For each goal fact $(\text{communicated-rock-data}, w)$ there is a rover r and a lander l such that $w^l \in VW(r)$, and either $(\text{have-rock-analysis}, r, w) \in s$, or $(\text{at-rock-sample}, w) \in s$, r is equipped for rock analysis, and $w \in W(r)$.

3. For each goal fact $(\text{communicated-image-data}, o, m)$ there is a rover r, a camera c, and a lander l such that r is equipped for imaging, c is on board

r, c supports m, $o \in VO(r)$, and $w^l \in VW(r)$; if c is not calibrated in s then we also need an objective $o' \in VO(r)$ that is a calibration target for c.

It is obvious that there can only be a plan for s when the above three conditions are fulfilled. The other way round, note that the only state-dependent properties required are the status of the soil/rock samples (at a waypoint or, in the form of the respective analysis, in a rover), as well as the calibration of c. As for the latter, we postulate that c is not calibrated initially, so if it is calibrated then there must be an appropriate calibration target $o' \in VO(r)$, and we can always re-calibrate c. As for the former, achieving the goal for one soil/rock analysis does not affect the status of any other soil/rock analysis. One can thus, if the above three conditions are given, reach the goals one by one as follows. For each single goal fact an achieving action sequence can easily be constructed: navigate an appropriate rover (here it is important that the movability relations are undirected so we can always navigate the rover back); if needed, drop a store or calibrate a camera; analyze a sample or take an image; and communicate with an appropriate lander. With the argumentation above the three conditions will still hold true in the resulting state, so we can iterate the process until we have reached the goal. Note that goal facts are not deleted by any action.

The lemma follows from observing that the three conditions above also identify exactly when there is a *relaxed* plan for s. If one of the conditions is not fulfilled then the respective goal fact can not be reached with- or without delete lists. The other direction is trivial. ■

Note that Lemma 6 also identifies when, exactly, a *Rovers* instance is solvable. The non-existence of unrecognized dead ends is a trivial consequence of the lemma.

Corollary 12 *Given a solvable instance (C, I, G) of the Rovers domain, the corresponding state space is at most recognized.*

Proof: From Lemma 6 it follows immediately that all dead ends are recognized. ■

The maximal bench exit distance and the maximal local minimum exit distance are both unbounded in *Rovers*. To see that for the maximum bench exit distance, consider the following example. There are one rover r, one lander l, and $n+1$ waypoints $w_0, \dots, w_n$ which are connected via $(\text{wp-visible}, w_i, w_{i+1})$ (and vice versa) for $i < n$, and $(\text{can-traverse}, r, w_i, w_{i+1})$ (and vice versa) for $i < n$. The rover is equipped for soil analysis, there is a soil sample at w_n, l is at w_0, and r is initially at w_1. The goal is to communicate the soil data. We got $h^+(I) = n+1$: $n-1$ actions navigating r to w_n, sample the soil, communicate the data to the lander. The nearest state s' with $h^+(s') < n+1$ is the one where r is in w_n, and has sampled the soil already: then, the relaxed plan navigates back to w_1 and communicates the data. For all states between I and s' where r is at a point w_i, $i < n$, the relaxed plan must include $n-1$ navigation and the two sampling and communication actions. Note that no other actions but navigating the rover are applicable in these states. The distance between I and s' is n.

To see that the maximal local minimum exit distance is unbounded, consider the following example. There are one rover r, one lander l, and $n+1$ waypoints $w_0, \ldots, w_n$ connected in the same way as above. The lander is at w_0 (r's initial position does not matter). The rover is equipped for image analysis. There are one camera c, one mode m which c supports, and three objectives o^c, o^i_1, and o^i_2. Say o^c is visible from w_n, and that both o^i_1 and o^i_2 are visible from w_1. Say o^c is a calibration target for c, and the goal is to communicate image data for both o^i_1 and o^i_2 (in mode m). We consider the reachable state s where the rover r is at w_1, and the camera c is already calibrated. We got $h^+(s) = 4$: take the two images, and communicate the two data pieces. We get larger h^+ values when navigating away from w_1. However, if we do take one of the images, the h^+ value of the resulting state s'' increases to $h^+(s'') = n+3$: c is no longer calibrated so the relaxed plan must navigate all the way to w_n ($n-1$ steps), calibrate c, take the other image, and communicate the two data pieces. By communicating the image data already taken in s'', we get a state s''' with $h^+(s''') = n+2$. The distance from s to a state s' with $h^+(s') < 4$, however, is $2n+1$: the nearest such state is the one where, starting from s''', one has navigated the rover to w_n ($n-1$ steps, h^+ constantly $n+2$), re-calibrated c (1 step, resulting h^+ value $n+1$), and navigated back to w_2 ($n-2$ steps, resulting h^+ value 3 – navigate to w_1, take the remaining image, communicate the data).

The two above examples, in order to produce arbitrarily large bench or local minimum exit distances, both make use of arbitrarily large distances that must be navigated. This observation holds true in general: as we prove now, both distances have an upper bound linear in the diameter of the waypoint graph. Given a solvable instance (C, I, G) of the *Rovers* domain, we define the *diameter* d of the instance as follows. For a graph (V, E) the diameter is the longest distance between any two connected points in the graph. For the *Rovers* instance, the diameter is the maximum, over all rovers r, diameter of the graph given by $W(r)$ (all waypoints r can navigate to) and the edges according to the (wp-visible, w, w') and (can-traverse, r, w, w') relations. From any reachable state s in a *Rovers* instance, with $h^+(s) < \infty$, one can reach a state s' with $h^+(s') < h^+(s)$ within $3d+2$ steps.

Theorem 27 *Given a solvable instance (C, I, G) of the Rovers domain, with diameter d. Given a reachable state s, $0 < h^+(s) < \infty$. The distance from s to a state s' with $h^+(s') < h^+(s)$ is at most $3d+2$.*

Proof: Let P^+_s be an optimal relaxed plan for (A, s, G) (by prerequisite, there is at least one such P^+_s). We distinguish between four cases:

1. P^+_s contains an action a dropping a store, calibrating an instrument, or communicating a data piece, such that a is applicable in s.

2. P^+_s contains an action a taking a soil or rock sample, such that a is applicable in s.

3. P^+_s contains an action a taking an image, such that a is applicable in s.

4. Cases 1, 2, and 3 do not hold (implying that P_s^+ contains an action a applicable in s that navigates a rover).

In case 1, the desired state s' can be reached with a single step. A relaxed plan for $s' = Result(s, \langle a \rangle)$ can be constructed simply by removing a from P_s^+: neither calibrating an instrument nor communicating a data piece have any delete effects, and the only delete effect of dropping a store st, (full, st), is not needed in any precondition or the goal. Finished along the lines of Proposition 2.

In case 2, the desired state s' can be reached within two steps. Say a takes a soil or rock sample with rover r and store st at waypoint w. We reach s' by applying a, then dropping st. For s' a relaxed plan can be constructed by removing a from P_s^+: s' is the same as s except that r has the soil / rock sample, and the sample is no longer at w; the latter fact is not needed more than once, i.e., only by a, in the optimal relaxed plan P_s^+ (there is no point in taking the sample two times).

In case 3, the desired state s' can be reached within $2d + 2$ steps. Say a takes an image with rover r and camera c at waypoint w. Upon execution of a, r has the respective image data, but c is no longer calibrated. We might thus need to re-calibrate c. To do this, the following suffices. First, navigate r to a calibration target, i.e., to a waypoint $w' \in W(r)$ (as before, $W(r)$ denotes the set of waypoints reachable for r) from which a calibration target for c is visible (there is at least one such w' or else c could not have been calibrated in the first place). Then, calibrate c, and navigate back to w (note here that by definition all waypoint connections can be traversed in both ways). In the resulting state s', a relaxed plan can be constructed by removing a from P_s^+: s' is the same as s except that r already has the image data taken by a. The distance from s to s' is at most $2d + 2$: two traversals of the waypoint graph plus taking the image and re-calibrating c.

Say we are in none of the above 3 cases. Then P_s^+ starts with an action navigating a rover r away from the waypoint w it is currently at. Consider the set W of all waypoints that r is navigated to in P_s^+, including w. W, together with the waypoint connections used by the respective navigation actions in P_s^+, forms a connected sub-component of $W(r)$. Now, consider all actions a in P_s^+ that use r at a waypoint w', by taking a soil / rock sample, calibrating a camera, taking an image, or communicating a data piece. At least one such action a becomes applicable when, starting in s, we navigate the rover to the respective waypoint w': this is the case for actions a using r to take soil / rock samples, or to calibrate instruments; if there are no such actions then the case holds for the actions using r to take images; if there are no such actions either then the case holds for the actions using r to communicate data. We pick an appropriate action a, and, starting in s, move r along the shortest possible path to w'. For the resulting state s'', we can construct a relaxed plan $P_{s''}^+$ such that $|P_{s''}^+| = |P_s^+|$: w' is a member of the connected component W; as all waypoint connections can be traversed in both ways, we can remove all actions from P_s^+ that navigate r, and insert actions spanning the same component W starting from w' instead (which can obviously be done within the same number of actions). By construction, a is applicable in s'' so, with the argumentation given above for the other cases, from s'' one can reach a state s' with a strictly shorter relaxed plan within at most

$2d + 2$ steps. We have reached s'' from s with at most d steps (traversing the shortest possible path from w to w'), so our overall upper bound on the distance from s to s' is $3d + 2$. ■

To understand why we need the seemingly complex proof argument for the seemingly simple case 4 in the proof to Theorem 27, note that the path which the relaxed plan itself uses to get from w to w' might not be the shortest possible one: the relaxed plan must reach all points in the whole component W, and for that a sub-optimal path to w' might be better suited.

I remark that Theorem 27 entails Lemma 6: with the proof to Theorem 27, one can, in particular, construct a (real) plan from all states s for which there is a relaxed plan (by iteratively traversing to a state s' with $h^+(s') < h^+(s)$). More importantly, Theorem 27 shows that both the maximal local minimum exit distance and the maximal bench exit distance in *Rovers* grow exclusively in the diameter of the waypoint graph, *not* in things like the number of rovers, samples to be taken, etc.

10 Unrecognized Dead Ends

We finally consider the group of domains where there *can* arise unrecognized dead ends. The domains are *Mystery*, *Mprime*, *Freecell*, *Miconic-ADL*, and *Assembly*. We focus on them in that order. For each domain I give a brief description of the semantics, and an example of an arbitrarily deep unrecognized dead end state. This already concludes the topological discussion of the domain (with one exception, see below). By Proposition 1, in the presence of unrecognized dead ends there is no constant upper bound on the local minimum exit distance, which implies that it is not particularly relevant whether there is an upper bound on the maximal bench exit distance or not. The exceptional domain is *Assembly*. There, in order to explain FF's excellent behavior in the AIPS-1998 example suit, I include a rather detailed discussion of implications between problem structure and h^+ topology.

10.1 Mystery

Like a number of benchmark domains we have already had a look at, *Mystery* is a transportation kind of domain, where objects must be transported between locations by means of vehicles. Unlike in the previously seen transportation domains, however, there can be unrecognized dead ends in *Mystery*. This is due to the fact that, here, vehicles use non-replenishable fuel. More explicitly, in a *Mystery* instance we got a set L of locations, a set O of objects, and a set V of vehicles. We also have sets F and S of fuel numbers and space numbers. Each location initially has a certain fuel number – the number of fuel items available at the location – and each vehicle has a certain space number – the number of objects the vehicle can carry at a time. There are operators to move vehicles between locations, and to load (unload) objects onto (from) vehicles. A move from location l to location

l' can only be made if l and l' are connected (where the connection relation is an arbitrary graph), and if there is *at least one fuel unit available at l* (l has a fuel number that has a lower neighbor). In effect of the move, the respective vehicle is located at l', and the amount of fuel at l is decreased by one unit, i.e., l is assigned the next lower fuel number. In a similar fashion, an object can only be loaded onto a vehicle if there is space for that, and in effect the available space decreases. Unloading the object frees the space again. Note that there is no way to re-gain fuel items at a location. See the domain details in Appendix A.19.

It is easy to construct an unrecognized dead end state in *Mystery*. Consider the solvable instance where there are three locations l_1, l_2, and l_3 connected as a line $l_1 \leftrightarrow l_2 \leftrightarrow l_3$, there is one vehicle v, and there are two objects o_1 and o_2. The vehicle has two space units (the available space does not matter much in this example as long as there are at least two space units). At l_2 there are two fuel units, l_1 and l_3 each got one fuel unit. Object o_1 is initially at l_1, object o_2 is initially at l_2, and both objects shall be transported to l_3. The vehicle is initially at l_2. One can solve the instance by moving from l_2 to l_1, loading o_1, moving back to l_2, loading o_2, moving to l_3, and unloading both objects. But if one transports only o_2 to l_3 first, one is trapped: say, from the initial state, we load o_2 and move to l_3. Let's call the resulting state s. In s, there is only one fuel unit left at l_2, which is not enough to move the vehicle over to l_1 and then transport o_1 from l_1 to l_3. However, all three locations have one fuel unit left in s so a relaxed plan has no trouble achieving the goal as it does not take the fuel consumption into account.

In the above example the unrecognized dead end is not deep (c.f.Definition 9), i.e., from the state s we have identified only one more unrecognized dead end state is reachable (resulting from unloading o_2). One can however easily construct unrecognized dead ends that are arbitrarily deep. Say, in the above example, we got additional locations $l'_1, \dots, l'_n$ where l'_i is connected to l'_{i+1} (one can also connect l_2 or l_3 to l'_1 without changing the meaning of the example). Say that $l'_1, \dots, l'_n$ each have one fuel unit, that there is a vehicle v' with one space unit at l'_1, and that there is an object o' at l'_1 which shall be transported to l'_n. Now, consider the state s that we reach just as before, by loading o_2 onto v and moving v to l_3 in the initial state. Consider the n states that are reachable from s by first loading o' onto v' and then moving v' towards l'_n. All these states are unrecognized dead ends: working towards the goal for o' does not make a difference with respect to the situation for o_1, which causes unsolvability of these states in spite of the presence of relaxed plans.

10.2 Mprime

The *Mprime* domain is exactly identical to *Mystery* except that there is an additional *transfer* operator to transfer fuel units between locations. The operator can transfer one fuel unit from location l to location l' if l and l' are connected, and l has at least two fuel units left. As the result of applying the operator, l's fuel number decreases by one while l''s fuel number increases by one. See the domain details in Appendix A.18.

One can use exactly the same examples as we used in *Mystery* to show that there can be unrecognized dead end states in *Mprime*, and that these states can be arbitrarily deep. Reconsider Section 10.1. In both example descriptions, in the state s all locations have at most a single fuel unit left so no *transfer* action can become applicable anyway, neither in reality nor in the relaxation.

10.3 Freecell

The *Freecell* domain is a STRIPS formulation of the widely known solitaire card game that comes with Microsoft Windows. A number of cards from different suits are initially arranged in random stacks on a number of columns. The cards must be put home. For each suit of cards there is a separate home column on which the cards from that suit must be stacked in increasing order of card value. There is a number of free cells. The cards can be moved around according to certain rules. A card is *clear* if it has no other card on top of it. Any clear card can be put into a free cell (if it's not already there), each free cell holds only one card at a time. Any clear card can be moved onto an empty column. A clear card c can be put home when the last card put home in the same suit was the one preceeding c. If c and c' are clear cards from differently colored suits, then one can stack c on top of c' if c' is not in a free cell, and c's card value is one less than the card value of c' (so stacks can only be built on columns, in decreasing order of card value, and in alternating colors). The goal is reached when the topmost cards of all suits have been put home. See the domain details in Appendix A.8.

There are unrecognized dead ends in *Freecell*, and they can be arbitrarily deep. Consider the following example. We got two suits of cards, one red, one black, both with n cards in them. We denote the red cards with $r_1, \dots, r_n$ and the black cards with $b_1, \dots, b_n$ where the index of a card is its card value. There are two free cells, and two columns. On the left column, we initially have the following stack of cards, from bottom to top: r_{n-2}, r_{n-1}, r_n, b_{n-4}, $b_{n-5}, \dots$, b_1, r_{n-3}, $r_{n-4}, \dots$, r_1. On the right column there initially are the cards $b_{n-3}, b_{n-2}, b_{n-1}, b_n$, from bottom to top. In words, on the right column we got the four highest black cards in increasing order, while on the left column at the bottom we got the three highest red cards in increasing order, then on top of that the $n-4$ lower black cards in decreasing order, then on top of that the $n-3$ lower red cards in decreasing order. One can solve the instance as follows. First, put all the lower red cards from the left column home – they are in the right order as the cards must be put home by increasing card value. Then, do the same with the lower black cards. Afterwards, use the two free cells as an intermediate store for r_n and r_{n-1}, put r_{n-2} home, then put r_{n-1} and r_n home. Finally, use the left column as an intermediate store for b_n, use the two free cells as an intermediate store for b_{n-1} and b_{n-2}, put b_{n-3} home, then put b_{n-2}, b_{n-1}, and b_n home. An unrecognized dead end state arises when, in the initial state, one moves b_n from the top of the right column into a free cell. Let's call this state s. There is no solution to s: b_n can never be put out of the free cell again unless a column becomes empty or b_{n-1} has been put home (there is no card onto which b_n can be stacked). But the single free cell that is left in s does not suffice to get rid of the three top red cards at the bottom of the left column, nor to get rid of the three top black cards at the bottom of the right column. So s is a dead end. The existence of a relaxed plan, however, is obvious: a relaxed plan can use the same free cell multiple times so the single free cell that is empty in s can be used to create the trivial relaxed plan that uses this cell as an intermediate store for all cards that can not be put home straight away. Thus s is an unrecognized dead end. Note that, starting from s, one can still put all the $n - 3$ lower red cards and the $n - 4$ lower black cards home. All states that are thus reachable are unrecognized dead ends, too, for the same reasons as explained above for s itself. Thus, when generalizing over n, s is an arbitrarily deep unrecognized dead end.

10.4 Miconic-ADL

Miconic-ADL is an ADL formulation of a complex elevator control problem occuring in a real-world application of planning [51]. It is a more constrained version of the *Miconic-SIMPLE* domain we have discussed before. A number of passengers are waiting at a number of floors to be transported with a lift. The lift can move up from floor *f* to floor *f'* if *f'* is (transitively) above *f*, and vice versa for moving downwards. The lift can also stop at a floor. When it does so at floor *f*, by conditional effects of the stopping action all passengers waiting at *f* are boarded, and all passengers wanting to get out at *f* depart. The goal is to serve all passengers, i.e., to bring them to their destination floor. The additional constraints are the following.

- In some cases, a passenger p has no access to a floor *f*; the lift can then not stop at *f* while p is boarded.
- Some passengers are VIPs; as long as these are not all served, the lift can only stop at floors where a VIP is getting on or off.
- Some passengers must be transported non-stop, i.e., when they are boarded the lift can make no intermediate stops before stopping at their destination floor.
- Some passengers can not travel alone, others can attend them; if one of the former kind is boarded so must be at least one of the latter kind.
- There are groups A and B of passengers such that it is not allowed to have people from both groups boarded simultaneously.
- Some passengers can only be transported in the direction of their own travel, i.e., if they need to go up (down) then, while they are boarded, the lift can not move downwards (upwards).

All of these constraints are formulated by means of complex first order preconditions of the operators. See the domain details in Appendix A.14. It is fairly easy to see that there can be arbitrarily deep unrecognized dead end states in *Miconic-ADL*. To construct such states, it suffices to consider the first and the last of the constraints listed above. Say we have n floors $f_1, \ldots, f_n$, as well as two passengers p_1 and p_2. The lift is initially at f_n, p_1 wants to go down from f_n to f_2, p_2 wants to go down from f_{n-1} to f_1. The constraints are that p_1 can only be transported in the direction of her own travel, and that p_2 has no access to f_2. We get to an unrecognized dead end state by, in the initial state, stopping, moving the lift down to f_{n-1}, and stopping again. Let's call the resulting state s. In s, both p_1 and p_2 are boarded. There is no solution to s: as long as p_1 is boarded the lift can only move downwards; but as long as p_2 is boarded the lift can not stop at f_2. To let p_1 depart first one needs to stop at f_2 which one can not do; to let p_2 depart first one needs to move down to f_1 and can then not move back up to f_2, with p_1 on board. In a relaxed plan from s, however, one can simply move the lift down to f_2, move it on to f_1 (while, without delete lists, remaining at f_2), then stop at f_1 to let p_2 out, then stop at f_2 to let p_1 out. Note that, in s, one can apply one-step downwards lift moves in a sequence from f_n down to f_3. For the same reasons as outlined above for s, all the state thus reachable remain unrecognized dead ends (in fact, one could add arbitrarily many passengers to be transported downwards somewhere inside $f_n, \ldots, f_3$; all these passengers could be served within a sequence of unrecognized dead end states starting in s).

It is an open question whether and how the topological properties of *Miconic-ADL* change when one restricts oneself to subsets of the full constraint list.

10.5 Assembly

We finally consider the *Assembly* domain, where a complex object must be constructed by assembling its parts — other objects which stand in a *part-of* relation to it — together, obeying certain ordering constraints. The parts themselves might need to be assembled in the same way beforehand. With respect to h^+, the situation in *Assembly* is way more complex than in the domains we have seen before.

I give an example of an unrecognized dead end state which can be arbitrarily deep. Like in the domains we have seen in the preceeding subsections, this would suffice to classify the topology of the *Assembly* domain. It is however an interesting question why, if *Assembly* can have a complex topology in general, FF performs so excellently in the AIPS-1998 instance collection [46] (FF solves even the largest example from that collection by evaluating 441 states in less than a second, finding a plan with 112 steps). So, instead of leaving it at the counter-example, I present a detailed investigation of the topological properties of *Assembly* instances, depending on their part-of relations and ordering constraints. I prove under exactly which circumstances an *Assembly* instance is solvable. I prove a sufficient criterion for the non-existence of dead ends. I finally prove a sufficient criterion for the initial state (amongst other states) having a monotonically decreasing path to the goal under h^+ (to avoid the clumsiness of language, we henceforth also refer to such paths as *full exit paths*). As it will turn out, the results are very relevant for the AIPS-1998 benchmark collection.

The investigation involves a lot of details from the formal definition of the *Assembly* domain, and is probably hard to understand without first having a look at that definition in Appendix A.1. Nevertheless, I give a short summary of the crucial points and of the notation. In any instance, the goal is to have some assembly g completed. The set of all assemblies is denoted by As. Each assembly $y \in As$ has a (possibly empty) set of *parts* $pa(y) \subseteq As$, so that the relations (x, y) where $x \in pa(y)$ form a tree with root g (the tree specifying which assembly "consists" of which other ones). Each assembly $y \in As$ has a set of *transient parts* $tr(y) \subseteq As$, where $pa(y) \cap tr(y) = \emptyset$ and $y \notin tr(y)$, and $pa(y) = \emptyset \Rightarrow tr(y) = \emptyset$ (i.e., the leafs of the tree, the "atomic" assemblies, have no transient parts either). One can assemble x into y if either x is a part of y, $x \in pa(y)$, or x is a transient part of y, $x \in tr(y)$. To execute such an assembling action x must be available, and all assemblies z that have an *assemble order* before x in y, $(z, x) \in ao(y)$, must already have been assembled into y. Upon execution of the assembling action x is incorporated into y, and y is complete and available if no transient part is incorporated into it, and $x \in pa(y)$ is the only part of y that was not yet incorporated. A transient part $x \in tr(y)$ can be removed from y if it is incorporated into y, and all z that have a *remove order* before x in y, $(z, x) \in ro(y)$, are already incorporated into y. In effect, x is available and no longer incorporated into y. If all parts $z \in pa(y)$ are incorporated into y, and x is the last transient part that is removed, then the removing action makes y complete and available. One can also remove a part $x \in pa(y)$ from y, under the condition that it is incorporated, and that all z which have an assemble order before x in y, $(z, x) \in ao(y)$, are *not* incorporated into y (in difference to transient parts, where the assemblies with remove orders must be incorporated for removal). Removing x does *not* delete the completeness or availability of y, in fact there is no operator at all that can delete completeness (which appears to be a domain bug, but after all what we want to look at are the properties of the domain as it is used in the planning community). In the initial state, all assemblies y are available (but not complete) that have no parts, $pa(y) = \emptyset$ (i.e., the leafs of the tree given by the $x \in pa(y)$ relations, the "atomic" assemblies, are initially available). The assemble and remove orders are consistent, i.e., $ao(y)$ respectively $ro(y)$ is cycle-free for all assemblies y. Furthermore, if $(x, z) \in ao(y)$ and $z \in pa(y) \cup tr(y)$ then $x \in pa(y) \cup tr(y)$. Similarly, if $(x, z) \in ro(y)$ and $z \in tr(y)$ then $x \in pa(y) \cup tr(y)$ (we say some words below on the restrictions we make). Assembling x into y or removing x from y can require an arbitrary amount of resources (i.e., machines) to be committed to y. All resources are available in the initial state. A resource r can be committed to (released from) an assembly y simply by applying the commit(r, y) (release(r, y)) action, which exchanges availability of r for its being committed to y (vice versa). There are no additional constraints on the application of these actions.

The set of instances makes (on top of the pa relation being a tree and the obvious constraints on transient parts) the following restrictions. The assemble and remove orders are consistent. If $(x, z) \in ao(y)$ and $z \in pa(y) \cup tr(y)$ (z might have to be incorporated into y) then $x \in pa(y) \cup tr(y)$ (one has the chance to incorporate x into y). Similarly, if $(x, z) \in ro(y)$ and $z \in tr(y)$ then $x \in pa(y) \cup tr(y)$. These restrictions are fulfilled in the AIPS-1998 example collection, with the following exceptions. There are three examples with syntax errors (the initial states use undeclared constants). Furthermore, in two examples there are x, z, and y with

$(x,z) \in ro(y)$, $z \in tr(y)$, and $x \notin pa(y) \cup tr(y)$. As, additionally, in both tasks z must at some point be assembled into y, these two tasks are trivially unsolvable — x must be assembled into y in order to remove z, but cannot be because it is neither a part nor a transient part of y. We assume that these tasks' specifications are flawed. The remaining 25 tasks adhere to the restrictions of the formal definition described above. One could define the set of instances more generously, and include all of the AIPS-1998 examples (except the ones with syntax errors, of course). This would not offer new insights as the restrictions only exclude cases that trivially imply unsolvability if at least one of the affected assembling or removing actions is required for solving the task. Including all the special cases in the subsequent discussion would make some of the claims rather unreadable. I thus settle for the (intuitively reasonable) set of instances specified above.

Two more restrictions that hold in the AIPS-1998 instances are these.

1. Each assembly is only transient part of assemblies at a higher tree level (in the tree given by the $x \in pa(y)$ relations).

2. No transient parts are needed for removing other transients, i.e., if $(x,z) \in ro(y)$ and $z \in tr(y)$ then $x \in pa(y)$.

These restrictions do *not* imply the non-existence of unrecognized dead ends or local minima. We will see below, however, that the first restriction implies solvability of the instance. The second restriction is among the prerequisites of both the sufficient criterion for the non-existence of dead ends and the sufficient criterion for the existence of a full exit path to the initial state.

Assembly is an ADL domain and makes extensive use of complex preconditions and conditional effects. I start the investigation by looking at conditions for solvability, then I focus on dead ends, finally I focus on local minima.

10.5.1 Solvability

The following definition will be important throughout the subsequent investigation. It specifies, for a reachable state in an *Assembly* task, exactly which assemblies must still be assembled into which other ones.

Definition 19 *Given an instance (C,I,G) of the Assembly domain, and a reachable state $s \in S$. We define*

$$
\begin{aligned}
NA_0(s) := \ & \{(x,y) \mid y \in As, (complete,y) \notin s, x \in pa(y), (incorporated,x,y) \notin s\} \cup \\
& \{(x,y) \mid y \in As, (complete,y) \notin s, x \in tr(y), (incorporated,x,y) \notin s, \\
& \qquad \exists z : z \in tr(y), (x,z) \in ro(y), (incorporated,z,y) \in s\} \\
NA_{i+1}(s) := \ & \{(x,y) \mid y \in As, (complete,y) \notin s, x \in tr(y), (incorporated,x,y) \notin s, \\
& \qquad \exists z : (z,y) \in NA_i(s), (x,z) \in ao(y)\} \cup \\
& \{(x,y) \mid y \in As, (complete,y) \notin s, x \in tr(y), (incorporated,x,y) \notin s, \\
& \qquad \exists z : (z,y) \in NA_i(s), z \in tr(y), (x,z) \in ro(y)\} \\
NA(s) := \ & \bigcup_{i=0}^{\infty} NA_i(s)
\end{aligned}
$$

We call $NAG(s) := (As, NA(s))$ *the* needed assembles graph to s*, and* $NAG(I)$ *the* needed assembles graph to the instance*.*

$NA_0(s)$ captures all pairs (x, y) where x must still be assembled into y because it is either a part of y, or a transient part that must be incorporated in order to remove some other transient part. $NA_{i+1}(s)$ captures all pairs (x, y) where x must still be assembled into y because either it is needed due to an assemble order before some other needed part or transient part, or due to a remove order before some needed transient part.

In the rest of the investigation, a *leaf* in a graph will denote a node that has no ingoing edges, and a *relevant leaf* will denote a leaf that has at least one outgoing edge. The following is a sufficient condition for a state s to be no dead end.

Lemma 7 *Given an instance* (C, I, G) *of the Assembly domain, with action set* A*. Let* $s \in S$ *be a reachable state such that* $NAG(s)$ *is cycle-free. If, for all relevant leafs* x *in* $NAG(s)$*,* $(available, x) \in s$*, then* (A, s, G) *is solvable.*

Proof: If $(\text{complete}, g) \in s$, we have nothing to prove. Else, as $NAG(s)$ is cycle-free, there are either no relevant leafs at all, or there is at least one non-leaf node y such that all sons of y are leafs. Assume there are no relevant leafs at all. It follows that all parts, and transient parts needed for removing other transient parts, are incorporated into g already (because there are no ingoing edges to g), so one can remove the transient parts in an order obeying $ro(g)$ (note that $ro(g)$ is cycle-free by definition), after committing (and beforehand releasing, if necessary) the resources required by g.

In the other case, select a node y that is not a leaf, but for which all sons are leafs. Let us focus on y's ingoing edges $(x, y) \in NA(s)$. All x are available by prerequisite. Also, from the definition of $NA(s)$, the set of those x is all we need for assembling y: all parts of y, all transient parts needed for assembling that parts, all transient parts needed to remove other transient parts again, and all transient parts needed for assembling those, have an edge to y if they are not yet assembled. Assume we commit (and beforehand release) all resources required by y, and carry out the respective sequence of $\text{assemble}(x, y)$ actions (which can be applied in some order because $ao(y)$ is cycle-free by definition). We can then remove all transient parts from y by a $\text{remove}(x, y)$ sequence (which can be applied because all parts with a remove order have been assembled). This gets us to a state s' where $(\text{complete}, y) \in s'$. Afterwards, we can also remove all parts of y again in a bottom-to-top order regarding $ao(y)$; remember that this does not delete $(\text{complete}, y)$. We then have a state s'' where all assemblies that have been available in s are again available, and where there is one more complete assembly, namely y. The new assembles graph $NAG(s'')$ differs from $NAG(s)$ only in the sense that all edges (x, y) have been removed. The new graph is therefore again cycle-free; the only node that might be a new leaf is y, and $(\text{available}, y) \in s''$. We can thus iterate the argument and are finished with the monotonicity of completeness. ∎

The opposite implication holds in the following sense.

Lemma 8 *Given an instance* (C, I, G) *of the Assembly domain, with action set* A*. Let* $s \in S$ *be a reachable state. If* $NAG(s)$ *contains a cycle, then* (A, s, G) *is unsolvable.*

Proof: With $(x, y) \in NA(s)$, we know that y is not complete in s; this either means that y is not available (both properties are achieved together, and completeness is never deleted), or that y was already available initially; the latter can not be as otherwise there would be no edge $(x, y) \in NA(s)$ (as then $pa(y) = tr(y) = \emptyset$). Further, all such assemblies y must be made complete (and with that available) at some point because the pa relation is a tree with root g that includes all non-atomic assemblies. Finally, $(x, y) \in NA(s)$ implies that, in any solution plan from s, x must be available at some point before y can be made available. Putting all of this together, a cycle in $NAG(s)$ gives us a sequence $y_0, \dots, y_{n-1}, y_n = y_0$ of assemblies such that y_i must in any solution from s be made available before y_{i+1} is made available, which is a contradiction. ■

With Lemmata 7 and 8, we get an exact classification of solvability of the *initial state.*

Theorem 28 *Given an instance* (C, I, G) *of the Assembly domain. The instance is solvable if and only if the needed assembles graph* $NAG(I)$ *to the instance is cycle-free.*

Proof: Consequence of Lemmata 7 and 8 and the fact that initially the leafs of the needed assemble graph are exactly the atomic assemblies x for which $pa(x) = \emptyset$. Those are available. ■

Note that, if each assembly is transient part only of assemblies at a higher level in the tree given by the $x \in pa(y)$ relations, then the needed assembles graph to the instance is cycle free (because $(x, y) \in NAG(I)$ implies that either $x \in pa(y)$ or $x \in tr(y)$). Thus by Theorem 28 all AIPS-1998 benchmark examples are solvable, except the flawed ones.

10.5.2 Some Observations Concerning Dead Ends

We now focus on the role of dead ends in the *Assembly* domain. Have a look at Figure 1, where the part-of relations are depicted by dotted arrows, the transient part relations are depicted by dashed arrows, and the ordering relations are full arrows; for ordering relations, the kind of ordering and the respective higher level assembly are written at the left hand side of the relation, i.e., the beginning of the arrow.

The situation s where G is incorporated into D and I is incorporated into C is an unrecognized dead end: either G or I must be removed first; say one tries G. This requires assembling E into D, in order to assemble H. But once E is assembled into D, one must assemble I to get it out again; this is, however, impossible, as I is at that point still assembled into C, and needs E to be assembled in the first place for being removed. A symmetrical situation arises when one tries

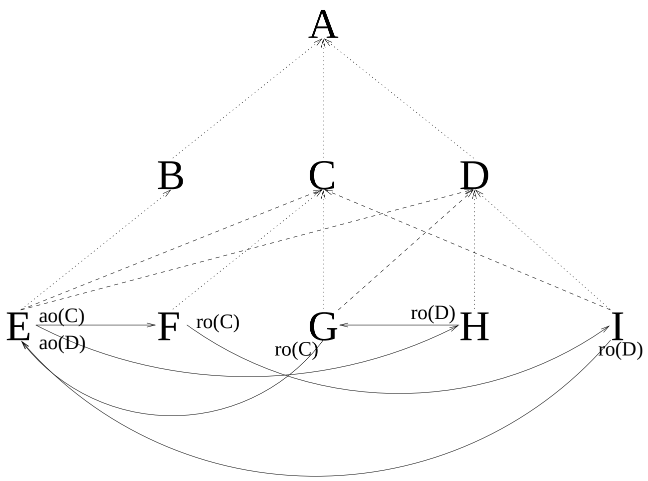

Figure 1: An *Assembly* instance where an unrecognized dead end arises in the situation where G is incorporated into D, and I is incorporated into C.

to remove I from C first, starting from s. On the other hand, a relaxed plan for (A, s, G) can be constructed by assembling E into both C and D (the availability of E remains undeleted), then assembling F and H, then removing G and I, and so on. Note that one can easily make s arbitrarily deep by adding additional parts and sub-parts of A that must be assembled completely independently of $B, \ldots, I$.

The instance depicted in Figure 1 adheres to all restrictions that apply in the AIPS-1998 benchmark suit, so one wonders why the AIPS-1998 examples are so unproblematic for FF. I will prove below that unless an instance contains a quite complex structure of remove order constraints there are no dead ends at all. All but one of the unflawed AIPS-1998 examples do not feature this complex structure.

The following definition formalizes the situation where some incorporated assembly needs other assemblies to be incorporated in order to be removed — this plays a key role in states where some parts that are needed elsewhere are "stuck" somewhere.

Definition 20 *Given an instance (C, I, G) of the Assembly domain, and a reachable state $s \in S$. For assemblies x such that $(incorporated, x, y) \in s$ for some y, we define the* needed removers $NR(x, s)$ for x in s *as:*

1. *if $x \in tr(y)$: those $z \in As$, $(incorporated, z, y) \notin s$ with $(z, x) \in ro(y)$, and*

2. *if $x \in pa(y)$: those $z \in As$, $(incorporated, z, y) \notin s$ such that there is some $z' \in tr(y), (z, z') \in ro(y)$ so that ex. $z' = z_0, \ldots, z_n = x$ with $(z_i, z_{i+1}) \in ao(y), (incorporated, z_i, y) \in s$ for $i \geq 0$ and $z_i \in pa(y)$ for $i \geq 1$.*

The first case, that of a transient part of y to be removed, is obvious. In the second case, a part x of y to be removed, all z_i must be removed before one can remove x, and z must be incorporated in order to remove z_0 (remember that all parts with an assemble order before x must not be incorporated when removing x). As we show now, if the needed removers for a number of assemblies that are incorporated into an assembly y are all empty, then one can remove all these assemblies.

Lemma 9 *Given an instance (C, I, G) of the Assembly domain, and a reachable state $s \in S$. If $x_1, \ldots x_n$ are assemblies such that, for some y, $(incorporated, x_i, y) \in s$ for $1 \leq i \leq n$, and $NR(x_i, s) = \emptyset$ for all i, then there is an action sequence that, when applied to s, yields a state s' where $(available, x_i) \in s'$ for all i.*

Proof: First of all, one can commit (and beforehand, if necessary, release) the resources required by y. Then collect together as the relevant transients all $x_i \in tr(y)$, and all $z \in tr(y)$ that are transitively ordered before some $x_i \in pa(y)$ regarding $ao(y)$. As $NR(x_i, s) = \emptyset$, all parts that have a remove order before a relevant transient are incorporated into y. Thus, one can apply a sequence of actions removing all relevant transients, in some order respecting $ro(y)$ (this is possible because $ro(y)$ is cycle-free by definition). Afterwards, all relevant parts — the $x_i \in pa(y)$ together with all $z \in pa(y)$ that are transitively ordered before some $x_i \in pa(y)$ regarding $ao(y)$ — can be removed in some order respecting inverse $ao(y)$ (which is, again, cycle-free by definition). ■

If a part x is incorporated in y', but needed in y, and can not be removed (and thus made available) easily, then there is a constraint between y' and y in the sense that y' must be disassembled before one assemble y. The following definition formalizes that situation.

Definition 21 *Given an instance (C, I, G) of the Assembly domain, and a reachable state $s \in S$. The* remove conflicts $RC(s)$ of *s are defined as*

$$RC(s) := \{(y', y) \mid y \in As, (complete, y) \notin s, \\ \exists\, x : (x, y) \in NA(s), (incorporated, x, y') \in s, NR(x, s) \neq \emptyset\}$$

The next is the core lemma of this subsection. We will later identify a situation — occurring, e.g., in all but the flawed AIPS-1998 examples — where the seemingly very strong prerequisite of the lemma is fulfilled. The proof is an extension of the proof to Lemma 7.

Lemma 10 *Given an instance (C, I, G) of the Assembly domain, with action set A. If, for all reachable states $s \in S$, the graph $(As, NA(s) \cup RC(s))$ is cycle-free, then (A, s, G) is solvable for all such s, i.e., the instance is dead-end free.*

Proof: If all $a \in As$ have $(\text{complete}, a) \in s$, then we are finished. If all nodes in $(As, NA(s) \cup RC(s))$ are leafs, then in particular the goal assembly g has no ingoing edges in $NA(s)$, so we can make it complete by committing the required resources, and removing all transient parts.

In the non-trivial case, let y be a node in $(As, NA(s) \cup RC(s))$ that is not a leaf, but for which all sons are leafs. In particular, $(\text{complete}, y) \notin s$. We construct an action sequence leading to a state s' where $(\text{complete}, y) \in s'$. Iterating the argument proves the lemma with the monotonicity of completeness. The construction proceeds as follows.

1. Process all sons y', $(y', y) \in RC(s)$: let, for such a y', $x \in pa(y) \cup tr(y), (x, y) \in NA(s)$ be incorporated into y'. If y' was not complete, then $NR(x, s) \neq \emptyset$ would imply an ingoing edge to y' in $NA(s)$ (obvious if $x \in tr(y')$; if $x \in pa(y')$, there is an incorporated transient part z' for which a remover z must be incorporated). Thus, y' is complete in s, and thus, all $z \in pa(y') \cup tr(y')$ are complete in s or initially available. As y' has no ingoing edges in $RC(s)$, all such z that are incorporated into some y'' have $NR(z, s) = \emptyset$. We can thus apply the following action sequence:
   (a) remove, using Lemma 9, all $z \in pa(y') \cup tr(y')$ from those y'' were they are incorporated (if they are);
   (b) commit the resources required by y', and assemble all $z \in pa(y') \cup tr(y')$ into y' in some order respecting $ao(y')$;
   (c) completely disassemble y', by first removing all transients in an order respecting $ro(y')$, then removing all parts in an order respecting $ao(y')$ (note that this does not delete the completeness of y').
2. Process all sons x, $(x, y) \in NA(s), (\text{complete}, x) \notin s, pa(x) \neq \emptyset$: as such x have no ingoing edges in $NA(s)$ — remember that they are leafs in $(As, NA(s) \cup RC(s))$ — all parts $z \in pa(x)$ are incorporated, and all parts needed for removing transients are incorporated, too. So x can be made complete by committing the resources, and removing all transients in an order respecting $ro(x)$.
3. Process all sons x, $(x, y) \in NA(s), (\text{incorporated}, x, y') \in s$, and there is no remove conflict between y' and y, i.e., $NR(x, s) = \emptyset$: with Lemma 9, all those x can be removed from the assemblies y'.

After the above steps have been carried out, it holds that all x with $(x, y) \in NA(s)$ are available. If, in s, they have been incorporated into some y' with $NR(x, s) \neq \emptyset$, then they have been removed from y' in the first process. If they have not been complete, they have been made so, and available, in process two. If they have been incorporated into some y' with $NR(x, s) = \emptyset$, then they have been removed from y' in the third process. We can therefore get to our desired state s' by committing the resources required by y, assembling all x with $(x, y) \in NA(s)$ into y in an order respecting $ao(y)$, and afterwards removing the transient parts in an order respecting $ro(y)$, making y complete. ∎

We now identify cases where Lemma 10 can be applied. Our first observation is this. If the remove orders behave in the manner present in the unflawed AIPS-1998 benchmarks, then the needed assembles graph is monotonic in the following sense.

Lemma 11 *Given an instance* (C, I, G) *of the Assembly domain, such that* $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$. *Then, for all reachable states* $s \in S$, $NA(s) \subseteq NA(I)$ *holds.*

Proof: With $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$, the role of incorporated transients in a state is diminished to the effect that all removers are parts and must therefore be incorporated anyway. The lemma then follows because in the initial state, both the set of parts that are not complete is maximal (all parts), and the set of pairs (x, y) where x is not incorporated into y is maximal (no part is incorporated into any other part). The other constituents of Definition 19 are static (the pa, tr, ao and ro functions). The proof proceeds by induction.

$NA_0(s) \subseteq NA_0(I)$ follows from the above, with the observation that the second set in the definition of $NA_0(s)$ always yields the empty set because of our prerequisite $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$: for a pair (x, y) in the second set, $x \in tr(y)$, $z \in tr(y)$ and $(x, z) \in ro(y)$ hold for some z, in contradiction. The induction step from $NA_i \longrightarrow NA_{i+1}$ is straightforward: in addition to I's maximality properties, we know from the induction hypothesis that each $(z, y) \in NA_i(s)$ is also contained in $NA_i(I)$. This gives us $NA_{i+1}(s) \subseteq NA_{i+1}(I)$, and consequently $NA(s) \subseteq NA(I)$. ■

Our second observation is that a remove conflict between y' and y implies the presence of the respective ordering constraints between y' and y in the instance, i.e., there must be a potential remove conflict between those two in the following sense.

Definition 22 *Given an instance* (C, I, G) *of the Assembly domain. The* potential remove conflicts pRC of the instance *are defined as*

$$\begin{aligned} pRC := \; & \{(y', y) \mid y \neq y', \exists x \in (pa(y) \cup tr(y)) \cap tr(y') : \exists z : (z, x) \in ro(y')\} \cup \\ & \{(y', y) \mid \exists\; x \in tr(y) \cap pa(y') : \\ & \qquad \exists\; z' \in tr(y'), z_1, \ldots, z_n \in pa(y') : \\ & \qquad\quad (z', z_1) \in ao(y'), (z_i, z_{i+1}) \in ao(y') \textit{ for all } i, (z_n, x) \in ao(y'), \\ & \qquad\quad \exists z : (z, z') \in ro(y')\} \end{aligned}$$

Reconsider Definitions 20 and 21. Imagine a remove conflict that is due to a transient part $x \in tr(y')$ that is incorporated into y' and needs some z to be incorporated for its removal. Such a conflict demands the existence of a potential remove conflict according to the first set in Definition 22 (remember that $(x, y) \in NA(s)$ for a state s implies $x \in pa(y) \cup tr(y)$). Slightly more complicated, say the conflict is due to a part $x \in pa(y')$ that is incorporated into y' such that removing x first involves removing all of $z_1, \ldots, z_n \in pa(y')$ (due to assemble orders), but removing z_1 requires incorporating z' (due to a remove order). That situation demands the existence of a potential remove conflict according to the second set in Definition 22. So in any state the remove conflicts are a subset of the potential remove conflicts of the instance, i.e., $RC(s) \subseteq pRC$ holds for all $s \in S$.

The set in the first line of the above definition has not many members if remove orders are seldom. Even more so for the second set, where there must be a complex construct of assemble orders in addition to the remove order. We will now see that, if the potential remove conflicts are cycle-free together with the needed assembles of an instance adhering to the restriction that $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$, then the instance is dead-end free.

Theorem 29 *Given an instance (C, I, G) of the Assembly domain, with action set A. If $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$, and $(As, NA(I) \cup pRC)$ is cycle-free, then (A, I, G) is dead-end free.*

Proof: With Lemma 10, it suffices to show that $(As, NA(s) \cup RC(s))$ is cycle-free for all $s \in S$. With Lemma 11 and $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$, we have $NA(s) \subseteq NA(I)$ for all states $s \in S$. Furthermore, we got $RC(s) \subseteq pRC$. Finished with the prerequisite that $(As, NA(I) \cup pRC)$ is cycle-free. ■

As an example, reconsider Figure 1. There is a cycle in (As, pRC): we have a potential remove conflict between C and D because $I \in pa(D)$, $I \in tr(C)$, and $(F, I) \in ro(C)$. The other way round, we have a potential remove conflict between D and C because $G \in pa(C)$, $G \in tr(D)$, and $(H, G) \in ro(D)$.

Theorem 29 proves that there are no dead ends in 24 of the 25 unflawed AIPS-1998 benchmark examples. All the unflawed examples fulfill $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$, and in all but one of them $(As, NA(I) \cup pRC)$ is cycle-free.

10.5.3 Some Observations Concerning Local Minima

There is more to be said, except their existence (which follows from the existence of unrecognized dead ends), about local minima in the *Assembly* domain. First, let us state that there can be local minima under h^+ even if there are no unrecognized dead ends. An example, adhering to the restrictions in the AIPS-1998 suit, is depicted in Figure 2. The meaning of the arrows is the same as in Figure 1.

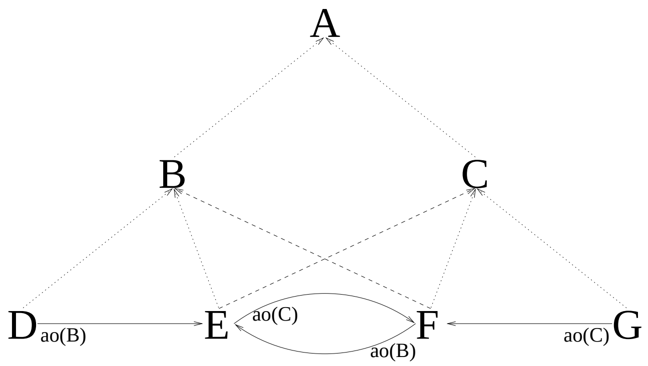

Figure 2: An *Assembly* instance where a local minimum arises when D is incorporated into B, and G is incorporated into C.

The phenomenon here is the following. With D and G incorporated, $h^+(s) = 6$: an optimal relaxed plan is to assemble E into C, assemble F into C, assemble F into B, assemble E into B, and assemble B and C into A. Note here that we must assemble the transient part E into C (F into B) before we can assemble F into C (E into B), but — in the relaxed plan — do not need to remove E from C (F from B) again. One must either first assemble E into C or F into B (the two other applicable actions are removing D from B or G from C but that does open up no new options except assembling them back). Say we decide to assemble E into C. In the resulting state s', we get, again, $h^+(s') = 6$ (like above, but remove E from C instead of assembling it). Assembling F into C afterwards, however, worsens the situation to $h^+(s'') = 7$ (where s'' is the respective resulting state): in addition to the above steps, one must remove G from C due to its assemble order before F. In more detail, an optimal relaxed plan from s'' is to remove E from C, remove G from C, remove F from C (which can not be done if G is incorporated into C, as $(G, F) \in ao(C)$), assemble F into B, assemble E into B (which can not be done unless F is incorporated into B), and assemble B and C into A. A symmetrical situation arises if we decide to go the other way round, i.e., to assemble F into B first.

Once again, one wonders why the AIPS-1998 examples are so unproblematic for FF. We now make an interesting observation concerning the behavior of local minima under h^+ in *Assembly* instances, and the ease of solving such instances using that heuristic. Unless there are complex structures concerning assemble orders, the *initial state* of an *Assembly* instance has a full exit path.

In the instance from Figure 2, the initial state does not have a full exit path. This is because there is a circle between B and C in the sense that they are conflicted (B, C) and (C, B) as follows.

Definition 23 *Given an instance* (C, I, G) *of the* Assembly *domain, and a reachable state* $s \in S$*. The* assemble conflicts $AC(s)$ *of* s *are defined as*

$$\begin{aligned} AC(s) := \{(y', y) \mid y \neq y', \exists x, z \in pa(y) : \; & (x, y) \in NA(s), \\ & (z, x) \in ao(y), \\ & (x, y') \in NA(s)\} \end{aligned}$$

$AC(I)$ *are the* assemble conflicts of the instance.

The difficulty with an assemble conflict between y' and y with regards to an assembly x is that x is needed for both y' and y, but once x is incorporated into y we will have trouble removing it because there is another part z with an assemble order before it. So one should care about y' first. This is not always possible in the example Figure 2 because that kind of conflict appears between B and C in both directions. If one can always care about y' first then states where all relevant leafs (in the needed assembles graph) are available have a monotonically decreasing path to the goal, under the additional condition that no needed assemble constraint is due only to a remove order.

Lemma 12 *Given an instance (C, I, G) of the Assembly domain with action set A, and a reachable state $s \in S$, such that for all relevant leafs x in $NAG(s)$, $(available, x) \in s$. If it holds that*

1. *$(As, NA(s) \cup AC(s))$ is cycle-free, and*
2. *for all $(x, y) \in NA(s)$, there is a sequence $x = x_0, x_1, \ldots, x_n$ such that $x_n \in pa(y)$, $(x_{i-1}, x_i) \in ao(y)$ and $(x_i, y) \in NA(s)$ for $1 \leq i \leq n$,*

then s has a monotonically decreasing path to the goal under h^+.

Proof: We prove that h^+ decreases monotonically on a solution path that can be obtained by Lemma 7 ($NAG(s)$ is cycle-free by the first prerequisite). This iteratively processes an arbitrary non-leaf assembly y where all sons x are leafs in $NAG(s)$; if all assemblies are leafs in $NAG(s)$, achieving the goal can be done by committing (and beforehand releasing) the resources required by the goal assembly g, and then removing the transient parts from g, all of which actions are contained in an optimal relaxed solution, and delete only propositions that are not needed anymore. We change the algorithm from Lemma 7 to iteratively process nodes y where all sons are leafs in $(As, NA(s) \cup AC(s))$ (i.e., we additionally consider the edges in $AC(s)$). If a node has no ingoing edges in $NA(s)$, then it has no outgoing edges in $AC(s)$ ($(y', y) \in AC(s)$ implies, by definition, $(x, y') \in NA(s)$ for some x), so the only ingoing edges (x, y) of y are such that $(x, y) \in NA(s)$. The solution algorithm first releases and commits all resources required by y, then assembles all x with $(x, y) \in NA(s)$ into y in an appropriate order, and afterwards removes all transient parts and parts of y in an appropriate order. The process ends up in a state where prerequisites 1 and 2 are again fulfilled; for prerequisite 2, this is obvious (all edges (x, y) are removed from $NA(s)$, and the edges (x, y') for other assemblies y' remain unchanged), and for prerequisite 1 it holds because no new edges are introduced into $NA(s)$, which implies that no new edges are introduced into $AC(s)$. So let us consider the optimal relaxed solution lengths on the constructed path. Removing an assembly x from y only deletes $(\text{incorporated}, x, y)$, which is not needed anymore once $(\text{complete}, y)$ has been achieved; it is also not needed anymore when all parts of y have been incorporated and x is a transient which is not needed for removing any other transient, both of which conditions are fulfilled when the algorithm removes a transient part x from y. As y is not complete, the required resources must be released and committed to y at some point in an optimal relaxed solution from s; as releasing and committing actions are invertible, those actions do therefore not increase optimal relaxed solution length. Thus, the only actions left to take care of are the $\text{assemble}(x, y)$ parts of the process, transforming a state s' into another state s''. It suffices to show that there is a relaxed solution to s'' that is at most as long as an optimal relaxed solution to s'. Assume $P^+_{s'}$ is an optimal relaxed solution to (A, s', G). x may be a part, or a transient part of y.

In the first case, $\text{assemble}(x, y)$ is contained in $P^+_{s'}$ simply because x must be incorporated into y at some point. If there is no $z \in pa(y)$, $(z, x) \in ao(y)$, then a relaxed plan $P^+_{s''}$ for (A, s'', G) can be constructed by removing $\text{assemble}(x, y)$

from $P^+_{s'}$, and placing remove(x, y) up front. If there is such a z, then, because y has no ingoing edges from $AC(s)$, it follows that there is no edge (x, y') in $NA(s)$ (otherwise, $(y', y) \in AC(s)$ would hold), so (available, x) is not needed anymore; neither do we need (not-incorporated, x, y) when x is a part (not a transient part) of y, so we can construct $P^+_{s''}$ by removing assemble(x, y) from $P^+_{s'}$ and are finished.

In the second case, x being a transient part of y, assemble(x, y) is contained in $P^+_{s'}$ because by prerequisite 2 x must be assembled into y at some point for the integration of all parts, due to assemble orders (none of $x_1, \dots, x_n$ can already be integrated into y in s', due to the assemble orders and $(x_i, y) \in NA(s)$ for all i). So our first step in the construction of a relaxed plan $P^+_{s''}$ for (A, s'', G) is to remove assemble(x, y) from $P^+_{s'}$. It then suffices to show that remove(x, y) can be applied at some point in the resulting relaxed plan before (available, x) or (not-incorporated, x, y) are needed. This follows again from prerequisite 2: remove(x, y) can be applied in $P^+_{s''}$ as soon as all z with $(z, x) \in ro(y)$ have been incorporated into y. With prerequisite 2, all z that have this property must either already be incorporated, or assembled into y at some point. All these actions — and the actions assembling their necessary (transient) parts $(z', z) \in ao(y)$ — can be moved to the front of $P^+_{s''}$ in the appropriate order. We insert remove(x, y) directly after those actions. This is early enough: obviously, none of the assemble(z, y) actions needs (available, x) as prerequisite; for (not-incorporated, x, y), matters are a bit more complicated, as the last of the assemble(z, y) actions might have been responsible for adding (complete, y); but in that case, all parts of y are incorporated at that point, and all transient parts except x have not been incorporated in s', so remove(x, y) achieves (complete, y) at that point of $P^+_{s''}$ starting from s''. Our construction $P^+_{s''}$ is therefore a relaxed plan for (A, s'', G), which concludes the argument. ∎

Let us identify situations where the prerequisites of Lemma 12 are fulfilled. First observe that with $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$ the assemble conflicts are monotonic, simply because the needed assembles are.

Lemma 13 *Given an instance (C, I, G) of the Assembly domain, such that $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$. Then, for all reachable states $s \in S$, $AC(s) \subseteq AC(I)$ holds.*

Proof: Follows directly from Lemma 11 and the definition of assemble conflicts. ∎

It is easy to see that the same restriction on remove orders implies the second prerequisite of Lemma 12 (no needed assembles due only to remove orders). So we know that, under the restriction, the initial state has a full exit path unless there is a cycle in the assemble conflicts with the needed assembles.

Theorem 30 *Given an instance (C, I, G) of the Assembly domain, such that*

1. *$(As, NA(I) \cup AC(I))$ is cycle-free, and*
2. *$\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$.*

Then, any reachable state $s \in S$ where all relevant leafs in $NAG(s)$ are available, in particular the initial state, has a monotonically decreasing path to the goal under h^+.

Proof: The prerequisites 1 and 2 are sufficient for prerequisites 1 and 2 of Lemma 12, respectively. As for prerequisite 2, with $\forall y \in As : (x, z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y)$, the only edges (x, y) in $NA(s)$ inserted due to a remove order are such that $x \in pa(y)$, implying that x must be incorporated anyway (in the terminology of the second prerequisite of Lemma 12, the trivial sequence $x = x_0 = x_n \in pa(y)$ suffices). The other implication follows directly from Lemmata 11 and 13: as there is no cycle in $(As, NA(I) \cup AC(I))$, and for all states s we got $NA(s) \subseteq NA(I)$ and $AC(s) \subseteq AC(I)$), there can not be a cycle in $(As, NA(s) \cup AC(s))$ for any s. ∎

Theorem 30 proves that the initial states of all unflawed AIPS-1998 instances have full exit paths under h^+: all these instances fulfill the theorem's prerequisites. Assuming that FF's search mechanism does stick to the full exit paths, this result explains, at least partly, the excellent runtime performance of the system in the AIPS-1998 collection.

11 A Planning Domain Taxonomy

My results divide the current planning benchmark domains into a taxonomy of classes which differ in terms of the topological behavior with respect to h^+. The overall picture is given in Figure 3.

The taxonomy, as it is drawn in Figure 3, has two dimensions: the x-axis, corresponding to the four dead end classes, and the y-axis, corresponding to the existence or non-existence of constant upper bounds on the maximal local minimum exit distance and on the maximal bench exit distance (where the domains with bounded bench exit distance are assumed to be a subset of the ones with bounded local minimum exit distance). The bottom right corner is crossed out because no domain can belong to the respective classes: by Proposition 1, the existence of unrecognized dead ends implies the non-existence of constant upper bounds on the maximal local minimum exit distance. Arranging both maximal exit distance properties together on the y-axis in the way Figure 3 does it is not generally justified. The domains with bounds on the maximal bench exit distance are in general not a subset of those with bounds on the maximal local minimum exit distance. However, as was said before, the question whether there is a bound on the difficulty of escaping benches does not seem as relevant when, anyway, it can be arbitrarily difficult to escape local minima.[7] The specific bounds proved for the individual domains are given in parentheses, local minimum exit distance bound preceeding bench exit distance bound in the cases where there are both. In

[7]Similarly, when benches can be arbitrarily large it is not as relevant if or if not the local minima are small or non-existent. In that sense the respective results for *Briefcaseworld*, *Fridge*, *Grid*, *Blocksworld-no-arm*, and *Hanoi* are only moderately important. I have included them as I find them intriguing anyway.

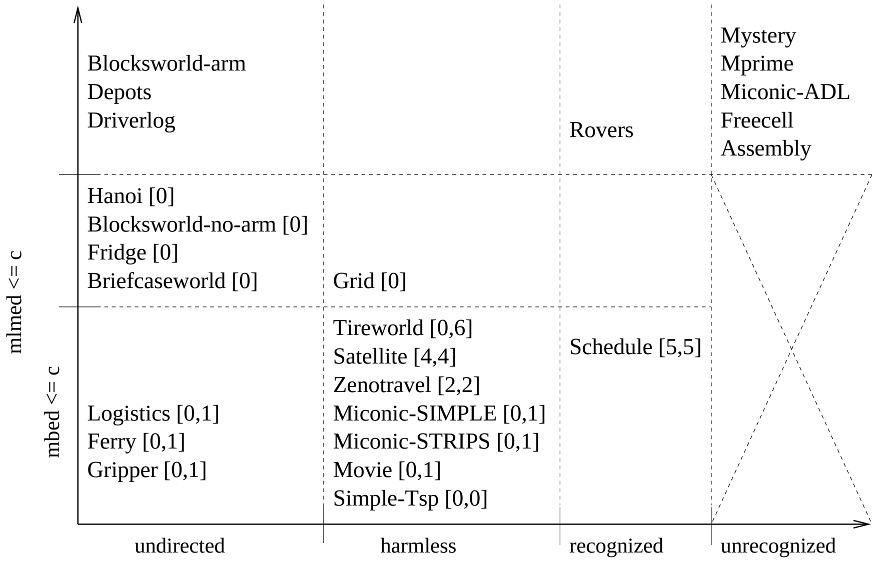

Figure 3: A planning domain taxonomy, overviewing the results.

most domains where there is an upper bound on the local minimum exit distance, the bound is 0, i.e., there are no local minima at all. In the domains where the latter is not the case, namely in *Zenotravel*, *Satellite*, and *Schedule*, it might be that the bench exit distance bound is lower than the local minimum exit distance bound. Again, whether this is the case or not does not seem particularly relevant.

From the point of view of relaxed plan based heuristic planning, Figure 3 provides a very appealing picture: put in colloquial terms, the "simple" domains are in the left bottom corner, while the "demanding" domains are in the top right corner. Obviously, the current planning benchmark domains tend to lie on the left bottom side, where there are no unrecognized dead ends, and local minima and/or benches can always be escaped from within a constant number of steps. Some more on the relevance of the taxonomy for the performance of state-of-the-art heuristic planners will be said below, in Section 12. Here, let me summarize what the most common patterns of problem structure are that *cause* the quality of h^+ in the benchmarks. At the level of the planning tasks' specifications, the most common pattern of structure causing the non-existence of dead ends is that all actions are (at least) invertible or need not be inverted (like when their adds are static and their deletes are irrelevant). Together with that, the most common reason for the non-existence of local minima is that the actions are respected by the relaxation, i.e., often, when an action starts an optimal plan, then there is simply no other way to achieve the goal than by applying it. As for bounds on the bench exit distance, these are present in domains without local minima, when one kind of actions have delete effects that are not needed anymore once the actions have been applied in an optimal plan, while the other kind of actions need not be applied more than a constant number of times in a row. Finally, we have seen that in 3 of the domains where there are local minima (*Zenotravel*, *Satellite*, and *Schedule*), both exit distances are bounded because important facts can always be achieved, without losing other important facts, within a constant number of steps (this being a property that also holds true for all the 8 other domains in the lowermost classes of the taxonomy).

12 Discussion

I have proven that most of the current planning benchmarks are of a structure that causes h^+ to be a very informative heuristic function. The question is, how relevant is this result in practice, where state-of-the-art heuristic planners use various approximations of relaxed planning in various search schemes (not necessarily forward state space search)? How much does the performance of the state-of-the-art heuristic planners really depend on the quality of h^+? The most specific connection is that between the domains in the lowermost classes of the taxonomy, and FF's search algorithm. As said before, FF performs an iteration of breadth-first searches for an exit state, and if the distance to such an exit is always bounded by a constant then search terminates in polynomial time. In addition, in an empirical investigation that is part of the author's PhD thesis [44], it turned out that the quality of h^+ gets largely preserved by FF's approximation of it. When taking random samples of example state spaces, there were rarely any samples on local minima in domains where there are none under h^+, and the maximal bench exit distance of the samples was identical to that under h^+. Indeed, when one compares the positions of the domains in the taxonomy with FF's runtime behavior in the respective example suits, then one finds that these match quite well. FF excells in all the 11 domains that belong to the lowermost classes of the taxonomy. In the 5 "middle" domains (no local minima but potentially large benches) FF performs well, but does not scale up as comfortably as in the 11 easier domains. As for the more complex domains: *Driverlog*, *Depots*, and *Blocksworld-arm* are amongst the most challenging domains for FF. In *Mystery* and *Mprime* FF performs just as bad as most other planners. In *Miconic-ADL* and *Freecell* FF is among the top performing planners, but often runs into unrecognized dead ends in the larger instances (e.g., the larger *Freecell* instances used at AIPS-2000). In *Rovers* and *Assembly*, FF excells in the respective competition example suits; however, just because a domain *potentially* contains complex search space topology that does not mean a specific example suit does. Indeed, we have seen that in the AIPS-1998 *Assembly* instances there are no dead ends, and the initial states have full exit paths under h^+. Also, we have seen that the road map diameter of a rovers instance yields an upper bound on the exit distance under h^+; the road map diameter in the AIPS-2002 instances varies around 1 to 6.

Let us focus on the other heuristic planners. Those that are state-of-the-art in runtime all use relaxed plan heuristics in one way or the other. There are also planners based on other heuristics [34, 35], but these – optimal – planners can not compete with the runtime performance of their sub-optimal relaxed plan based counterparts. Except FF, the most successful heuristic planners in the last 3 years (specifically in the AIPS-2000 and AIPS-2002 competitions) have been HSP2 [10], STAN4 [24], Mips [20], RePOP [57], SAPA [16], VHPOP [67], and LPG [29]. Of these, HSP2 (in the configuration used at AIPS-2000), STAN4, Mips, and SAPA search forward in the state space, while the others do variations of partial order search (in LPG, based on a dynamic planning graph structure). HSP2, STAN4, and Mips approximate h^+ to obtain heuristic information (in the case of STAN4, the heuristic also takes automatically inferred domain knowledge into account). At AIPS-2000 [2] the domains where these three planners performed well coincided pretty much with those were FF did, with the exception of *Schedule* (where FF's heuristic pruning technique makes a huge difference). In particular, the easiest domains for these planners were *Logistics*, *Miconic-STRIPS*, and *Miconic-SIMPLE* – those where h^+ has the highest quality. For RePOP (which adapts relaxed plan heuristics to the partial-order framework) and SAPA (which extends relaxed plan based heuristic planning to a temporal numerical context), the best results reported [57, 16] are in *Logistics*-type transportation domains. VHPOP and LPG have been awarded at AIPS-2002 (as best newcomer respectively for excellent performance across a wide range of example domains). VHPOP is a heuristic partial order planner closely related to RePOP. LPG performs a kind of partial order search by dynamically modifying a planning graph, using various heuristics including a relaxed planning technique (taking into account exclusion relations in the planning graph) to estimate the difficulty of achieving an action's preconditions. When looking at the STRIPS domains used at AIPS-2002, one finds that, for both VHPOP and LPG, the domains where they scale most comfortably are *Zenotravel* and *Satellite* – once again, those domains where h^+ has the highest quality. The other domains (*Depots*, *Driverlog*, *Rovers*, and *Freecell*) are more challenging. Similar observations can be made for Mips, SimPlanner [58], and Stella [64] (the latter two planners use variations of relaxed planning techniques to control forward search; SimPlanner, in particular, did very well in the AIPS-2002 STRIPS domains). Experiments performed by the authors of LPG [31] indicate that the relaxed plan heuristic has a large impact on LPG's runtime behavior.

Let me say some words on future work. Of course it would be desirable to extend the analysis techniques described herein to numerical and durational settings as used in IPC-3 [25]. A very ambitious research direction contributed to by the work at hand is to further planner runtime performance by elaborating relaxed planning techniques in those cases where h^+ is not a good heuristic, or, more generally, by devising heuristics that have qualities complementary to h^+. In the long run, the vision is to have an arsenal of complementary heuristics, and combine these into a hybrid system that can automatically be configured to best suit a given arbitrary planning task. The contribution made towards this vision by the results at hand is a very clear picture of where the strengths of h^+ lie. There already are attempts in the literature to elaborate on the quality of h^+: as indicated before, STAN4 [24] uses automatically extracted domain knowledge to improve its heuristic goal distance estimates, and LPG's relaxed planning process [32] takes exclusion relations in a planning graph into account.

A more short term project than the above vision, but still an ambitious task, is to find more concrete connections between problem structure and h^+ topology. Some of the definitions that form the structural core of the theory at hand, c.f. Section 5, are of a syntactical nature (namely, the definitions concerned with dead ends). Others, however, are not directly testable by a machine (at least not without combinatorial search). The relevant implications are Lemma 4 (for applying which we need to test whether the actions are respected by the relaxation), as well as the proof techniques used for proving exit distance bounds. These implications are, for a given planning task or domain, generally not overly difficult to test for a human observer (depending, of course, on the complexity of the domain). It is largely unclear, however, how a task or domain could be analyzed *automatically* to see whether the implications apply or not. A promising line of research seems to be the identification of more syntactical criteria sufficient for the prerequisites of the implications. The benefits of such domain analysis techniques would be twofold. First, information about the quality to be expected from individual heuristics is crucial for configuring hybrid systems, c.f. above. Second, an automatic analysis technique might spare the effort of investigating h^+ quality for each single domain over again – ideally, one would want to prove topological properties that hold true for domain *classes*.

I close the article by discussing one of the most pressing questions opened up by this work. Briefly put, the result we got here is that most of the current planning benchmarks share a structure that makes h^+ a very high quality heuristic. Now:

Is this a good *or a* bad *result for AI planning? Does it mean that we have identified widely spread structural properties that can make planning feasible, or does it only mean that the benchmarks are superficial?*

Put in less polemical terms, the question is whether the current benchmarks are an adequate means of evaluation in the sense that they capture the structures that are important in the envisioned applications. This is not a question one can answer just like that. There is not even a clear picture in the community of what the "envisioned applications" are. While domains like *Simple-Tsp*, *Movie*, or *Gripper* are so trivial they seem highly unlikely to contain any structures relevant in reality,[8] it is much harder to assess the relevance of domains like *Satellite*, *Grid*, or *Blocksworld-arm*. The author personally believes that, at this point in time, there are the following two things to do about the benchmarks/about evaluation in planning:

1. Design new benchmark domains and example suits that are as close as possible to existing real-world applications of planning. To name a few, examples for such applications are Jana Koehler's Miconic domain [51], Sylvie Thiebaux's PSR domain [3, 12], and Stefan Edelkamp's software verification domain [18]. The use of more such domains will hopefully yield a more conclusive picture of how efficient the current planning technology can be

[8]Indeed, these domains have been designed with the intention to highlight certain characteristic shortcomings of certain planning systems, rather than with the intention to model realistic problems.

expected to be in reality, and provide a testbed suitable for steering the field into a fruitful direction.

2. Avoid the repeated use of structurally similar benchmark domains. It has been no exception in the past to see papers evaluate planners in testbeds including, e.g., all of the *Gripper*, *Logistics*, and *Miconic-STRIPS* domains. This obviously does not make much sense, at least if the performance of (one of) the tested planner(s) depends a lot on the quality of h^+ – polemically put, the only thing that changes are the names of the predicates. What a good testbed should do is, test the planners on a variety of different problem structures. A criterion for a good testbed could be to cover the relevant complexity classes (at least P, NP, and PSPACE) for the complexity of deciding (bounded) plan existence within the domains; or, at least for planners that use relaxed plan based heuristics, to cover all the relevant distinctions in the planning domain taxonomy provided by the investigation at hand.

While the above two directives are basically common sense, a better thought through choice of benchmark examples, and interpretation of results obtained that way, would certainly do good to the AI planning community.[9] Together with Stefan Edelkamp, the author intends to promote just this, by applying the above directives in the next international planning competition alongside ICAPS-2004.

Acknowledgements

I would like to thank Drew McDermott, Fahiem Bacchus, Maria Fox, and Derek Long for their responses to various questions concerning the definitions of/intentions behind the competition domains. Thanks also go to Malte Helmert for some useful comments on the published benchmark instances, and the definition of planning domains.

A Formalized Benchmark Domains

This appendix contains the formal definitions of all investigated benchmark domains. As said in Section 2, to ease readability the formalizations presented here are made such that the domains are negation-free, which is in some ADL domains slightly different (syntactically, but not semantically) from the original formulation used in the planning community. I state the differences explicitly. As has been mentioned in Section 3, the decisions about what exactly the instances of the domains are have been made by abstracting from the published benchmarks. In most cases, this is obvious. In the less obvious cases, a few comments are added.

[9] While there have traditionally been a lot of complaints about the *Blocksworld* domain, there has rarely been an argument against *Blocksworld* except that it intuitively is a "toy" domain. But that's a property that most STRIPS domains are likely to share, and it does not exclude their potential ability to capture *structures* that are relevant in the real world. With respect to the topology under h^+, the investigation at hand indeed proves that *Blocksworld-arm* is one of the domains with a more interesting structure.

Generally, this appendix contains only the formal definitions and some explanation of the formal means involved. More intuitive descriptions of the domains are given in the respective individual subsections.

The 25 domains are listed in alphabetical order. For getting used to the notation, it is probably better to start with simple examples like *Simple-Tsp*, *Movie*, *Gripper*, and *Logistics*.

A.1 Assembly

Definition 24 *Assembly is the ADL domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\begin{aligned}\mathcal{P} = \{ \ & (equal, 2), (assembly, 1), (not\text{-}assembly, 1), (resource, 1), (not\text{-}resource, 1), \\ & (requires, 2), (not\text{-}requires, 2), (part\text{-}of, 2), (not\text{-}part\text{-}of, 2), (transient\text{-}part, 2), \\ & (not\text{-}transient\text{-}part, 2), (assemble\text{-}order, 3), (not\text{-}assemble\text{-}order, 3), \\ & (remove\text{-}order, 3), (not\text{-}remove\text{-}order, 3), (commited, 2), \\ & (available, 1), (incorporated, 2), (not\text{-}incorporated, 2), (complete, 1)\}\end{aligned}$$

$\mathcal{O}$ *consists of the operators*

- *commit:*

$$\begin{aligned}(\ & (x, y), \\ & \{(resource, x), (assembly, y), (available, x)\}, \\ & \{(committed, x, y)\}, \\ & \{(available, x)\})\end{aligned}$$

- *release:*

$$\begin{aligned}(\ & (x, y), \\ & \{(resource, x), (assembly, y), (commited, x, y)\}, \\ & \{(available, x)\}, \\ & \{(committed, x, y)\})\end{aligned}$$

- *assemble:*

$$\begin{aligned}(\ & (x, y), \\ & (assembly, x) \wedge (assembly, y) \wedge (available, x) \wedge \\ & (\forall z : (not\text{-}resource, z) \vee (not\text{-}requires, y, z) \vee (committed, z, y)) \wedge \\ & ((part\text{-}of, x, y) \vee (transient\text{-}part, x, y)) \wedge \\ & (\forall z : (not\text{-}assembly, z) \vee (not\text{-}assemble\text{-}order, z, x, y) \vee \\ & \qquad (incorporated, z, y)), \\ & \{ \ ((), \top, \{(incorporated, x, y)\}, \{(available, x), (not\text{-}incorporated, x, y)\}), \\ & \quad (\ (),\end{aligned}$$

$$
\begin{array}{l}
(\textit{part-of}, x, y) \wedge \\
(\forall z : (\textit{not-assembly}, z) \vee (\textit{not-part-of}, z, y) \vee (\textit{equal}, z, x) \vee \\
\qquad (\textit{incorporated}, z, y)) \wedge \\
(\forall z : (\textit{not-assembly}, z) \vee (\textit{not-transient-part}, z, y) \vee \\
\qquad (\textit{not-incorporated}, z, y)), \\
\{(\textit{complete}, y), (\textit{available}, y)\}, \\
\emptyset)\})
\end{array}
$$

- *remove:*

$$
\begin{array}{l}
(\ (x, y), \\
\ \ (\textit{assembly}, x) \wedge (\textit{assembly}, y) \wedge (\textit{incorporated}, x, y) \wedge \\
\ \ (\forall z : (\textit{not-resource}, z) \vee (\textit{not-requires}, y, z) \vee (\textit{committed}, z, y)) \wedge \\
\ \ (\ (\textit{transient-part}, x, y) \wedge \\
\ \ \ \ (\forall z : (\textit{not-assembly}, z) \vee (\textit{not-remove-order}, z, x, y) \vee \\
\ \ \ \ (\textit{incorporated}, z, y)) \vee (\textit{part-of}, x, y) \wedge \\
\ \ \ \ (\forall z : (\textit{not-assembly}, z) \vee (\textit{not-assemble-order}, z, x, y) \vee \\
\qquad\quad (\textit{not-incorporated}, z, y))), \\
\ \ \{\ ((), \top, \{(\textit{available}, x), (\textit{not-incorporated}, x, y)\}, \{(\textit{incorporated}, x, y)\}), \\
\ \ \ \ (\ (), \\
\ \ \ \ \ \ (\textit{transient-part}, x, y) \wedge \\
\ \ \ \ \ \ (\forall z \in C : (\textit{not-assembly}, z) \vee (\textit{not-part-of}, z, y) \vee (\textit{incorporated}, z, y)) \wedge \\
\ \ \ \ \ \ (\forall z \in C : (\textit{not-assembly}, z) \vee (\textit{not-transient-part}, z, y) \vee (\textit{equal}, z, x) \vee \\
\qquad\qquad\quad (\textit{not-incorporated}, z, y)), \\
\ \ \ \ \ \ \{(\textit{complete}, y), (\textit{available}, y)\}, \\
\ \ \ \ \ \ \emptyset)\})
\end{array}
$$

and the set of instances is

$$
\begin{array}{l}
\mathcal{I} = \{\ (\ As \cup R, \\
\qquad \{(\textit{equal}, x, x) \mid x \in As \cup R\} \cup \\
\qquad \{(\textit{assembly}, x) \mid x \in As\} \cup \{(\textit{not-assembly}, x) \mid x \in R\} \cup \\
\qquad \{(\textit{resource}, x) \mid x \in R\} \cup \{(\textit{not-resource}, x) \mid x \in As\} \cup \\
\qquad \{(\textit{requires}, x, y) \mid x \in As, y \in re(x)\} \cup \\
\qquad \{(\textit{not-requires}, x, y) \mid x \in As, y \in R \setminus re(x)\} \cup \\
\qquad \{(\textit{part-of}, x, y) \mid y \in As, x \in pa(y)\} \cup \\
\qquad \{(\textit{not-part-of}, x, y) \mid y \in As, x \in As \setminus pa(y)\} \cup \\
\qquad \{(\textit{transient-part}, x, y) \mid y \in As, x \in tr(y)\} \cup \\
\qquad \{(\textit{not-transient-part}, x, y) \mid y \in As, x \in As \setminus tr(y)\} \cup \\
\qquad \{(\textit{assemble-order}, x, z, y) \mid y \in As, (x, z) \in ao(y)\} \cup \\
\qquad \{(\textit{not-assemble-order}, x, z, y) \mid y \in As, (x, z) \in (As \times As) \setminus ao(y)\} \cup \\
\qquad \{(\textit{remove-order}, x, z, y) \mid y \in As, (x, z) \in ro(y)\} \cup \\
\qquad \{(\textit{not-remove-order}, x, z, y) \mid y \in As, (x, z) \in (As \times As) \setminus ro(y)\} \cup \\
\qquad \{(\textit{available}, x) \mid x \in As, pa(x) = \emptyset\} \cup \\
\qquad \{(\textit{not-incorporated}, x, y) \mid y \in As, x \in pa(y) \cup tr(y)\}, \\
\qquad \{(\textit{complete}, g)\})
\end{array}
$$

$$
\begin{aligned}
| \ & As \cup R \subseteq \Sigma^+, As \cap R = \emptyset, |As| \geq 2 \\
& pa : As \mapsto 2^{As}, (As, \{(x,y) \mid y \in As, x \in pa(y)\}) \ \textit{is tree with root } g, \\
& tr : As \mapsto 2^{As}, \\
& \forall y \in As : pa(y) = \emptyset \Rightarrow tr(y) = \emptyset \wedge tr(y) \cap pa(y) = \emptyset \wedge y \notin tr(y), \\
& re : As \mapsto 2^{R}, \\
& ao : As \mapsto 2^{(As \times As)}, (As, ao(y)) \ \textit{is cycle-free for } y \in As, \\
& \forall y \in As : (x,z) \in ao(y) \wedge z \in pa(y) \cup tr(y) \Rightarrow x \in pa(y) \cup tr(y) \\
& ro : As \mapsto 2^{(As \times As)}, (As, ro(y)) \ \textit{is cycle-free for } y \in As, \\
& \forall y \in As : (x,z) \in ro(y) \wedge z \in tr(y) \Rightarrow x \in pa(y) \cup tr(y)\}
\end{aligned}
$$

Remember that we allow ADL domains also to comprise STRIPS operators, c.f. Section 2. The *commit* and *release* operators are STRIPS formulations; *assemble* and *remove* are ADL operators, with complex preconditions and complex effect conditions but with empty effect parameters. In the instances, the pa function defines the part-of relation, which forms a tree where the root node is the goal assembly. Each non-leaf node of the tree ($pa(y) \neq \emptyset$) can have arbitrary transient parts except its own parts, given by the tr function, and each non-leaf node can require an arbitrary amount of resources specified via the re function. The assemble order $ao(y)$ is a consistent partial ordering between parts and transient parts of an assembly y, and the remove order $ro(y)$ is a consistent partial ordering between (transient) parts and transient parts of an assembly y. In the original domain there are a lot of negations, which are formulated by the extra predicates *not-assembly*, *not-resource*, *not-requires*, *not-part-of*, *not-transient-part*, *not-assemble-order*, *not-remove-order*, and *not-incorporated*.

The set of instances makes a number of restrictions excluding cases that are obvious nonsense. The restrictions are the following. The pa relation forms a tree — otherwise one part would need to be incorporated into several others simultaneously, or similar nonsense. There are no transient parts in the atomic assemblies (the leafs of the pa-tree), no assembly is transient part and part (of the same higher-level assembly) simultaneously, and no assembly is transient part of itself. The assemble and remove orders are consistent. Given an assemble order $(x,z) \in ao(y)$ where it might be that z must be incorporated into y at some point (as $z \in pa(y) \cup tr(y)$) we postulate that x can in principle be incorporated into y, $x \in pa(y) \cup tr(y)$. Similarly, given a remove order $(x,z) \in ro(y)$ where it might be that x must be incorporated into y at some point in order to get z out (which can only happen if $z \in tr(y)$) we postulate that this can in principle be done, $x \in pa(y) \cup tr(y)$.

With respect to the above restrictions, in the AIPS-1998 benchmark collection the situation is as follows. There are three tasks where the initial state uses undeclared constants, which is a violation of the ADL language. There are furthermore two tasks that do not adhere to the last of the restrictions described above, i.e., where there are x, z, and y with $(x,z) \in ro(y)$, $z \in tr(y)$, and $x \notin pa(y) \cup tr(y)$. These two tasks are trivially unsolvable, and we assume that their specification is flawed. The remaining 25 tasks adhere to the restrictions of our definition.

A.2 Blocksworld-arm

Definition 25 *Blocksworld-arm is the STRIPS domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\mathcal{P} = \{(clear, 1), (holding, 1), (arm\text{-}empty, 0), (on\text{-}table, 1), (on, 2)\}$$

$\mathcal{O}$ *consists of the operators*

- *pickup:*

$$\begin{array}{l} (\ (x), \\ \quad \{(clear, x), (on\text{-}table, x), (arm\text{-}empty)\}, \\ \quad \{(holding, x)\}, \\ \quad \{(clear, x), (on\text{-}table, x), (arm\text{-}empty)\}) \end{array}$$

- *putdown:*

$$\begin{array}{l} (\ (x), \\ \quad \{(holding, x)\}, \\ \quad \{(clear, x), (on\text{-}table, x), (arm\text{-}empty)\}, \\ \quad \{(holding, x)\}) \end{array}$$

- *unstack:*

$$\begin{array}{l} (\ (x, y), \\ \quad \{(clear, x), (on, x, y), (arm\text{-}empty)\}, \\ \quad \{(holding, x), (clear, y)\}, \\ \quad \{(clear, x), (on, x, y), (arm\text{-}empty)\}) \end{array}$$

- *stack:*

$$\begin{array}{l} (\ (x, y), \\ \quad \{(holding, x), (clear, y)\}, \\ \quad \{(clear, x), (on, x, y), (arm\text{-}empty)\}, \\ \quad \{(holding, x), (clear, y)\}) \end{array}$$

and the set of instances is

$$\begin{array}{rl} \mathcal{I} = \{ & (\ B, \\ & \quad \{(arm\text{-}empty)\} \cup \{(on\text{-}table, x) \mid x \in B, \forall y : (x, y) \notin I\} \cup \\ & \quad \{(on, x, y) \mid (x, y) \in I\} \cup \{(clear, x) \mid x \in B, \forall y : (y, x) \notin I\}, \\ & \quad \{(on, x, y) \mid (x, y) \in G\}) \\ & \mid\ B \subseteq \Sigma^+, |B| \geq 1, \\ & \quad I \subseteq B \times B, (B, I) \textit{ is cycle-free graph with in- and out-degree 1}, \\ & \quad G \subseteq B \times B, (B, G) \textit{ is cycle-free graph with in- and out-degree 1}\} \end{array}$$

In the instances, the edge sets I and G represent the initial and goal states, respectively, by the semantics that $(x, y) \in I$ ($\in G$) if and only if x is initially (shall be finally) located on y. Note that an edge set represents a legal state iff there are no cycles in the corresponding on relation, each block has at most one block located on top of it, and each block is located on top of at most one other block.

A.3 Blocksworld-no-arm

Definition 26 *Blocksworld-no-arm is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$\mathcal{P} = \{(\textit{not-eq}, 2), (\textit{clear}, 1), (\textit{on-table}, 1), (\textit{on}, 2)\}$$

$\mathcal{O}$ consists of the operators

- *move-table-to-block:*

$$\begin{array}{l} (\ (x, y), \\ \quad \{(\textit{not-eq}, x, y), (\textit{clear}, x), (\textit{clear}, y), (\textit{on-table}, x)\}, \\ \quad \{(\textit{on}, x, y)\}, \\ \quad \{(\textit{clear}, y), (\textit{on-table}, x)\}) \end{array}$$

- *move-block-to-table:*

$$((x, y), \{(\textit{clear}, x), (\textit{on}, x, y)\}, \{(\textit{on-table}, x), (\textit{clear}, y)\}, \{(\textit{on}, x, y)\}$$

- *move-block-to-block:*

$$\begin{array}{l} (\ (x, y, z), \\ \quad \{(\textit{not-eq}, x, z), (\textit{clear}, x), (\textit{on}, x, y), (\textit{clear}, z)\}, \\ \quad \{(\textit{on}, x, z), (\textit{clear}, y)\}, \\ \quad \{(\textit{on}, x, y), (\textit{clear}, z)\}) \end{array}$$

and the set of instances is

$$\begin{array}{rl} \mathcal{I} = \{ & (\ B, \\ & \quad \{(\textit{not-eq}, x, y) \mid x, y \in B, x \neq y\} \cup \{(\textit{on-table}, x) \mid x \in B, \forall y : (x, y) \notin I\} \cup \\ & \quad \{(\textit{on}, x, y) \mid (x, y) \in I\} \cup \{(\textit{clear}, x) \mid x \in B, \forall y : (y, x) \notin I\}, \\ & \quad \{(\textit{on}, x, y) \mid (x, y) \in G\}) \\ & \mid\ B \subseteq \Sigma^+, |B| \geq 1, \\ & \quad I \subseteq B \times B, (B, I) \textit{ is cycle-free graph with in- and out-degree 1}, \\ & \quad G \subseteq B \times B, (B, G) \textit{ is cycle-free graph with in- and out-degree 1}\} \end{array}$$

The instances are exactly the same as for *Blocksworld-arm*, see above. We have introduced non-equality to get rid of actions that try to move a block on top of itself.

A.4 Briefcaseworld

Definition 27 *Briefcaseworld is the ADL domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\mathcal{P} = \{(\mathit{not\text{-}eq}, 2), (\mathit{location}, 1), (\mathit{portable}, 1), (\mathit{at}, 2), (\mathit{in}, 1), (\mathit{not\text{-}in}, 1), (\mathit{is\text{-}at}, 1)\}$$

$\mathcal{O}$ *consists of the operators*

- *move:*

$$\begin{aligned} &(\ (x, y), \\ &\quad (\mathit{not\text{-}eq}, x, y) \wedge (\mathit{location}, x) \wedge (\mathit{location}, y) \wedge (\mathit{is\text{-}at}, x), \\ &\quad \{\ ((), \top, \{(\mathit{is\text{-}at}, y)\}, \{(\mathit{is\text{-}at}, x)\}), \\ &\qquad ((z), (\mathit{portable}, z) \wedge (\mathit{in}, z), \{(\mathit{at}, z, y)\}, \{(\mathit{at}, z, x)\})\}) \end{aligned}$$

- *take-out:* $((x), \{(\mathit{portable}, x)\}, \{(\mathit{not\text{-}in}, x)\}, \{(\mathit{in}, x)\})$
- *put-in:*

$$\begin{aligned} &(\ (x, y, z), \\ &\quad \{(\mathit{portable}, x), (\mathit{location}, y), (\mathit{not\text{-}in}, x), (\mathit{at}, x, y), (\mathit{is\text{-}at}, y)\}, \\ &\quad \{(\mathit{in}, x)\}, \\ &\quad \{(\mathit{not\text{-}in}, x)\}) \end{aligned}$$

and the set of instances is

$$\begin{aligned} \mathcal{I} = \{\ (\ & L \cup P, \\ & \{(\mathit{not\text{-}eq}, x, y) \mid x, y \in L, x \neq y\} \cup \{(\mathit{location}, x) \mid x \in L\} \cup \\ & \{(\mathit{portable}, x) \mid x \in P\} \cup \{(\mathit{is\text{-}at}, \mathit{il})\} \cup \{(\mathit{at}, x, \mathit{pil}(x)) \mid x \in P\}, \\ & (\mathit{is\text{-}at}, \mathit{gl}) \wedge \bigwedge\{(\mathit{at}, x, \mathit{pgl}(x)) \mid x \in P'\}) \\ \mid\ & (L \cup P) \subseteq \Sigma^+, L \cap P = \emptyset, |L| \geq 1, \\ & \mathit{il} \in L, \mathit{pil} : P \mapsto L, \mathit{gl} \in L, P' \subseteq P, \mathit{pgl} : P' \mapsto L\} \end{aligned}$$

Note that we have a mixture between STRIPS and ADL operators, the only ADL element being the conditional effect of the *move* operator (which has one effect parameter). In the definition of the instances, *il* and *gl* are the initial and goal locations of the briefcase, while *pil* and *pgl* are the initial and goal location functions for portables. $P' \subseteq P$ is the subset of portables that must be transported, i.e., that are mentioned in the goal specification. We have introduced non-equality to get rid of actions that try to move within a location. The *not-in* predicate is part of the original domain formulation.

A.5 Depots

Definition 28 *Depots is the STRIPS domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\begin{aligned}\mathcal{P} = \{\ & (\textit{not-equal}, 2), (\textit{location}, 1), (\textit{truck}, 1),\\ & (\textit{crate}, 1), (\textit{hoist}, 1), (\textit{pallet}, 1), (\textit{surface}, 1),\\ & (\textit{at}, 2), (\textit{in}, 2), (\textit{on}, 2), (\textit{lifting}, 2), (\textit{available}, 1), (\textit{clear}, 1)\}\end{aligned}$$

$\mathcal{O}$ consists of the operators

- *drive:*

$$\begin{aligned}(\ & (x, y, z),\\ & \{(\textit{truck}, x), (\textit{location}, y), (\textit{location}, z), (\textit{not-equal}, y, z), (\textit{at}, x, y)\},\\ & \{(\textit{at}, x, z)\},\\ & \{(\textit{at}, x, y)\})\end{aligned}$$

- *lift:*

$$\begin{aligned}(\ & (x, y, z, u),\\ & \{\ (\textit{crate}, x), (\textit{hoist}, y), (\textit{surface}, z), (\textit{location}, u),\\ & \quad (\textit{at}, y, u), (\textit{available}, y), (\textit{at}, x, u), (\textit{on}, x, z), (\textit{clear}, x)\},\\ & \{(\textit{lifting}, y, x), (\textit{clear}, z)\},\\ & \{(\textit{at}, x, u), (\textit{clear}, x), (\textit{available}, y), (\textit{on}, x, z)\})\end{aligned}$$

- *drop:*

$$\begin{aligned}(\ & (x, y, z, u),\\ & \{\ (\textit{crate}, x), (\textit{hoist}, y), (\textit{surface}, z), (\textit{location}, u),\\ & \quad (\textit{at}, y, u), (\textit{lifting}, y, x), (\textit{at}, z, u), (\textit{clear}, z)\},\\ & \{(\textit{at}, x, u), (\textit{clear}, x), (\textit{available}, y), (\textit{on}, x, z)\},\\ & \{(\textit{lifting}, y, x), (\textit{clear}, z)\})\end{aligned}$$

- *load:*

$$\begin{aligned}(\ & (x, y, z, u),\\ & \{\ (\textit{crate}, x), (\textit{hoist}, y), (\textit{truck}, z), (\textit{location}, u),\\ & \quad (\textit{at}, y, u), (\textit{lifting}, y, x), (\textit{at}, z, u)\},\\ & \{(\textit{in}, x, z), (\textit{available}, y)\},\\ & \{(\textit{lifting}, y, x)\})\end{aligned}$$

- *unload:*

$$\begin{aligned}(\ & (x, y, z, u),\\ & \{\ (\textit{crate}, x), (\textit{hoist}, y), (\textit{truck}, z), (\textit{location}, u),\\ & \quad (\textit{at}, y, u), (\textit{available}, y), (\textit{at}, z, u), (\textit{in}, x, z)\},\\ & \{(\textit{lifting}, y, x)\},\\ & \{(\textit{in}, x, z), (\textit{available}, y)\})\end{aligned}$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{ \ (\ & L \cup T \cup C \cup H \cup P, \\
& \{(location, x) \mid x \in L\} \cup \{(truck, x) \mid x \in T\} \cup \{(crate, x) \mid x \in C\} \cup \\
& \{(hoist, x) \mid x \in H\} \cup \{(pallet, x) \mid x \in P\} \cup \\
& \{(surface, x) \mid x \in C \cup P\} \cup \{(available, x) \mid x \in H\} \cup \\
& \{(at, x, il(x)) \mid x \in T \cup C \cup H \cup P\} \cup \\
& \{(on, x, y) \mid (x, y) \in O^i\} \cup \{(clear, x) \mid x \in C \cup P, \forall y : (y, x) \notin O^i\}, \\
& \{(on, x, y) \mid (x, y) \in O^g\} \\
\mid \ & (L \cup T \cup C \cup H \cup P) \subseteq \Sigma^+, L, T, C, H, \ \textit{and } P \textit{ are pairwise disjoint}, \\
& |L| \geq 1, il : T \cup C \cup H \mapsto L, \\
& O^i \subseteq (C \times (C \cup P)), \ \textit{s.t. } (C \cup P, O^i) \textit{ is cycle-free graph} \\
& \qquad \textit{with in-degree 1 and out-degree 1,} \\
& \qquad \forall x \in C : \exists y : (x, y) \in O^i, \textit{ and} \\
& \qquad \forall x, y : (x, y) \in O^i \rightarrow il(x) = il(y), \\
& C^g \subseteq C, \\
& O^g \subseteq (C^g \times (C^g \cup P)), \ \textit{s.t. } (C^g \cup P, O^g) \textit{ is cycle-free graph} \\
& \qquad \textit{with in-degree 1 and out-degree 1, and} \\
& \qquad \forall x \in C^g : \exists y : (x, y) \in O^g\}
\end{aligned}
$$

In the instances, the edge sets O^i and O^g together with the function il represent the initial and goal states. The semantics of the edge sets are that $(x, y) \in O^i$ ($\in O^g$) if and only if x is initially (shall be finally) on y. Note that: crates can be (required to be) on pallets but not vice versa; that by definition at the bottom of each initial/goal stack of crates there is a pallet; and that an edge set represents a legal state only if there are no cycles in the corresponding *on* relation, each crate/pallet has at most one crate located on top of it, and each crate is located on top of at most one other crate/pallet. The il function simply specifies the initial locations of all trucks, crates, hoists, and pallets. The additional constraint there is that the given *on* and *at* relations are consistent with each other: if x is on y initially then x and y must be at the same location. Operator parameter orderings have been changed slightly from the original domain encoding in order to improve readability.

A.6 Driverlog

Definition 29 *Driverlog is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$
\begin{aligned}
\mathcal{P} = \{ \ & (location, 1), (truck, 1), (obj, 1), (driver, 1), \\
& (link, 2), (path, 2), \\
& (at, 2), (in, 2), (driving, 2), (empty, 1)\}
\end{aligned}
$$

$\mathcal{O}$ consists of the operators

- *load-truck:*

  $(\ (x, y, z),$

$\{(obj, x), (truck, y), (location, z), (at, x, z), (at, y, z)\},$
$\{(in, x, y)\},$
$\{(at, x, z)\})$

- *unload-truck:*

$$
\begin{aligned}
&(\ (x, y, z),\\
&\quad \{(obj, x), (truck, y), (location, z), (in, x, y), (at, y, z)\},\\
&\quad \{(at, x, z)\},\\
&\quad \{(in, x, y)\})
\end{aligned}
$$

- *board-truck:*

$$
\begin{aligned}
&(\ (x, y, z),\\
&\quad \{(driver, x), (truck, y), (location, z), (at, x, z), (at, y, z), (empty, y)\},\\
&\quad \{(driving, x, y)\},\\
&\quad \{(at, x, z), (empty, y)\})
\end{aligned}
$$

- *disembark-truck:*

$$
\begin{aligned}
&(\ (x, y, z),\\
&\quad \{(driver, x), (truck, y), (location, z), (driving, x, y), (at, y, z)\},\\
&\quad \{(at, x, z), (empty, y)\},\\
&\quad \{(driving, x, y)\})
\end{aligned}
$$

- *drive-truck:*

$$
\begin{aligned}
&(\ (x, y, z, u),\\
&\quad \{ \ (truck, x), (location, y), (location, z), (driver, u),\\
&\qquad (link, y, z), (driving, u, x), (at, x, y)\},\\
&\quad \{(at, x, z)\},\\
&\quad \{(at, x, y)\})
\end{aligned}
$$

- *walk:*

$$
\begin{aligned}
&(\ (x, y, z),\\
&\quad \{ \ (driver, x), (location, y), (location, z),\\
&\qquad (path, y, z), (at, x, y)\},\\
&\quad \{(at, x, z)\},\\
&\quad \{(at, x, y)\})
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{ \ (\ & L \cup T \cup O \cup D,\\
& \{(location, x) \mid x \in L\} \cup \{(truck, x) \mid x \in T\} \cup\\
& \{(object, x) \mid x \in O\} \cup \{(driver, x) \mid x \in D\} \cup
\end{aligned}
$$

$$
\begin{aligned}
& \{(\mathit{link}, x, y) \mid (x,y) \in LI\} \cup \\
& \{(\mathit{path}, x, y) \mid (x,y) \in PA\} \cup \\
& \{(\mathit{at}, x, il(x)) \mid x \in T \cup O \cup D\}, \\
& \{(\mathit{at}, x, gl(x)) \mid x \in T' \cup O' \cup D'\}) \\
\mid\ & (L \cup T \cup O \cup D) \subseteq \Sigma^+, L, T, O, \ \textit{and } D \textit{ are pairwise disjoint}, \\
& |L| \geq 1, \\
& LI \subseteq L \times L, \forall x, y : (x,y) \in LI \rightarrow (y,x) \in LI, \\
& PA \subseteq L \times L, \forall x, y : (x,y) \in PA \rightarrow (y,x) \in PA, \\
& il : T \cup O \cup D \mapsto L, \\
& T' \subseteq T, O' \subseteq O, D' \subseteq D, gl : T' \cup O' \cup D' \mapsto L\}
\end{aligned}
$$

In the definition of the instances, il and gl are the initial location and goal location functions, respectively. There is at least one location. Subsets of the trucks, objects, and drivers are required to be at certain locations to achieve the goal. The locations are linked for truck movements by the edges in LI (links), and for driver movements by the edges in PA (paths). Note that the movement links are bi-directional in the sense that, if one can move from x to y, one can also move back. This restriction holds true in the instances as can be generated with the IPC-3 generator. The restriction does make a significant difference – with directed edges, there can be unrecognized dead ends, see Section 9.3.

(ARE THESE THE RESTRICTIONS ON LI AND PA IN THE IPC-3 GENERATOR??)

A.7 Ferry

Definition 30 *Ferry is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$\mathcal{P} = \{(\mathit{not\text{-}eq}, 2), (\mathit{car}, 1), (\mathit{location}, 1), (\mathit{at\text{-}ferry}, 1), (\mathit{at}, 2), (\mathit{empty\text{-}ferry}, 0), (\mathit{on}, 1)\}$$

$\mathcal{O}$ consists of the operators

- *sail:*

$$
\begin{aligned}
(\ & (x, y), \\
& \{(\mathit{not\text{-}eq}, x, y), (\mathit{location}, x), (\mathit{location}, y), (\mathit{at\text{-}ferry}, x)\}, \\
& \{(\mathit{at\text{-}ferry}, y)\}, \\
& \{(\mathit{at\text{-}ferry}, x)\})
\end{aligned}
$$

- *board:*

$$
\begin{aligned}
(\ & (x, y), \\
& \{(\mathit{car}, x), (\mathit{location}, y), (\mathit{at}, x, y), (\mathit{at\text{-}ferry}, y), (\mathit{empty\text{-}ferry})\}, \\
& \{(\mathit{on}, x)\}, \\
& \{(\mathit{at}, x, y), (\mathit{empty\text{-}ferry})\})
\end{aligned}
$$

- *debark:*

$$
\begin{array}{l}
(\ (x, y), \\
\quad \{(car, x), (location, y), (on, x), (at\text{-}ferry, y)\}, \\
\quad \{(at, x, y), (empty\text{-}ferry)\}, \\
\quad \{(on, x)\})
\end{array}
$$

and the set of instances is

$$
\begin{array}{rl}
\mathcal{I} = \{ & (\ C \cup L, \\
& \quad \{(not\text{-}eq, x, y) \mid x, y \in L, x \neq y\} \cup \{(car, x) \mid x \in C\} \cup \\
& \quad \{(location, x) \mid x \in L\} \cup \{(at\text{-}ferry, il)\} \cup \{(at, x, cil(x)) \mid x \in C\}, \\
& \quad \{(at, x, cgl(x)) \mid x \in C'\}) \\
& \mid (C \cup L) \subseteq \Sigma^+, C \cap L = \emptyset, |L| \geq 1, \\
& \quad il \in L, cil : C \mapsto L, C' \subseteq C, cgl : C' \mapsto L\}
\end{array}
$$

In the instances, il is the initial location of the ferry, and cil is the car initial location function mapping the cars to their respective initial locations. Likewise, cgl maps cars to their goal locations, where only the subset C' of all cars is required to be at a specific location to fulfill the goal. We have introduced non-equality to get rid of actions that try to sail within a location.

A.8 Freecell

Definition 31 *Freecell is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$
\begin{array}{rl}
\mathcal{P} = \{ & (card, 1), (suit\text{-}type, 1), (num, 1), (successor, 2), (suit, 2), \\
& (value, 2), (canstack, 2), (home, 1), (bottomcol, 1), (on, 2), \\
& (clear, 1), (incell, 1), (cellspace, 1), (colspace, 1)\}
\end{array}
$$

$\mathcal{O}$ consists of the operators

- *move:*

$$
\begin{array}{l}
(\ (x, y, z), \\
\quad \{(card, x), (card, y), (card, z), (clear, x), (clear, z), (on, x, y), (canstack, x, z)\}, \\
\quad \{(on, x, z), (clear, y)\}, \\
\quad \{(on, x, y), (clear, z)\})
\end{array}
$$

- *move-b:*

$$
\begin{array}{l}
(\ (x, y, z, u), \\
\quad \{\ (card, x), (card, y), (num, z), (num, u), (clear, x), (bottomcol, x), \\
\quad\quad (clear, y), (canstack, x, y), (colspace, z), (successor, u, z)\}, \\
\quad \{(on, x, y), (colspace, u)\}, \\
\quad \{(bottomcol, x), (clear, y), (colspace, z)\})
\end{array}
$$

- *sendtofree:*

$$
\begin{aligned}
&(\ (x,y,z,u),\\
&\quad \{\ (card,x),(card,y),(num,z),(num,u),\\
&\qquad (clear,x),(on,x,y),(cellspace,z),(successor,z,u)\},\\
&\quad \{(incell,x),(clear,y),(cellspace,u)\},\\
&\quad \{(on,x,y),(clear,x),(cellspace,z)\})
\end{aligned}
$$

- *sendtofree-b:*

$$
\begin{aligned}
&(\ (x,y,z,u,v),\\
&\quad \{\ (card,x),(num,y),(num,z),(num,u),(num,v),(clear,x),(bottomcol,x),\\
&\qquad (cellspace,y),(successor,y,z),(colspace,u),(successor,v,u)\},\\
&\quad \{(incell,x),(cellspace,z),(colspace,v)\},\\
&\quad \{(bottomcol,x),(clear,x),(cellspace,y),(colspace,u)\})
\end{aligned}
$$

- *sendtonewcol:*

$$
\begin{aligned}
&(\ (x,y,z,u),\\
&\quad \{\ (card,x),(card,y),(num,z),(num,u),\\
&\qquad (clear,x),(on,x,y),(colspace,z),(successor,z,u)\},\\
&\quad \{(bottomcol,x),(clear,y),(colspace,u)\},\\
&\quad \{(on,x,y),(colspace,z)\})
\end{aligned}
$$

- *sendtohome:*

$$
\begin{aligned}
&(\ (x,y,z,u,v,w),\\
&\quad \{\ (card,x),(card,y),(suit\text{-}type,z),(num,u),(card,v),\\
&\qquad (num,w),(on,x,y),(clear,x),(home,v),(suit,x,z),\\
&\qquad (suit,v,z),(value,x,u),(value,v,w),(successor,u,w)\},\\
&\quad \{(home,x),(clear,y)\},\\
&\quad \{(on,x,y),(clear,x),(home,v)\})
\end{aligned}
$$

- *sendtohome-b:*

$$
\begin{aligned}
&(\ (x,y,z,u,v,w,q),\\
&\quad \{\ (card,x),(suit\text{-}type,y),(num,z),(card,u),(num,v),(num,w),(num,q),\\
&\qquad (bottomcol,x),(clear,x),(home,u),(suit,x,y),(suit,u,y),(value,x,z),\\
&\qquad (value,u,v),(successor,z,v),(colspace,w),(successor,q,w)\},\\
&\quad \{(home,x),(colspace,q)\},\\
&\quad \{(bottomcol,x),(clear,x),(home,u),(colspace,w)\})
\end{aligned}
$$

- *homefromfreecell:*

$$
\begin{aligned}
&(\ (x,y,z,u,v,w,q),\\
&\quad \{\ (card,x),(suit\text{-}type,y),(num,z),(card,u),(num,v),(num,w),(num,q),
\end{aligned}
$$

$(incell, x), (home, u), (suit, x, y), (suit, u, y), (value, x, z), (value, u, v),$
$(successor, z, v), (cellspace, w), (successor, q, w)\},$
$\{(home, x), (cellspace, q)\},$
$\{(incell, x), (home, u), (cellspace, w)\})$

- *colfromfreecell:*

$$
\begin{aligned}
(\ & (x, y, z, u), \\
& \{ \ (card, x), (card, y), (num, z), (num, u), (incell, x), (clear, y), \\
& \quad (canstack, x, y), (cellspace, z), (successor, u, z)\}, \\
& \{(on, x, y), (clear, x), (cellspace, u)\}, \\
& \{(incell, x), (clear, y), (cellspace, z)\})
\end{aligned}
$$

- *newcolfromfreecell:*

$$
\begin{aligned}
(\ & (x, y, z, u, v), \\
& \{ \ (card, x), (num, y), (num, z), (num, u), (num, v), (incell, x), \\
& \quad (cellspace, y), (successor, z, y), (colspace, u), (successor, u, v)\}, \\
& \{(bottomcol, x), (clear, x), (cellspace, z), (colspace, v)\}, \\
& \{(incell, x), (cellspace, y), (colspace, u)\})
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{ \ (\ & C \cup S \cup N, \textit{where } C = \{c_{10}, \ldots, c_{1m_1}, \ldots, c_{n0}, \ldots, c_{nm_n}\}, \\
& \qquad S = \{s_1, \ldots, s_n\}, N = \{n_0, \ldots, n_m\}, \\
& \{(card, x) \mid x \in C\} \cup \{(suit\text{-}type, x) \mid x \in S\} \cup \{(num, x) \mid x \in N\} \cup \\
& \{(successor, n_{i+1}, n_i) \mid 0 \leq i < m\} \cup \\
& \{(suit, c_{ij}, s_i) \mid 1 \leq i \leq n, 0 \leq j \leq m_i\} \cup \\
& \{(value, c_{ij}, n_j) \mid 1 \leq i \leq n, 0 \leq j \leq m_i\} \cup \\
& \{(canstack, c_{ij}, c_{i'j'}) \mid co(s_i) \neq co(s_{i'}), j = j' - 1\} \cup \\
& \{(home, c_{i0}) \mid 1 \leq i \leq n\} \cup \{(bottomcol, x) \mid x \in C, \forall y : (x, y) \notin I\} \cup \\
& \{(on, x, y) \mid (x, y) \in I\} \cup \{(clear, x) \mid x \in C, \forall y : (y, x) \notin I\} \cup \\
& \{(cellspace, n_{nce})\} \cup \\
& \{(colspace, n_{nrco}) \mid nrco = nco - |\{x \in C \mid \forall y : (x, y) \notin I\}|\}, \\
& \{(home, c_{im_i}) \mid 1 \leq i \leq n\}) \\
\mid \ & (S \cup C \cup N) \subseteq \Sigma^+, S, C, \textit{ and } N \textit{ are pairwise disjoint}, \\
& |S| \geq 1, m_i \in \mathbb{N} \textit{ for } 1 \leq i \leq n, nce \in \mathbb{N}, nco \in \mathbb{N}, \\
& m \geq max\{m_1, \ldots, m_n, nce, nco\}, co : S \mapsto \{0, 1\}, \\
& I \subseteq C \times C, (C, I) \textit{ is cycle-free graph with in- and out-degree 1}, \\
& |\{x \in C \mid \forall y : (x, y) \notin I\}| \leq nco\}
\end{aligned}
$$

The different operators express different ways of changing the position of a card. Options are to move a card on top of another card if the respective cards can be stacked, to move a card to a free column or a free cell, or to move a card home. The operators keep track of the respective numbers of free columns or free cells left, and of the appropriate informations about the current configuration of the cards. In the instances, there are n suits of cards each of which has at least one element: the one that is initially home already. The cards within a suit have different values, encoded by numbers. There are at least as many numbers as are needed to account for all given items. Each suit has one of two colors given by the *co* function, and a card can be stacked on top of another one if and only if its color (the one of its suit) is different, and its value is one less than the one of the destination card. The total number of free cells is *nce*, the total number of columns is *nco*. The initial configuration of the cards is given by the edge set I, specifying which cards are on top of other ones. The edge set represents a legal state as there are no cycles, each card has at most one card located on top of it, and each card is located on top of at most one other card; also, the number of bottom cards — those that are placed directly on a column — is at most as high as the number of columns *nco*, such that the difference between both numbers, the number of (initially) remaining free columns, is $nrco \geq 0$.

A.9 Fridge

Definition 32 *Fridge is the ADL domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$\begin{aligned}\mathcal{P} = \{\ & (\mathit{fridge}, 1), (\mathit{compressor}, 1), (\mathit{fits}, 2), (\mathit{not\text{-}fits}, 2),\\ & (\mathit{screwed}, 1), (\mathit{not\text{-}screwed}, 1), (\mathit{fridge\text{-}on}, 1), (\mathit{not\text{-}fridge\text{-}on}, 1),\\ & (\mathit{screw}, 1), (\mathit{not\text{-}screw}, 1), (\mathit{attached}, 1), (\mathit{free}, 1)\}\end{aligned}$$

$\mathcal{O}$ consists of the operators

- *stop-fridge:*

$$((x), \{(\mathit{fridge}, x), (\mathit{fridge\text{-}on}, x)\}, \{(\mathit{not\text{-}fridge\text{-}on}, x)\}, \{(\mathit{fridge\text{-}on}, x)\})$$

- *start-fridge:*

$$((x), \{(\mathit{fridge}, x), (\mathit{not\text{-}fridge\text{-}on}, x)\}, \{(\mathit{fridge\text{-}on}, x)\}, \{(\mathit{not\text{-}fridge\text{-}on}, x)\})$$

- *unfasten:*

$$\begin{aligned}(\ & (x, y, z),\\ & \{\ (\mathit{screw}, x), (\mathit{compressor}, y), (\mathit{fridge}, z),\\ & \quad (\mathit{screwed}, x), (\mathit{fits}, x, y), (\mathit{attached}, y, z), (\mathit{not\text{-}fridge\text{-}on}, z)\},\\ & \{(\mathit{not\text{-}screwed}, x)\},\\ & \{(\mathit{screwed}, x)\})\end{aligned}$$

- *fasten:*

$$\begin{aligned}(\ & (x, y, z),\\ & \{\ (\mathit{screw}, x), (\mathit{compressor}, y), (\mathit{fridge}, z),\\ & \quad (\mathit{not\text{-}screwed}, x), (\mathit{fits}, x, y), (\mathit{attached}, y, z), (\mathit{not\text{-}fridge\text{-}on}, z)\},\\ & \{(\mathit{screwed}, x)\},\\ & \{(\mathit{not\text{-}screwed}, x)\})\end{aligned}$$

- *remove-compressor:*

$$
\begin{aligned}
&(\ (x,y),\\
&\quad (compressor,x) \wedge (fridge,y) \wedge (attached,x,y) \wedge (not\text{-}fridge\text{-}on,y) \wedge\\
&\quad \forall z : (not\text{-}screw,z) \vee (not\text{-}fits,z,x) \vee (not\text{-}screwed,z),\\
&\quad \{((), \top, \{(free,x),(free,y)\}, \{(attached,x,y)\})\})
\end{aligned}
$$

- *attach-compressor:*

$$
\begin{aligned}
&(\ (x,y),\\
&\quad \{(compressor,x),(fridge,y),(free,x),(free,y),(fits,x,y),(not\text{-}fridge\text{-}on,y)\},\\
&\quad \{(attached,x,y)\},\\
&\quad \{(free,x),(free,y)\})
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ (\ & F \cup C \cup S, \textit{where}\ \ F = \{f_1,\ldots,f_n\}, C = \{b_1,n_1,\ldots,b_n,n_n\},\\
& \qquad\qquad\qquad\quad S = \{s_{11},\ldots s_{1m},\ldots,s_{n1},\ldots,s_{nm}\},\\
& \{(fridge,x) \mid x \in F\} \cup \{(compressor,x) \mid x \in C\} \cup \{(screw,x) \mid x \in S\} \cup\\
& \{(not\text{-}screw,x) \mid x \in F \cup C\} \cup \{(fits,b_i,f_i),(fits,n_i,f_i) \mid 1 \leq i \leq n\} \cup\\
& \{(fits,s_{ij},b_i),(fits,s_{ij},n_i) \mid 1 \leq i \leq n, 1 \leq j \leq m\} \cup\\
& \{(not\text{-}fits,s_{ij},b_{i'}) \mid 1 \leq i \neq i' \leq n, 1 \leq j \leq m\} \cup\\
& \{(not\text{-}fits,s_{ij},n_{i'}) \mid 1 \leq i \neq i' \leq n, 1 \leq j \leq m\} \cup\\
& \{(fridge\text{-}on,x) \mid x \in F\} \cup \{(screwed,s_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq m\} \cup\\
& \{(attached,b_i,f_i) \mid 1 \leq i \leq n\} \cup \{(free,n_i) \mid 1 \leq i \leq n\},\\
& \bigwedge\{(fridge\text{-}on,x) \mid x \in F\} \wedge \bigwedge\{(screwed,s_{ij}) \mid 1 \leq i \leq n, 1 \leq j \leq m\} \wedge\\
& \bigwedge\{(attached,n_i,f_i) \mid 1 \leq i \leq n\})\\
\mid\ & (F \cup C \cup S) \subseteq \Sigma^+, F, C,\ \textit{and}\ S\ \textit{are pairwise disjoint}, n \geq 1, m \geq 1\}
\end{aligned}
$$

Note that we have a mixture between STRIPS and ADL operators, the only ADL element being the complex precondition of the *remove* operator. In the instances, F is the set of fridges, C the set of compressors, and S the set of screws. For each fridge f_i, there is the broken compressor b_i that is initially attached, and the new compressor n_i that shall be mounted. For each compressor b_i or n_i, there are the m screws $s_{i1},\ldots,s_{im}$ that fit for the compressor. In the original formulation there are various negations, which are formulated by the extra predicates *not-screw*, *not-fits*, *not-fridge-on*, and *not-screwed.*

A.10 Grid

Definition 33 *Grid is the STRIPS domain* $(\mathcal{P},\mathcal{O},\mathcal{I})$ *where the set of predicates is*

$$
\begin{aligned}
\mathcal{P} = \{\ & (place,1),(key,1),(shape,1),(conn,2),(holding,1),(arm\text{-}empty,1),\\
& (key\text{-}shape,2),(lock\text{-}shape,2),(open,1),(locked,1),(at,2),(at\text{-}robot,1)\}
\end{aligned}
$$

$\mathcal{O}$ *consists of the operators*

- *move:*

$$\begin{aligned}
&(\ (x, y),\\
&\quad \{(place, x), (place, y), (conn, x, y), (at\text{-}robot, x), (open, y)\},\\
&\quad \{(at\text{-}robby, y)\},\\
&\quad \{(at\text{-}robby, x)\})
\end{aligned}$$

- *pickup:*

$$\begin{aligned}
&(\ (x, y),\\
&\quad \{(place, x), (key, y), (at\text{-}robot, x), (at, y, x), (arm\text{-}empty)\},\\
&\quad \{(holding, y)\},\\
&\quad \{(at, y, x), (arm\text{-}empty)\})
\end{aligned}$$

- *pickup-and-lose:*

$$\begin{aligned}
&(\ (x, y, z),\\
&\quad \{(place, x), (key, y), (key, z), (at\text{-}robot, x), (holding, y), (at, z, x)\},\\
&\quad \{(holding, z), (at, y, x)\},\\
&\quad \{(holding, y), (at, z, x)\})
\end{aligned}$$

- *putdown:*

$$\begin{aligned}
&(\ (x, y),\\
&\quad \{(place, x), (key, y), (at\text{-}robot, x), (holding, y)\},\\
&\quad \{(at, y, x), (arm\text{-}empty)\},\\
&\quad \{(holding, y)\})
\end{aligned}$$

- *unlock:*

$$\begin{aligned}
&(\ (x, y, z, v),\\
&\quad \{\ (place, x), (place, y), (key, z), (shape, v), (conn, x, y), (at\text{-}robot, x),\\
&\qquad (holding, z), (key\text{-}shape, z, v), (lock\text{-}shape, y, v), (locked, y)\},\\
&\quad \{(open, y)\},\\
&\quad \{(locked, y)\})
\end{aligned}$$

and the set of instances is

$$\begin{aligned}
\mathcal{I} = \{\ (\ & P \cup S \cup K, \textit{where } P = \{p_{11}, \ldots, p_{nm}\},\\
& \{(place, x) \mid x \in P\} \cup \{(shape, x) \mid x \in S\} \cup \{(key, x) \mid x \in K\} \cup\\
& \{(conn, p_{ij}, p_{i'j'}) \mid p_{ij}, p_{i'j'} \in P, i = i', |j - j'| = 1 \ \textit{or}\ j = j', |i - i'| = 1\} \cup\\
& \{(open, x) \mid x \in P'\} \cup \{(locked, x) \mid x \in P \setminus P'\} \cup\\
& \{(lock\text{-}shape, x, ls(x)) \mid x \in P \setminus P'\} \cup \{(key\text{-}shape, x, ks(x)) \mid x \in K\} \cup\\
& \{(at\text{-}robot, ip) \cup \{(at, x, kip(x)) \mid x \in K\} \cup \{(arm\text{-}empty)\},\\
& \{(at, x, kgp(x)) \mid x \in K'\})\\
\mid\ & (P \cup S \cup K) \subseteq \Sigma^+, P, S, \textit{ and } K \textit{ are pairwise disjoint}, |P| \geq 1,\\
& P' \subseteq P, ls : P \setminus P' \mapsto S, ks : K \mapsto S,\\
& ip \in P', kip : K \mapsto P, K' \subseteq K, kgp : K' \mapsto P\}
\end{aligned}$$

In the instances, P' is the set of places that are initially already open. The *ls* function assigns to the other, locked, places, their lock shape, and *ks* assigns to all keys their shape. The initial place of the robot is $ip \in P'$, which *must be open.* This is an intuitively justified restriction (after all the robot is supposed to only be standing at unlocked places), and holds in the published benchmarks. The restriction is necessary to prove the non-existence of local minima under h^+, c.f. Section 7.4. The *kip* function assigns the initial key places, and *kgp* assigns the goal key places for those keys that must be transported, i.e., that are in $K' \subseteq K$.

A.11 Gripper

Definition 34 *Gripper is the STRIPS domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\begin{aligned}\mathcal{P} = \{ & (\textit{not-eq}, 2), (\textit{room}, 1), (\textit{ball}, 1), (\textit{gripper}, 1), \\ & (\textit{at-robby}, 1), (\textit{at}, 2), (\textit{free}, 1), (\textit{carry}, 2)\}\end{aligned}$$

$\mathcal{O}$ *consists of the operators*

- *move:*

$$\begin{aligned}(\; & (x, y), \\ & \{(\textit{not-eq}, x, y), (\textit{room}, x), (\textit{room}, y), (\textit{at-robby}, x)\}, \\ & \{(\textit{at-robby}, y)\}, \\ & \{(\textit{at-robby}, x)\})\end{aligned}$$

- *pick:*

$$\begin{aligned}(\; & (x, y, z), \\ & \{(\textit{ball}, x), (\textit{room}, y), (\textit{gripper}, z), (\textit{at}, x, y), (\textit{at-robby}, y), (\textit{free}, z)\}, \\ & \{(\textit{carry}, x, z)\}, \\ & \{(\textit{at}, x, y), (\textit{free}, z)\})\end{aligned}$$

- *drop:*

$$\begin{aligned}(\; & (x, y, z), \\ & \{(\textit{ball}, x), (\textit{room}, y), (\textit{gripper}, z), (\textit{carry}, x, z), (\textit{at-robby}, y)\}, \\ & \{(\textit{at}, x, y), (\textit{free}, z)\}, \\ & \{(\textit{carry}, x, z)\})\end{aligned}$$

and the set of instances is

$$\begin{aligned}\mathcal{I} = \{ \; (\; & \{a, b, l, r\} \cup B, \\ & \{(\textit{not-eq}, a, b), (\textit{not-eq}, b, a), (\textit{room}, a), (\textit{room}, b), (\textit{gripper}, l), (\textit{gripper}, r)\} \cup \\ & \{(\textit{ball}, x), (\textit{at}, x, a) \mid x \in B\} \cup \{(\textit{free}, l), (\textit{free}, r), (\textit{at-robby}, a)\}, \\ & \{(\textit{at}, x, b) \mid x \in B\}) \\ \mid \; & (\{a, b, l, r\} \cup B) \subseteq \Sigma^+, \{a, b, l, r\} \cap B = \emptyset, \\ & a, b, l, \textit{ and } r \textit{ are pairwise different}, |B| \geq 1\}\end{aligned}$$

We have introduced non-equality to get rid of actions that try to move within a room.

A.12 Hanoi

Definition 35 *Hanoi is the STRIPS domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\mathcal{P} = \{(smaller, 2), (clear, 1), (on, 2)\}$$

$\mathcal{O}$ *consists of the single operator*

- *move:*

$$\begin{aligned}(& (x, y, z), \\ & \{(on, x, y), (clear, x), (clear, z), (smaller, x, z)\}, \\ & \{(on, x, z), (clear, y)\}, \\ & \{(on, x, y), (clear, z)\})\end{aligned}$$

and the set of instances is

$$\begin{aligned}\mathcal{I} = \{ & (\{p_1, p_2, p_3, d_1, \dots, d_n\}, \\ & \quad \{(smaller, d_i, p_1), (smaller, d_i, p_2), (smaller, d_i, p_3) \mid 1 \leq i \leq n\} \cup \\ & \quad \{(smaller, d_i, d_j) \mid 1 \leq i < j \leq n\} \cup \\ & \quad \{(on, d_n, p_1)\} \cup \{(on, d_i, d_{i+1}) \mid 1 \leq i \leq n-1\} \cup \\ & \quad \{(clear, d_1), (clear, p_2), (clear, p_3)\}, \\ & \quad \{(on, d_n, p_3)\} \cup \{(on, d_i, d_{i+1}) \mid 1 \leq i \leq n-1\}) \\ & \mid \{p_1, p_2, p_3, d_1, \dots, d_n\} \subseteq \Sigma^+, \\ & \quad p_1, p_2, p_3, d_1, \dots, d_n \, \textit{are pairwise different}, n \geq 1\}\end{aligned}$$

In the instance encodings, p_1, p_2, and p_3 are the three pegs, while $d_1, \dots, d_n$ are the discs in increasing order of size. A move(x, y, z) action moves disc x from disc (or peg) y to disc (or peg) z.

A.13 Logistics

Definition 36 *Logistics is the STRIPS domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\begin{aligned}\mathcal{P} = \{ & (not\text{-}eq, 2), (obj, 1), (truck, 1), (location, 1), (airplane, 1), (city, 1), \\ & (airport, 1), (at, 2), (in, 2), (in\text{-}city, 2)\}\end{aligned}$$

$\mathcal{O}$ *consists of the operators*

- *drive-truck:*

$$
\begin{aligned}
(\ & (x,y,z,u),\\
& \{\ (\mathit{truck},x),(\mathit{not\text{-}eq},y,z),(\mathit{location},y),(\mathit{location},z),(\mathit{city},u),\\
& \quad (\mathit{in\text{-}city},y,u),(\mathit{in\text{-}city},z,u),(\mathit{at},x,y)\},\\
& \{(\mathit{at},x,z)\},\\
& \{(\mathit{at},x,y)\})
\end{aligned}
$$

- *fly-airplane:*

$$
\begin{aligned}
(\ & (x,y,z),\\
& \{(\mathit{airplane},x),(\mathit{not\text{-}eq},y,z),(\mathit{airport},y),(\mathit{airport},z),(\mathit{at},x,y)\},\\
& \{(\mathit{at},x,z)\},\\
& \{(\mathit{at},x,y)\})
\end{aligned}
$$

- *load-truck:*

$$
\begin{aligned}
(\ & (x,y,z),\\
& \{(\mathit{obj},x),(\mathit{truck},y),(\mathit{location},z),(\mathit{at},x,z),(\mathit{at},y,z)\},\\
& \{(\mathit{in},x,y)\},\\
& \{(\mathit{at},x,z)\})
\end{aligned}
$$

- *load-airplane:*

$$
\begin{aligned}
(\ & (x,y,z),\\
& \{(\mathit{obj},x),(\mathit{airplane},y),(\mathit{location},z),(\mathit{at},x,z),(\mathit{at},y,z)\},\\
& \{(\mathit{in},x,y)\},\\
& \{(\mathit{at},x,z)\})
\end{aligned}
$$

- *unload-truck:*

$$
\begin{aligned}
(\ & (x,y,z),\\
& \{(\mathit{obj},x),(\mathit{truck},y),(\mathit{location},z),(\mathit{in},x,y),(\mathit{at},y,z)\},\\
& \{(\mathit{at},x,z)\},\\
& \{(\mathit{in},x,y)\})
\end{aligned}
$$

- *unload-airplane:*

$$
\begin{aligned}
(\ & (x,y,z),\\
& \{(\mathit{obj},x),(\mathit{airplane},y),(\mathit{location},z),(\mathit{in},x,y),(\mathit{at},y,z)\},\\
& \{(\mathit{at},x,z)\},\\
& \{(\mathit{in},x,y)\})
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ (\ & O \cup T \cup A \cup C \cup L_1 \cup \ldots \cup L_n, \mathit{where}\ C = \{c_1, \ldots, c_n\},\\
& \{(\mathit{not\text{-}eq},x,y) \mid x,y \in L_1 \cup \ldots \cup L_n, x \neq y\} \cup\\
& \{(\mathit{obj},x) \mid x \in O\} \cup \{(\mathit{truck},x) \mid x \in T\} \cup \{(\mathit{airplane},x) \mid x \in A\} \cup
\end{aligned}
$$

$$
\begin{aligned}
&\{(\mathit{city}, x) \mid x \in C\} \cup \{(\mathit{location}, x) \mid x \in L_1 \cup \ldots \cup L_n\} \cup \\
&\{(\mathit{in\text{-}city}, x, c_i) \mid x \in L_i\} \cup \{(\mathit{airport}, l_1), \ldots, (\mathit{airport}, l_n)\} \cup \\
&\{(\mathit{at}, x, \mathit{il}(x)) \mid x \in O \cup T \cup A\}, \\
&\{(\mathit{at}, x, \mathit{gl}(x)) \mid x \in O'\}) \\
\mid\ &(O \cup T \cup A \cup C \cup L_1 \cup \ldots \cup L_n) \subseteq \Sigma^+, \\
&O, T, A, C, L_1, \ldots, L_n \textit{ pairwise disjoint}, \\
&n \geq 1, |L_i| \geq 1, l_i \in L_i \textit{ for } 1 \leq i \leq n, \\
&\mathit{il} : O \cup T \cup A \mapsto L_1 \cup \ldots \cup L_n, \forall x \in A : \mathit{il}(x) \in \{l_1, \ldots, l_n\}, \\
&O' \subseteq O, \mathit{gl} : O' \mapsto L_1 \cup \ldots \cup L_n\}
\end{aligned}
$$

In the definition of the instances, *il* and *gl* are the initial location and goal location functions, respectively. There is at least one city, each city has at least one location, airplanes are only allowed at airports (of which each city has exactly one), and only a subset of the transportable objects need to have goal locations specified. We have introduced non-equality to get rid of actions that try to drive a truck, or fly an airplane, within a single location.

A.14 Miconic-ADL

Definition 37 *Miconic-ADL is the ADL domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$
\begin{aligned}
\mathcal{P} = \{\ &(\mathit{passenger}, 1), (\mathit{not\text{-}passenger}, 1), (\mathit{floor}, 1), (\mathit{origin}, 2), (\mathit{not\text{-}origin}, 2), \\
&(\mathit{above}, 2), (\mathit{destin}, 2), (\mathit{not\text{-}destin}, 2), (\mathit{not\text{-}no\text{-}access}, 2), (\mathit{attendant}, 1), \\
&(\mathit{not\text{-}going\text{-}up}, 1), (\mathit{not\text{-}going\text{-}down}, 1), (\mathit{not\text{-}going\text{-}nonstop}, 1), \\
&(\mathit{not\text{-}never\text{-}alone}, 1), (\mathit{vip}, 1), (\mathit{not\text{-}vip}, 1), (\mathit{not\text{-}conflict\text{-}A}, 1), \\
&(\mathit{not\text{-}conflict\text{-}B}, 1), (\mathit{boarded}, 1), (\mathit{not\text{-}boarded}, 1), \\
&(\mathit{served}, 1), (\mathit{not\text{-}served}, 1), (\mathit{lift\text{-}at}, 1)\}
\end{aligned}
$$

$\mathcal{O}$ consists of the operators

- *stop:*

$$
\begin{aligned}
(\ &(x), \\
&(\mathit{floor}, x) \wedge (\mathit{lift\text{-}at}, x) \wedge \\
&(\forall y : (\mathit{not\text{-}passenger}, y) \vee (\mathit{not\text{-}no\text{-}access}, y, x) \vee (\mathit{not\text{-}boarded}, y)) \wedge \\
&(\ \forall y : ((\mathit{not\text{-}vip}, y) \vee (\mathit{served}, y)) \vee \\
&\quad \exists y : (\mathit{vip}, y) \wedge ((\mathit{origin}, y, x) \vee (\mathit{destin}, y, x))) \wedge \\
&(\forall y : (\mathit{not\text{-}going\text{-}nonstop}, y) \vee (\mathit{not\text{-}boarded}, y) \vee (\mathit{destin}, y, x)) \wedge \\
&(\ \forall y : (\mathit{not\text{-}never\text{-}alone}, y) \vee \\
&\qquad\quad (\ ((\mathit{not\text{-}boarded}, y) \vee (\mathit{destin}, y, x)) \wedge \\
&\qquad\qquad ((\mathit{served},\ y) \vee (\mathit{not\text{-}origin}, y, x))) \vee \\
&\quad \exists z : (\mathit{attendant}, z) \wedge \\
&\qquad\quad (\ ((\mathit{boarded}, z) \wedge (\mathit{not\text{-}destin}, z, x)) \vee \\
&\qquad\qquad ((\mathit{not\text{-}served}, z) \wedge (\mathit{origin}, z, x)))) \wedge \\
&(\ \forall y : (\mathit{not\text{-}conflict\text{-}A}, y) \vee
\end{aligned}
$$

$$
\begin{aligned}
&\quad\quad (\ ((\textit{not-boarded}, y) \vee (\textit{destin}, y, x)) \wedge \\
&\quad\quad\quad ((\textit{served},\ y) \vee (\textit{not-origin}, y, x))) \vee \\
&\quad \forall y : (\textit{not-conflict-B}, y) \vee \\
&\quad\quad (\ ((\textit{not-boarded}, y) \vee (\textit{destin}, y, x)) \wedge \\
&\quad\quad\quad ((\textit{served},\ y) \vee (\textit{not-origin}, y, x)))), \\
&\{\ (\ (y), \\
&\quad (\textit{passenger}, y) \wedge (\textit{boarded}, y) \wedge (\textit{destin}, y, x), \\
&\quad \{(\textit{served}, y), (\textit{not-boarded}, y)\}, \\
&\quad \{(\textit{boarded}, y), (\textit{not-served}, y)\}), \\
&\ \ (\ (y), \\
&\quad (\textit{passenger}, y) \wedge (\textit{not-served}, y) \wedge (\textit{origin}, y, x), \\
&\quad \{(\textit{boarded}, y)\}, \\
&\quad \{(\textit{not-boarded}, y)\})\})
\end{aligned}
$$

- *up:*

$$
\begin{aligned}
&(\ (x, y), \\
&\ \ (\textit{floor}, x) \wedge (\textit{floor}, y) \wedge (\textit{lift-at}, x) \wedge (\textit{above}, y, x) \wedge \\
&\ \ \forall z : (\textit{not-going-down}, z) \vee (\textit{not-boarded}, z), \\
&\ \ \{((), \top, \{(\textit{lift-at}, y)\}, \{(\textit{lift-at}, x)\})\})
\end{aligned}
$$

- *down:*

$$
\begin{aligned}
&(\ (x, y), \\
&\ \ (\textit{floor}, x) \wedge (\textit{floor}, y) \wedge (\textit{lift-at}, x) \wedge (\textit{above}, x, y) \wedge \\
&\ \ \forall z : (\textit{not-going-up}, z) \vee (\textit{not-boarded}, z), \\
&\ \ \{((), \top, \{(\textit{lift-at}, y)\}, \{(\textit{lift-at}, x)\})\})
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ (\ & P \cup F, \textit{where } F = \{\mathrm{f}_1, \ldots, \mathrm{f}_n\}, \\
& \{(\textit{passenger}, x) \mid x \in P\} \cup \\
& \{(\textit{not-passenger}, x) \mid x \in F\} \cup \{(\textit{floor}, x) \mid x \in F\} \cup \\
& \{(\textit{origin}, x, \mathit{of}(x)) \mid x \in P\} \cup \{(\textit{not-origin}, x, y) \mid x \in P, y \neq \mathit{of}(x)\} \cup \\
& \{(\textit{destin}, x, \mathit{df}(x)) \mid x \in P\} \cup \{(\textit{not-destin}, x, y) \mid x \in P, y \neq \mathit{df}(x)\} \cup \\
& \{(\textit{above}, \mathrm{f}_i, \mathrm{f}_j) \mid i < j\} \cup \{(\textit{not-no-access}, x, y) \mid x \in P, y \in F, y \notin \mathit{na}(x)\} \cup \\
& \{(\textit{not-going-up}, x) \mid x \in (F \cup P) \setminus GU\} \cup \\
& \{(\textit{not-going-down}, x) \mid x \in (F \cup P) \setminus GD\} \cup \\
& \{(\textit{not-going-nonstop}, x) \mid x \in (F \cup P) \setminus GN\} \cup \\
& \{(\textit{attendant}, x) \mid x \in AT\} \cup \{(\textit{not-never-alone}, x) \mid x \in (F \cup P) \setminus NA\} \cup \\
& \{(\textit{vip}, x) \mid x \in VI\} \cup \{(\textit{not-vip}, x) \mid x \in (F \cup P) \setminus VI\} \cup \\
& \{(\textit{not-conflict-A}, x) \mid x \in (F \cup P) \setminus CA\} \cup \\
& \{(\textit{not-conflict-B}, x) \mid x \in (F \cup P) \setminus CB\} \cup \\
& \{(\textit{not-boarded}, x), (\textit{not-served}, x) \mid x \in P\} \cup \{(\textit{lift-at}, \mathit{if})\}, \\
& \bigwedge\{(\textit{served}, x) \mid x \in P\}) \\
\mid\ \ & (P \cup F) \subseteq \Sigma^+, P \cap F = \emptyset, |F| \geq 1,
\end{aligned}
$$

$of : P \mapsto F, df : P \mapsto F, if \in F, na : P \mapsto 2^F,$
$GU \subseteq P, GD \subseteq P, GN \subseteq P, AT \subseteq P, NA \subseteq P, VI \subseteq P, CA \subseteq P, CB \subseteq P\}$

All operators are ADL operators, making use of complex preconditions to formulate the constraints concerning different types of passengers. The *move* operator has two conditional effects with one effect parameter each. The *up* and *down* operators each have a single unconditional effect. One help in understanding the rather complicated precondition of *stop*: there often appears a sub-expression of the form $((\text{boarded}, y) \wedge (\text{not-destin}, y, x)) \vee ((\text{not-served}, y) \wedge (\text{origin}, y, x))$, or the respective negated version, for a floor x and a passenger y; the expression is true if and only if y will be boarded during the next lift move, after stopping at x. In the definition of the instances, *of* and *df* are the origin and destination floors of the passengers, and *if* is the floor where the elevator (the lift) is initially positioned. The *na* function specifies, for each passenger, the set of floors he does not have access to. The sets GU, GD, GN, AT, NA, VI, CA, and CB specify the different types of passengers. In the original domain there are a lot of negations, which are formulated by the extra predicates *not-passenger*, *not-origin*, *not-destin*, *not-going-up*, *not-going-down*, *not-going-nonstop*, *not-never-alone*, *not-vip*, *non-conflict-A*, *non-conflict-B*, *not-boarded*, and *not-served*.

A.15 Miconic-SIMPLE

Definition 38 *Miconic-SIMPLE is the ADL domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$\begin{aligned}\mathcal{P} = \{ & (passenger, 1), (floor, 1), (origin, 2), (destin, 2), (above, 2), (boarded, 1), \\ & (served, 1), (not\text{-}served, 1), (lift\text{-}at, 1)\}\end{aligned}$$

$\mathcal{O}$ consists of the operators

- *stop:*

$$\begin{aligned}
&(\ (x), \\
&\quad (floor, x) \wedge (lift\text{-}at, x), \\
&\quad \{\ (\ (y), \\
&\qquad\quad (passenger, y) \wedge (boarded, y) \wedge (destin, y, x), \\
&\qquad\quad \{(served, y)\}, \\
&\qquad\quad \{(boarded, y), (not\text{-}served, y)\}), \\
&\qquad ((y), (passenger, y) \wedge (not\text{-}served, y) \wedge (origin, y, x), \{(boarded, y)\}, \emptyset)\})
\end{aligned}$$

- *up:*

$$\begin{aligned}
&(\ (x, y), \\
&\quad \{(floor, x), (floor, y), (lift\text{-}at, x), (above, y, x)\}, \\
&\quad \{(lift\text{-}at, y)\}, \\
&\quad \{(lift\text{-}at, x)\})
\end{aligned}$$

- *down:*

$$
\begin{aligned}
&(\ (x, y),\\
&\quad \{(\mathit{floor}, x), (\mathit{floor}, y), (\mathit{lift\text{-}at}, x), (\mathit{above}, x, y)\},\\
&\quad \{(\mathit{lift\text{-}at}, y)\},\\
&\quad \{(\mathit{lift\text{-}at}, x)\})
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ (\ & P \cup F, \mathit{where}\ F = \{f_1, \dots, f_n\},\\
& \{(\mathit{passenger}, x) \mid x \in P\} \cup \{(\mathit{floor}, x) \mid x \in F\} \cup\\
& \{(\mathit{above}, f_i, f_j) \mid i < j\} \cup \{(\mathit{not\text{-}served}, x) \mid x \in P\} \cup\\
& \{(\mathit{origin}, x, \mathit{of}(x)), (\mathit{destin}, x, \mathit{df}(x)) \mid x \in P\} \cup \{(\mathit{lift\text{-}at}, \mathit{if})\},\\
& \textstyle\bigwedge \{(\mathit{served}, x) \mid x \in P\})\\
\mid\ & (P \cup F) \subseteq \Sigma^+, P \cap F = \emptyset, |F| \geq 1, \mathit{of}: P \mapsto F, \mathit{df}: P \mapsto F, \mathit{if} \in F\}
\end{aligned}
$$

The only ADL element are the two conditional effects of the *move* operator, both of which have one effect parameter. In the definition of the instances, *of* and *df* are the origin and destination floors of the passengers, and *if* is the floor where the elevator (the "lift") is initially positioned. The second conditional effect of the *stop* operator requires the respective passengers not to be served already. This negative condition is formulated by the *not-served* predicate.

A.16 Miconic-STRIPS

Definition 39 *Miconic-STRIPS is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$
\begin{aligned}
\mathcal{P} = \{\ & (\mathit{passenger}, 1), (\mathit{floor}, 1), (\mathit{origin}, 2), (\mathit{destin}, 2), (\mathit{above}, 2), (\mathit{boarded}, 1),\\
& (\mathit{served}, 1), (\mathit{lift\text{-}at}, 1)\}
\end{aligned}
$$

$\mathcal{O}$ consists of the operators

- *board:*

$$((x, y), \{(\mathit{floor}, x), (\mathit{passenger}, y), (\mathit{lift\text{-}at}, x), (\mathit{origin}, y, x)\}, \{(\mathit{boarded}, y)\}, \emptyset)$$

- *depart:*

$$
\begin{aligned}
&(\ (x, y),\\
&\quad \{(\mathit{floor}, x), (\mathit{passenger}, y), (\mathit{lift\text{-}at}, x), (\mathit{destin}, y, x), (\mathit{boarded}, y)\},\\
&\quad \{(\mathit{served}, y)\},\\
&\quad \{(\mathit{boarded}, y)\})
\end{aligned}
$$

- *up:*

$$
\begin{aligned}
&(\ (x, y),\\
&\quad \{(\mathit{floor}, x), (\mathit{floor}, y), (\mathit{lift\text{-}at}, x), (\mathit{above}, y, x)\},\\
&\quad \{(\mathit{lift\text{-}at}, y)\},\\
&\quad \{(\mathit{lift\text{-}at}, x)\})
\end{aligned}
$$

- *down:*

$$
\begin{aligned}
&(\ (x, y),\\
&\quad \{(\mathit{floor}, x), (\mathit{floor}, y), (\mathit{lift\text{-}at}, x), (\mathit{above}, x, y)\},\\
&\quad \{(\mathit{lift\text{-}at}, y)\},\\
&\quad \{(\mathit{lift\text{-}at}, x)\})
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ (\ & P \cup F, \mathit{where}\ F = \{f_1, \ldots, f_n\},\\
& \{(\mathit{passenger}, x) \mid x \in P\} \cup \{(\mathit{floor}, x) \mid x \in F\} \cup \{(\mathit{above}, f_i, f_j) \mid i < j\} \cup\\
& \{(\mathit{origin}, x, \mathit{of}(x)), (\mathit{destin}, x, \mathit{df}(x)) \mid x \in P\} \cup \{(\mathit{lift\text{-}at}, \mathit{if})\},\\
& \{(\mathit{served}, x) \mid x \in P\})\\
\mid\ & (P \cup F) \subseteq \Sigma^+, P \cap F = \emptyset, |F| \geq 1, \mathit{of} : P \mapsto F, \mathit{df} : P \mapsto F, \mathit{if} \in F\}
\end{aligned}
$$

In the definition of the instances, *of* and *df* are the origin and destination floors of the passengers, and *if* is the floor where the elevator (the "lift") is initially positioned.

A.17 Movie

Definition 40 *Movie is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$
\begin{aligned}
\mathcal{P} = \{\ & (\mathit{movie\text{-}rewound}, 0), (\mathit{counter\text{-}at\text{-}other\text{-}than\text{-}two\text{-}hours}, 0),\\
& (\mathit{counter\text{-}at\text{-}zero}, 0), (\mathit{have\text{-}chips}, 0), (\mathit{have\text{-}dip}, 0), (\mathit{have\text{-}pop}, 0),\\
& (\mathit{have\text{-}cheese}, 0), (\mathit{have\text{-}crackers}, 0), (\mathit{chips}, 1),\\
& (\mathit{dip}, 1), (\mathit{pop}, 1), (\mathit{cheese}, 1), (\mathit{crackers}, 1)\}
\end{aligned}
$$

$\mathcal{O}$ consists of the operators

- *rewind-movie:*

$$
\begin{aligned}
&(\ (),\\
&\quad \{(\mathit{counter\text{-}at\text{-}other\text{-}than\text{-}two\text{-}hours})\},\\
&\quad \{(\mathit{movie\text{-}rewound})\},\\
&\quad \{(\mathit{counter\text{-}at\text{-}zero})\})
\end{aligned}
$$

- *reset-counter:* $((), \emptyset, \{(\mathit{counter\text{-}at\text{-}zero})\}, \emptyset)$
- *get-chips:* $((x), \{(\mathit{chips}, x)\}, \{(\mathit{have\text{-}chips})\}, \emptyset)$

- *get-dip:* $((x), \{(dip, x)\}, \{(have\text{-}dip)\}, \emptyset)$
- *get-pop:* $((x), \{(pop, x)\}, \{(have\text{-}pop)\}, \emptyset)$
- *get-cheese:* $((x), \{(cheese, x)\}, \{(have\text{-}cheese)\}, \emptyset)$
- *get-crackers:* $((x), \{(crackers, x)\}, \{(have\text{-}crackers)\}, \emptyset)$

and the set of instances is

$$\begin{aligned}
\mathcal{I} = \{ \ (\ & C \cup D \cup P \cup Z \cup K, \\
& \{(chips, x) \mid x \in C\} \cup \{(dip, x) \mid x \in D\} \cup \{(pop, x) \mid x \in P\} \cup \\
& \{(cheese, x) \mid x \in Z\} \cup \{(crackers, x) \mid x \in K\} \cup \\
& \{(counter\text{-}at\text{-}other\text{-}than\text{-}two\text{-}hours)\}, \\
& \{ \ (movie\text{-}rewound), (counter\text{-}at\text{-}zero), (have\text{-}chips), \\
& \quad (have\text{-}dip), (have\text{-}pop), (have\text{-}cheese), (have\text{-}crackers)\}) \\
\mid \ & (C \cup D \cup P \cup Z \cup K) \subseteq \Sigma^+, C, D, P, Z, \text{ and } K \text{ are pairwise disjoint}, \\
& |C| \geq 1, |D| \geq 1, |P| \geq 1, |Z| \geq 1, |K| \geq 1\}
\end{aligned}$$

Note that none of the effects contains an operator parameter, implying that the set of reachable states is, as said in Section 6.2, essentially the same across instances.

A.18 Mprime

Definition 41 *Mprime is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$\begin{aligned}
\mathcal{P} = \{ \ & (location, 1), (vehicle, 1), (object, 1), (fuel, 1), (space, 1), (conn, 2), (has\text{-}fuel, 2), \\
& (at, 2), (fuel\text{-}nextlower, 2), (in, 2), (has\text{-}space, 2), (space\text{-}nextlower, 2)\}
\end{aligned}$$

$\mathcal{O}$ consists of the operators

- *move:*

$$\begin{aligned}
(\ & (x, y, z, u, v), \\
& \{ \ (vehicle, x), (location, y), (location, z), (fuel, u), (fuel, v), \\
& \quad (at, x, y), (conn, y, z), (has\text{-}fuel, y, u), (fuel\text{-}nextlower, v, u)\}, \\
& \{(at, x, z), (has\text{-}fuel, y, v)\}, \\
& \{(at, x, y), (has\text{-}fuel, y, u)\})
\end{aligned}$$

- *load:*

$$\begin{aligned}
(\ & (x, y, z, u, v), \\
& \{ \ (object, x), (vehicle, y), (location, z), (space, u), (space, v), \\
& \quad (at, x, z), (at, y, z), (has\text{-}space, x, u), (space\text{-}nextlower, v, u)\}, \\
& \{(in, x, y), (has\text{-}space, y, v)\}, \\
& \{(at, x, z), (has\text{-}space, y, u)\})
\end{aligned}$$

- *unload:*

$$
\begin{array}{l}
(\ (x, y, z, u, v), \\
\quad \{\ (object, x), (vehicle, y), (location, z), (space, u), (space, v), \\
\qquad (in, x, y), (at, y, z), (has\text{-}space, x, u), (space\text{-}nextlower, u, v)\}, \\
\quad \{(at, x, z), (has\text{-}space, y, u)\}, \\
\quad \{(in, x, y), (has\text{-}space, y, v)\})
\end{array}
$$

- *transfer:*

$$
\begin{array}{l}
(\ (x, y, z, u, v, w, q), \\
\quad \{\ (location, x), (location, y), (fuel, z), (fuel, u), (fuel, v), (fuel, w), (fuel, q), \\
\qquad (conn, x, y), (has\text{-}fuel, x, z), (fuel\text{-}nextlower, u, z), (fuel\text{-}nextlower, v, u), \\
\qquad (has\text{-}fuel, y, w), (fuel\text{-}nextlower, w, q)\}, \\
\quad \{(has\text{-}fuel, x, u), (has\text{-}fuel, y, q)\}, \\
\quad \{(has\text{-}fuel, x, z), (has\text{-}fuel, y, w)\})
\end{array}
$$

and the set of instances is

$$
\begin{array}{rl}
\mathcal{I} = \{\ (& L \cup V \cup O \cup F \cup S, \\
& where\ F = \{f_0, \ldots, f_n\}, S = \{s_0, \ldots, s_m\} \\
& \{(location, x) \mid x \in L\} \cup \{(vehicle, x) \mid x \in V\} \cup \{(object, x) \mid x \in O\} \cup \\
& \{(fuel, x) \mid x \in F\} \cup \{(space, x) \mid x \in S\} \cup \\
& \{(fuel\text{-}nextlower, f_i, f_{i+1}) \mid 0 \leq i < n\} \cup \\
& \{(space\text{-}nextlower, s_i, s_{i+1}) \mid 0 \leq i < m\} \cup \\
& \{(conn, x, y) \mid (x, y) \in M\} \cup \{(has\text{-}fuel, x, f(x)) \mid x \in L\} \cup \\
& \{(has\text{-}space, x, s(x)) \mid x \in V\} \cup \\
& \{(at, x, il(x)) \mid x \in V\} \cup \{(at, x, il(x)) \mid x \in O\}, \\
& \{(at, x, gl(x)) \mid x \in O'\}) \\
\mid & (L \cup V \cup O \cup F \cup S) \subseteq \Sigma^+, L, V, O, F,\ and\ S\ are\ pairwise\ disjoint, \\
& |L| \geq 2, |F| \geq 1, |S| \geq 1, \\
& M \subseteq L \times L, f : L \mapsto F, s : V \mapsto S, il : V \cup O \mapsto L, O' \subseteq O, gl : O' \mapsto L\}
\end{array}
$$

Varying fuel and space is realized by the objects in F and S, which are ordered via the (fuel-nextlower, 2) and (space-nextlower, 2) predicates: the left hand side is the next lower fuel or space amount; f_0 respectively s_0 are the zero-amounts, having no lower neighbor. The transfer operator transfers one fuel unit from location x to a connected location y, as long as x has still at least one fuel unit left after the transaction. The locations are connected according to the edge set M, f and g determine the initial fuel and space amounts for locations and vehicles, il specifies the initial positions of vehicles and objects, finally gl are the goal locations for the subset O' of objects. We remark that in the version of the domain used at AIPS-1998 all predicates were given counter-intuitive names to conceal the nature of the domain. We have translated all names back to ease readability.

A.19 Mystery

The *Mystery* domain is exactly identical with *Mprime*, see above, except that there is no *transfer* operator.

A.20 Rovers

Definition 42 *Rovers is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$
\begin{aligned}
\mathcal{P} = \{ \; & (\mathit{waypoint}, 1), (\mathit{rover}, 1), (\mathit{lander}, 1), (\mathit{store}, 1), \\
& (\mathit{camera}, 1), (\mathit{objective}, 1), (\mathit{mode}, 1), \\
& (\mathit{wp\text{-}visible}, 2), (\mathit{ob\text{-}visible}, 2), (\mathit{can\text{-}traverse}, 3), \\
& (\mathit{at\text{-}soil\text{-}sample}, 1), (\mathit{at\text{-}rock\text{-}sample}, 1), (\mathit{at\text{-}lander}, 2), (\mathit{at}, 2), \\
& (\mathit{equipped\text{-}for\text{-}soil\text{-}analysis}, 1), (\mathit{equipped\text{-}for\text{-}rock\text{-}analysis}, 1), \\
& (\mathit{equipped\text{-}for\text{-}imaging}, 1), (\mathit{store\text{-}of}, 2), (\mathit{empty}, 1), (\mathit{full}, 1), \\
& (\mathit{on\text{-}board}, 2), (\mathit{supports}, 2), (\mathit{calibration\text{-}target}, 2), (\mathit{calibrated}, 1), \\
& (\mathit{have\text{-}rock\text{-}analysis}, 2), (\mathit{have\text{-}soil\text{-}analysis}, 2), (\mathit{have\text{-}image}, 3), \\
& (\mathit{communicated\text{-}soil\text{-}data}, 1), (\mathit{communicated\text{-}rock\text{-}data}, 1), \\
& (\mathit{communicated\text{-}image\text{-}data}, 2)\}
\end{aligned}
$$

$\mathcal{O}$ consists of the operators

- *navigate:*

$$
\begin{aligned}
(\; & (x, y, z), \\
& \{ \; (\mathit{rover}, x), (\mathit{waypoint}, y), (\mathit{waypoint}, z), \\
& \quad (\mathit{can\text{-}traverse}, x, y, z), (\mathit{wp\text{-}visible}, y, z), (\mathit{at}, x, y)\}, \\
& \{(\mathit{at}, x, z)\}, \\
& \{(\mathit{at}, x, y)\})
\end{aligned}
$$

- *sample-soil:*

$$
\begin{aligned}
(\; & (x, y, z), \\
& \{ \; (\mathit{rover}, x), (\mathit{waypoint}, y), (\mathit{store}, z), \\
& \quad (\mathit{equipped\text{-}for\text{-}soil\text{-}analysis}, x), (\mathit{store\text{-}of}, z, x), \\
& \quad (\mathit{at\text{-}soil\text{-}sample}, y), (\mathit{at}, x, y), (\mathit{empty}, z)\}, \\
& \{(\mathit{have\text{-}soil\text{-}analysis}, x, y), (\mathit{full}, z)\}, \\
& \{(\mathit{empty}, z), (\mathit{at\text{-}soil\text{-}sample}, y)\})
\end{aligned}
$$

- *sample-rock:*

$$
\begin{aligned}
(\; & (x, y, z), \\
& \{ \; (\mathit{rover}, x), (\mathit{waypoint}, y), (\mathit{store}, z), \\
& \quad (\mathit{equipped\text{-}for\text{-}rock\text{-}analysis}, x), (\mathit{store\text{-}of}, z, x), \\
& \quad (\mathit{at\text{-}rock\text{-}sample}, y), (\mathit{at}, x, y), (\mathit{empty}, z)\}, \\
& \{(\mathit{have\text{-}rock\text{-}analysis}, x, y), (\mathit{full}, z)\}, \\
& \{(\mathit{empty}, z), (\mathit{at\text{-}rock\text{-}sample}, y)\})
\end{aligned}
$$

- *drop:*

$$
\begin{aligned}
(\ & (x, y),\\
& \{(\mathit{rover}, x), (\mathit{store}, y), (\mathit{store\text{-}of}, y, x), (\mathit{full}, y)\},\\
& \{(\mathit{empty}, y)\},\\
& \{(\mathit{full}, y)\})
\end{aligned}
$$

- *calibrate:*

$$
\begin{aligned}
(\ & (x, y, z, u),\\
& \{\ (\mathit{rover}, x), (\mathit{waypoint}, y), (\mathit{camera}, z), (\mathit{objective}, u),\\
& \quad (\mathit{equipped\text{-}for\text{-}imaging}, x), (\mathit{calibration\text{-}target}, z, u), (\mathit{on\text{-}board}, z, x),\\
& \quad (\mathit{ob\text{-}visible}, y, u), (\mathit{at}, x, y)\},\\
& \{(\mathit{calibrated}, z)\},\\
& \emptyset)
\end{aligned}
$$

- *take-image:*

$$
\begin{aligned}
(\ & (x, y, z, u, v),\\
& \{\ (\mathit{rover}, x), (\mathit{waypoint}, y), (\mathit{camera}, z), (\mathit{objective}, u), (\mathit{mode}, v),\\
& \quad (\mathit{equipped\text{-}for\text{-}imaging}, x), (\mathit{on\text{-}board}, z, x), (\mathit{supports}, z, v),\\
& \quad (\mathit{ob\text{-}visible}, y, u), (\mathit{at}, x, y), (\mathit{calibrated}, z)\},\\
& \{(\mathit{have\text{-}image}, x, u, v)\},\\
& \{(\mathit{calibrated}, z)\})
\end{aligned}
$$

- *communicate-soil-data:*

$$
\begin{aligned}
(\ & (x, y, z, u, v),\\
& \{\ (\mathit{rover}, x), (\mathit{lander}, y), (\mathit{waypoint}, z), (\mathit{waypoint}, u), (\mathit{waypoint}, v),\\
& \quad (\mathit{wp\text{-}visible}, z, u), (\mathit{at\text{-}lander}, y, u), (\mathit{at}, x, z), (\mathit{have\text{-}soil\text{-}analysis}, x, v)\},\\
& \{(\mathit{communicated\text{-}soil\text{-}data}, v)\},\\
& \emptyset)
\end{aligned}
$$

- *communicate-rock-data:*

$$
\begin{aligned}
(\ & (x, y, z, u, v),\\
& \{\ (\mathit{rover}, x), (\mathit{lander}, y), (\mathit{waypoint}, z), (\mathit{waypoint}, u), (\mathit{waypoint}, v),\\
& \quad (\mathit{wp\text{-}visible}, z, u), (\mathit{at\text{-}lander}, y, u), (\mathit{at}, x, z), (\mathit{have\text{-}rock\text{-}analysis}, x, v)\},\\
& \{(\mathit{communicated\text{-}rock\text{-}data}, v)\},\\
& \emptyset)
\end{aligned}
$$

- *communicate-image-data:*

$$
\begin{aligned}
(\ & (x, y, z, u, v, w),\\
& \{\ (\mathit{rover}, x), (\mathit{lander}, y), (\mathit{waypoint}, z), (\mathit{waypoint}, u),\\
& \quad (\mathit{objective}, v), (\mathit{mode}, w),\\
& \quad (\mathit{wp\text{-}visible}, z, u), (\mathit{at\text{-}lander}, y, u), (\mathit{at}, x, z), (\mathit{have\text{-}image}, x, v, w)\},\\
& \{(\mathit{communicated\text{-}image\text{-}data}, v, w)\},\\
& \emptyset)
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ (\ & W \cup R \cup L \cup S \cup C \cup O \cup M, \\
& \{(\mathit{waypoint}, x) \mid x \in W\} \cup \{(\mathit{rover}, x) \mid x \in R\} \cup \\
& \{(\mathit{lander}, x) \mid x \in L\} \cup \{(\mathit{store}, x) \mid x \in S\} \cup \\
& \{(\mathit{camera}, x) \mid x \in C\} \cup \{(\mathit{objective}, x) \mid x \in O\} \cup \{(\mathit{mode}, x) \mid x \in M\} \cup \\
& \{(\mathit{wp\text{-}visible}, x, y) \mid (x, y) \in WV\} \cup \\
& \{(\mathit{ob\text{-}visible}, x, y) \mid (x, y) \in OV\} \cup \\
& \{(\mathit{can\text{-}traverse}, x, y, z) \mid x \in R, (y, z) \in T(x)\} \cup \\
& \{(\mathit{at\text{-}soil\text{-}sample}, x) \mid x \in W^s\} \cup \{(\mathit{at\text{-}rock\text{-}sample}, x) \mid x \in W^r\} \cup \\
& \{(\mathit{at\text{-}lander}, x, iw(x)) \mid x \in L\} \cup \{(\mathit{at}, x, iw(x)) \mid x \in R\} \cup \\
& \{(\mathit{equipped\text{-}for\text{-}soil\text{-}analysis}, x) \mid x \in R^s\} \cup \\
& \{(\mathit{equipped\text{-}for\text{-}rock\text{-}analysis}, x) \mid x \in R^r\} \cup \\
& \{(\mathit{equipped\text{-}for\text{-}imaging}, x) \mid x \in R^i\} \cup \\
& \{(\mathit{store\text{-}of}, x, so(x)) \mid x \in S\} \cup \{(\mathit{empty}, x) \mid x \in S\} \cup \\
& \{(\mathit{on\text{-}board}, x, ob(x)) \mid x \in C\} \cup \{(\mathit{supports}, x, y) \mid x \in C, y \in sp(x)\} \cup \\
& \{(\mathit{calibration\text{-}target}, x, y) \mid x \in C, y \in ct(x)\}, \\
& \{(\mathit{communicated\text{-}soil\text{-}data}, x) \mid x \in W^{sg}\} \cup \\
& \{(\mathit{communicated\text{-}rock\text{-}data}, x) \mid x \in W^{rg}\} \cup \\
& \{(\mathit{communicated\text{-}image\text{-}data}, x, y) \mid x \in O, y \in cid(x)\}) \\
\mid\ & (W \cup R \cup L \cup S \cup C \cup O \cup M) \subseteq \Sigma^+, \\
& W, R, L, S, C, O, \ \textit{and } M \ \textit{are pairwise disjoint}, \\
& |W| \geq 1, |R| \geq 1, \\
& WV \subseteq W \times W, \forall x, y : (x, y) \in WV \to (y, x) \in WV, \\
& OV \subseteq W \times O, \\
& T : R \mapsto 2^{(W \times W)}, \forall x, y, z : (y, z) \in T(x) \to (z, y) \in T(x), \\
& W^s \subseteq W, W^r \subseteq W, iw : R \cup L \mapsto W, \\
& R^s \subseteq R, R^r \subseteq R, R^i \subseteq R, \\
& so : S \mapsto R, sp : C \mapsto 2^M, ct : C \mapsto 2^O, \\
& W^{sg} \subseteq W, W^{rg} \subseteq W, cid : O \mapsto 2^M \}
\end{aligned}
$$

In the definition of the instances, WV provides the edges defining waypoint visibility, OV defines the object visibility, and $T(x)$ for a rover x defines the traversable waypoint edges. Note that WV and $T(x)$ are postulated to be bi-directional. This is a restriction that holds true in the IPC-3 generator, according to personal communication with Derek Long. The restriction does make a significant difference – in difference to undirected movability, with directed movability there can be unrecognized dead ends, see Section 9.4. The *iw* function specifies the initial waypoint locations of the rovers and landers, R^s, R^r, and R^i specify the abilities of the rovers, *so* maps the stores to the rovers they are on, *sp* says what modes the cameras support, and *ct* defines the calibration targets. W^{sg} and W^{rg} specify the soil and rock sample goals, *cid* says which image data must be communicated.

In the original version of the domain used at IPC-3, there was an additional predicate (available, 1). This was required for the rover in the precondition of navigating, and was added *as well as deleted* by all the communication actions.

All rovers were available initially. By definition, the delete effects are applied before the add effects so the predicate never changed its truth value. The intention behind the construct – according to personal communication with Derek Long – was to avoid parallel applications of navigation and communication actions using the same rover. In the sequential setting we consider here the predicate does not make any difference so it is ignored in Definition 9.4. Similarly, at IPC-3 there was a predicate (channel-free, 1) which was true of all landers initially, and was added as well as deleted by the communication actions. The predicate served to avoid parallel applications of communication actions using the same lander, and is left out of Definition 9.4 as it does not make a difference in our sequential setting. Finally, we have slightly modified the predicate names and parameter orderings to improve readability.

A.21 Schedule

Definition 43 *Schedule is the ADL domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$
\begin{aligned}
\mathcal{P} = \{\ & (\textit{not-equal}, 2), (\textit{p-cylindrical}, 1), (\textit{p-cold}, 1), (\textit{p-hot}, 1), (\textit{p-polished}, 1),\\
& (\textit{p-rough}, 1), (\textit{p-smooth}, 1), (\textit{p-roller}, 1), (\textit{p-lathe}, 1), (\textit{p-polisher}, 1),\\
& (\textit{p-grinder}, 1), (\textit{p-punch}, 1), (\textit{p-drill-press}, 1), (\textit{p-spray-painter}, 1),\\
& (\textit{p-immersion-painter}, 1), (\textit{part}, 1), (\textit{ashape}, 1), (\textit{atemperature}, 1),\\
& (\textit{surface}, 1), (\textit{color}, 1), (\textit{width}, 1), (\textit{anorient}, 1), (\textit{machine}, 1),\\
& (\textit{shape}, 2), (\textit{temperature}, 2), (\textit{surface-condition}, 2), (\textit{painted}, 2),\\
& (\textit{has-hole}, 3), (\textit{not-has-hole}, 3), (\textit{busy}, 1), (\textit{not-busy}, 1),\\
& (\textit{scheduled}, 1), (\textit{not-scheduled}, 1), (\textit{objscheduled}, 1),\\
& (\textit{not-objscheduled}, 1), (\textit{has-paint}, 2), (\textit{has-bit}, 2), (\textit{can-orient}, 2)\}
\end{aligned}
$$

$\mathcal{O}$ *consists of the operators*

- *do-roll:*

$$
\begin{aligned}
&(\ (x, y, z, u),\\
&\quad (\textit{part}, x) \wedge (\textit{p-roller}, y) \wedge (\textit{p-hot}, z) \wedge\\
&\quad (\textit{p-cylindrical}, u) \wedge (\textit{not-busy}, \textit{roller}) \wedge (\textit{not-scheduled}, x),\\
&\quad \{\ (\ (),\\
&\qquad\quad \top,\\
&\qquad\quad \{(\textit{busy}, y), (\textit{scheduled}, x), (\textit{temperature}, x, z), (\textit{shape}, x, u)\},\\
&\qquad\quad \{(\textit{not-busy}, \textit{roller}), (\textit{not-scheduled}, x)\}),\\
&\qquad ((), (\textit{not-objscheduled}), \{(\textit{objscheduled})\}, \{(\textit{not-objscheduled})\}),\\
&\qquad ((v), (\textit{surface}, v) \wedge (\textit{surface-condition}, x, v), \emptyset, \{(\textit{surface-condition}, x, v)\}),\\
&\qquad ((v), (\textit{color}, v) \wedge (\textit{painted}, x, v), \emptyset, \{(\textit{painted}, x, v)\}),\\
&\qquad ((v), (\textit{ashape}, v) \wedge (\textit{shape}, x, v) \wedge (\textit{not-equal}, v, u), \emptyset, \{(\textit{shape}, x, v)\}),\\
&\qquad (\ (v),\\
&\qquad\quad (\textit{atemperature}, v) \wedge (\textit{temperature}, x, v) \wedge (\textit{not-equal}, y, z),\\
&\qquad\quad \emptyset,
\end{aligned}
$$

$\{(temperature, x, v)\}),$
$(\ (v, w),$
$(width, v) \wedge (anorient, w) \wedge (has\text{-}hole, x, v, w),$
$\{(not\text{-}has\text{-}hole, x, v, w)\},$
$\{(has\text{-}hole, x, v, w)\})\})$

- *do-lathe:*

$(\ (x, y, z, u),$
$(part, x) \wedge (p\text{-}lathe, y) \wedge (p\text{-}rough, z) \wedge (p\text{-}cylindrical, u) \wedge$
$(not\text{-}busy, lathe) \wedge (not\text{-}scheduled, x),$
$\{\ (\ (),$
$\top,$
$\{(busy, y), (scheduled, x), (surface\text{-}condition, x, z), (shape, x, u)\},$
$\{(not\text{-}busy, y), (not\text{-}scheduled, x)\}),$
$((), (not\text{-}objscheduled), \{(objscheduled)\}, \{(not\text{-}objscheduled)\}),$
$((v), (color, v) \wedge (painted, x, v), \emptyset, \{(painted, x, v)\}),$
$((v), (ashape, v) \wedge (shape, x, v) \wedge (not\text{-}equal, v, u), \emptyset, \{(shape, x, v)\}),$
$(\ (v),$
$(surface, v) \wedge (surface\text{-}condition, x, v) \wedge (not\text{-}equal, v, z),$
$\emptyset,$
$\{(surface\text{-}condition, x, v)\})\})$

- *do-polish:*

$(\ (x, y, z, u),$
$(part, x) \wedge (p\text{-}polisher, y) \wedge (p\text{-}polished, z) \wedge (p\text{-}cold, u) \wedge$
$(not\text{-}busy, y) \wedge (not\text{-}scheduled, x) \wedge (temperature, x, u),$
$\{\ (\ (),$
$\top,$
$\{(busy, y), (scheduled, x), (surface\text{-}condition, x, z)\},$
$\{(not\text{-}busy, y), (not\text{-}scheduled, x)\}),$
$((), (not\text{-}objscheduled), \{(objscheduled)\}, \{(not\text{-}objscheduled)\}),$
$(\ (v),$
$(surface, v) \wedge (surface\text{-}condition, x, v) \wedge (not\text{-}equal, v, polished),$
$\emptyset,$
$\{(surface\text{-}condition, x, v)\})\})$

- *do-grind:*

$(\ (x, y, z),$
$(part, x) \wedge (p\text{-}grinder, y) \wedge (p\text{-}smooth, z) \wedge$
$(not\text{-}busy, y) \wedge (not\text{-}scheduled, x),$
$\{\ (\ (),$

$\top$,
$\{(busy, y), (scheduled, x), (surface\text{-}condition, x, z)\}$,
$\{(not\text{-}busy, y), (not\text{-}scheduled, x)\})$,
$((), (not\text{-}objscheduled), \{(objscheduled)\}, \{(not\text{-}objscheduled)\})$,
$((u), (color, u) \wedge (painted, x, u), \emptyset, \{(painted, x, u)\})$,
$(\ (u)$,
$(surface, u) \wedge (surface\text{-}condition, x, u) \wedge (not\text{-}equal, y, z)$,
$\emptyset$,
$\{(surface\text{-}condition, x, u)\})\})$

- *do-punch:*

$(\ (x, y, z, u, v, w)$,
$(part, x) \wedge (width, y) \wedge (anorient, z) \wedge (p\text{-}punch, u) \wedge (p\text{-}cold, v) \wedge$
$(p\text{-}rough, w) \wedge (not\text{-}busy, u) \wedge (not\text{-}scheduled, x) \wedge (not\text{-}has\text{-}hole, x, y, z) \wedge$
$(has\text{-}bit, u, y) \wedge (can\text{-}orient, u, z) \wedge (temperature, x, v)$,
$\{\ (\ ()$,
$\top$,
$\{(busy, u), (scheduled, x), (has\text{-}hole, x, y, z), (surface\text{-}condition, x, w)\}$,
$\{(not\text{-}busy, u), (not\text{-}scheduled, x), (not\text{-}has\text{-}hole, x, y, z)\})$,
$((), (not\text{-}objscheduled), \{(objscheduled)\}, \{(not\text{-}objscheduled)\})$,
$(\ (t)$,
$(surface, t) \wedge (surface\text{-}condition, x, t) \wedge (not\text{-}equal, t, w)$,
$\emptyset$,
$\{(surface\text{-}condition, x, t)\})\})$

- *do-drill-press:*

$(\ (x, y, z, u, v)$,
$(part, x) \wedge (width, y) \wedge (anorient, z) \wedge (p\text{-}drill\text{-}press, u) \wedge (p\text{-}cold, v) \wedge$
$(not\text{-}busy, u) \wedge (not\text{-}scheduled, x) \wedge (not\text{-}has\text{-}hole, x, y, z) \wedge$
$(has\text{-}bit, u, y) \wedge (can\text{-}orient, u, z) \wedge (temperature, x, v)$,
$\{\ (\ ()$,
$\top$,
$\{(busy, u), (scheduled, x), (has\text{-}hole, x, y, z)\}$,
$\{(not\text{-}busy, u), (not\text{-}scheduled, x), (not\text{-}has\text{-}hole, x, y, z)\})$,
$((), (not\text{-}objscheduled), \{(objscheduled)\}, \{(not\text{-}objscheduled)\})\})$

- *do-spray-paint:*

$(\ (x, y, z, u)$,
$(part, x) \wedge (color, y) \wedge (p\text{-}spray\text{-}painter, z) \wedge (p\text{-}cold, u) \wedge$
$(not\text{-}busy, z) \wedge (not\text{-}scheduled, x) \wedge (has\text{-}paint, z, y) \wedge (temperature, x, u)$,

$$
\begin{array}{l}
\{\ (\ (),\\
\quad\ \top,\\
\quad\ \{(\mathit{busy}, z), (\mathit{scheduled}, x), (\mathit{painted}, x, y)\},\\
\quad\ \{(\mathit{not\text{-}busy}, z), (\mathit{not\text{-}scheduled}, x)\}),\\
\quad ((), (\mathit{not\text{-}objscheduled}), \{(\mathit{objscheduled})\}, \{(\mathit{not\text{-}objscheduled})\}),\\
\quad ((v), (\mathit{color}, v) \wedge (\mathit{painted}, x, v) \wedge (\mathit{not\text{-}equal}, v, y), \emptyset, \{(\mathit{painted}, x, v)\}),\\
\quad (\ (v),\\
\quad\ (\mathit{surface}, v) \wedge (\mathit{surface\text{-}condition}, x, v),\\
\quad\ \emptyset,\\
\quad\ \{(\mathit{surface\text{-}condition}, x, v)\})\})
\end{array}
$$

- *do-immersion-paint:*

$$
\begin{array}{l}
(\ (x, y, z),\\
\ (\mathit{part}, x) \wedge (\mathit{color}, y) \wedge (\mathit{p\text{-}immersion\text{-}painter}, z) \wedge\\
\ (\mathit{not\text{-}busy}, z) \wedge (\mathit{not\text{-}scheduled}, x) \wedge (\mathit{has\text{-}paint}, z, y),\\
\ \{\ (\ (),\\
\quad\ \top,\\
\quad\ \{(\mathit{busy}, z), (\mathit{scheduled}, x), (\mathit{painted}, x, y)\},\\
\quad\ \{(\mathit{not\text{-}busy}, z), (\mathit{not\text{-}scheduled}, x)\}),\\
\quad ((), (\mathit{not\text{-}objscheduled}), \{(\mathit{objscheduled})\}, \{(\mathit{not\text{-}objscheduled})\}),\\
\quad ((u), (\mathit{color}, u) \wedge (\mathit{painted}, x, u) \wedge (\mathit{not\text{-}equal}, u, y), \emptyset, \{(\mathit{painted}, x, u)\})\})
\end{array}
$$

- *do-time-step:*

$$
\begin{array}{l}
(\ (),\\
\ (\mathit{objscheduled}),\\
\ \{\ ((x), (\mathit{part}, x) \wedge (\mathit{scheduled}, x), \emptyset, \{(\mathit{scheduled}, x)\}),\\
\quad ((x), (\mathit{machine}, x) \wedge (\mathit{busy}, x), \emptyset, \{(\mathit{busy}, x)\})\})
\end{array}
$$

and the set of instances is

$$
\begin{array}{l}
\mathcal{I} = \{\ (\ \{\ \mathit{cylindrical}, \mathit{cold}, \mathit{hot}, \mathit{polished}, \mathit{rough}, \mathit{smooth}, \mathit{roller}, \mathit{lathe}, \mathit{polisher},\\
\qquad\qquad \mathit{grinder}, \mathit{punch}, \mathit{drill\text{-}press}, \mathit{spray\text{-}painter}, \mathit{immersion\text{-}painter}\} = C' \cup\\
\qquad P \cup S \cup L \cup W \cup O,\\
\qquad \{(\mathit{not\text{-}equal}, x, y) \mid x, y \in C, x \neq y\} \cup\\
\qquad \{\ (\mathit{p\text{-}cylindrical}, \mathit{cylindrical}), (\mathit{p\text{-}cold}, \mathit{cold}), (\mathit{p\text{-}hot}, \mathit{hot}),\\
\qquad\quad \{(\mathit{p\text{-}polished}, \mathit{polished}), (\mathit{p\text{-}rough}, \mathit{rough}), (\mathit{p\text{-}smooth}, \mathit{smooth}),\\
\qquad\quad \{(\mathit{p\text{-}roller}, \mathit{roller}), (\mathit{p\text{-}lathe}, \mathit{lathe}), (\mathit{p\text{-}polisher}, \mathit{polisher}),\\
\qquad\quad \{(\mathit{p\text{-}grinder}, \mathit{grinder}), (\mathit{p\text{-}punch}, \mathit{punch}), (\mathit{p\text{-}drill\text{-}press}, drill - press),\\
\qquad\quad \{(\mathit{p\text{-}spray\text{-}painter}, spray - painter),\\
\qquad\quad \{(\mathit{p\text{-}immersion\text{-}painter}, immersion - painter)\} \cup\\
\qquad \{(\mathit{part}, x) \mid x \in P\} \cup \{(\mathit{ashape}, \mathit{cylindrical})\} \cup \{(\mathit{ashape}, x) \mid x \in S\} \cup
\end{array}
$$

$$
\begin{aligned}
&\{(atemperature, cold), (atemperature, hot)\}\cup \\
&\{(surface, polished), (surface, rough), (surface, smooth)\}\cup \\
&\{(color, x) \mid x \in L\} \cup \{(width, x) \mid x \in W\} \cup \{(anorient, x) \mid x \in O\}\cup \\
&\{\ (machine, polisher), (machine, roller), (machine, lathe), \\
&\quad (machine, grinder), (machine, punch), (machine, drill\text{-}press), \\
&\quad ((machine, spray\text{-}painter), (machine, immersion\text{-}painter)\}\cup \\
&\{(has\text{-}paint, spray\text{-}painter, x) \mid x \in L\}\cup \\
&\{(has\text{-}paint, immersion\text{-}painter, x) \mid x \in L\}\cup \\
&\{(has\text{-}bit, punch, x), (has\text{-}bit, drill\text{-}press, x) \mid x \in W\}\cup \\
&\{(can\text{-}orient, punch, x), (can\text{-}orient, drill\text{-}press, x) \mid x \in O\}\cup \\
&\{(temperature, x, cold) \mid x \in P\} \cup \{(shape, x, is(x)) \mid x \in P\}\cup \\
&\{(surface\text{-}condition, x, ic(x)) \mid x \in P\} \cup \{(painted, x, ip(x)) \mid x \in P^{ip}\}\cup \\
&\{(has\text{-}hole, x, y, z) \mid (y, z) \in ih(x)\}, \\
&\bigwedge\{(shape, x, cylindrical) \mid x \in P^{gs}\}\wedge \\
&\bigwedge\{(has\text{-}hole, x, y, z) \mid (y, z) \in gh(x)\}\wedge \\
&\bigwedge\{(painted, x, gp(x)) \mid x \in P^{gp}\}\wedge \\
&\bigwedge\{(surface\text{-}condition, x, gc(x)) \mid x \in P^{gc}\}) \\
\mid\ &(C' \cup P \cup S \cup L \cup W \cup O) \subseteq \Sigma, \\
&C', P, S, L, W, \ \ and\ O\ \ are\ pairwise\ \ disjoint, \\
&|P| \geq 1, |L| \geq 1, |W| \geq 1, |O| \geq 1, is : P \mapsto \{cylindrical\} \cup S, \\
&ic : P \mapsto \{polished, rough, smooth\}, \\
&P^{ip} \subseteq P, ip : P^{ip} \mapsto L, ih : P \mapsto 2^{(W \times O)}, \\
&P^{gs} \subseteq P, P^{gc} \subseteq P, gc : P^{gc} \mapsto \{polished, rough, smooth\}, \\
&P^{gp} \subseteq P, gp : P^{gp} \mapsto L, gh : P \mapsto 2^{(W \times O)}\}
\end{aligned}
$$

All *Schedule* operators are ADL operators, and make use of conditional effects, sometimes with effect parameters, sometimes without effect parameters; complex conditions are not used. All instances share the constants *cylindrical*, *cold*, *hot*, *polished*, *rough*, *smooth*, *roller*, *lathe*, *polisher*, *grinder*, *punch*, *drill-press*, *spray-painter*, and *immersion-painter*: the relevant shape, temperatures, surface conditions, and machines. In the operators, these are all encoded via specific predicates, which deviates slightly from the original operators that explicitly refer to these constants. We have avoided such "domain constants" in order to keep our formal framework (i.e., the framework for ADL) as simple as possible. After removing static predicates, there is no difference at all.

Initially, all parts are cold, and have a shape and a surface condition specified. Some of the parts are also painted initially, and a part can have none or several holes. In the goal condition, some of the parts can be required to have cylindrical shape (the only shape that can be produced by the machines), some need an arbitrary surface condition (note that only the three given surface conditions can be produced by the machines), some must be painted, and each part can be required to have an arbitrary number of holes (note here that the parts can not be required to have *no* holes). In the original domain there are various negations, which are formulated by the extra predicates *not-equal* (the original formulation makes use of an *equality* predicate which is negated in all its appearances), *not-has-hole*, *not-busy*, *not-scheduled*, and *not-objscheduled*. It might seem restrictive that all appropriate machines can achieve all the respective goal conditions (like both painting machines having access to all colors), but that's what's the case in the published benchmarks.

A.22 Satellite

Definition 44 *Satellite is the STRIPS domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\begin{aligned}\mathcal{P} = \{\ & (\textit{not-eq}, 2), (\textit{satellite}, 1), (\textit{direction}, 1), (\textit{instrument}, 1), (\textit{mode}, 1),\\ & (\textit{on-board}, 2), (\textit{supports}, 2), (\textit{pointing}, 2), (\textit{power-avail}, 1),\\ & (\textit{power-on}, 1), (\textit{calibrated}, 1), (\textit{have-image}, 2), (\textit{calibration-target}, 2)\}\end{aligned}$$

$\mathcal{O}$ *consists of the operators*

- *turn-to:*

$$\begin{aligned}(\ & (x, y, z),\\ & \{(\textit{not-eq}, y, z), (\textit{satellite}, x), (\textit{direction}, y), (\textit{direction}, z), (\textit{pointing}, x, y)\},\\ & \{(\textit{pointing}, x, z)\},\\ & \{(\textit{pointing}, x, y)\})\end{aligned}$$

- *switch-on:*

$$\begin{aligned}(\ & (x, y),\\ & \{(\textit{instrument}, x), (\textit{satellite}, y), (\textit{on-board}, x, y), (\textit{power-avail}, y)\},\\ & \{(\textit{power-on}, x)\},\\ & \{(\textit{calibrated}, x), (\textit{power-avail}, y)\})\end{aligned}$$

- *switch-off:*

$$\begin{aligned}(\ & (x, y),\\ & \{(\textit{instrument}, x), (\textit{satellite}, y), (\textit{on-board}, x, y), (\textit{power-on}, x)\},\\ & \{(\textit{power-avail}, y)\},\\ & \{(\textit{power-on}, x)\})\end{aligned}$$

- *calibrate:*

$$\begin{aligned}(\ & (x, y, z),\\ & \{\ (\textit{satellite}, x), (\textit{instrument}, y), (\textit{direction}, z),\\ & \quad (\textit{on-board}, y, x), (\textit{calibration-target}, y, z), (\textit{pointing}, x, z), (\textit{power-on}, y)\},\\ & \{(\textit{calibrated}, y)\},\\ & \emptyset)\end{aligned}$$

- *take-image:*

$$
\begin{aligned}
&(\ (x, y, z, u),\\
&\quad \{\ (\textit{satellite}, x), (\textit{direction}, y), (\textit{instrument}, z), (\textit{mode}, u), (\textit{calibrated}, z),\\
&\quad\quad (\textit{on-board}, z, x), (\textit{supports}, z, u), (\textit{power-on}, z), (\textit{pointing}, x, y)\},\\
&\quad \{(\textit{have-image}, y, u)\},\\
&\quad \emptyset)
\end{aligned}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ (\ & S \cup D \cup I \cup M,\\
& \{(\textit{satellite}, x) \mid x \in S\} \cup \{(\textit{direction}, x) \mid x \in D\} \cup\\
& \{(\textit{instrument}, x) \mid x \in P\} \cup \{(\textit{mode}, x) \mid x \in P\} \cup\\
& \{(\textit{power-avail}, x) \mid x \in S\} \cup \{(\textit{pointing}, x, ip(x)) \mid x \in S\} \cup\\
& \{(\textit{on-board}, x, ob(x)) \mid x \in I\} \cup \{(\textit{supports}, x, y) \mid x \in I, y \in sp(x)\} \cup\\
& \{(\textit{calibration-target}, x, y) \mid x \in I, y \in ct(x)\},\\
& \{(\textit{pointing}, x, gp(x)) \mid x \in S^g\} \cup\\
& \{(\textit{have-image}, x, y) \mid x \in D, y \in hi(x)\})\\
\mid\ & (S \cup D \cup I \cup M) \subseteq \Sigma^+, S, D, I, \textit{ and } M \textit{ are pairwise disjoint},\\
& |S| \geq 1, |D| \geq 1,\\
& ip : S \mapsto D, ob : I \mapsto S, sp : I \mapsto 2^M, ct : I \mapsto 2^D,\\
& S^g \subseteq S, gp : S^g \mapsto D, hi : D \mapsto 2^M\}
\end{aligned}
$$

In the definition of the instances, the *ip* function defines the initial orientations of the satellites, *ob* maps the instruments to the satellites they are on, *sp* defines which modes the instruments support, and *ct* says which directions are calibration targets for which instruments. The satellites S^g are, for the goal, required to point into the direction given by *gp*. Finally, *hi* defines which images need to be taken.

The initial states are such that each satellite points in exactly one direction, each instrument is on board of exactly one satellite, each satellite (but no instrument) has power available, and no instrument is calibrated. The first two are purely physical restrictions, the third restriction is just one way of saying that each satellite has the power to run one instrument at a time; the last restriction seems natural. For a more detailed discussion see Section 8.2.

A.23 Simple-Tsp

Definition 45 *Simple-Tsp is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$\mathcal{P} = \{(\textit{not-eq}, 2), (\textit{at}, 1), (\textit{visited}, 1)\}$$

$\mathcal{O}$ consists of the single operator

- *move:* $((x, y), \{(\textit{not-eq}, x, y), (\textit{at}, x)\}, \{(\textit{at}, y), (\textit{visited}, y)\}, \{(\textit{at}, x)\})$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{\ (& L,\\
& \{(\textit{not-eq}, x, y) \mid x, y \in L, x \neq y\} \cup \{(\textit{at}, l_0)\},\\
& \{(\textit{visited}, x) \mid x \in L\})\\
\mid\ & L \subseteq \Sigma^+, |L| \geq 2, l_0 \in L\}
\end{aligned}
$$

We have introduced non-equality to get rid of actions that try to move within a location.

A.24 Tireworld

Definition 46 *Tireworld is the STRIPS domain* $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ *where the set of predicates is*

$$\begin{aligned}\mathcal{P} = \{ & (\mathit{wheel}, 1), (\mathit{nut}, 1), (\mathit{obj}, 1), (\mathit{hub}, 1), (\mathit{ppump}, 1), (\mathit{pwrench}, 1), (\mathit{pjack}, 1), \\ & (\mathit{open\text{-}boot}, 0), (\mathit{closed\text{-}boot}, 0), (\mathit{in\text{-}boot}, 1), (\mathit{have}, 1), (\mathit{intact}, 1), \\ & (\mathit{inflated}, 1), (\mathit{not\text{-}inflated}, 1), (\mathit{loose}, 2), (\mathit{tight}, 2), (\mathit{on\text{-}ground}, 1), \\ & (\mathit{not\text{-}on\text{-}ground}, 1), (\mathit{fastened}, 1), (\mathit{not\text{-}fastened}, 1), (\mathit{free}, 1), (\mathit{on}, 2)\}\end{aligned}$$

$\mathcal{O}$ *consists of the operators*

- open-boot*:* $((), \{(\mathit{closed\text{-}boot})\}, \{(\mathit{open\text{-}boot})\}, \{(\mathit{closed\text{-}boot})\})$
- close-boot*:* $((), \{(\mathit{open\text{-}boot})\}, \{(\mathit{closed\text{-}boot})\}, \{(\mathit{open\text{-}boot})\})$
- fetch*:* $((x), \{(\mathit{obj}, x), (\mathit{in\text{-}boot}, x), (\mathit{open\text{-}boot})\}, \{(\mathit{have}, x)\}, \{(\mathit{in\text{-}boot}, x)\})$
- put-away*:* $((x), \{(\mathit{obj}, x), (\mathit{have}, x), (\mathit{open\text{-}boot})\}, \{(\mathit{in\text{-}boot}, x)\}, \{(\mathit{have}, x)\})$
- inflate*:*

$$\begin{aligned}(\; & (x, y), \\ & \{(\mathit{wheel}, x), (\mathit{ppump}, y), (\mathit{have}, y), (\mathit{not\text{-}inflated}, x), (\mathit{intact}, x)\}, \\ & \{(\mathit{inflated}, x)\}, \\ & \{(\mathit{not\text{-}inflated}, x)\})\end{aligned}$$

- loosen*:*

$$\begin{aligned}(\; & (x, y, z), \\ & \{(\mathit{nut}, x), (\mathit{hub}, y), (\mathit{pwrench}, z), (\mathit{have}, z), (\mathit{tight}, x, y), (\mathit{on\text{-}ground}, y)\}, \\ & \{(\mathit{loose}, x, y)\}, \\ & \{(\mathit{tight}, x, y)\})\end{aligned}$$

- tighten*:*

$$\begin{aligned}(\; & (x, y, z), \\ & \{(\mathit{nut}, x), (\mathit{hub}, y), (\mathit{pwrench}, z), (\mathit{have}, z), (\mathit{loose}, x, y), (\mathit{on\text{-}ground}, y)\}, \\ & \{(\mathit{tight}, x, y)\}, \\ & \{(\mathit{loose}, x, y)\})\end{aligned}$$

- jack-up*:*

$$
\begin{array}{l}
(\ (x,y),\\
\quad \{(hub,x),(pjack,y),(have,y),(on\text{-}ground,x)\},\\
\quad \{(not\text{-}on\text{-}ground,x)\},\\
\quad \{(have,y),(on\text{-}ground,x)\})
\end{array}
$$

- *jack-down:*

$$
\begin{array}{l}
(\ (x,y),\\
\quad \{(hub,x),(pjack,y),(not\text{-}on\text{-}ground,x)\},\\
\quad \{(have,y),(on\text{-}ground,x)\},\\
\quad \{(not\text{-}on\text{-}ground,x)\})
\end{array}
$$

- *undo:*

$$
\begin{array}{l}
(\ (x,y,z),\\
\quad \{\ (nut,x),(hub,y),(pwrench,z),\\
\qquad (have,z),(fastened,y),(loose,x,y),(not\text{-}on\text{-}ground,y)\},\\
\quad \{(have,x),(not\text{-}fastened,y)\},\\
\quad \{(fastened,y),(loose,x,y)\})
\end{array}
$$

- *do-up:*

$$
\begin{array}{l}
(\ (x,y,z),\\
\quad \{\ (nut,x),(hub,y),(pwrench,z),\\
\qquad (have,z),(have,x),(not\text{-}fastened,y),(not\text{-}on\text{-}ground,y)\},\\
\quad \{(fastened,y),(loose,x,y)\},\\
\quad \{(have,x),(not\text{-}fastened,y)\})
\end{array}
$$

- *remove-wheel:*

$$
\begin{array}{l}
(\ (x,y),\\
\quad \{(wheel,x),(hub,y),(on,x,y),(not\text{-}on\text{-}ground,y),(not\text{-}fastened,y)\},\\
\quad \{(have,x),(free,y)\},\\
\quad \{(on,x,y)\})
\end{array}
$$

- *put-on-wheel:*

$$
\begin{array}{l}
(\ (x,y),\\
\quad \{\ (wheel,x),(hub,y),(have,x),(free,y),\\
\qquad (not\text{-}on\text{-}ground,y),(not\text{-}fastened,y)\},\\
\quad \{(on,x,y)\},\\
\quad \{(have,x),(free,y)\})
\end{array}
$$

and the set of instances is

$$
\begin{aligned}
\mathcal{I} = \{ \ (\ & \{pump, wrench, jack\} \cup N \cup H \cup W, \\
& where \ N = \{n_1, \dots, n_n\}, H = \{h_1, \dots, h_n\}, W = \{w_1, s_1, \dots, w_n, s_n\}, \\
& \{(obj, pump), (obj, wrench), (obj, jack)\} \cup \{(nut, x) \mid x \in N\} \cup \\
& \{(hub, x) \mid x \in H\} \cup \{(wheel, x) \mid x \in W\} \cup \{(obj, x) \mid x \in N \cup W\} \cup \\
& \{(ppump, pump), (pwrench, wrench), (pjack, jack)\} \cup \\
& \{(on, w_i, h_i), (on\text{-}ground, h_i), (tight, n_i, h_i) \mid 1 \le i \le n\} \cup \\
& \{(fastened, h_i), (intact, s_i), (in\text{-}boot, s_i), (not\text{-}inflated, s_i) \mid 1 \le i \le n\} \cup \\
& \{(in\text{-}boot, pump), (in\text{-}boot, wrench), \} \cup \\
& \{(in\text{-}boot, jack), (closed\text{-}boot)\}, \\
& \{(on, s_i, h_i), (inflated, s_i), (tight, n_i, h_i), (in\text{-}boot, w_i) \mid 1 \le i \le n\} \cup \\
& \{(in\text{-}boot, pump), (in\text{-}boot, wrench), (in\text{-}boot, jack), (closed\text{-}boot)\}) \\
\mid \ & \{pump, wrench, jack\} \cup N \cup H \cup W \subseteq \Sigma^+, \\
& \{pump, wrench, jack\}, N, H, \ and \ W \ are \ pairwise \ disjoint, n \ge 1\}
\end{aligned}
$$

All instances share the constants *pump*, *wrench*, and *jack*, i.e., the tools required for performing the different working steps. These are encoded in the operators by using a specific predicate for each tool. We remark that the original operators directly refer to the tools as constants. We have avoided such "domain constants" in order to keep our formal framework (i.e., the framework for ADL) as simple as possible. After removing static predicates, there is no difference at all.

A.25 Zenotravel

Definition 47 *Zenotravel is the STRIPS domain $(\mathcal{P}, \mathcal{O}, \mathcal{I})$ where the set of predicates is*

$$
\begin{aligned}
\mathcal{P} = \{ \ & (not\text{-}eq, 2), (city, 1), (aircraft, 1), (person, 1), (flevel, 1), \\
& (at, 2), (in, 2), (fuel\text{-}level, 2), (next, 2)\}
\end{aligned}
$$

$\mathcal{O}$ consists of the operators

- *board:*

$$
\begin{aligned}
(\ & (x, y, z), \\
& \{(person, x), (aircraft, y), (city, z), (at, x, z), (at, y, z)\}, \\
& \{(in, x, y)\}, \\
& \{(at, x, z)\})
\end{aligned}
$$

- *debark:*

$$
\begin{aligned}
(\ & (x, y, z), \\
& \{(person, x), (aircraft, y), (city, z), (in, x, y), (at, y, z)\}, \\
& \{(at, x, z)\}, \\
& \{(in, x, y)\})
\end{aligned}
$$

- *fly:*

$$
\begin{array}{l}
(\ (x,y,z,u,v), \\
\quad \{\ (\mathit{not\text{-}eq},y,z),(\mathit{aircraft},x),(\mathit{city},y),(\mathit{city},z),(\mathit{flevel},u),(\mathit{flevel},v), \\
\quad\quad (\mathit{at},x,y),(\mathit{fuel\text{-}level},x,u),(\mathit{next},v,u)\}, \\
\quad \{(\mathit{at},x,z),(\mathit{fuel\text{-}level},x,v)\}, \\
\quad \{(\mathit{at},x,y),(\mathit{fuel\text{-}level},x,u)\})
\end{array}
$$

- *zoom:*

$$
\begin{array}{l}
(\ (x,y,z,u,v,w), \\
\quad \{\ (\mathit{not\text{-}eq},y,z),(\mathit{aircraft},x),(\mathit{city},y),(\mathit{city},z), \\
\quad\quad (\mathit{flevel},u),(\mathit{flevel},v),(\mathit{flevel},w), \\
\quad\quad (\mathit{at},x,y),(\mathit{fuel\text{-}level},x,u),(\mathit{next},v,u),(\mathit{next},w,v)\}, \\
\quad \{(\mathit{at},x,z),(\mathit{fuel\text{-}level},x,w)\}, \\
\quad \{(\mathit{at},x,y),(\mathit{fuel\text{-}level},x,u)\})
\end{array}
$$

- *refuel:*

$$
\begin{array}{l}
(\ (x,y,z,u), \\
\quad \{\ (\mathit{aircraft},x),(\mathit{city},y),(\mathit{flevel},z),(\mathit{flevel},u), \\
\quad\quad (\mathit{at},x,y),(\mathit{fuel\text{-}level},x,z),(\mathit{next},z,u)\}, \\
\quad \{(\mathit{fuel\text{-}level},x,u)\}, \\
\quad \{(\mathit{fuel\text{-}level},x,z)\})
\end{array}
$$

and the set of instances is

$$
\begin{array}{rl}
\mathcal{I} = \{\ (& C \cup A \cup P \cup F, \mathit{where}\ F = \{f_0,\ldots,f_n\}, \\
& \{(\mathit{city},x) \mid x \in C\} \cup \{(\mathit{aircraft},x) \mid x \in A\} \cup \{(\mathit{person},x) \mid x \in P\} \cup \\
& \{(\mathit{flevel},x) \mid x \in F\} \cup \{(\mathit{next},f_i,f_{i+1}) \mid 0 \leq i < n\} \cup \\
& \{(\mathit{fuel\text{-}level},x,f(x)) \mid x \in A\} \cup \\
& \{(\mathit{at},x,il(x)) \mid x \in A\} \cup \{(\mathit{at},x,il(x)) \mid x \in P\}, \\
& \{(\mathit{at},x,gl(x)) \mid x \in P^g\}) \\
\mid & (C \cup A \cup P \cup F) \subseteq \Sigma^+, C, A, P,\ \mathit{and}\ F\ \mathit{are\ pairwise\ disjoint}, \\
& |C| \geq 1, |F| \geq 1, \\
& f : A \mapsto F, il : A \cup P \mapsto C, P^g \subseteq P, gl : P^g \mapsto C\}
\end{array}
$$

In the definition of the instances, varying fuel is realized by the objects in F, which are ordered via the $(\mathrm{next}, 2)$ predicate: the left hand side is the next lower fuel level; f_0 is the zero-amount, having no lower neighbor. The function f determines the initial fuel levels of the aircrafts, il specifies the initial positions of aircrafts and persons, finally gl are the goal positions for the subset P^g of persons that must be transported.

References

[1] *Proceedings of the 17th National Conference of the American Association for Artificial Intelligence (AAAI-00)*, Austin, TX, July 2000. MIT Press.

[2] Fahiem Bacchus. The AIPS'00 planning competition. *The AI Magazine*, 22(3):47–56, 2001.

[3] Piergioirgio Bertoli, Alessandro Cimatti, John Slaney, and Sylvie Thiebaux. Solving power supply restoration problems with planning via symbolic model checking. In ECAI-02 [17], pages 576–80.

[4] S. Biundo and M. Fox, editors. *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, Durham, UK, September 1999. Springer-Verlag.

[5] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1636–1642, Montreal, Canada, August 1995. Morgan Kaufmann.

[6] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):279–298, 1997.

[7] Blai Bonet and Héctor Geffner. HSP: Heuristic search planner. In *AIPS-98 Planning Competition*, Pittsburgh, PA, 1998.

[8] Blai Bonet and Héctor Geffner. Planning as heuristic search: New results. In Biundo and Fox [4], pages 60–72.

[9] Blai Bonet and Héctor Geffner. Heuristic search planner 2.0. *The AI Magazine*, 22(3):77–80, 2001.

[10] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[11] Blai Bonet, Gábor Loerincs, and Héctor Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference of the American Association for Artificial Intelligence (AAAI-97)*, pages 714–719. MIT Press, July 1997.

[12] Blai Bonet and Sylvie Thiebaux. GPT meets PSR. In ICAPS-03 [47]. Accepted for publication.

[13] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

[14] A. Cesta and D. Borrajo, editors. *Recent Advances in AI Planning. 6th European Conference on Planning (ECP'01)*, Toledo, Spain, September 2001. Springer-Verlag.

[15] S. Chien, R. Kambhampati, and C. Knoblock, editors. *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*. AAAI Press, Menlo Park, 2000.

[16] Minh. B. Do and Subbarao Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In Cesta and Borrajo [14], pages 109–120.

[17] *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, Lyon, France, July 2002. Wiley.

[18] Stefan Edelkamp. Promela planning. In *Proceedings SPIN-03*, 2003. Accepted for publication.

[19] Stefan Edelkamp. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, 2003. Special issue on the 3rd International Planning Competition, to appear.

[20] Stefan Edelkamp and Malte Helmert. MIPS: The model checking integrated planning system. *AI Magazine*, 22(3):67–71, 2001.

[21] Richard E. Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[22] Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.

[23] Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 956–961, Stockholm, Sweden, August 1999. Morgan Kaufmann.

[24] Maria Fox and Derek Long. STAN4: A hybrid planning strategy based on subproblem abstraction. *The AI Magazine*, 22(3):81–84, 2001.

[25] Maria Fox and Derek Long. The third international planning competition: Temporal and metric planning. In Ghallab et al. [33], pages 333–335.

[26] Jeremy Frank, Peter Cheeseman, and John Stutz. When gravity fails: Local search topology. *Journal of Artificial Intelligence Research*, 7:249–281, 1997.

[27] B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel and Alami [66], pages 221–233.

[28] A. Gerevini and L. Schubert. Inferring state constraints in DISCOPLAN: Some new results. In AAAI-00 [1], pages 761–767.

[29] Alfonso Gerevini and Ivan Serina. LPG: A planner based on local search for planning graphs with action costs. In Ghallab et al. [33], pages 13–22.

[30] Alfonso Gerevini and Ivan Serina. Fast planning through local search and action graphs. *Journal of Artificial Intelligence Research*, 2003. Special issue on the 3rd International Planning Competition, to appear.

[31] Alfonso Gerevini and Ivan Serina. LPG tests, 2003. Personal communication.

[32] Alfonso Gerevini, Ivan Serina, Alessandro Saetti, and Sergio Spinoni. Local search techniques for temporal planning in LPG. In ICAPS-03 [47]. Accepted for publication.

[33] M. Ghallab, J. Hertzberg, and P. Traverso, editors. *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, Toulouse, France, April 2002. Morgan Kaufmann.

[34] Patrick Haslum and Hector Geffner. Admissible heuristics for optimal planning. In Chien et al. [15], pages 140–149.

[35] Patrick Haslum and Hector Geffner. Heuristic planning with time and resources. In Cesta and Borrajo [14], pages 121–132.

[36] Malte Helmert. On the complexity of planning in transportation domains. In Cesta and Borrajo [14], pages 349–360.

[37] Malte Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 2003. Accepted for publication.

[38] Jörg Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proceedings of the 12th International Symposium on Methodologies for Intelligent Systems (ISMIS-00)*, pages 216–227. Springer-Verlag, October 2000.

[39] Jörg Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. Technical Report 133, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany, January 2000. available at ftp://ftp.informatik.uni-freiburg.de/papers/ki/hoffmann-tr99.ps.gz.

[40] Jörg Hoffmann. FF: The fast-forward planning system. *The AI Magazine*, 22(3):57–62, 2001.

[41] Jörg Hoffmann. Local search topology in planning benchmarks: An empirical analysis. In Nebel [55], pages 453–458.

[42] Jörg Hoffmann. Extending FF to numerical state variables. In ECAI-02 [17], pages 571–575.

[43] Jörg Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In Ghallab et al. [33], pages 92–100.

[44] Jörg Hoffmann. *Utilizing Problem Structure in Planning: A Local Search Approach.* PhD thesis, University of Freiburg, Freiburg, Germany, 2002. postscript available at http://www.informatik.uni-freiburg.de/~hoffmann/diss.ps.gz".

[45] Jörg Hoffmann. Where ignoring delete lists works: Local search topology in planning benchmarks. *Artificial Intelligence*, 2003. Submitted.

[46] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[47] *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, Trento, Italy, 2003. Morgan Kaufmann.

[48] Jana Koehler and Jörg Hoffmann. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, 2000.

[49] Jana Koehler and Jörg Hoffmann. On the instantiation of ADL operators involving arbitrary first-order formulas. In *Proceedings ECAI-00 Workshop on New Results in Planning, Scheduling and Design*, 2000.

[50] Jana Koehler, Bernhard Nebel, Jörg Hoffmann, and Yannis Dimopoulos. Extending planning graphs to an ADL subset. In Steel and Alami [66], pages 273–285.

[51] Jana Koehler and Kilian Schuster. Elevator control as a planning problem. In Chien et al. [15], pages 331–338.

[52] Drew McDermott. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 142–149. AAAI Press, Menlo Park, 1996.

[53] Drew McDermott. The 1998 AI planning systems competition. *The AI Magazine*, 21(2):35–55, 2000.

[54] Drew V. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111–159, 1999.

[55] B. Nebel, editor. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, Washington, USA, August 2001. Morgan Kaufmann.

[56] Xuanlong Nguyen and Subbarao Kambhampati. Extracting effective and admissible heuristics from the planning graph. In AAAI-00 [1], pages 798–805.

[57] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In Nebel [55], pages 459–464.

[58] E. Onaindia, O. Sapena, L. Sebastia, and E. Marzal. Simplanner: an execution-monitoring system for replanning in dynamic worlds. In *Proceedings EPIA-01*, pages 393–400, 2001.

[59] Edwin P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR-89)*, pages 324–331, Toronto, ON, May 1989. Morgan Kaufmann.

[60] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In B. Nebel, W. Swartout, and C. Rich, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*, pages 103–114, Cambridge, MA, October 1992. Morgan Kaufmann.

[61] Ioannis Refanidis and Ioannis Vlahavas. GRT: a domain independent heuristic for STRIPS worlds based on greedy regression tables. In Biundo and Fox [4], pages 47–59.

[62] Ioannis Refanidis and Ioannis Vlahavas. The GRT planning system: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research*, 15:115–161, 2001.

[63] Jussi Rintanen. An iterative algorithm for synthesizing invariants. In AAAI-00 [1], pages 806–811.

[64] L. Sebastia, E. Onaindia, and E. Marzal. Stella: An optimal sequential and parallel planner. In *Proceedings EPIA-01*, pages 409–416, 2001.

[65] Biplav Srivastava, Xuanlong Nguyen, Subbarao Kambhampati, Minh B. Do, Ullas Nambiar, Zaiqing Nie, Romeo Nigenda, and Terry Zimmermann. Altalt: Combining graphplan and heuristic state search. *The AI Magazine*, 22(3):88–90, 2001.

[66] S. Steel and R. Alami, editors. *Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97)*, volume 1348 of *Lecture Notes in Artificial Intelligence*, Toulouse, France, September 1997. Springer-Verlag.

[67] Hakan Younes and Reid Simmons. On the role of ground actions in refinement planning. In Ghallab et al. [33], pages 54–61.