Prompt, Prove, Patch: The Neuro-Symbolic Loop for General Policy Synthesis

Dominik Drexler[1], **Markus Fritzsche**[1], **Farid Musayev**[1], **Simon Ståhlberg**[2]

[1]Linköping University, Sweden
[2]RWTH Aachen University, Germany
dominik.drexler@liu.se, markus.fritzsche@liu.se, farid.musayev@liu.se, simon.stahlberg@rwth-aachen.de

Abstract

Generalized planning asks for a single controller that solves all tasks of a domain family. Existing approaches are brittle in characteristic ways. On one hand, symbolic program-synthesis methods compile all training tasks into a single, monolithic classical planning episode. This creates a massive search space and leaves no mechanism for iterative refinement when the planner fails. On the other hand, language models used as one-shot plan or policy generators use broad world knowledge but lack formal verification guarantees; when a generated candidate fails, there is no principled way to isolate the root cause or deduce missing state abstractions. We propose a neuro-symbolic framework that unifies the strengths of both paradigms. A language model proposes policies over an expressive description-logic feature grammar governed by formal denotational semantics. A symbolic verifier then either certifies the policy across training tasks or returns precise structural counterexamples: all unhandled open states and, when present, an infinite loop. A feature generator analyzes these failure states to algorithmically discover missing relational features, allowing the language model to iteratively refine its policy until it is formally proven. We evaluate this loop across fifteen classical planning domains, including several entirely beyond the reach of prior program-synthesis methods. In all except one domain, our framework successfully synthesizes a policy that is formally proven on training tasks and generalizes perfectly to held-out test tasks under greedy execution. In the remaining domain, we trace the failure to a feature-language expressivity limitation and identify a concrete extension that would address it.

1 Introduction

Generalized planning asks for a single controller that solves all tasks of a domain family. A useful controller must not depend on the names or number of objects in one task; it must instead refer to abstract state properties such as whether some package is in the wrong city, whether a vehicle is co-located with it, or whether a hiker couple is ready to advance to the next waypoint.

Several approaches have tackled this problem, each with characteristic limitations. Program-synthesis methods (Segovia-Aguas, Jiménez, and Jonsson 2019, 2021) compile all training tasks into a single, monolithic classical planning episode and search for imperative programs with conditionals and loops. However, their expressivity is confined to flat relational features, such as bounded conjunctive queries, leaving them unable to express the inductive or recursive relational abstractions required for complex multi-object domains. Furthermore, because synthesis relies on a monolithic combinatorial search, it offers no mechanism for iterative refinement when the planner fails due to inadequate bounds or dead-ends. GNN-based learners (Ståhlberg, Bonet, and Geffner 2022a,b, 2023) learn value functions or transition policies that generalize across tasks but are bounded by C_2 and systematically fail on domains that require more expressivity. Alternatively, language models have been applied to generate plans or programs directly (Huang et al. 2022; Silver et al. 2024), but their outputs lack formal verification guarantees. When a candidate policy fails, empirical feedback is limited to a binary pass/fail signal over the training tasks, providing no structural indication of what abstraction is missing (Valmeekam et al. 2023).

We present a tool-mediated workflow for learning general policies from example tasks that applies the methodology around the symbolic verification loop (Yao et al. 2023b) and tree of thoughts (Yao et al. 2023a) in the generalized planning setting. A language model proposes a policy expressed over description-logic features, which are first-order relational expressions whose denotations depend on predicate structure, not on which objects populate a particular task. This constrains the hypothesis class to policies that are task-independent by construction, preventing the model from overfitting to specific training environments. The system then evaluates the policy over the grounded transition graphs of the training tasks. Failed proofs return precise symbolic counterexamples that isolate exactly why the candidate fails, such as an unhandled open state or a policy-induced infinite loop. A feature generator analyzes these failure states to algorithmically enumerate relational features that resolve the ambiguity, guiding the language model as it iteratively refines the policy until it is formally proven on the training set and checked by execution on held-out tasks.

The main contribution of this work is a compact formal account of this workflow. We define the feature language and general policies, together with the four verification tools that drive the refinement loop. We evaluate the loop across fifteen classical planning domains and provide a detailed analysis of four synthesized policies, illustrating how the refinement loop constructs relational abstractions for complex domains. The practical consequence is reach: earlier rule-based approaches

over description-logic features (Bonet and Geffner 2018; Francès, Bonet, and Geffner 2021) first construct an explicit pool of all features up to a fixed complexity bound, and that pool grows exponentially with the bound, so the bound must be kept low and high-complexity features stay out of scope. Because our workflow proposes and generates candidates on demand, it never materializes such a pool, and the synthesized policies rely on description-logic features of a complexity far beyond what an explicitly enumerated pool can afford, yielding general policies that earlier methods could not reach.

2 Related Work

This work belongs to generalized planning that represents controllers as rules over relational state features. Description-logic concepts and roles were introduced as a feature language for generalized policies (Martín and Geffner 2004), and later developed into feature pools and rule-based general policies whose denotation is independent of the objects of any task (Bonet and Geffner 2018; Bonet, Francès, and Geffner 2019; Francès, Bonet, and Geffner 2021; Bonet and Geffner 2021); sketches extend this to policies that decompose a problem into subproblems of bounded width (Drexler, Seipp, and Geffner 2022, 2024). We inherit this feature language and rule-based policy form. The same goal has also been pursued with other representations (Srivastava, Immerman, and Zilberstein 2011; Hu and Giacomo 2011): program synthesis searches for an imperative program over the training tasks (Segovia-Aguas, Jiménez, and Jonsson 2019, 2021), and deep learning trains neural controllers that generalize across task sizes, including action-schema networks (Toyer et al. 2018, 2020), transferable policies for relational domains (Garg, Bajpai, and Mausam 2019), deep reinforcement learning (Rivlin, Hazan, and Karpas 2020), and graph neural networks (Ståhlberg, Bonet, and Geffner 2022a,b, 2023). All fix the representation, whether expert-supplied or pre-enumerated features (de Graaff, Corrêa, and Pommerening 2021), a program grammar, or a network. They then search it monolithically, with no structural account of why a candidate fails. Recent LLM-based work also targets abstraction generation for generalized planning in a relational language (Cui et al. 2026). Like that work, we benefit from a language whose denotation is object-independent: the language model cannot violate generalized-planning invariance by referring directly to object names from particular tasks. We are closest to this work in representation, but differ in the role assigned to the language model: here it proposes features and rules incrementally inside a counterexample-guided loop, and a generator extends the feature pool only as failures demand.

Language models have been used as generators of plans or domain-general programs (Huang et al. 2022; Silver et al. 2024), including Python programs that operate over PDDL domains while invoking planning machinery (Silver et al. 2024). Other planner-in-the-loop approaches include LLM+P, which translates a task into PDDL for a planner (Liu et al. 2023), and LLM-Modulo frameworks, which pair a language-model generator with external critics that vet its proposals (Kambhampati et al. 2024). These systems differ from ours in the artifact being generated: the language model emits a per-task plan, model, or imperative program around a planning system, whereas here its output is itself the formal object, a general policy over relational features with denotational semantics for every task. Its feedback, too, is a structural counterexample, not a binary verdict. Finally, the loop relates to counterexample-guided synthesis. Counterexample-guided inductive synthesis searches a fixed grammar, querying a verification oracle for inputs on which a candidate fails (Solar-Lezama et al. 2006); counterexample-guided abstraction refinement applies it in model checking and in abstraction heuristics for classical planning (Clarke et al. 2000; Seipp and Helmert 2013, 2018). Our loop shares this shape but differs twice: the search operator is a language model rather than a constraint solver, prompted iteratively in the style of reasoning-and-acting methods (Yao et al. 2023b,a); and the oracle returns symbolic counterexamples (an unhandled open state or a policy-induced loop) and a generator that extends the grammar itself rather than only searching it.

3 Background

This section sets up the formal background used throughout the paper. We first recall *classical planning* and the state models induced by planning tasks. We then introduce the *description-logic feature language* in which our policies are expressed, a relational vocabulary whose denotation is independent of the objects of any particular task. Finally, we define *generalized planning* and the *general policies* that solve a whole class of tasks at once.

3.1 Classical Planning

A *classical planning task* is a tuple $\tau = \langle \mathcal{P}, \mathcal{A}, \mathcal{O}, I, \mathcal{G} \rangle$. The *planning domain* is $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$, and the *task information* is $\langle \mathcal{O}, I, \mathcal{G} \rangle$, namely:

Predicates and Literals. $\mathcal{P}$ is a set of predicate symbols. Each *predicate symbol* P in $\mathcal{P}$ has an associated arity $ar(P)$ in $\mathbb{N}$. An *atom* over a predicate P of arity $ar(P) = k$ is an expression $P(x_1, \ldots, x_k)$ where x_i with $i = 1, \ldots, k$ are variables or constants. A *literal* is either an atom p or its negation $\neg p$.

Objects and Grounding. $\mathcal{O}$ is a set of objects (constants). An atom, literal, or action that contains no variables is *ground*.

States. I is the *initial state*. A *state* s consists of a set of ground atoms $\overline{P}(s)$ that hold in s. Ground atoms not in $\overline{P}(s)$ are false. A positive ground literal $\overline{p}$ *holds* in s iff $\overline{p} \in \overline{P}(s)$, and a negative ground literal $\neg\overline{p}$ *holds* iff $\overline{p} \notin \overline{P}(s)$.

Goals. $\mathcal{G}$ is the goal, which is a set of ground literals.

Actions and Successors. $\mathcal{A}$ is a set of action schemas. Each *action schema* a in $\mathcal{A}$ consists of two sets $pre(a)$ and $\mathit{eff}(a)$, where $pre(a)$ is a set of precondition literals and $\mathit{eff}(a)$ is a set of effect literals. We write $pre^+(a)$, $pre^-(a)$, $\mathit{eff}^+(a)$, and $\mathit{eff}^-(a)$ for the sets of atoms of positive and negative literals in $pre(a)$ and $\mathit{eff}(a)$. A ground action $\overline{a}$ is *applicable* in a state s iff every ground literal $\overline{\ell}$ in $pre(\overline{a})$ holds in s. Applying a ground action $\overline{a}$ in a state s where it is applicable yields the *successor state* s' with $\overline{P}(s') = (\overline{P}(s) \setminus \mathit{eff}^-(\overline{a})) \cup \mathit{eff}^+(\overline{a})$.

Plans. The objective for a given classical planning task is to find a *plan*, which is a sequence of ground actions that, when successively applied starting from the initial state, yields a state in which all ground literals mentioned in the goal hold.

State Models and Paths. The *state model* of τ is $\mathcal{S}(\tau) = \langle S_\tau, T_\tau, I_\tau, G_\tau \rangle$. Here S_τ is the set of states reachable from I_τ by applying ground actions, $T_\tau \subseteq S_\tau \times S_\tau$ contains (s, s') iff some applicable ground action maps s to s', and $G_\tau \subseteq S_\tau$ is the set of reachable states satisfying $\mathcal{G}$. A *path* in $\mathcal{S}(\tau)$ is either a finite sequence $s_0, \ldots, s_k$ with $(s_i, s_{i+1}) \in T_\tau$ for all $0 \leq i < k$, or an infinite sequence $s_0, s_1, \ldots$ with $(s_i, s_{i+1}) \in T_\tau$ for all $i \in \mathbb{N}$. A finite path is *maximal* if its last state has no outgoing edge in T_τ; every infinite path is maximal. A *cycle* is a finite path $s_0, \ldots, s_k$ with $k > 0$ and $s_0 = s_k$.

3.2 Description-Logic Features

A controller that must work across every task of a domain cannot refer to individual objects or to how many there are, since these vary from task to task. It must instead act on abstract, task-independent properties of a state, such as whether some package is still in the wrong city or whether every vehicle has reached its destination. We capture such properties with *features* that are computed from a state-goal pair and whose denotation depends only on the relational structure of a task, not on which objects populate it.

Feature Interpretations. Features are evaluated over a relational structure derived from a state-goal pair. For a pair $(s, \mathcal{G})$, let $\mathcal{I}_{s,\mathcal{G}}$ be the finite relational structure with domain $\Delta = \mathcal{O}$. A state predicate denotes the tuples true in s, and a goal predicate denotes the tuples required by $\mathcal{G}$, with separate positive and negative goal interpretations. A unary predicate gives rise to a concept and a binary predicate to a role, where concepts denote subsets of Δ and roles denote binary relations over Δ (Baader et al. 2003).

Syntax and Semantics. The feature grammar has concept and role terms. These terms are combined to boolean and numerical features, which are interpreted as described below. Concept terms are defined by the grammar

$$\begin{aligned} C ::= {} & \top \mid \bot \mid A_s \mid A_g^+ \mid A_g^- \mid \neg C \mid C \sqcap C \mid C \sqcup C \\ & \mid \exists R.C \mid \forall R.C \mid \geq n R.C \mid \leq n R.C \mid = n R.C \\ & \mid R \subseteq S \mid R \doteq S \mid R : \{i_1, \ldots, i_k\} \\ & \mid \{i_1, \ldots, i_k\}. \end{aligned}$$

where A_s is a state concept, A_g^+ and A_g^- are positive and negative goal concepts, and $i_1, \ldots, i_k$ are individual names; $\{i_1, \ldots, i_k\}$ is a one-of concept, with nominals as the singleton case (reserved for domain constants). Role terms are

$$\begin{aligned} R ::= {} & U \mid P_s \mid P_g^+ \mid P_g^- \mid \neg R \mid R \sqcap R \mid R \sqcup R \mid R^- \\ & \mid R \circ R \mid R^+ \mid R^* \mid R|C \mid id(C). \end{aligned}$$

Writing $\mathcal{I} = \mathcal{I}_{s,\mathcal{G}}$, the grammar constructs are interpreted as follows:

- $\top^\mathcal{I} = \Delta$ and $\bot^\mathcal{I} = \emptyset$. For each unary predicate symbol A, $A_s^\mathcal{I}$ is the set of objects satisfying A in s, while $(A_g^+)^\mathcal{I}$ and $(A_g^-)^\mathcal{I}$ are the sets of objects occurring in positive and negative goal literals over A, respectively. For each binary predicate symbol P, $P_s^\mathcal{I}$ is the set of pairs satisfying P in s, while $(P_g^+)^\mathcal{I}$ and $(P_g^-)^\mathcal{I}$ are the sets of pairs occurring in positive and negative goal literals over P, respectively. Each individual name i denotes an object $i^\mathcal{I} \in \Delta$.
- Boolean concept connectives are set operations: $(\neg C)^\mathcal{I} = \Delta \setminus C^\mathcal{I}$, $(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$, and $(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$.
- Role quantifiers select objects by R-successors: $(\exists R.C)^\mathcal{I} = \{o \mid \exists o'.(o, o') \in R^\mathcal{I}, o' \in C^\mathcal{I}\}$, and $(\forall R.C)^\mathcal{I} = \{o \mid \forall o'.(o, o') \in R^\mathcal{I} \Rightarrow o' \in C^\mathcal{I}\}$.
- Number restrictions count restricted successors: $(\geq n R.C)^\mathcal{I}$, $(\leq n R.C)^\mathcal{I}$, and $(= n R.C)^\mathcal{I}$ contain objects o for which $|\{o' \mid (o, o') \in R^\mathcal{I}, o' \in C^\mathcal{I}\}|$ is at least, at most, or equal to n.
- Role comparisons used as concepts are row-wise tests: $(R \subseteq S)^\mathcal{I} = \{o \mid \forall p.(o, p) \in R^\mathcal{I} \Rightarrow (o, p) \in S^\mathcal{I}\}$, and $(R \doteq S)^\mathcal{I} = \{o \mid \forall p.(o, p) \in R^\mathcal{I} \Leftrightarrow (o, p) \in S^\mathcal{I}\}$. Role fillers require all listed objects as successors: $(R : \{i_1, \ldots, i_k\})^\mathcal{I} = \{o \mid \{i_1^\mathcal{I}, \ldots, i_k^\mathcal{I}\} \subseteq \{p \mid (o, p) \in R^\mathcal{I}\}\}$. One-of concepts are interpreted as $\{i_1, \ldots, i_k\}^\mathcal{I} = \{i_1^\mathcal{I}, \ldots, i_k^\mathcal{I}\}$.
- Role constructors denote binary relations: $U^\mathcal{I} = \Delta \times \Delta$, $(\neg R)^\mathcal{I} = (\Delta \times \Delta) \setminus R^\mathcal{I}$, $(R \sqcap S)^\mathcal{I} = R^\mathcal{I} \cap S^\mathcal{I}$, $(R \sqcup S)^\mathcal{I} = R^\mathcal{I} \cup S^\mathcal{I}$, $(R^-)^\mathcal{I} = \{(o', o) \mid (o, o') \in R^\mathcal{I}\}$, and $(R \circ S)^\mathcal{I} = \{(o, o'') \mid \exists o'.(o, o') \in R^\mathcal{I}, (o', o'') \in S^\mathcal{I}\}$.
- R^+ and R^* are transitive and reflexive-transitive closure. For $n \in \mathbb{N}$, $(R^\mathcal{I})^0 = \{(o, o) \mid o \in \Delta\}$ and $(R^\mathcal{I})^{n+1} = (R^\mathcal{I})^n \circ R^\mathcal{I}$. $(R|C)^\mathcal{I} = \{(o, o') \in R^\mathcal{I} \mid o' \in C^\mathcal{I}\}$, and $id(C)^\mathcal{I} = \{(o, o) \mid o \in C^\mathcal{I}\}$.
- Boolean features are truth-valued: zero-arity state/goal tests, $C \neq \emptyset$, and $R \neq \emptyset$, true exactly when the condition holds in $\mathcal{I}$.
- Numerical features take values in $\mathbb{N} \cup \{\infty\}$: the cardinalities $|C|$ and $|R|$, and the role distance $d_R(C, D) = \min\{n \mid (C^\mathcal{I} \times D^\mathcal{I}) \cap (R^\mathcal{I})^n \neq \emptyset\}$, with value ∞ if no such n exists.

Example. Consider a Blocks task with objects a, b, and c. In the state s, block c sits on b, b sits on a, and a sits on the table, and the goal $\mathcal{G}$ requires a to be on b. The interpretation $\mathcal{I}_{s,\mathcal{G}}$ then has domain $\Delta = \{a, b, c\}$ and maps the state role on_s to $\{(c, b), (b, a)\}$, the state concept $clear_s$ to $\{c\}$, and the positive goal role on_g^+ to $\{(a, b)\}$; the negative goal interpretations are empty, since the goal has no negative literals. Transitive closure turns this direct-support relation into the *above* relation: on_s^+ denotes $\{(c, b), (b, a), (c, a)\}$, where the closure contributes the pair (c, a), as block c lies above a without resting directly on it. Finally, the numerical feature $|\, on_g^+ \sqcap \neg\, on_s \,|$ counts the goal atoms still unsatisfied: the role $on_g^+ \sqcap \neg\, on_s$ denotes the single pair (a, b), so the feature evaluates to 1 in s, and a policy solves the task exactly when it drives this feature to 0.

Expressivity. The description-logic feature languages used in prior generalized-planning work are closely related to C_2, first-order logic with counting and at most two variables, and inherit its expressive limitations (Drexler et al. 2024). The same bound applies to GNN-based policy learners, whose expressivity does not exceed C_2 either (Ståhlberg, Bonet, and Geffner 2022b; Drexler et al. 2024); distinctions that lie outside C_2 are therefore beyond the reach of both pre-enumerated feature pools and GNN controllers. Our grammar

keeps the constructors of this standard, C_2-equivalent feature language and adds two extensions that increase expressivity beyond C_2: role composition and transitive closure. Role composition stays within first-order logic but requires more than two variables, so it already lies beyond C_2; transitive closure is not first-order definable at all, and the distance feature build on it.[1] The grammar's remaining constructors, such as role comparisons over C_2-definable roles, stay within C_2. The full grammar is captured by first-order logic extended with a transitive-closure operator. This expressivity is needed in practice: the learned Logistics policy relies on role composition to recognize whether a package or vehicle is in the package's destination city, while transitive closure and distance features capture recursive structure such as the *above* relation in Blocks, and multiple-source, multiple-target shortest path distances in grid-navigation domains.

3.3 Generalized Planning

Problem Classes. A *problem class* $\mathcal{Q}$ over a common domain $\mathcal{D}$ is a set of tasks whose first two components are $\mathcal{D}$ and whose objects, initial state, and goal may vary.

General Policies. Let $\Phi = B \cup N$ be a finite set of Boolean features B and numerical features N. For a feature $f \in \Phi$ and a state s, $f(s)$ denotes the value of f in s. A policy π for a problem class $\mathcal{Q}$ assigns to each task $\tau \in \mathcal{Q}$ a relation $\pi(\tau) \subseteq T_\tau$. It is a relation, not a function: from a state, π may admit several outgoing transitions rather than prescribe a unique next step. In our syntax, π is given by a finite set of *rules*. A rule $\rho = (\mathit{pre}, \mathit{eff})$ pairs a set pre of *tests* with a set eff of *effects*. A test constrains the source state: for a Boolean feature $b \in B$ it is b or $\neg b$, and for a numerical feature $n \in N$ it is $n = 0$ or $n > 0$. An effect constrains how a feature changes from the source state s to the target state s': a Boolean feature is made true (b) or false ($\neg b$), and a numerical feature must strictly increase ($n\uparrow$), strictly decrease ($n\downarrow$), or keep its value ($n=$).[2] A transition (s, s') is *compatible* with a rule ρ if every test in pre holds in s and every effect in eff holds for (s, s'). The policy selects exactly the compatible transitions: $\pi(\tau)$ contains the edges of T_τ compatible with at least one rule. Finally, π solves $\mathcal{Q}$ if, for every $\tau \in \mathcal{Q}$, every maximal path from I_τ in the graph selected by $\pi(\tau)$ is finite and ends in a state in G_τ.

4 Policy Learning with Verification Tools

The language model never touches the planner internals. It authors a single artifact, the policy, consisting of a feature set Φ and a set of rules over those features. All learning happens through four tools that check this artifact against finite sets of planning tasks and report their results as operations over finite transition graphs; the tools only inspect the policy and never modify it.

[1] Unlike Francès, Bonet, and Geffner (2021), we allow arbitrary source and target concepts in $d_R(C, D)$.

[2] Prior rule-based policies also define an effect $n?$ that permits arbitrary change, and treat any feature absent from eff as unchanged; we instead make arbitrary change the default, so eff records only the features whose change is constrained.

The four tools split into two phases. One tool screens the task data once, before learning begins. The other three drive an iterative refinement loop: the model proposes a policy, PROVEPOLICY checks it on the training tasks and on failure returns a *counterexample* that pinpoints what to fix, GENERATEFEATURES turns counterexamples into candidate features, and EXECUTEPOLICY provides a final held-out check. Every counterexample is one of two kinds:

- an *open state*: a reachable non-goal state with no π-accepted outgoing transition, which the policy thus leaves unhandled;
- a *cycle*: a cycle in the policy-induced graph, on which the policy can loop forever without reaching the goal.

The tools perform no solvability analysis, so a π-accepted transition into a state from which no goal can be reached is not flagged at that step; entering such a state surfaces later as an open state or a cycle. The model uses these counterexamples to introduce or revise features and rules, and repeats the loop until the policy is proven on training tasks and succeeds under held-out execution.

4.1 Preprocessing: Solvability Screening

Before the loop begins, a single tool screens the task data so that policy learning starts from well-formed examples.

Tool 1: PROVESOLVABILITY. Let $\mathcal{Q}' \subseteq \mathcal{Q}$ be a finite set of planning tasks. PROVESOLVABILITY maps $\mathcal{Q}'$ to a table $\{(\tau_i, \sigma_i, \ell_i, c_i) \mid \tau_i \in \mathcal{Q}'\}$, where σ_i is the search status, ℓ_i is the plan length, and c_i is the plan cost. Operationally, the tool constructs each lifted task and searches with greedy best-first search using an FF-style relaxation heuristic. It runs once, before policy learning begins, to reject degenerate examples, i.e., unsolvable tasks or tasks that are overly complex.

4.2 Verification Tools

The remaining three tools form the refinement loop. We describe them as mathematical operations over finite task sets, policies, transition graphs, and feature valuations.

Tool 2: PROVEPOLICY. Given a policy π and a finite task set $\mathcal{Q}' \subseteq \mathcal{Q}$, it decides whether π solves every task in $\mathcal{Q}'$, and when it does not, returns a precise reason. We call it a *proof* rather than a test because it does not run the policy once and observe the outcome; it checks *every* behavior the policy permits. This is necessary because a policy is a *relation*, not a function: $\pi(\tau)$ may accept several outgoing transitions from the same state, so the policy does not prescribe a single trajectory but a branching set of them. PROVEPOLICY must therefore guarantee that a goal is reached no matter which accepted transitions are taken; equivalently, it asks whether an adversary, free to choose among the transitions that π accepts, could steer execution so that no goal is ever reached. The policy is proven exactly when no such adversary exists.

For each $\tau \in \mathcal{Q}'$ the tool grounds the task and constructs its state model $\mathcal{S}(\tau) = \langle S_\tau, T_\tau, I_\tau, G_\tau \rangle$. It then performs on-policy exploration, keeping exactly the states reachable from I_τ by following transitions accepted by π and discarding the rest of $\mathcal{S}(\tau)$. The result is the policy-induced reachable

graph $\mathcal{S}^\pi_\tau = \langle S^\pi_\tau, T^\pi_\tau \rangle$, built as a least fixed point: from $S^\pi_{\tau,0} = \{I_\tau\}$, each layer adds the accepted successors of the states reached so far,

$$S^\pi_{\tau,i+1} = S^\pi_{\tau,i} \cup \{s' \mid \exists\, s \in S^\pi_{\tau,i} : (s, s') \in \pi(\tau)\},$$

and the construction stabilizes at $S^\pi_\tau = \bigcup_i S^\pi_{\tau,i}$, with edges $T^\pi_\tau = \pi(\tau) \cap (S^\pi_\tau \times S^\pi_\tau)$. Because $\mathcal{S}(\tau)$ is finite, this graph is finite and the exploration always terminates.

The policy is proven on τ iff every maximal path from I_τ in $\mathcal{S}^\pi_\tau$ is finite and ends in a goal state. This is the same condition that defines when a policy solves a problem class, here decided for a single task on a finite graph. The condition can fail in two ways, and the tool reports each as the corresponding counterexample. An infinite maximal path shows up, on a finite graph, as a *cycle* in $\mathcal{S}^\pi_\tau$ along which the policy loops without progress. A finite maximal path that stops short of the goal ends in an *open state*, a reachable non-goal state from which the policy accepts no further transition. PROVEPOLICY works entirely on the policy-induced reachable graph $\mathcal{S}^\pi_\tau$ and does not analyze whether states of the underlying task graph $\mathcal{S}(\tau)$ remain solvable; it therefore does not flag a *dead-end transition*, an accepted transition (s, s') whose target s' has no path to any goal, as a counterexample of its own. Entering such an unsolvable state is not detected at that step: it must later produce an open state or a cycle further along $\mathcal{S}^\pi_\tau$, and PROVEPOLICY reports that downstream symptom. PROVEPOLICY succeeds iff none of the two arises for any task in $\mathcal{Q}'$; otherwise it returns one such counterexample, which the model inspects and repairs, often after passing it to GENERATEFEATURES.

Tool 3: GENERATEFEATURES. Let C be the set of counterexamples returned by PROVEPOLICY. Then GENERATEFEATURES extracts a set of *conflicting state pairs* from C: for open states the state itself paired against a nearby goal state, and for cycles the states involved. Given a complexity bound, it enumerates Boolean and numerical description-logic features and keeps those that assign different values to at least one conflicting pair, after semantic pruning that discards a candidate whose valuations coincide with an already-retained feature.

The output presented to the model is a concrete table: for each retained feature, its syntactic form together with its value on each conflicting state. The model sees exactly which low-complexity abstract properties currently separate the states, and can use this as a scaffold for understanding what kind of distinction the policy is missing.

Crucially, the model is not restricted to the generated candidates. It may add any description-logic expression it forms to Φ, independently of the table. In practice, the generated pool is almost always set aside: the complexity bound keeps the enumerated features simple, and simple features rarely capture the relational structure the policy actually needs. The model typically inspects the table to identify the nature of the required distinction, then constructs a higher-complexity expression that encodes it in a form that generalizes beyond the extracted states.

Tool 4: EXECUTEPOLICY. A proof on the finite training set does not guarantee that the policy generalizes to unseen tasks; EXECUTEPOLICY provides an empirical check on a separate validation set before the loop terminates. Whereas PROVEPOLICY certifies a policy over the training tasks that drive the loop by checking all maximal π-paths, EXECUTEPOLICY executes the policy on a separate validation set, committing to one resolution of nondeterminism at each state rather than exploring all accepted transitions. Given a policy π and a finite validation set $\mathcal{Q}' \subseteq \mathcal{Q}$, it returns success iff every $\tau \in \mathcal{Q}'$ reaches a goal state under transitions accepted by π. On failure it returns a witness: either a cycle in the policy-induced search graph or an unsolved reachable state from which the guided search finds no goal-reaching continuation. As with PROVEPOLICY, dead-end transitions are not reported: neither tool performs a solvability analysis of the underlying task graph, so an accepted transition into an unsolvable state shows up instead as a downstream unsolved state when the search cannot make further progress. Because it only exercises the policy on the validation tasks, it is not a certificate for all tasks of the domain.

4.3 The Refinement Loop

The prompt. The interaction is initialized by a single prompt of fewer than 5000 characters that gives the model its instructions, constraints, allowances, and goal, together with file paths to the LaTeX sources defining the description-logic syntax and semantics. To strengthen generalization, analogous to previous work on learning general policies via weighted Max-SAT (Francès, Bonet, and Geffner 2021), the prompt steers the model toward syntactically simple solutions. For example, it discourages feature families that enumerate fixed path lengths, such as one-step, two-step, and three-step features, when a distance feature can express the expected generalization to arbitrary lengths. No additional domain-specific prompts are injected during the interaction. Instead, the interaction proceeds within a single agent session whose context accumulates the transcript of all tool calls, failed candidates, counterexamples, generated features, and successful checks; every subsequent model action is conditioned on this growing transcript. The prompt does not prescribe fixed resource bounds for the tools; instead, the model chooses bounds during the interaction based on the current task, failed attempts, and tool feedback.

The loop. The loop is a single ordered cycle. For each domain, the interaction runs as one continuous agent session: at every iteration the model is conditioned on the full history of prior iterations, not only on the latest counterexample. This history includes proposed policies, tool calls, counterexamples, and generated features. Each refinement step is therefore a continuation of earlier reasoning rather than an independent attempt. The model starts from the empty policy and calls PROVEPOLICY on the training tasks. Each returned counterexample is either inspected or passed to GENERATEFEATURES for candidate features; the model then revises the policy, adding or editing features in Φ and the rules over them, and calls PROVEPOLICY again. The cycle repeats until PROVEPOLICY succeeds on the training set and EXECUTEPOLICY succeeds on the validation set, which is the termination goal of the interaction.

Domain	Params	Train	Valid	Test
Barman	(c, i, s, g)	$[1,3]\times[2,5]\times[2,6]\times P,\ s \geq c$	$[4,6]\times[6,9]\times[7,10]\times P,\ s \geq c$	$[7,10]\times[10,14]\times[11,15]\times P,\ s \geq c$
Blocks-3	b	$[4, 10]$	$[11, 15]$	$[16, 24]$
Blocks-4	b	$[4, 10]$	$[11, 15]$	$[16, 24]$
Childsnack	(c, t, g, r)	$[2,6]\times[1,2]\times Q\times R$	$[7,10]\times[1,3]\times Q\times R$	$[11,16]\times[2,4]\times Q\times R$
Delivery	(n, p)	$[2,5]\times[1,4]$	$[6,8]\times[5,7]$	$[9,13]\times[8,12]$
Driverlog	$(l, d, t, p, g_p, g_d, g_t)$	$[4,7]\times[2,4]\times[2,4]\times[2,4]\times G^3$	$[9,12]\times[1,2]\times[1,2]\times[5,7]\times G^3$	$[13,18]\times[2,4]\times[2,4]\times[8,11]\times G^3$
Ferry	(c, l)	$[2, 7]\times[3, 5]$	$[8, 11]\times[6, 8]$	$[12, 17]\times[9, 12]$
Floortile	(r, c, b)	$[2,4]\times[3,5]\times[1,2]$	$[5,6]\times[6,7]\times[2,3]$	$[7,9]\times[8,10]\times[3,4]$
Goldminer	(r, c)	$[3,6]\times[4,7]$	$[7,9]\times[8,10]$	$[10,13]\times[11,14]$
Grid	(w, h, s, k, l, g)	$[2,4]\times[2,4]\times[1,2]\times[1,3]\times[1,3]\times B$	$[5,6]\times[5,6]\times[2,3]\times[3,4]\times[3,4]\times B$	$[7,8]\times[7,8]\times[3,4]\times[4,6]\times[4,6]\times B$
Gripper	b	$[1, 5]$	$[6, 10]$	$[11, 20]$
Hiking-Bin.	(h, c, p)	$[1,3]\times[2,4]\times[2,3]$	$4\times5\times[4,5]$	$5\times[6,7]\times[5,6]$
Logistics	(c, l, p, a, g)	$[1,2]\times[2,4]\times[1,3]\times[1,2]\times B$	$[3,4]\times[5,6]\times[4,6]\times[1,2]\times B$	$[5,7]\times[7,9]\times[7,10]\times[2,3]\times B$
Miconic	(f, p)	$[1,6]\times[1,6]$	$[7,10]\times[7,10]$	$[11,16]\times[11,16]$
Satellite	(s, i, m, t, o, g)	$[1,2]\times[1,3]\times[2,3]\times[1,2]\times[1,3]\times Q$	$[3,4]\times[4,5]\times[3,4]\times[3,4]\times[4,5]\times Q$	$[5,6]\times[6,8]\times[4,5]\times[5,6]\times[6,8]\times Q$

Table 1: Generator parameter value sets. The interval $[n, m]$ denotes the integer set $\{n, \ldots, m\}$; singleton values are written without braces; $P = \{0, 0.5, 1.0\}$, $B = \{0.5, 1.0\}$, $G = \{0.5, 0.75, 1.0\}$, $Q = \{0.25, 0.5, 0.75\}$, and $R = \{1.2, 1.5\}$. Each split contains 100 tasks except Gripper, which contains 5 train, 5 validation, and 10 test tasks. Parameters are abbreviated as follows: Barman (c, i, s, g) cocktails, ingredients, shots, ingredient-goal probability; Blocks b blocks; Childsnack (c, t, g, r) children, trays, gluten factor, sandwich ratio; Delivery (n, p) grid size, packages; Driverlog $(l, d, t, p, g_p, g_d, g_t)$ locations, drivers, trucks, packages, package/driver/truck goal probabilities; Ferry (c, l) cars, locations; Floortile (r, c, b) rows, columns, robots; Goldminer (r, c) rows, columns; Grid (w, h, s, k, l, g) width, height, shapes, keys, locks, goal probability; Gripper b balls; Hiking-Binary (h, c, p) couples, cars, places; Logistics (c, l, p, a, g) cities, city size, packages, airplanes, goal-package probability; Miconic (f, p) floors, passengers; Satellite (s, i, m, t, o, g) satellites, instruments, modes, targets, observations, pointing-goal probability.

5 Experiments

Our experiments address three questions: can the prompted learning loop discover general policies for a diverse set of classical planning domains; do the learned policies generalize to instances far larger than those seen during learning; and how do they compare to previously learned policies and to state-of-the-art domain-independent planners?

5.1 Data Generation

We generated training, validation, and test splits for 15 classical planning domains, most of which have appeared in the International Planning Competition, using PDDL generators (Seipp, Torralba, and Hoffmann 2022). When an IPC generator was available, we adapted it to sample initial states and goals according to its IPC-style distribution. Hiking-Binary (abbreviated Hiking-Bin.) is an equivalent reformulation of the original Hiking domain where we compiled a ternary relation persons/3 into a single binary relation personof/2 that links each person to a couple. We adopted this reformulation because our grammar does not support ternary relations.

Table 1 shows the Cartesian parameter sets used for each generated split. We asked ChatGPT 5.5 to propose initial Cartesian parameter ranges after describing the tools used to generate, prove, and execute policies. Crucially, we then manually validated and refined these bounds to enforce a strict curriculum of increasing difficulty. While some individual parameter dimensions may overlap, the resulting Cartesian parameter sets are strictly pairwise disjoint across the train, validation, and test splits. Furthermore, we ensured that for every domain, at least one core scaling dimension strictly increases across the splits without any overlap. This means that the most difficult instances are reserved for the test set. We repeated this step for Hiking-Binary and Childsnack, where the initial training ranges were too large for ProvePolicy because of high branching factors and symmetries. Repeated seeds are used only to sample multiple tasks from the same structural space.

5.2 Learning Setup

The core learning framework is implemented in C++, and its four tools are exposed to the LLM through a Python interface. For each domain, we ran the prompted learning loop from Section 4 using Codex with ChatGPT 5.5 with medium thinking mode as the proposal model. The exact prompting instructions are given in Appendix A. We used the same objective for every domain, without domain-specific prompt adjustments or domain-specific resource bounds such as an upper bound on feature complexity. The only domain-specific input was the training, validation, and test splits, and the loop produced one final policy per domain.

5.3 Learning Results

Table 2 summarizes the final policy obtained for each domain during learning. The iteration count varies substantially, ranging from one or a few refinement steps in Delivery, Ferry, Miconic, and Satellite to hundreds of steps in Floortile. In simple domains such as Delivery, Miconic, and Gripper, general policies in the same language had already been learned with Weighted Max-SAT and presented in the literature (Francès, Bonet, and Geffner 2021), which may explain the fast convergence. However, our general Delivery policy differs from previous ones, as we discuss in the analysis section. The most

Domain	#I	#Φ	C^{max}	C^{sum}	#R	Proved	Valid.
Barman	69	18	49	221	18	98/98	77/77
Blocks-3	11	11	53	198	9	100/100	100/100
Blocks-4	8	8	39	109	5	100/100	100/100
Childsnack	3	9	19	78	10	100/100	66/66
Delivery	1	6	18	74	4	100/100	100/100
Driverlog	8	28	31	462	22	100/100	100/100
Ferry	1	4	10	26	4	100/100	100/100
Floortile	329	12	100	465	11	97/100	100/100
Goldminer	10	9	9	27	7	100/100	88/88
Grid	18	12	28	141	14	100/100	100/100
Gripper	2	5	10	33	4	5/5	5/5
Hiking-Bin.	28	21	41	419	31	95/95	0/0
Logistics	9	16	29	335	14	100/100	100/100
Miconic	2	12	8	65	6	100/100	100/100
Satellite	3	9	24	125	7	100/100	100/100

Table 2: Final learned policies. #I is the iteration count, #Φ is the number of features, C^{max} and C^{sum} are the maximum and sum of feature syntactic complexities, and #R is the number of policy rules. In Hiking-Binary, no validation tasks could be determined as solvable due to scaling.

Domain	LAMA		BFWS		Policy	
	S	T	S	T	S	T
Barman (100)	98	1.11	**100**	0.04	**100**	0.64
Blocks-3 (100)	**100**	0.14	98	0.69	**100**	0.84
Blocks-4 (100)	**100**	0.04	**100**	0.07	**100**	0.14
Childsnack (100)	56	0.47	62	1.01	**100**	1.19
Delivery (100)	**100**	0.41	**100**	0.05	**100**	0.51
Driverlog (100)	**100**	0.20	**100**	0.03	60	0.22
Ferry (100)	**100**	0.02	**100**	0.00	**100**	0.10
Floortile (100)	0	–	0	–	**100**	–
Goldminer (100)	43	0.74	**100**	0.07	**100**	0.87
Grid (100)	**100**	0.03	**100**	0.01	**100**	0.25
Gripper (10)	**10**	0.01	**10**	0.00	**10**	0.10
Hiking-Bin. (100)	**100**	–	0	–	**100**	–
Logistics (100)	**100**	0.04	**100**	0.09	**100**	0.65
Miconic (100)	**100**	0.01	**100**	0.00	**100**	0.10
Satellite (100)	**100**	0.01	**100**	0.02	**100**	0.19
Total (1410)	1207	0.08	1170	0.03	**1370**	0.31

Table 3: Testing results on generated test tasks. S is the number of solved tasks and T is the geometric mean total time in seconds over *commonly* solved tasks.

interesting cases are Barman, Blocks-4, Driverlog, Floortile, Hiking-Binary, Logistics, and Satellite: these domains are significantly more complex, require more iterations, and, to the best of our knowledge, do not have previously learned policies in the same language.

The learned policies are compact in terms of rule count, but their features can be structurally complex: several domains require double-digit numbers of features and cumulative syntactic complexity above 100. The closest existing features in terms of syntactic complexity are the hand-crafted features used to represent hand-crafted policy sketches (Drexler, Seipp, and Geffner 2024) and generalized potential heuristics (Francès et al. 2019). However, these features differ from ours not only in syntactic complexity, but also in the grammars that induce them.

The proof column reports how many training tasks were solved by the policy proof, whereas the validation column reports how many validation tasks were solved by greedy execution. Most domains are fully proved on training tasks and solved by execution on validation tasks. The main exceptions are Floortile, which leaves a few training tasks unproved, and Hiking-Binary, where the final policy proves the training tasks but has no sufficiently small validation tasks under the selected validation split.

5.4 Test Evaluation

After learning, we evaluated each policy by greedy execution on generated test tasks and on existing IPC benchmarks, whose scaling is known to be challenging. We conducted this evaluation with the Lab toolkit (Seipp et al. 2017). All evaluation runs used an 8 GB memory limit and a 30-minute timeout. As reference points, we also ran two well-known, state-of-the-art domain-independent planners, LAMA (Richter and Westphal 2010) and Dual-BFWS (Lipovetzky and Geffner 2017), on the same benchmarks.

Tables 3 and 4 show that, in all but two domains, a single greedy execution reaches a goal state. The exceptions are Driverlog and Satellite. In Satellite, performance scales poorly because feature evaluation is costly: the hardest solved tasks require close to 300 seconds. In Driverlog, the feature language is not expressive enough; we return to this limitation, and possible ways to address it, in the analysis.

5.5 Analysis

We inspect three domains in more detail and relate the learned policies to previous results and analyses from the literature.

Delivery. Delivery tasks require delivering packages to target locations within a grid using a single truck. The learned policy follows a simple decomposition: it unloads when possible, otherwise uses trucks for delivery. The set of features Φ_D is $\{r, c, ru, pt, dp, dg\}$ where r counts the number of packages not at their goal location, c counts the number of packages currently carried by the truck, ru counts the number of carried packages whose truck is at the package goal location, pt counts the number of undelivered uncarried packages at the truck location, dp is the truck grid distance to an undelivered package location, and dg is the truck grid distance to the goal location of the carried package. The policy rules are:

$\{ru > 0\} \mapsto \{r\downarrow, c\downarrow, ru\downarrow\}$; deliver pkg
$\{c > 0, ru = 0\} \mapsto \{dg\downarrow, r =\}$; go to target
$\{c = 0, pt > 0\} \mapsto \{c\uparrow, r =\}$; pick it up
$\{c = 0, pt = 0, r > 0\} \mapsto \{dp\downarrow, c =, r =\}$; go to nearest pkg

In contrast to previously learned general policies for Delivery (Francès, Bonet, and Geffner 2021), our learned policy generalizes to a more interesting class of Delivery tasks, where

Domain	LAMA		BFWS		Policy	
	S	T	S	T	S	T
Barman-11 (20)	**20**	0.90	**20**	0.05	**20**	0.55
Barman-14 (20)	**20**	2.07	**20**	0.10	**20**	0.81
Blocks (35)	**35**	0.01	**35**	0.01	**35**	0.11
Childsnack (20)	6	0.21	9	0.83	**20**	1.01
Driverlog (20)	**20**	0.03	**20**	0.00	15	0.16
Floortile-11 (20)	7	41.84	4	0.07	**20**	0.16
Floortile-14 (20)	2	3.54	2	0.03	**20**	0.17
Grid (5)	**5**	0.08	**5**	0.03	**5**	0.61
Gripper (20)	**20**	0.01	**20**	0.00	**20**	0.12
Hiking-Bin. (20)	**20**	0.88	14	0.05	**20**	0.93
Logistics-00 (28)	**28**	0.01	**28**	0.01	**28**	0.17
Logistics-98 (35)	**35**	0.18	34	1.74	**35**	4.34
Miconic (150)	**150**	0.02	**150**	0.01	**150**	0.23
Satellite (36)	**36**	0.14	33	0.47	32	3.28
Total (449)	404	0.19	394	0.04	**440**	0.43

Table 4: Testing results on IPC tasks. S is the number of solved tasks and T is the geometric mean total time in seconds over *commonly* solved tasks.

package target locations are not restricted to a single location but may be arbitrary grid locations. The key new feature is dg, which has syntactic complexity 18, compared to the complexity 15 feature sufficient for the single-target setting.

Logistics. Logistics tasks require delivering packages between locations in different cities, using trucks within cities and airplanes between airports. The learned controller follows this decomposition: it unloads when possible, otherwise uses trucks for same-city delivery, routes different-city packages to airports, flies them to the destination city, and finishes delivery with a destination-city truck. Its guards prevent loaded vehicles from being diverted and avoid truck-airplane cycles at airports.

The policy depends on recognizing whether packages and vehicles are in the package destination city. This matches the observation by Ståhlberg, Bonet, and Geffner (2023) that such relations are important for Logistics, and requires role composition. Although bottom-up approaches often exclude role composition because it makes the feature pool grow too quickly (Francès, Bonet, and Geffner 2021), our learned policy uses it, supporting its importance in this domain.

Driverlog. Driverlog tasks require satisfying goals for packages, trucks, and drivers. Packages are delivered by trucks, but trucks can only move when a driver is inside them. Drivers can walk between locations, board trucks, and drive them along roads, so a solution must coordinate package delivery with the target locations of both trucks and drivers.

We do not learn a generalizing Driverlog policy because the feature language is not expressive enough for one part of the domain. Consider states where all packages and trucks are at their goals, so only driver goals remain. The available distance features express shortest distances between sets of source and target objects, but cannot bind each driver to its own target. If one driver is at another driver's goal location, such a set-distance feature can be zero although the driver assignment is wrong. This issue was also observed by Drexler, Seipp, and Geffner (2024) in a hand-crafted width-1 policy sketch and solved with a feature that sums each driver's distance to its own goal. Indexical policies (Bonet, Drexler, and Geffner 2024) would offer another remedy by fixing an object from a concept denotation, such as a driver not at its target, and then measuring distance relative to that object.

Hiking-Binary. Hiking-Binary is a binary-relation reformulation of IPC Hiking in which couples advance along a linear trail, requiring both partners and a tent at each next place while cars transport people and tents forward. The domain is difficult because progress depends on preserving enough local resources: moving a person, tent, or car at the wrong time can leave a couple unable to re-establish the conditions needed for the next walk. The learned policy is complex, with 21 features of complexity up to 41 and 31 rules. It proves all solvable training tasks and solves all generated and IPC test tasks by greedy execution, although no sufficiently small validation tasks could be established as solvable. The policy relies on $\geq nR.C$ features to express multiplicity conditions, such as whether both partners are co-located at a relevant place and whether two cars are available nearby, showing another constructor that is hard for explicit enumeration to manage.

6 Conclusions

We have shown that language models can synthesize rule-based general policies over a rich description-logic feature grammar when every proposal is embedded in a symbolic verification loop. This is a substantially stronger target than one-shot plan generation: the output is a compact policy with a denotational semantics, and its candidate failures are exposed as graph-theoretic counterexamples. The resulting policies solve domains that have been difficult for prior generalized-planning approaches because they jointly stress scalability and feature expressivity.

The learned policies also show that language models can construct and use complex relational abstractions, not only select among hand-designed ones. In our case studies, useful features arise from description-logic expressions with syntactic complexity well above 30 and up to 100 in the final policies. The hard part of the workflow remains delegated to symbolic tools: grounding, graph search, policy proof, counterexample extraction, feature generation, and validation. The language model supplies the structured hypotheses that connect these tools.

Taken together, these results mark a milestone toward practical generalized planning: complex symbolic policies, previously out of reach for enumerative synthesis methods, can be discovered by combining language-model proposals with formal counterexample-guided verification. Future work should replace validation-only test evidence with proof obligations over parameterized task families, automate more of the repair loop, compare against systematic policy-synthesis baselines, and use obfuscated or encrypted PDDL to isolate name cues from relational structure.

7 Acknowledgements

This work was supported by the Swedish Foundation for Strategic Research and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. The research has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research, Germany, by the European Research Council (ERC), Grant agreement No. 885107, and by the Excellence Strategy of the Federal Government and the NRW Länder, Germany.

References

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Bonet, B.; Drexler, D.; and Geffner, H. 2024. On Policy Reuse: An Expressive Language for Representing and Executing General Policies that Call Other Policies. In Bernardini, S.; and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, 31–39. AAAI Press.

Bonet, B.; Francès, G.; and Geffner, H. 2019. Learning Features and Abstract Actions for Computing Generalized Plans. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, 2703–2710. AAAI Press.

Bonet, B.; and Geffner, H. 2018. Features, Projections, and Representation Change for Generalized Planning. In Lang, J., ed., *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, 4667–4673. IJCAI.

Bonet, B.; and Geffner, H. 2021. General Policies, Representations, and Planning Width. In Leyton-Brown, K.; and Mausam, eds., *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11764–11773. AAAI Press.

Clarke, E. M.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-Guided Abstraction Refinement. In Emerson, E. A.; and Sistla, A. P., eds., *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, 154–169.

Cui, Z.; Xia, H.; Shen, H.; Luo, K.; He, Y.; and Liang, W. 2026. Abstraction Generation for Generalized Planning with Pretrained Large Language Models. arXiv:2602.10485 [cs.AI].

de Graaff, R.; Corrêa, A. B.; and Pommerening, F. 2021. Concept Languages as Expert Input for Generalized Planning: Preliminary Results. In *ICAPS 2021 Workshop on Knowledge Engineering for Planning and Scheduling*.

Drexler, D.; Seipp, J.; and Geffner, H. 2022. Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width. In Thiébaux, S.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 62–70. AAAI Press.

Drexler, D.; Seipp, J.; and Geffner, H. 2024. Expressing and Exploiting Subgoal Structure in Classical Planning Using Sketches. *Journal of Artificial Intelligence Research*, 80: 171–208.

Drexler, D.; Ståhlberg, S.; Bonet, B.; and Geffner, H. 2024. Symmetries and Expressive Requirements for Learning General Policies. In *Proceedings of the Twenty-First International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*. IJCAI Organization.

Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning General Planning Policies from Small Examples Without Supervision. In Leyton-Brown, K.; and Mausam, eds., *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11801–11808. AAAI Press.

Francès, G.; Corrêa, A. B.; Geissmann, C.; and Pommerening, F. 2019. Generalized Potential Heuristics for Classical Planning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5554–5561. IJCAI.

Garg, S.; Bajpai, A.; and Mausam. 2019. Size Independent Neural Transfer for RDDL Planning. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 631–636. AAAI Press.

Hu, Y.; and Giacomo, G. D. 2011. Generalized Planning: Synthesizing Plans that Work for Multiple Environments. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 918–923. AAAI Press.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. arXiv:2201.07207 [cs.LG].

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 22895–22907. PMLR.

Lipovetzky, N.; and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. In Singh, S.; and Markovitch, S., eds., *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3590–3596. AAAI Press.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. arXiv:2304.11477 [cs.AI].

Martín, M.; and Geffner, H. 2004. Learning Generalized Policies from Planning Examples Using Concept Languages. *Applied Intelligence*, 20(1): 9–19.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized Planning With Deep Reinforcement Learning. In *ICAPS 2020 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, 16–24.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2019. Computing programs for generalized planning using a classical planner. *Artificial Intelligence*, 272: 52–85.

Segovia-Aguas, J.; Jiménez, S.; and Jonsson, A. 2021. Generalized Planning as Heuristic Search. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 569–577. AAAI Press.

Seipp, J.; and Helmert, M. 2013. Counterexample-guided Cartesian Abstraction Refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

Seipp, J.; and Helmert, M. 2018. Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning. *Journal of Artificial Intelligence Research*, 62: 535–577.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL Generators. https://doi.org/10.5281/zenodo.6382173.

Silver, T.; Dan, S.; Srinivas, K.; Tenenbaum, J.; Pack Kaelbling, L.; and Katz, M. 2024. Generalized Planning in PDDL Domains with Pretrained Large Language Models. In Dy, J.; and Natarajan, S., eds., *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, 20256–20264. AAAI Press.

Solar-Lezama, A.; Tancau, L.; Bodík, R.; Seshia, S. A.; and Saraswat, V. A. 2006. Combinatorial Sketching for Finite Programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2006)*, 404–415. ACM.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2): 393–401.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In Thiébaux, S.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 629–637. AAAI Press.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In Kern-Isberner, G.; Lakemeyer, G.; and Meyer, T., eds., *Proceedings of the Nineteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2022)*, 474–483. IJCAI Organization.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2023. Learning General Policies with Policy Gradient Methods. In Marquis, P.; Son, T. C.; and Kern-Isberner, G., eds., *Proceedings of the Twentieth International Conference on Principles of Knowledge Representation and Reasoning (KR 2023)*, 647–657. IJCAI Organization.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence Research*, 68: 1–68.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2018. Action Schema Networks: Generalised Policies with Deep Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 6294–6301. AAAI Press.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023. On the Planning Abilities of Large Language Models - A Critical Investigation. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 75993–76005.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2023a. Tree of thoughts: Deliberate Problem Solving with Large Language Models. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 11809–11822.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; and Cao, Y. 2023b. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*. OpenReview.net.

A Input Format for the Language Model

This section contains the input prompt used for the language model in our experiments. It is included here for reproducibility and to provide insight into how we structured the problem for the model. The prompt includes a description of the task, the format of the input and output, and examples of valid policies.

A.1 Objective

The objective is stated equally for every domain and stored as a file in `objective.md`. The input is:

```
# Experimentation

This file describes the general task. An additional markdown file grounds the placeholder
    into a concrete experiment. The input is `<domain_name>`, the name of the domain.

**The goal is simple:** Find a policy for which
   1) generality can be proven on the solvable training tasks, and
   2) plans can be found on the solvable validation tasks.

**Available Tools:**
1) `generate_features`: for generating features exhaustively up to a given complexity.
2) `prove_policy`: prove the policy on the grounded training tasks. It returns
    counterexamples and the constructed policy graph.
3) `execute_policy`: greedily follow the policy on a given task.
4) `prove_solvability`: run lifted GBFS lazy with hFF on a task to prove solvability.

**Available Documentation:**
1) `grammar.tex`: the syntax, abstract syntax, and semantic of implemented description logics
     constructors.

**What you CAN do:**
- Create helper code files in `workspace_<domain_name>/`.
- You can create feature manually as well through semantic reasoning on what features may be
    useful.
- You can build specialized tools based on the existing tools by combining existing tools or
    writing small domain-specific scripts.
- Prefer using the in-memory policy graph returned by `prove_policy` when deriving features
    or inspecting counterexamples.

**What you CANNOT do:**
- You are not allowed to modify `src/lpg/core/` or `src/lpg/tools/`.
- Do not run `prove_policy` on validation tasks because the exhaustive policy graph can be
    too large there; use `execute_policy` instead.
- You cannot use width > 0 in the `execute_policy` tool. Always use width 0 to execute the
    general policy and not a sketch.

**Initial steps:**
1. Filter train and validation tasks for solvability using the `prove_solvability` tool.
   Discard unsolvable tasks for subsequent steps. Use time limit 30 seconds per task.
   Report which tasks were solvable and unsolvable within that limit.
2. Write the empty policy to `output_<domain_name>/policy_0.txt`.
3. Use the empty policy to find counterexamples on the training tasks with
   `prove_policy`. This establishes the baseline failure mode and should guide
   the first real policy.

**Core loop:**
1. Propose or refine a policy based on the previous counterexamples.
2. Write every considered policy to `output_<domain_name>/policy_<i>.txt`.
   Do not overwrite earlier policy attempts; the sequence of files is the
   experiment history.
3. Use `prove_policy` only on the training tasks. Use its counterexamples and
   policy graph to understand which states are open, or which cycles remain.
4. Use `execute_policy` on the validation tasks only after the current policy
   proves the training tasks.
5. If execution on validation tasks fails, refine the policy from the failure and
   prove the refined policy again on the training tasks before executing it again.
```

```
5. If execution on validation tasks succeeds, assume that the goal was achieved.

**Constraints:**
- You should start by using the empty policy.
- You are required to report each candidate policy in `output_<domain_name>/policy_<i>.txt`
    where `<i>` is the iteration.
- You should avoid trying to overfit. For example, you are not allowed to use 1-step, 2-step,
     3-step-features and so on,
  if generalization toward arbitrary number can be expected. Instead, you should use distance
      features for example.
- You are only allowed to stop if you are sure that no policy that satisfies the goal can be
    found.
- You are required to write a log file `output_<domain_name>/run.log` that
  explains every tool invocation, why that tool was used, what was learned from
  the result, and the reasoning behind each change from one policy attempt to
  the next.
- When a policy attempt fails, record the failure category that motivated the
  next change: open state, transition into a deadend, cycle, validation failure,
  parser error, or another concrete blocker.
- When a policy attempt succeeds, copy it to `output_<domain_name>/policy.txt`
  and keep the numbered policy file that produced the success.
  For the final policy also report the maximum syntactic complexity of a feature
  and the syntactic complexity of the policy itself.

**Inputs:**
- `data/planning-benchmarks/learning/classical/<domain_name>/domain.pddl` is the PDDL
    planning domain.
- `data/planning-benchmarks/learning/classical/<domain_name>/train`: contains the training
    PDDL planning tasks.
- `data/planning-benchmarks/learning/classical/<domain_name>/valid`: contains the validation
    PDDL planning tasks.
```

A.2 Domain Specific Input

Per domain, we give the language model the following prompt:

```
# Experimentation: Learning a general policy for the <domain_name> domain

Arguments:
- `domain_name`: `<domain name>`

**Goal:** Learn a general policy for `<domain_name>` and the workflow specified in `objective.
    md`.

**Important:** Do not stop until finding a general policy!
```

B Learned General Policies

This section contains the learned general policies for all four domains, which were used in the experiments. The policies are presented in two sections, the features and the rules.

B.1 Barman

```
(policy
  (:features
       (numerical g "g" "unsatisfied contains-goals" (n_count (r_and (r_atomic_goal "contains"
            true) (r_complement (r_atomic_state "contains")))))
       (numerical gc "gc" "unsatisfied cocktail contains-goals" (n_count (r_restriction (r_and
            (r_atomic_goal "contains" true) (r_complement (r_atomic_state "contains"))) (
           c_atomic_state "cocktail"))))
       (numerical gi "gi" "unsatisfied ingredient contains-goals" (n_count (r_restriction (
           r_and (r_atomic_goal "contains" true) (r_complement (r_atomic_state "contains"))) (
           c_atomic_state "ingredient"))))
       (numerical hs "hs" "held shots count" (n_count (c_and (c_atomic_state "shot") (c_some (
           r_inverse (r_atomic_state "holding")) (c_top)))))
```

```
    (numerical hk "hk" "held shakers count" (n_count (c_and (c_atomic_state "shaker") (
       c_some (r_inverse (r_atomic_state "holding")) (c_top)))))
    (numerical hf "hf" "held shots containing a beverage" (n_count (c_and (c_and (
       c_atomic_state "shot") (c_some (r_inverse (r_atomic_state "holding")) (c_top))) (
       c_some (r_atomic_state "contains") (c_top)))))
    (numerical lv "lv" "shaker level distance" (n_distance (c_some (r_inverse (
       r_atomic_state "shaker-empty-level")) (c_top)) (r_atomic_state "next") (c_some (
       r_inverse (r_atomic_state "shaker-level")) (c_top))))
    (boolean skc "skc" "shaker clean" (b_nonempty (c_and (c_atomic_state "shaker") (
       c_atomic_state "clean"))))
    (boolean ske "ske" "shaker empty" (b_nonempty (c_and (c_atomic_state "shaker") (
       c_atomic_state "empty"))))
    (boolean sks "sks" "shaker shaked" (b_nonempty (c_atomic_state "shaked")))
    (boolean sku "sku" "shaker unshaked" (b_nonempty (c_atomic_state "unshaked")))
    (boolean skr "skr" "shaker has both parts of a needed cocktail" (b_nonempty (c_and (
       c_and (c_atomic_state "cocktail") (c_some (r_inverse (r_and (r_atomic_goal "
       contains" true) (r_complement (r_atomic_state "contains")))) (c_top))) (c_and (
       c_some (r_atomic_state "cocktail-part1") (c_some (r_inverse (r_atomic_state "
       contains")) (c_atomic_state "shaker"))) (c_some (r_atomic_state "cocktail-part2") (
       c_some (r_inverse (r_atomic_state "contains")) (c_atomic_state "shaker")))))))
    (numerical po "po" "needed cocktails shaked in shaker" (n_count (c_and (c_and (
       c_atomic_state "cocktail") (c_some (r_inverse (r_and (r_atomic_goal "contains" true)
        (r_complement (r_atomic_state "contains")))) (c_top))) (c_and (c_and (
       c_atomic_state "cocktail") (c_some (r_inverse (r_atomic_state "contains")) (
       c_atomic_state "shaker"))) (c_some (r_inverse (r_atomic_state "contains")) (c_and (
       c_atomic_state "shaker") (c_atomic_state "shaked")))))))
    (boolean hsu "hsu" "held shot has ingredient useful for some needed cocktail" (
       b_nonempty (c_and (c_some (r_inverse (r_atomic_state "contains")) (c_and (
       c_atomic_state "shot") (c_some (r_inverse (r_atomic_state "holding")) (c_top)))) (
       c_or (c_some (r_inverse (r_atomic_state "cocktail-part1")) (c_and (c_atomic_state "
       cocktail") (c_some (r_inverse (r_and (r_atomic_goal "contains" true) (r_complement
       (r_atomic_state "contains")))) (c_top)))) (c_some (r_inverse (r_atomic_state "
       cocktail-part2")) (c_and (c_atomic_state "cocktail") (c_some (r_inverse (r_and (
       r_atomic_goal "contains" true) (r_complement (r_atomic_state "contains")))) (c_top))
       ))))))
    (numerical hsig "hsig" "held shots that are goal shots for unsatisfied ingredient goal"
        (n_count (c_and (c_and (c_atomic_state "shot") (c_some (r_inverse (r_atomic_state "
       holding")) (c_top))) (c_some (r_and (r_atomic_goal "contains" true) (r_complement (
       r_atomic_state "contains"))) (c_atomic_state "ingredient")))))
    (numerical hng "hng" "held shots that are non-goal shots" (n_count (c_and (c_and (
       c_atomic_state "shot") (c_some (r_inverse (r_atomic_state "holding")) (c_top))) (
       c_and (c_atomic_state "shot") (c_not (c_some (r_atomic_goal "contains" true) (c_top)
       ))))))
    (boolean hcomp "hcomp" "held shot contains ing that completes a needed cocktail with
       shaker contents" (b_nonempty (c_and (c_some (r_inverse (r_atomic_state "contains"))
        (c_and (c_atomic_state "shot") (c_some (r_inverse (r_atomic_state "holding")) (
       c_top)))) (c_or (c_some (r_inverse (r_atomic_state "cocktail-part2")) (c_and (c_and
        (c_atomic_state "cocktail") (c_some (r_inverse (r_and (r_atomic_goal "contains"
       true) (r_complement (r_atomic_state "contains")))) (c_top))) (c_some (
       r_atomic_state "cocktail-part1") (c_some (r_inverse (r_atomic_state "contains")) (
       c_atomic_state "shaker"))))) (c_some (r_inverse (r_atomic_state "cocktail-part1"))
       (c_and (c_and (c_atomic_state "cocktail") (c_some (r_inverse (r_and (r_atomic_goal "
       contains" true) (r_complement (r_atomic_state "contains")))) (c_top))) (c_some (
       r_atomic_state "cocktail-part2") (c_some (r_inverse (r_atomic_state "contains")) (
       c_atomic_state "shaker")))))))))
    (boolean hsd "hsd" "held shot is dirty (has used flag)" (b_nonempty (c_and (c_and (
       c_atomic_state "shot") (c_some (r_inverse (r_atomic_state "holding")) (c_top))) (
       c_some (r_atomic_state "used") (c_top)))))
)
(:rules
    ((:conditions (greater_zero gi) (equal_zero hs) (equal_zero hk)) (:effects (increases
       hs) (increases hsig) (unchanged g) (unchanged gi) (unchanged gc) (unchanged hk) (
       unchanged hf) (unchanged lv) (positive skc) (positive ske) (negative sks) (negative
        sku) (negative skr) (unchanged po) (negative hsu) (unchanged hng) (negative hsd)))
```

((:conditions (greater_zero gi) (greater_zero hs) (greater_zero hsig) (equal_zero hk) (equal_zero hf) (positive skc) (positive ske) (negative hsd)) (:effects (decreases g) (decreases gi) (unchanged gc) (unchanged hs) (unchanged hk) (increases hf) (decreases hsig) (unchanged lv) (positive skc) (positive ske) (negative sks) (negative sku) (negative skr) (unchanged po) (unchanged hng) (negative hcomp) (positive hsd)))
((:conditions (greater_zero hs) (greater_zero hf) (equal_zero hk) (equal_zero hsig) (positive skc) (positive ske)) (:effects (unchanged g) (unchanged gi) (unchanged gc) (decreases hs) (unchanged hk) (decreases hf) (unchanged lv) (unchanged hsig) (unchanged hng) (positive skc) (positive ske) (negative sks) (negative sku) (negative skr) (unchanged po) (negative hsu) (negative hcomp) (negative hsd)))
((:conditions (greater_zero gc) (equal_zero hs) (equal_zero hk) (equal_zero gi) (positive skc) (positive ske)) (:effects (increases hs) (increases hng) (unchanged g) (unchanged gc) (unchanged gi) (unchanged hk) (unchanged hf) (unchanged lv) (positive skc) (positive ske) (negative sks) (negative sku) (negative skr) (unchanged po) (negative hsu) (unchanged hsig) (negative hsd)))
((:conditions (greater_zero gc) (greater_zero hs) (greater_zero hng) (equal_zero hk) (equal_zero hf) (positive skc) (positive ske) (negative hsd)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs) (unchanged hk) (increases hf) (unchanged lv) (unchanged hsig) (unchanged hng) (positive skc) (positive ske) (negative sks) (negative sku) (negative skr) (unchanged po) (positive hsu) (positive hsd)))
((:conditions (greater_zero gc) (greater_zero hs) (greater_zero hng) (equal_zero hk) (greater_zero hf) (positive skc) (positive ske) (positive hsu) (positive hsd)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs) (unchanged hk) (decreases hf) (increases lv) (unchanged hsig) (unchanged hng) (negative skc) (negative ske) (negative sks) (positive sku) (negative skr) (unchanged po) (negative hsu) (positive hsd)))
((:conditions (greater_zero gc) (greater_zero hs) (greater_zero hng) (equal_zero hk) (equal_zero hf) (negative skc) (negative ske) (positive sku) (negative skr) (positive hsd)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs) (unchanged hk) (unchanged hf) (unchanged lv) (unchanged hsig) (unchanged hng) (negative skc) (negative ske) (negative sks) (positive sku) (negative skr) (unchanged po) (negative hsu) (negative hsd)))
((:conditions (greater_zero gc) (greater_zero hs) (greater_zero hng) (equal_zero hk) (equal_zero hf) (negative skc) (negative ske) (positive sku) (negative skr) (negative hcomp) (negative hsd)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs) (unchanged hk) (increases hf) (unchanged lv) (unchanged hsig) (unchanged hng) (negative skc) (negative ske) (negative sks) (positive sku) (negative skr) (unchanged po) (positive hsu) (positive hcomp) (positive hsd)))
((:conditions (greater_zero gc) (greater_zero hs) (greater_zero hng) (equal_zero hk) (greater_zero hf) (negative skc) (negative ske) (positive sku) (negative skr) (positive hsu) (positive hcomp) (positive hsd)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs) (unchanged hk) (decreases hf) (increases lv) (unchanged hsig) (unchanged hng) (negative skc) (negative ske) (negative sks) (positive sku) (positive skr) (unchanged po) (negative hsu) (negative hcomp) (positive hsd)))
((:conditions (greater_zero gc) (greater_zero hs) (greater_zero hng) (equal_zero hk) (equal_zero hf) (negative skc) (negative ske) (positive sku) (positive skr) (positive hsd)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs) (unchanged hk) (unchanged hf) (unchanged lv) (unchanged hsig) (unchanged hng) (negative skc) (negative ske) (negative sks) (positive sku) (positive skr) (unchanged po) (negative hsu) (negative hsd)))
((:conditions (greater_zero gc) (greater_zero hs) (greater_zero hng) (equal_zero hk) (equal_zero hf) (negative skc) (negative ske) (positive sku) (positive skr) (negative hsd)) (:effects (unchanged g) (unchanged gi) (unchanged gc) (decreases hs) (unchanged hk) (unchanged hf) (unchanged lv) (unchanged hsig) (decreases hng) (negative skc) (negative ske) (negative sks) (positive sku) (positive skr) (unchanged po) (negative hsu) (negative hsd)))
((:conditions (greater_zero gc) (equal_zero hs) (equal_zero hk) (negative skc) (negative ske) (positive sku) (positive skr)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs) (increases hk) (unchanged hf) (unchanged lv) (unchanged hsig) (unchanged hng) (negative skc) (negative ske) (negative sks) (positive sku) (positive skr) (unchanged po) (negative hsu)))

```
      ((:conditions (greater_zero gc) (equal_zero hs) (greater_zero hk) (negative sks) (
         positive sku) (positive skr)) (:effects (unchanged g) (unchanged gc) (unchanged gi)
          (unchanged hs) (unchanged hk) (unchanged hf) (unchanged lv) (unchanged hsig) (
         unchanged hng) (negative skc) (negative ske) (positive sks) (negative sku) (
         negative skr) (increases po) (negative hsu)))
      ((:conditions (greater_zero gc) (equal_zero hs) (greater_zero hk) (positive sks) (
         greater_zero lv) (greater_zero po)) (:effects (decreases g) (decreases gc) (
         unchanged gi) (unchanged hs) (unchanged hk) (unchanged hf) (decreases lv) (
         unchanged hsig) (unchanged hng) (negative skc) (negative ske) (positive sks) (
         negative sku) (negative skr) (negative hsu) (negative hcomp)))
      ((:conditions (equal_zero hs) (greater_zero hk) (positive sks) (equal_zero po)) (:
         effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs) (unchanged hk) (
         unchanged hf) (unchanged hsig) (unchanged hng) (negative skc) (positive ske) (
         negative sks) (negative sku) (negative skr) (unchanged po) (negative hsu) (negative
          hcomp)))
      ((:conditions (equal_zero hs) (greater_zero hk) (positive sks) (greater_zero po) (
         equal_zero lv)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (unchanged hs)
          (unchanged hk) (unchanged hf) (unchanged hsig) (unchanged hng) (unchanged lv) (
         negative skc) (positive ske) (negative sks) (negative sku) (negative skr) (
         decreases po) (negative hsu) (negative hcomp)))
      ((:conditions (equal_zero hs) (greater_zero hk) (negative skc) (positive ske) (negative
          sks) (negative sku)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (
         unchanged hs) (unchanged hk) (unchanged hf) (unchanged lv) (unchanged hsig) (
         unchanged hng) (positive skc) (positive ske) (negative sks) (negative sku) (
         negative skr) (unchanged po) (negative hsu)))
      ((:conditions (equal_zero hs) (greater_zero hk) (positive skc) (positive ske) (negative
          sks) (negative sku)) (:effects (unchanged g) (unchanged gc) (unchanged gi) (
         unchanged hs) (decreases hk) (unchanged hf) (unchanged lv) (unchanged hsig) (
         unchanged hng) (positive skc) (positive ske) (negative sks) (negative sku) (
         negative skr) (unchanged po) (negative hsu)))
  )
)
```

B.2 Blocks-3

```
(policy
  (:features
    (numerical missing "m" "blocks with a goal support that are not currently on it" (n_count
        (c_and (c_some (r_atomic_goal "on" true) c_top) (c_not (c_same_as (r_atomic_state "on")
         (r_atomic_goal "on" true))))))
    (numerical stackable_ready "sr" "ready blocks whose goal support is clear and safe to use
        now" (n_count (c_and (c_atomic_state "clear") (c_and (c_some (r_atomic_goal "on" true)
         c_top) (c_and (c_not (c_same_as (r_atomic_state "on") (r_atomic_goal "on" true))) (
        c_some (r_atomic_goal "on" true) (c_and (c_atomic_state "clear") (c_and (c_not (c_and
        (c_some (r_atomic_goal "on" true) c_top) (c_not (c_same_as (r_atomic_state "on") (
        r_atomic_goal "on" true))))) (c_and (c_not (c_some (r_transitive_closure (
        r_atomic_state "on")) (c_and (c_some (r_atomic_goal "on" true) c_top) (c_not (
        c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))) (c_not (c_some (
        r_transitive_closure (r_atomic_state "on")) (c_some (r_inverse (r_atomic_goal "on"
        true)) (c_and (c_some (r_atomic_goal "on" true) c_top) (c_not (c_same_as (
        r_atomic_state "on") (r_atomic_goal "on" true))))))))))))))))
    (numerical inv_mis "im" "satisfied goal links whose support is itself misplaced" (n_count
        (c_and (c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)) (c_some (
        r_atomic_goal "on" true) (c_and (c_some (r_atomic_goal "on" true) c_top) (c_not (
        c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))))))
    (numerical inv_src "is" "satisfied goal links whose support is above a missing source" (
        n_count (c_and (c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)) (c_some (
        r_atomic_goal "on" true) (c_some (r_transitive_closure (r_atomic_state "on")) (c_and (
        c_some (r_atomic_goal "on" true) c_top) (c_not (c_same_as (r_atomic_state "on") (
        r_atomic_goal "on" true)))))))))
    (numerical inv_tgt "it" "satisfied goal links whose support is above a missing target" (
        n_count (c_and (c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)) (c_some (
        r_atomic_goal "on" true) (c_some (r_transitive_closure (r_atomic_state "on")) (c_some
        (r_inverse (r_atomic_goal "on" true)) (c_and (c_some (r_atomic_goal "on" true) c_top)
        (c_not (c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))))))))
```

```
   (numerical inv_mis_above "ima" "blocks above satisfied links whose support is misplaced" (
      n_count (c_some (r_transitive_closure (r_atomic_state "on")) (c_and (c_same_as (
      r_atomic_state "on") (r_atomic_goal "on" true)) (c_some (r_atomic_goal "on" true) (
      c_and (c_some (r_atomic_goal "on" true) c_top) (c_not (c_same_as (r_atomic_state "on")
       (r_atomic_goal "on" true)))))))))
   (numerical inv_src_above "isa" "blocks above satisfied links whose support is above a
      missing source" (n_count (c_some (r_transitive_closure (r_atomic_state "on")) (c_and (
      c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)) (c_some (r_atomic_goal "on"
       true) (c_some (r_transitive_closure (r_atomic_state "on")) (c_and (c_some (
      r_atomic_goal "on" true) c_top) (c_not (c_same_as (r_atomic_state "on") (r_atomic_goal
       "on" true)))))))))))
   (numerical inv_tgt_above "ita" "blocks above satisfied links whose support is above a
      missing target" (n_count (c_some (r_transitive_closure (r_atomic_state "on")) (c_and (
      c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)) (c_some (r_atomic_goal "on"
       true) (c_some (r_transitive_closure (r_atomic_state "on")) (c_some (r_inverse (
      r_atomic_goal "on" true)) (c_and (c_some (r_atomic_goal "on" true) c_top) (c_not (
      c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))))))))
   (numerical src_above "sa" "blocks above misplaced source blocks" (n_count (c_some (
      r_transitive_closure (r_atomic_state "on")) (c_and (c_some (r_atomic_goal "on" true)
      c_top) (c_not (c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))))
   (numerical tgt_above "ta" "blocks above goal supports of misplaced source blocks" (n_count
       (c_some (r_transitive_closure (r_atomic_state "on")) (c_some (r_inverse (
      r_atomic_goal "on" true)) (c_and (c_some (r_atomic_goal "on" true) c_top) (c_not (
      c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))))))
   (numerical tabled "t" "blocks on the table" (n_count (c_atomic_state "on-table")))
  )
  (:rules
   ((:conditions (greater_zero inv_mis_above)) (:effects (decreases inv_mis_above) (increases
       tabled)))
   ((:conditions (equal_zero inv_mis_above) (greater_zero inv_mis)) (:effects (decreases
      inv_mis) (increases tabled)))
   ((:conditions (equal_zero inv_mis_above) (equal_zero inv_mis) (greater_zero inv_src_above)
      ) (:effects (decreases inv_src_above) (increases tabled)))
   ((:conditions (equal_zero inv_mis_above) (equal_zero inv_mis) (equal_zero inv_src_above) (
      greater_zero inv_src)) (:effects (decreases inv_src) (increases tabled)))
   ((:conditions (equal_zero inv_mis_above) (equal_zero inv_mis) (equal_zero inv_src_above) (
      equal_zero inv_src) (greater_zero inv_tgt_above)) (:effects (decreases inv_tgt_above)
      (increases tabled)))
   ((:conditions (equal_zero inv_mis_above) (equal_zero inv_mis) (equal_zero inv_src_above) (
      equal_zero inv_src) (equal_zero inv_tgt_above) (greater_zero inv_tgt)) (:effects (
      decreases inv_tgt) (increases tabled)))
   ((:conditions (equal_zero inv_mis_above) (equal_zero inv_mis) (equal_zero inv_src_above) (
      equal_zero inv_src) (equal_zero inv_tgt_above) (equal_zero inv_tgt) (greater_zero
      stackable_ready)) (:effects (decreases missing) (unchanged inv_mis) (unchanged inv_src)
       (unchanged inv_tgt)))
   ((:conditions (equal_zero inv_mis_above) (equal_zero inv_mis) (equal_zero inv_src_above) (
      equal_zero inv_src) (equal_zero inv_tgt_above) (equal_zero inv_tgt) (equal_zero
      stackable_ready) (greater_zero src_above)) (:effects (decreases src_above) (increases
      tabled)))
   ((:conditions (equal_zero inv_mis_above) (equal_zero inv_mis) (equal_zero inv_src_above) (
      equal_zero inv_src) (equal_zero inv_tgt_above) (equal_zero inv_tgt) (equal_zero
      stackable_ready) (equal_zero src_above) (greater_zero tgt_above)) (:effects (decreases
       tgt_above) (increases tabled)))
  )
)
```

B.3 Blocks-4

```
(policy
  (:features
   (numerical remaining "r" "goal on-relations not currently true" (n_count (r_and (
      r_atomic_goal "on" true) (r_complement (r_atomic_state "on")))))
   (numerical holding "h" "blocks currently held" (n_count (c_atomic_state "holding")))
   (numerical on_count "oc" "current on-relations" (n_count (r_atomic_state "on")))
```

```
   (numerical ready_stack "rs" "held blocks whose goal support is clear and stable" (n_count
       (c_and (c_atomic_state "holding") (c_some (r_atomic_goal "on" true) (c_and (
       c_atomic_state "clear") (c_and (c_or (c_and (c_atomic_state "on-table") (c_not (c_some
        (r_atomic_goal "on" true) c_top))) (c_same_as (r_atomic_state "on") (r_atomic_goal "
       on" true))) (c_not (c_some (r_transitive_closure (r_atomic_state "on")) (c_not (c_or (
       c_and (c_atomic_state "on-table") (c_not (c_some (r_atomic_goal "on" true) c_top))) (
       c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))))))))
   (numerical wrong_clear_on "wco" "clear blocks on another block but not correctly on their
       goal support" (n_count (c_and (c_atomic_state "clear") (c_and (c_some (r_atomic_state "
       on") c_top) (c_not (c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))))
   (numerical bad_depth "bd" "transitive on-pairs above blocks that are not in place" (
       n_count (r_restriction (r_transitive_closure (r_atomic_state "on")) (c_not (c_or (
       c_and (c_atomic_state "on-table") (c_not (c_some (r_atomic_goal "on" true) c_top))) (
       c_same_as (r_atomic_state "on") (r_atomic_goal "on" true)))))))
   (numerical ready_table "rt" "clear table blocks ready to be picked up for correct stacking
       " (n_count (c_and (c_atomic_state "clear") (c_and (c_atomic_state "on-table") (c_and (
       c_not (c_same_as (r_atomic_state "on") (r_atomic_goal "on" true))) (c_some (
       r_atomic_goal "on" true) (c_and (c_atomic_state "clear") (c_and (c_or (c_and (
       c_atomic_state "on-table") (c_not (c_some (r_atomic_goal "on" true) c_top))) (
       c_same_as (r_atomic_state "on") (r_atomic_goal "on" true))) (c_not (c_some (
       r_transitive_closure (r_atomic_state "on")) (c_not (c_or (c_and (c_atomic_state "on-
       table") (c_not (c_some (r_atomic_goal "on" true) c_top))) (c_same_as (r_atomic_state "
       on") (r_atomic_goal "on" true))))))))))))))
   (numerical bottom_ready "br" "clear table blocks with no goal support" (n_count (c_and (
       c_atomic_state "clear") (c_and (c_atomic_state "on-table") (c_not (c_some (
       r_atomic_goal "on" true) c_top))))))
  )
  (:rules
   ((:conditions (greater_zero holding) (greater_zero ready_stack)) (:effects (decreases
       remaining) (decreases holding) (decreases ready_stack) (increases on_count)))
   ((:conditions (greater_zero holding) (equal_zero ready_stack)) (:effects (decreases
       holding) (unchanged on_count) (unchanged remaining)))
   ((:conditions (equal_zero holding) (greater_zero wrong_clear_on)) (:effects (decreases
       on_count) (increases holding) (unchanged remaining)))
   ((:conditions (equal_zero holding) (equal_zero wrong_clear_on) (greater_zero bad_depth)) (:
       effects (decreases on_count) (increases holding) (increases remaining) (decreases
       bad_depth)))
   ((:conditions (equal_zero holding) (equal_zero wrong_clear_on) (equal_zero bad_depth) (
       greater_zero ready_table)) (:effects (increases holding) (decreases ready_table) (
       unchanged bottom_ready) (unchanged remaining)))
  )
)
```

B.4 Childsnack

```
(policy
  (:features
   (numerical allergic_unserved "au" "allergic children not yet served" (n_count (c_and (
       c_atomic_state "child") (c_and (c_atomic_state "allergic_gluten") (c_not (
       c_atomic_state "served"))))))
   (numerical nonallergic_unserved "nu" "non-allergic children not yet served" (n_count (
       c_and (c_atomic_state "child") (c_and (c_atomic_state "not_allergic_gluten") (c_not (
       c_atomic_state "served"))))))
   (numerical tray_at_kitchen "tk" "trays at the kitchen" (n_count (c_and (c_atomic_state "
       tray") (c_some (r_atomic_state "at") (c_one_of "kitchen"))))))
   (numerical ng_unplaced "ngu" "gluten-free sandwiches at the kitchen" (n_count (c_and (
       c_atomic_state "sandwich") (c_and (c_atomic_state "at_kitchen_sandwich") (
       c_atomic_state "no_gluten_sandwich"))))))
   (numerical regular_unplaced "ru" "regular sandwiches at the kitchen" (n_count (c_and (
       c_atomic_state "sandwich") (c_and (c_atomic_state "at_kitchen_sandwich") (c_not (
       c_atomic_state "no_gluten_sandwich"))))))
   (numerical ng_ontray "ngo" "gluten-free sandwiches on trays" (n_count (c_and (
       c_atomic_state "sandwich") (c_and (c_atomic_state "no_gluten_sandwich") (c_some (
       r_atomic_state "ontray") (c_atomic_state "tray"))))))
```

```
  (numerical regular_ontray "ro" "regular sandwiches on trays" (n_count (c_and (
      c_atomic_state "sandwich") (c_and (c_not (c_atomic_state "no_gluten_sandwich")) (
      c_some (r_atomic_state "ontray") (c_atomic_state "tray"))))))
  (numerical ready_allergic "ra" "allergic children at a tray carrying a gluten-free
      sandwich" (n_count (c_and (c_atomic_state "child") (c_and (c_atomic_state "
      allergic_gluten") (c_and (c_not (c_atomic_state "served")) (c_some (r_atomic_state "
      waiting") (c_some (r_inverse (r_atomic_state "at")) (c_some (r_inverse (r_atomic_state
       "ontray")) (c_atomic_state "no_gluten_sandwich")))))))))
  (numerical ready_nonallergic "rn" "non-allergic children at a tray carrying a regular
      sandwich" (n_count (c_and (c_atomic_state "child") (c_and (c_atomic_state "
      not_allergic_gluten") (c_and (c_not (c_atomic_state "served")) (c_some (r_atomic_state
       "waiting") (c_some (r_inverse (r_atomic_state "at")) (c_some (r_inverse (
      r_atomic_state "ontray")) (c_and (c_atomic_state "sandwich") (c_not (c_atomic_state "
      no_gluten_sandwich")))))))))))
 )
 (:rules
  ((:conditions (greater_zero ready_allergic)) (:effects (decreases allergic_unserved) (
      decreases ng_ontray)))
  ((:conditions (greater_zero allergic_unserved) (equal_zero ready_allergic) (greater_zero
      ng_ontray)) (:effects (increases ready_allergic) (unchanged allergic_unserved) (
      unchanged ng_ontray)))
  ((:conditions (greater_zero allergic_unserved) (equal_zero ready_allergic) (equal_zero
      ng_ontray) (greater_zero ng_unplaced) (greater_zero tray_at_kitchen)) (:effects (
      decreases ng_unplaced) (increases ng_ontray) (unchanged allergic_unserved)))
  ((:conditions (greater_zero allergic_unserved) (equal_zero ready_allergic) (equal_zero
      ng_ontray) (greater_zero ng_unplaced) (equal_zero tray_at_kitchen)) (:effects (
      increases tray_at_kitchen) (unchanged ng_unplaced) (unchanged allergic_unserved)))
  ((:conditions (greater_zero allergic_unserved) (equal_zero ready_allergic) (equal_zero
      ng_ontray) (equal_zero ng_unplaced)) (:effects (increases ng_unplaced) (unchanged
      allergic_unserved)))
  ((:conditions (equal_zero allergic_unserved) (greater_zero ready_nonallergic)) (:effects (
      decreases nonallergic_unserved) (decreases regular_ontray)))
  ((:conditions (equal_zero allergic_unserved) (greater_zero nonallergic_unserved) (
      equal_zero ready_nonallergic) (greater_zero regular_ontray)) (:effects (increases
      ready_nonallergic) (unchanged nonallergic_unserved) (unchanged regular_ontray)))
  ((:conditions (equal_zero allergic_unserved) (greater_zero nonallergic_unserved) (
      equal_zero ready_nonallergic) (equal_zero regular_ontray) (greater_zero
      regular_unplaced) (greater_zero tray_at_kitchen)) (:effects (decreases
      regular_unplaced) (increases regular_ontray) (unchanged nonallergic_unserved)))
  ((:conditions (equal_zero allergic_unserved) (greater_zero nonallergic_unserved) (
      equal_zero ready_nonallergic) (equal_zero regular_ontray) (greater_zero
      regular_unplaced) (equal_zero tray_at_kitchen)) (:effects (increases tray_at_kitchen)
      (unchanged regular_unplaced) (unchanged nonallergic_unserved)))
  ((:conditions (equal_zero allergic_unserved) (greater_zero nonallergic_unserved) (
      equal_zero ready_nonallergic) (equal_zero regular_ontray) (equal_zero regular_unplaced)
      ) (:effects (increases regular_unplaced) (unchanged nonallergic_unserved)))
 )
)
```

B.5 Delivery

```
(policy
 (:features
  (numerical remaining "r" "packages not at their goal location" (n_count (c_and (
      c_atomic_state "package") (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at"
      true))))))
  (numerical carried "c" "packages currently carried by the truck" (n_count (c_and (
      c_atomic_state "package") (c_some (r_inverse (r_atomic_state "carrying")) (
      c_atomic_state "truck")))))
  (numerical ready_unload "ru" "carried packages whose truck is at the package goal location
      " (n_count (c_and (c_atomic_state "package") (c_and (c_some (r_inverse (r_atomic_state
       "carrying")) (c_atomic_state "truck")) (c_same_as (r_composition (r_inverse (
      r_atomic_state "carrying")) (r_atomic_state "at")) (r_atomic_goal "at" true))))))
  (numerical pkg_at_truck "pt" "undelivered uncarried packages at the truck location" (
      n_count (c_and (c_atomic_state "package") (c_and (c_some (r_atomic_state "at") (
```

```
      c_atomic_state "cell")) (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "
      at" true))) (c_some (r_atomic_state "at") (c_some (r_inverse (r_atomic_state "at")) (
      c_atomic_state "truck"))))))))
  (numerical dist_to_pkg "dp" "truck grid distance to an undelivered package location" (
      n_distance (c_some (r_inverse (r_atomic_state "at")) (c_atomic_state "truck")) (
      r_atomic_state "adjacent") (c_some (r_inverse (r_atomic_state "at")) (c_and (
      c_atomic_state "package") (c_and (c_some (r_atomic_state "at") (c_atomic_state "cell"))
       (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))))))))
  (numerical dist_to_goal "dg" "truck grid distance to the goal location of the carried
      package" (n_distance (c_some (r_inverse (r_atomic_state "at")) (c_atomic_state "truck")
      ) (r_atomic_state "adjacent") (c_some (r_inverse (r_atomic_goal "at" true)) (c_and (
      c_atomic_state "package") (c_some (r_inverse (r_atomic_state "carrying")) (
      c_atomic_state "truck"))))))
 )
 (:rules
  ((:conditions (greater_zero ready_unload)) (:effects (decreases remaining) (decreases
      carried) (decreases ready_unload)))
  ((:conditions (greater_zero carried) (equal_zero ready_unload)) (:effects (decreases
      dist_to_goal) (unchanged carried) (unchanged remaining)))
  ((:conditions (equal_zero carried) (greater_zero pkg_at_truck)) (:effects (increases
      carried) (unchanged remaining)))
  ((:conditions (equal_zero carried) (equal_zero pkg_at_truck) (greater_zero remaining)) (:
      effects (decreases dist_to_pkg) (unchanged carried) (unchanged remaining)))
 )
)
```

B.6 Driverlog

```
(policy
 (:features
  (numerical pkg_remaining "pr" "packages not at their goal location" (n_count (c_and (c_and
       (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "location")))
       (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true))))))
  (numerical pkg_carried "pc" "packages in trucks" (n_count (c_and (c_atomic_state "obj") (
      c_some (r_atomic_state "in") (c_atomic_state "truck"))))))
  (numerical pkg_ready_unload "pu" "carried packages whose truck is at the package goal" (
      n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_state "in") (
      c_atomic_state "truck"))) (c_same_as (r_composition (r_atomic_state "in") (
      r_atomic_state "at")) (r_atomic_goal "at" true)))))
  (numerical pkg_at_truck "pt" "undelivered packages at a truck location" (n_count (c_and (
      c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "
      location"))) (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))
       (c_some (r_atomic_state "at") (c_some (r_inverse (r_atomic_state "at")) (
      c_atomic_state "truck")))))))
  (numerical pkg_truck_distance "pd" "link distance from a driven truck to an undelivered
      package location" (n_distance (c_some (r_inverse (r_atomic_state "at")) (c_some (
      r_inverse (r_atomic_state "driving")) (c_atomic_state "driver"))) (r_atomic_state "link
      ") (c_some (r_inverse (r_atomic_state "at")) (c_and (c_atomic_state "obj") (c_and (
      c_some (r_atomic_goal "at" true) (c_atomic_state "location")) (c_not (c_same_as (
      r_atomic_state "at") (r_atomic_goal "at" true)))))) ))
  (numerical pkg_goal_distance "pgd" "link distance from a truck carrying a package to a
      carried package goal" (n_distance (c_some (r_inverse (r_atomic_state "at")) (c_some (
      r_inverse (r_atomic_state "in")) (c_atomic_state "obj"))) (r_atomic_state "link") (
      c_some (r_inverse (r_atomic_goal "at" true)) (c_some (r_atomic_state "in") (
      c_atomic_state "truck")))))
  (numerical driving "dr" "drivers currently driving" (n_count (c_and (c_atomic_state "
      driver") (c_some (r_atomic_state "driving") (c_atomic_state "truck")))))
  (numerical driver_at_empty_truck "det" "drivers colocated with an empty truck" (n_count (
      c_and (c_atomic_state "driver") (c_some (r_atomic_state "at") (c_some (r_inverse (
      r_atomic_state "at")) (c_and (c_atomic_state "truck") (c_atomic_state "empty")))))))
  (numerical driver_to_empty_truck_distance "ded" "path distance from a driver to an empty
      truck" (n_distance (c_some (r_inverse (r_atomic_state "at")) (c_atomic_state "driver"))
       (r_atomic_state "path") (c_some (r_inverse (r_atomic_state "at")) (c_and (
      c_atomic_state "truck") (c_atomic_state "empty")))))
```

```
(numerical truck_remaining "tr" "trucks not at their goal location" (n_count (c_and (c_and
     (c_atomic_state "truck") (c_some (r_atomic_goal "at" true) (c_atomic_state "location")
    )) (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true))))))
(numerical goal_truck_driven "gtd" "unsatisfied goal trucks currently being driven" (
    n_count (c_and (c_and (c_atomic_state "truck") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_and (c_not (c_same_as (r_atomic_state "at") (
    r_atomic_goal "at" true))) (c_some (r_inverse (r_atomic_state "driving")) (
    c_atomic_state "driver"))))))
(numerical driver_at_goal_truck "dgt" "drivers colocated with an unsatisfied goal truck" (
    n_count (c_and (c_atomic_state "driver") (c_some (r_atomic_state "at") (c_some (
    r_inverse (r_atomic_state "at")) (c_and (c_and (c_atomic_state "truck") (c_some (
    r_atomic_goal "at" true) (c_atomic_state "location"))) (c_not (c_same_as (
    r_atomic_state "at") (r_atomic_goal "at" true)))))))))
(numerical driver_to_goal_truck_distance "dgd" "path distance from a driver to an
    unsatisfied goal truck" (n_distance (c_some (r_inverse (r_atomic_state "at")) (
    c_atomic_state "driver")) (r_atomic_state "path") (c_some (r_inverse (r_atomic_state "
    at")) (c_and (c_and (c_atomic_state "truck") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "
    at" true)))))))
(numerical goal_truck_distance "gtdist" "link distance from driven unsatisfied goal trucks
     to their goals" (n_distance (c_some (r_inverse (r_atomic_state "at")) (c_and (c_and (
    c_atomic_state "truck") (c_some (r_atomic_goal "at" true) (c_atomic_state "location")))
     (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true))) (c_some (
    r_inverse (r_atomic_state "driving")) (c_atomic_state "driver"))))) (r_atomic_state "
    link") (c_some (r_inverse (r_atomic_goal "at" true)) (c_and (c_atomic_state "truck") (
    c_some (r_inverse (r_atomic_state "driving")) (c_atomic_state "driver"))))))
(numerical driver_remaining "der" "drivers not at their goal location" (n_count (c_and (
    c_and (c_atomic_state "driver") (c_some (r_atomic_goal "at" true) (c_atomic_state "
    location"))) (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true))))))
(numerical driver_ready_exit "dre" "driving drivers whose truck is at their goal" (n_count
     (c_and (c_and (c_atomic_state "driver") (c_some (r_atomic_state "driving") (
    c_atomic_state "truck"))) (c_same_as (r_composition (r_atomic_state "driving") (
    r_atomic_state "at")) (r_atomic_goal "at" true)))))
(numerical driver_goal_distance "dgd2" "path distance from an unsatisfied on-foot driver
    to a driver goal" (n_distance (c_some (r_inverse (r_atomic_state "at")) (c_and (c_and
    (c_atomic_state "driver") (c_some (r_atomic_goal "at" true) (c_atomic_state "location")
    )) (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true))))) (
    r_atomic_state "path") (c_some (r_inverse (r_atomic_goal "at" true)) (c_atomic_state "
    driver"))))
(numerical driver_unsat_goal_distance "dugd" "path distance from unsatisfied on-foot
    drivers to unsatisfied driver goals" (n_distance (c_some (r_inverse (r_atomic_state "
    at")) (c_and (c_and (c_atomic_state "driver") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "
    at" true))))) (r_atomic_state "path") (c_some (r_inverse (r_atomic_goal "at" true)) (
    c_and (c_and (c_atomic_state "driver") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "
    at" true)))))))
(numerical pkg_carrier_driven "pcd" "carried packages in a truck with a driver" (n_count (
    c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_state "in") (c_atomic_state "
    truck"))) (c_some (r_composition (r_atomic_state "in") (r_inverse (r_atomic_state "
    driving"))) (c_atomic_state "driver")))))
(numerical driver_at_carried_truck "dct" "drivers colocated with a truck carrying an
    undelivered package" (n_count (c_and (c_atomic_state "driver") (c_some (r_atomic_state
     "at") (c_some (r_inverse (r_atomic_state "at")) (c_and (c_atomic_state "truck") (
    c_some (r_inverse (r_atomic_state "in")) (c_and (c_atomic_state "obj") (c_and (c_some
    (r_atomic_goal "at" true) (c_atomic_state "location")) (c_not (c_same_as (
    r_composition (r_atomic_state "in") (r_atomic_state "at")) (r_atomic_goal "at" true))))
    ))))))))
(numerical driver_to_carried_truck_distance "dctd" "path distance from a driver to a truck
     carrying an undelivered package" (n_distance (c_some (r_inverse (r_atomic_state "at"))
     (c_atomic_state "driver")) (r_atomic_state "path") (c_some (r_inverse (r_atomic_state
     "at")) (c_and (c_atomic_state "truck") (c_some (r_inverse (r_atomic_state "in")) (
    c_and (c_atomic_state "obj") (c_and (c_some (r_atomic_goal "at" true) (c_atomic_state "
    location")) (c_not (c_same_as (r_composition (r_atomic_state "in") (r_atomic_state "at
    ")) (r_atomic_goal "at" true)))))))))))
```

```
  (numerical driver_at_free_truck "dft" "drivers colocated with an empty truck that has no
     goal" (n_count (c_and (c_atomic_state "driver") (c_some (r_atomic_state "at") (c_some
     (r_inverse (r_atomic_state "at")) (c_and (c_and (c_atomic_state "truck") (
     c_atomic_state "empty")) (c_not (c_some (r_atomic_goal "at" true) (c_atomic_state "
     location")))))))))
  (numerical driver_to_free_truck_distance "dftd" "path distance from a driver to an empty
     truck that has no goal" (n_distance (c_some (r_inverse (r_atomic_state "at")) (
     c_atomic_state "driver")) (r_atomic_state "path") (c_some (r_inverse (r_atomic_state "
     at")) (c_and (c_and (c_atomic_state "truck") (c_atomic_state "empty")) (c_not (c_some
     (r_atomic_goal "at" true) (c_atomic_state "location")))))))
  (numerical driven_driver_goal_distance "ddgd" "link distance from a driven unsatisfied
     driver's truck to that driver's goal" (n_distance (c_some (r_inverse (r_atomic_state "
     at")) (c_some (r_inverse (r_atomic_state "driving")) (c_and (c_and (c_atomic_state "
     driver") (c_some (r_atomic_goal "at" true) (c_atomic_state "location"))) (c_not (
     c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))))) (r_atomic_state "link")
      (c_some (r_inverse (r_atomic_goal "at" true)) (c_some (r_atomic_state "driving") (
     c_atomic_state "truck")))))
  (numerical wrong_driver_goal_occupant "wdg" "unsatisfied drivers currently at some driver
     goal location" (n_count (c_and (c_and (c_and (c_atomic_state "driver") (c_some (
     r_atomic_goal "at" true) (c_atomic_state "location"))) (c_not (c_same_as (
     r_atomic_state "at") (r_atomic_goal "at" true)))) (c_some (r_atomic_state "at") (
     c_some (r_inverse (r_atomic_goal "at" true)) (c_atomic_state "driver"))))))
  (numerical swapped_driver_goal_pair "sdg" "unsatisfied drivers mutually occupying each
     others goal locations" (n_count (c_and (c_and (c_and (c_atomic_state "driver") (c_some
      (r_atomic_goal "at" true) (c_atomic_state "location"))) (c_not (c_same_as (
     r_atomic_state "at") (r_atomic_goal "at" true)))) (c_some (r_and (r_composition (
     r_atomic_state "at") (r_inverse (r_atomic_goal "at" true))) (r_composition (
     r_atomic_goal "at" true) (r_inverse (r_atomic_state "at")))) (c_and (c_and (
     c_atomic_state "driver") (c_some (r_atomic_goal "at" true) (c_atomic_state "location"))
     ) (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))))))
  (numerical driver_one_step "dos" "unsatisfied drivers one path step from their own goal" (
     n_count (r_and (r_identity (c_and (c_and (c_atomic_state "driver") (c_some (
     r_atomic_goal "at" true) (c_atomic_state "location"))) (c_not (c_same_as (
     r_atomic_state "at") (r_atomic_goal "at" true))))) (r_composition (r_composition (
     r_atomic_state "at") (r_atomic_state "path")) (r_inverse (r_atomic_goal "at" true))))))

  (numerical driven_goal_truck_one_step "t-os" "driven unsatisfied goal trucks one link step
      from their own goal" (n_count (r_and (r_identity (c_and (c_and (c_and (c_atomic_state
      "truck") (c_some (r_atomic_goal "at" true) (c_atomic_state "location"))) (c_not (
     c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))) (c_some (r_inverse (
     r_atomic_state "driving")) (c_atomic_state "driver")))) (r_composition (r_composition
     (r_atomic_state "at") (r_atomic_state "link")) (r_inverse (r_atomic_goal "at" true)))))
     )
)
(:rules
  ((:conditions (greater_zero pkg_ready_unload)) (:effects (decreases pkg_remaining) (
     decreases pkg_carried) (decreases pkg_ready_unload)))
  ((:conditions (greater_zero pkg_carried) (equal_zero pkg_ready_unload) (greater_zero
     pkg_carrier_driven)) (:effects (decreases pkg_goal_distance) (unchanged pkg_carried) (
     unchanged pkg_remaining)))
  ((:conditions (greater_zero pkg_carried) (equal_zero pkg_ready_unload) (equal_zero
     pkg_carrier_driven) (greater_zero driver_at_carried_truck)) (:effects (increases
     pkg_carrier_driven) (unchanged pkg_carried) (unchanged pkg_remaining)))
  ((:conditions (greater_zero pkg_carried) (equal_zero pkg_ready_unload) (equal_zero
     pkg_carrier_driven) (equal_zero driver_at_carried_truck)) (:effects (decreases
     driver_to_carried_truck_distance) (unchanged pkg_carried) (unchanged pkg_remaining)))
  ((:conditions (equal_zero pkg_carried) (greater_zero pkg_at_truck)) (:effects (increases
     pkg_carried) (decreases pkg_at_truck) (unchanged pkg_remaining)))
  ((:conditions (equal_zero pkg_carried) (equal_zero pkg_at_truck) (greater_zero
     pkg_remaining) (greater_zero driving)) (:effects (decreases pkg_truck_distance) (
     unchanged pkg_remaining)))
  ((:conditions (equal_zero pkg_carried) (equal_zero pkg_at_truck) (greater_zero
     pkg_remaining) (equal_zero driving) (greater_zero driver_at_empty_truck)) (:effects (
     increases driving) (unchanged pkg_remaining)))
  ((:conditions (equal_zero pkg_carried) (equal_zero pkg_at_truck) (greater_zero
```

```
      pkg_remaining) (equal_zero driving) (equal_zero driver_at_empty_truck)) (:effects (
      decreases driver_to_empty_truck_distance) (unchanged pkg_remaining)))
   ((:conditions (equal_zero pkg_remaining) (greater_zero goal_truck_driven) (greater_zero
      driven_goal_truck_one_step)) (:effects (decreases truck_remaining) (decreases
      driven_goal_truck_one_step)))
   ((:conditions (equal_zero pkg_remaining) (greater_zero goal_truck_driven) (equal_zero
      driven_goal_truck_one_step)) (:effects (decreases goal_truck_distance) (unchanged
      goal_truck_driven) (unchanged truck_remaining)))
   ((:conditions (equal_zero pkg_remaining) (greater_zero goal_truck_driven) (equal_zero
      driven_goal_truck_one_step)) (:effects (increases driven_goal_truck_one_step) (
      unchanged goal_truck_driven) (unchanged truck_remaining)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero goal_truck_driven) (greater_zero
      truck_remaining) (greater_zero driver_at_goal_truck)) (:effects (increases
      goal_truck_driven) (unchanged truck_remaining)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero goal_truck_driven) (greater_zero
      truck_remaining) (equal_zero driver_at_goal_truck)) (:effects (decreases
      driver_to_goal_truck_distance) (unchanged truck_remaining)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (greater_zero
      driver_ready_exit)) (:effects (decreases driver_remaining) (decreases
      driver_ready_exit)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (greater_zero
      driver_ready_exit)) (:effects (decreases driver_remaining) (decreases
      driver_ready_exit)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (equal_zero
      driver_ready_exit) (greater_zero driving)) (:effects (decreases
      driven_driver_goal_distance) (unchanged driving) (unchanged driver_remaining) (
      unchanged truck_remaining)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (equal_zero driving)
       (greater_zero driver_remaining) (greater_zero driver_at_free_truck)) (:effects (
      increases driving) (unchanged driver_remaining) (unchanged truck_remaining)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (equal_zero driving)
       (greater_zero driver_remaining) (equal_zero driver_at_free_truck)) (:effects (
      decreases driver_to_free_truck_distance) (unchanged driver_remaining) (unchanged
      truck_remaining)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (equal_zero
      driver_ready_exit) (greater_zero driving)) (:effects (decreases driving) (unchanged
      driver_remaining)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (equal_zero driving)
       (greater_zero driver_remaining) (greater_zero swapped_driver_goal_pair) (equal_zero
      driver_one_step)) (:effects (decreases swapped_driver_goal_pair) (unchanged
      driver_remaining) (unchanged driving)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (equal_zero driving)
       (greater_zero driver_remaining) (greater_zero driver_one_step)) (:effects (decreases
      driver_remaining) (decreases driver_one_step)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (equal_zero driving)
       (greater_zero driver_remaining) (equal_zero driver_one_step)) (:effects (decreases
      driver_unsat_goal_distance) (unchanged driver_remaining)))
   ((:conditions (equal_zero pkg_remaining) (equal_zero truck_remaining) (equal_zero driving)
       (greater_zero driver_remaining) (equal_zero driver_one_step)) (:effects (increases
      driver_one_step) (unchanged driver_remaining)))
  )
)
```

B.7 Ferry

```
(policy
  (:features
    (numerical remaining "r" "cars not at their goal location" (n_count (c_and (c_atomic_state
        "car") (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true))))))
    (numerical carried "c" "cars currently on the ferry" (n_count (c_and (c_atomic_state "car
       ") (c_atomic_state "on"))))
    (numerical ready_unload "ru" "carried cars whose goal is the ferry location" (n_count (
       c_and (c_atomic_state "car") (c_and (c_atomic_state "on") (c_some (r_atomic_goal "at"
       true) (c_atomic_state "at-ferry")))))))
```

```
   (numerical car_at_ferry "cf" "undelivered cars at the ferry location" (n_count (c_and (
       c_atomic_state "car") (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "
       at" true))) (c_some (r_atomic_state "at") (c_atomic_state "at-ferry"))))))
  )
  (:rules
   ((:conditions (greater_zero ready_unload)) (:effects (decreases remaining) (decreases
       carried) (decreases ready_unload)))
   ((:conditions (greater_zero carried) (equal_zero ready_unload)) (:effects (increases
       ready_unload) (unchanged carried) (unchanged remaining)))
   ((:conditions (equal_zero carried) (greater_zero car_at_ferry)) (:effects (increases
       carried) (unchanged remaining)))
   ((:conditions (equal_zero carried) (equal_zero car_at_ferry) (greater_zero remaining)) (:
       effects (increases car_at_ferry) (unchanged remaining)))
  )
)
```

B.8 Floortile

```
(policy
  (:features
   (numerical remaining "r" "goal tiles not yet painted with their goal color" (n_count (
       c_and (c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted" true) c_top)) (
       c_not (c_same_as (r_atomic_state "painted") (r_atomic_goal "painted" true))))))
   (numerical frontier "f" "clear unpainted goal tiles with no unpainted goal tile below" (
       n_count (c_and (c_and (c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted"
       true) c_top)) (c_and (c_atomic_state "clear") (c_not (c_same_as (r_atomic_state "
       painted") (r_atomic_goal "painted" true))))) (c_not (c_some (r_or (r_inverse (
       r_atomic_state "up")) (r_composition (r_transitive_closure (r_or (r_or (r_atomic_state
        "left") (r_inverse (r_atomic_state "left"))) (r_or (r_atomic_state "right") (
       r_inverse (r_atomic_state "right"))))) (r_inverse (r_atomic_state "up")))) (c_and (
       c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted" true) c_top)) (c_not (
       c_same_as (r_atomic_state "painted") (r_atomic_goal "painted" true)))))))))
   (numerical paintable "p" "frontier tiles vertically adjacent to a robot" (n_count (c_and (
       c_and (c_and (c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted" true)
       c_top)) (c_and (c_atomic_state "clear") (c_not (c_same_as (r_atomic_state "painted") (
       r_atomic_goal "painted" true))))) (c_not (c_some (r_or (r_inverse (r_atomic_state "up")
       ) (r_composition (r_transitive_closure (r_or (r_or (r_atomic_state "left") (r_inverse
       (r_atomic_state "left"))) (r_or (r_atomic_state "right") (r_inverse (r_atomic_state "
       right"))))) (r_inverse (r_atomic_state "up")))) (c_and (c_and (c_atomic_state "tile")
       (c_some (r_atomic_goal "painted" true) c_top)) (c_not (c_same_as (r_atomic_state "
       painted") (r_atomic_goal "painted" true)))))))) (c_some (r_atomic_state "up") (c_some (
       r_inverse (r_atomic_state "robot-at")) (c_atomic_state "robot"))))))
   (numerical ready_paint "rp" "paintable frontier tiles whose adjacent robot has the goal
       color" (n_count (c_and (c_and (c_and (c_and (c_atomic_state "tile") (c_some (
       r_atomic_goal "painted" true) c_top)) (c_and (c_atomic_state "clear") (c_not (
       c_same_as (r_atomic_state "painted") (r_atomic_goal "painted" true))))) (c_not (c_some
        (r_or (r_inverse (r_atomic_state "up")) (r_composition (r_transitive_closure (r_or (
       r_or (r_atomic_state "left") (r_inverse (r_atomic_state "left"))) (r_or (
       r_atomic_state "right") (r_inverse (r_atomic_state "right"))))) (r_inverse (
       r_atomic_state "up"))))) (c_and (c_and (c_atomic_state "tile") (c_some (r_atomic_goal "
       painted" true) c_top)) (c_not (c_same_as (r_atomic_state "painted") (r_atomic_goal "
       painted" true)))))))) (c_same_as (r_atomic_goal "painted" true) (r_composition (
       r_atomic_state "up") (r_composition (r_inverse (r_atomic_state "robot-at")) (
       r_atomic_state "robot-has")))))))
   (numerical dist_to_frontier_support "dfs" "clear-grid distance from a robot to a clear
       vertical support tile for the frontier" (n_distance (c_some (r_inverse (r_atomic_state
        "robot-at")) (c_atomic_state "robot")) (r_restriction (r_or (r_or (r_or (
       r_atomic_state "up") (r_inverse (r_atomic_state "up"))) (r_or (r_inverse (
       r_atomic_state "up")) (r_inverse (r_inverse (r_atomic_state "up"))))) (r_or (r_or (
       r_atomic_state "left") (r_inverse (r_atomic_state "left"))) (r_or (r_atomic_state "
       right") (r_inverse (r_atomic_state "right"))))) (c_atomic_state "clear")) (c_and (c_or
        (c_atomic_state "clear") (c_some (r_inverse (r_atomic_state "robot-at")) (
       c_atomic_state "robot"))) (c_some (r_inverse (r_atomic_state "up")) (c_and (c_and (
       c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted" true) c_top)) (c_and (
```

```
    c_atomic_state "clear") (c_not (c_same_as (r_atomic_state "painted") (r_atomic_goal "
    painted" true))))) (c_not (c_some (r_or (r_inverse (r_atomic_state "up")) (
    r_composition (r_transitive_closure (r_or (r_or (r_atomic_state "left") (r_inverse (
    r_atomic_state "left"))) (r_or (r_atomic_state "right") (r_inverse (r_atomic_state "
    right")))))) (r_inverse (r_atomic_state "up")))) (c_and (c_and (c_atomic_state "tile")
    (c_some (r_atomic_goal "painted" true) c_top)) (c_not (c_same_as (r_atomic_state "
    painted") (r_atomic_goal "painted" true))))))))))
  (numerical four_or_more_robots "fr" "objects seeing at least four robots" (n_count (
    c_at_least 4 (r_universal) (c_atomic_state "robot"))))
  (numerical edge_blocked_frontier_goal "ebfg" "occupied active goals at a horizontal row
    edge" (n_count (c_and (c_and (c_and (c_and (c_and (c_atomic_state "tile") (c_some (
    r_atomic_goal "painted" true) c_top)) (c_not (c_same_as (r_atomic_state "painted") (
    r_atomic_goal "painted" true)))) (c_some (r_inverse (r_atomic_state "robot-at")) (
    c_atomic_state "robot"))) (c_not (c_some (r_or (r_inverse (r_atomic_state "up")) (
    r_composition (r_transitive_closure (r_or (r_or (r_atomic_state "left") (r_inverse (
    r_atomic_state "left"))) (r_or (r_atomic_state "right") (r_inverse (r_atomic_state "
    right")))))) (r_inverse (r_atomic_state "up")))) (c_and (c_and (c_atomic_state "tile")
    (c_some (r_atomic_goal "painted" true) c_top)) (c_not (c_same_as (r_atomic_state "
    painted") (r_atomic_goal "painted" true)))))))) (c_or (c_not (c_some (r_atomic_state "
    left") (c_atomic_state "tile"))) (c_not (c_some (r_atomic_state "right") (
    c_atomic_state "tile")))))))
  (numerical crowded_blocked_frontier_goal "cbfg" "occupied active goals with another
    occupied active goal in the same row" (n_count (c_and (c_and (c_and (c_and (c_and (
    c_atomic_state "tile") (c_some (r_atomic_goal "painted" true) c_top)) (c_not (
    c_same_as (r_atomic_state "painted") (r_atomic_goal "painted" true)))) (c_some (
    r_inverse (r_atomic_state "robot-at")) (c_atomic_state "robot"))) (c_not (c_some (r_or
     (r_inverse (r_atomic_state "up")) (r_composition (r_transitive_closure (r_or (r_or (
    r_atomic_state "left") (r_inverse (r_atomic_state "left"))) (r_or (r_atomic_state "
    right") (r_inverse (r_atomic_state "right")))))) (r_inverse (r_atomic_state "up")))) (
    c_and (c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted" true) c_top)) (
    c_not (c_same_as (r_atomic_state "painted") (r_atomic_goal "painted" true)))))))) (
    c_some (r_transitive_closure (r_or (r_or (r_atomic_state "left") (r_inverse (
    r_atomic_state "left"))) (r_or (r_atomic_state "right") (r_inverse (r_atomic_state "
    right"))))) (c_and (c_and (c_and (c_and (c_atomic_state "tile") (c_some (r_atomic_goal
     "painted" true) c_top)) (c_not (c_same_as (r_atomic_state "painted") (r_atomic_goal "
    painted" true)))) (c_some (r_inverse (r_atomic_state "robot-at")) (c_atomic_state "
    robot"))) (c_not (c_some (r_or (r_inverse (r_atomic_state "up")) (r_composition (
    r_transitive_closure (r_or (r_or (r_atomic_state "left") (r_inverse (r_atomic_state "
    left"))) (r_or (r_atomic_state "right") (r_inverse (r_atomic_state "right"))))) (
    r_inverse (r_atomic_state "up")))) (c_and (c_and (c_atomic_state "tile") (c_some (
    r_atomic_goal "painted" true) c_top)) (c_not (c_same_as (r_atomic_state "painted") (
    r_atomic_goal "painted" true)))))))))))
  (numerical clear_tiles "ct" "currently clear tiles" (n_count (c_atomic_state "clear")))
  (numerical blocked_goal "bg" "missing goal tiles currently occupied by a robot" (n_count (
    c_and (c_and (c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted" true)
    c_top)) (c_not (c_same_as (r_atomic_state "painted") (r_atomic_goal "painted" true))))
     (c_some (r_inverse (r_atomic_state "robot-at")) (c_atomic_state "robot")))))
  (numerical blocked_frontier_goal "bfg" "missing goal tiles in the active deepest row
    currently occupied by a robot" (n_count (c_and (c_and (c_and (c_and (c_atomic_state "
    tile") (c_some (r_atomic_goal "painted" true) c_top)) (c_not (c_same_as (
    r_atomic_state "painted") (r_atomic_goal "painted" true)))) (c_some (r_inverse (
    r_atomic_state "robot-at")) (c_atomic_state "robot"))) (c_not (c_some (r_or (r_inverse
     (r_atomic_state "up")) (r_composition (r_transitive_closure (r_or (r_or (
    r_atomic_state "left") (r_inverse (r_atomic_state "left"))) (r_or (r_atomic_state "
    right") (r_inverse (r_atomic_state "right")))))) (r_inverse (r_atomic_state "up")))) (
    c_and (c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted" true) c_top)) (
    c_not (c_same_as (r_atomic_state "painted") (r_atomic_goal "painted" true)))))))))
  (numerical blocked_support "bs" "support tiles for occupied missing goal tiles that are
    occupied by a robot" (n_count (c_and (c_some (r_inverse (r_atomic_state "robot-at")) (
    c_atomic_state "robot")) (c_some (r_inverse (r_atomic_state "up")) (c_and (c_and (
    c_and (c_atomic_state "tile") (c_some (r_atomic_goal "painted" true) c_top)) (c_not (
    c_same_as (r_atomic_state "painted") (r_atomic_goal "painted" true)))) (c_some (
    r_inverse (r_atomic_state "robot-at")) (c_atomic_state "robot")))))))
)
(:rules
```

```
   ((:conditions (greater_zero blocked_frontier_goal)) (:effects (decreases
      blocked_frontier_goal) (unchanged frontier) (unchanged remaining) (unchanged
      clear_tiles)))
   ((:conditions (greater_zero blocked_frontier_goal)) (:effects (decreases
      blocked_frontier_goal) (increases frontier) (unchanged remaining) (unchanged
      clear_tiles)))
   ((:conditions (greater_zero blocked_support)) (:effects (decreases blocked_support) (
      unchanged remaining) (unchanged clear_tiles) (unchanged blocked_frontier_goal)))
   ((:conditions (equal_zero frontier) (equal_zero blocked_support) (greater_zero
      blocked_goal)) (:effects (decreases blocked_goal) (unchanged remaining) (unchanged
      clear_tiles)))
   ((:conditions (greater_zero ready_paint)) (:effects (decreases remaining) (decreases
      frontier) (decreases clear_tiles)))
   ((:conditions (greater_zero ready_paint)) (:effects (decreases remaining) (increases
      frontier) (decreases clear_tiles)))
   ((:conditions (greater_zero ready_paint)) (:effects (decreases remaining) (unchanged
      frontier) (decreases paintable) (decreases clear_tiles)))
   ((:conditions (equal_zero ready_paint) (greater_zero paintable)) (:effects (increases
      ready_paint) (unchanged remaining) (unchanged clear_tiles) (unchanged blocked_support))
      )
   ((:conditions (greater_zero blocked_frontier_goal) (greater_zero
      edge_blocked_frontier_goal) (equal_zero four_or_more_robots) (equal_zero
      blocked_support) (equal_zero ready_paint) (greater_zero remaining)) (:effects (
      decreases dist_to_frontier_support) (unchanged remaining) (unchanged clear_tiles) (
      unchanged blocked_support)))
   ((:conditions (greater_zero blocked_frontier_goal) (greater_zero
      crowded_blocked_frontier_goal) (greater_zero blocked_support) (equal_zero ready_paint)
       (greater_zero remaining)) (:effects (decreases dist_to_frontier_support) (unchanged
      remaining) (unchanged clear_tiles) (unchanged blocked_support)))
   ((:conditions (equal_zero ready_paint) (equal_zero blocked_frontier_goal) (greater_zero
      remaining)) (:effects (decreases dist_to_frontier_support) (unchanged remaining) (
      unchanged clear_tiles) (unchanged blocked_support)))
  )
)
```

B.9 Goldminer

```
(policy
  (:features
    (boolean arm_empty "ae" "robot arm is empty" (b_atomic_state "arm-empty" true))
    (boolean holds_bomb "hb" "robot holds a bomb" (b_atomic_state "holds-bomb" true))
    (numerical soft "s" "soft rocks remaining" (n_count (c_atomic_state "soft-rock-at")))
    (numerical gold_soft "gs" "gold cell still has soft rock" (n_count (c_and (c_atomic_state
       "gold-at") (c_atomic_state "soft-rock-at"))))
    (numerical gold_clear "gc" "gold cell is clear" (n_count (c_and (c_atomic_state "gold-at")
       (c_atomic_state "clear"))))
    (numerical clear "c" "clear cells" (n_count (c_atomic_state "clear")))
    (numerical dist_bomb "db" "clear-path distance from robot to bomb supply" (n_distance (
       c_atomic_state "robot-at") (r_restriction (r_atomic_state "connected") (c_atomic_state
       "clear")) (c_atomic_state "bomb-at")))
    (numerical dist_frontier "df" "clear-path distance from robot to a clear cell adjacent to
       soft rock" (n_distance (c_atomic_state "robot-at") (r_restriction (r_atomic_state "
       connected") (c_atomic_state "clear")) (c_and (c_atomic_state "clear") (c_some (
       r_atomic_state "connected") (c_atomic_state "soft-rock-at")))))
    (numerical dist_gold "dg" "clear-path distance from robot to gold" (n_distance (
       c_atomic_state "robot-at") (r_restriction (r_atomic_state "connected") (c_atomic_state
       "clear")) (c_atomic_state "gold-at")))
  )
  (:rules
    ((:conditions (positive arm_empty) (greater_zero gold_clear)) (:effects (negative
       arm_empty)))
    ((:conditions (positive arm_empty) (greater_zero gold_clear) (greater_zero dist_gold)) (:
       effects (decreases dist_gold) (unchanged soft) (unchanged gold_soft) (unchanged
       gold_clear) (unchanged clear)))
```

```
   ((:conditions (positive arm_empty) (equal_zero gold_clear) (greater_zero soft) (equal_zero
        dist_bomb)) (:effects (positive holds_bomb) (negative arm_empty) (unchanged gold_soft)
        (unchanged gold_clear) (unchanged clear)))
   ((:conditions (positive arm_empty) (equal_zero gold_clear) (greater_zero soft) (
       greater_zero dist_bomb)) (:effects (decreases dist_bomb) (unchanged soft) (unchanged
       gold_soft) (unchanged gold_clear) (unchanged clear)))
   ((:conditions (positive holds_bomb) (equal_zero gold_clear) (equal_zero dist_frontier)) (:
       effects (negative holds_bomb) (positive arm_empty) (decreases soft) (increases clear)
       (unchanged gold_soft) (unchanged gold_clear)))
   ((:conditions (positive holds_bomb) (equal_zero gold_clear) (equal_zero dist_frontier) (
       greater_zero gold_soft)) (:effects (negative holds_bomb) (positive arm_empty) (
       decreases soft) (increases clear) (decreases gold_soft) (increases gold_clear)))
   ((:conditions (positive holds_bomb) (equal_zero gold_clear) (greater_zero dist_frontier))
       (:effects (decreases dist_frontier) (unchanged soft) (unchanged gold_soft) (unchanged
       gold_clear) (unchanged clear)))
  )
)
```

B.10 Grid

```
(policy
  (:features
    (numerical remaining "r" "goal keys not at their goal place" (n_count (c_and (
       c_atomic_state "key") (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)
       )))))
    (numerical held "h" "keys currently held" (n_count (c_and (c_atomic_state "key") (
       c_atomic_state "holding"))))
    (numerical locked "l" "locked places" (n_count (c_and (c_atomic_state "place") (
       c_atomic_state "locked"))))
    (numerical ready_put "rp" "held goal keys whose goal is the robot place" (n_count (c_and (
       c_and (c_atomic_state "key") (c_atomic_state "holding")) (c_some (r_atomic_goal "at"
       true) (c_atomic_state "at-robot")))))
    (numerical unlockable "u" "adjacent locked places matching the held key shape" (n_count (
       c_and (c_and (c_atomic_state "place") (c_atomic_state "locked")) (c_and (c_some (
       r_inverse (r_atomic_state "conn")) (c_atomic_state "at-robot")) (c_some (
       r_atomic_state "lock-shape") (c_some (r_inverse (r_atomic_state "key-shape")) (c_and (
       c_atomic_state "key") (c_atomic_state "holding"))))))))
    (numerical held_matching_locks "ml" "locked places matching the held key shape" (n_count (
       c_and (c_and (c_atomic_state "place") (c_atomic_state "locked")) (c_some (
       r_atomic_state "lock-shape") (c_some (r_inverse (r_atomic_state "key-shape")) (c_and (
       c_atomic_state "key") (c_atomic_state "holding")))))))
    (numerical dist_to_lock_frontier "dlf" "open-grid distance to a place adjacent to a
       matching held-key lock" (n_distance (c_atomic_state "at-robot") (r_restriction (
       r_atomic_state "conn") (c_atomic_state "open")) (c_and (c_atomic_state "open") (c_some
        (r_atomic_state "conn") (c_and (c_and (c_atomic_state "place") (c_atomic_state "
       locked")) (c_some (r_atomic_state "lock-shape") (c_some (r_inverse (r_atomic_state "key
       -shape")) (c_and (c_atomic_state "key") (c_atomic_state "holding"))))))))))
    (numerical held_open_goal "hog" "held goal keys whose goal place is currently open" (
       n_count (c_and (c_and (c_atomic_state "key") (c_atomic_state "holding")) (c_some (
       r_atomic_goal "at" true) (c_atomic_state "open")))))
    (numerical dist_to_held_goal "dhg" "open-grid distance to the held key goal place" (
       n_distance (c_atomic_state "at-robot") (r_restriction (r_atomic_state "conn") (
       c_atomic_state "open")) (c_some (r_inverse (r_atomic_goal "at" true)) (c_and (
       c_atomic_state "key") (c_atomic_state "holding")))))
    (numerical lock_key_here "lkh" "keys at robot whose shape opens some locked place" (
       n_count (c_and (c_and (c_atomic_state "key") (c_some (r_atomic_state "at") (
       c_atomic_state "at-robot"))) (c_some (r_atomic_state "key-shape") (c_some (r_inverse (
       r_atomic_state "lock-shape")) (c_and (c_atomic_state "place") (c_atomic_state "locked")
       ))))))
    (numerical open_goal_key_here "ogh" "undelivered keys at robot with an open goal place" (
       n_count (c_and (c_and (c_atomic_state "key") (c_some (r_atomic_state "at") (
       c_atomic_state "at-robot"))) (c_and (c_not (c_same_as (r_atomic_state "at") (
       r_atomic_goal "at" true))) (c_some (r_atomic_goal "at" true) (c_atomic_state "open"))))
       ))
```

```
   (numerical dist_to_useful_key "duk" "open-grid distance to an open place containing a lock
        key or open-goal key" (n_distance (c_atomic_state "at-robot") (r_restriction (
       r_atomic_state "conn") (c_atomic_state "open")) (c_and (c_atomic_state "open") (c_some
        (r_inverse (r_atomic_state "at")) (c_and (c_atomic_state "key") (c_or (c_some (
       r_atomic_state "key-shape") (c_some (r_inverse (r_atomic_state "lock-shape")) (c_and (
       c_atomic_state "place") (c_atomic_state "locked")))) (c_and (c_not (c_same_as (
       r_atomic_state "at") (r_atomic_goal "at" true))) (c_some (r_atomic_goal "at" true) (
       c_atomic_state "open")))))))))
  )
  (:rules
   ((:conditions (greater_zero unlockable)) (:effects (decreases locked) (decreases
       unlockable) (unchanged held) (unchanged remaining)))
   ((:conditions (equal_zero unlockable) (equal_zero held_matching_locks) (greater_zero
       ready_put)) (:effects (decreases remaining) (decreases held) (decreases ready_put)))
   ((:conditions (greater_zero held) (equal_zero unlockable) (greater_zero
       held_matching_locks) (greater_zero dist_to_lock_frontier)) (:effects (decreases
       dist_to_lock_frontier) (unchanged held) (unchanged remaining) (unchanged locked)))
   ((:conditions (greater_zero held) (equal_zero unlockable) (equal_zero held_matching_locks)
        (greater_zero held_open_goal) (greater_zero dist_to_held_goal)) (:effects (decreases
       dist_to_held_goal) (unchanged held) (unchanged remaining)))
   ((:conditions (greater_zero held) (equal_zero unlockable) (equal_zero held_matching_locks)
        (equal_zero held_open_goal) (greater_zero dist_to_useful_key)) (:effects (decreases
       dist_to_useful_key) (unchanged held) (unchanged remaining)))
   ((:conditions (greater_zero held) (equal_zero unlockable) (equal_zero held_matching_locks)
        (equal_zero held_open_goal) (greater_zero locked)) (:effects (unchanged held) (
       increases held_matching_locks) (increases remaining)))
   ((:conditions (greater_zero held) (equal_zero unlockable) (equal_zero held_matching_locks)
        (equal_zero held_open_goal)) (:effects (unchanged held) (increases
       held_matching_locks)))
   ((:conditions (greater_zero held) (equal_zero unlockable) (equal_zero held_matching_locks)
        (equal_zero held_open_goal)) (:effects (unchanged held) (increases held_open_goal)))
   ((:conditions (equal_zero held) (greater_zero lock_key_here) (greater_zero locked)) (:
       effects (increases held) (decreases lock_key_here) (increases remaining)))
   ((:conditions (equal_zero held) (greater_zero lock_key_here)) (:effects (increases held) (
       decreases lock_key_here) (decreases remaining)))
   ((:conditions (equal_zero held) (greater_zero lock_key_here)) (:effects (increases held) (
       decreases lock_key_here) (unchanged remaining)))
   ((:conditions (equal_zero held) (equal_zero lock_key_here) (greater_zero
       open_goal_key_here)) (:effects (increases held) (decreases open_goal_key_here) (
       decreases remaining)))
   ((:conditions (equal_zero held) (equal_zero lock_key_here) (greater_zero
       open_goal_key_here)) (:effects (increases held) (decreases open_goal_key_here) (
       unchanged remaining)))
   ((:conditions (equal_zero held) (equal_zero lock_key_here) (equal_zero open_goal_key_here)
        (greater_zero dist_to_useful_key)) (:effects (decreases dist_to_useful_key) (
       unchanged held) (unchanged remaining)))
  )
)
```

B.11 Gripper

```
(policy
  (:features
   (numerical remaining "r" "balls not at their goal room" (n_count (c_and (c_atomic_state "
       ball") (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true))))))
   (numerical carried "c" "balls currently carried by a gripper" (n_count (c_and (
       c_atomic_state "ball") (c_some (r_atomic_state "carry") (c_atomic_state "gripper")))))
   (numerical free_grippers "f" "free grippers" (n_count (c_and (c_atomic_state "gripper") (
       c_atomic_state "free"))))
   (numerical ready_drop "rd" "carried balls whose goal is the robot room" (n_count (c_and (
       c_atomic_state "ball") (c_and (c_some (r_atomic_state "carry") (c_atomic_state "gripper
       ")) (c_some (r_atomic_goal "at" true) (c_atomic_state "at-robby"))))))
   (numerical ball_at_robot "br" "undelivered balls at the robot room" (n_count (c_and (
       c_atomic_state "ball") (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "
       at" true))) (c_some (r_atomic_state "at") (c_atomic_state "at-robby"))))))
```

```
  )
  (:rules
    ((:conditions (greater_zero ready_drop)) (:effects (decreases remaining) (decreases
        carried) (increases free_grippers) (decreases ready_drop)))
    ((:conditions (greater_zero carried) (equal_zero ready_drop)) (:effects (increases
        ready_drop) (unchanged carried) (unchanged remaining)))
    ((:conditions (equal_zero ready_drop) (greater_zero ball_at_robot) (greater_zero
        free_grippers)) (:effects (increases carried) (decreases free_grippers) (unchanged
        remaining)))
    ((:conditions (equal_zero ready_drop) (equal_zero carried) (greater_zero remaining) (
        equal_zero ball_at_robot)) (:effects (increases ball_at_robot) (unchanged remaining)))
  )
)
```

B.12 Hiking-Bin.

```
(policy
  (:features
    (numerical r "r" "remaining next-chain distance over unfinished couples" (n_count (
        r_composition (r_atomic_state "walked") (r_transitive_closure (r_atomic_state "next")))
        ))
    (numerical a "a" "partners away from their couple source" (n_count (c_and (c_atomic_state
        "person") (c_and (c_some (r_inverse (r_atomic_state "partner_of")) (c_and (
        c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
        walked" true))))) (c_not (c_same_as (r_atomic_state "at_person") (r_composition (
        r_inverse (r_atomic_state "partner_of")) (r_atomic_state "walked")))))))
    (numerical an "an" "away partners at next place" (n_count (c_and (c_and (c_atomic_state "
        person") (c_and (c_some (r_inverse (r_atomic_state "partner_of")) (c_and (
        c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
        walked" true))))) (c_not (c_same_as (r_atomic_state "at_person") (r_composition (
        r_inverse (r_atomic_state "partner_of")) (r_atomic_state "walked")))))) (c_same_as (
        r_atomic_state "at_person") (r_composition (r_inverse (r_atomic_state "partner_of")) (
        r_composition (r_atomic_state "walked") (r_atomic_state "next")))))))
    (numerical ap "ap" "away partners at previous place" (n_count (c_and (c_and (
        c_atomic_state "person") (c_and (c_some (r_inverse (r_atomic_state "partner_of")) (
        c_and (c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (
        r_atomic_goal "walked" true))))) (c_not (c_same_as (r_atomic_state "at_person") (
        r_composition (r_inverse (r_atomic_state "partner_of")) (r_atomic_state "walked"))))))
         (c_same_as (r_composition (r_atomic_state "at_person") (r_atomic_state "next")) (
        r_composition (r_inverse (r_atomic_state "partner_of")) (r_atomic_state "walked"))))))
    (boolean ab "ab" "some away partner remains" (b_nonempty (c_and (c_atomic_state "person")
        (c_and (c_some (r_inverse (r_atomic_state "partner_of")) (c_and (c_atomic_state "
        couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true)))))
         (c_not (c_same_as (r_atomic_state "at_person") (r_composition (r_inverse (
        r_atomic_state "partner_of")) (r_atomic_state "walked")))))))))
    (boolean s "s" "some source partner present" (b_nonempty (c_and (c_and (c_atomic_state "
        couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true))))
        (c_some (r_atomic_state "partner_of") (c_and (c_atomic_state "person") (c_and (c_some
        (r_inverse (r_atomic_state "partner_of")) (c_and (c_atomic_state "couple") (c_not (
        c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true))))) (c_same_as (
        r_atomic_state "at_person") (r_composition (r_inverse (r_atomic_state "partner_of")) (
        r_atomic_state "walked")))))))))
    (boolean st "st" "both source partners present" (b_nonempty (c_and (c_and (c_atomic_state
        "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true))))
         (c_all (r_atomic_state "partner_of") (c_and (c_atomic_state "person") (c_and (c_some
        (r_inverse (r_atomic_state "partner_of")) (c_and (c_atomic_state "couple") (c_not (
        c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true))))) (c_same_as (
        r_atomic_state "at_person") (r_composition (r_inverse (r_atomic_state "partner_of")) (
        r_atomic_state "walked")))))))))
    (boolean at "at" "both partners at next" (b_nonempty (c_and (c_and (c_atomic_state "couple
        ") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true)))) (c_all
         (r_atomic_state "partner_of") (c_and (c_and (c_atomic_state "person") (c_and (c_some
        (r_inverse (r_atomic_state "partner_of")) (c_and (c_atomic_state "couple") (c_not (
        c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true))))) (c_not (
```

```
    c_same_as (r_atomic_state "at_person") (r_composition (r_inverse (r_atomic_state "
    partner_of")) (r_atomic_state "walked")))))) (c_same_as (r_atomic_state "at_person") (
    r_composition (r_inverse (r_atomic_state "partner_of")) (r_composition (r_atomic_state
     "walked") (r_atomic_state "next"))))))))))
(boolean apt "apt" "two away partners at one place" (b_nonempty (c_and (c_atomic_state "
    place") (c_at_least 2 (r_inverse (r_atomic_state "at_person")) (c_and (c_atomic_state "
    person") (c_and (c_some (r_inverse (r_atomic_state "partner_of")) (c_and (
    c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
    walked" true))))) (c_not (c_same_as (r_atomic_state "at_person") (r_composition (
    r_inverse (r_atomic_state "partner_of")) (r_atomic_state "walked")))))))))))
(numerical fl "fl" "final-leg couples" (n_count (c_and (c_and (c_atomic_state "couple") (
    c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true)))) (c_same_as
     (r_composition (r_atomic_state "walked") (r_atomic_state "next")) (r_atomic_goal "
    walked" true)))))
(numerical ch "ch" "cars at current walked places" (n_count (c_and (c_atomic_state "car")
    (c_some (r_atomic_state "at_car") (c_some (r_inverse (r_atomic_state "walked")) (c_and
     (c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
    walked" true)))))))))
(boolean ch2 "ch2" "two cars at current walked place" (b_nonempty (c_and (c_some (
    r_inverse (r_atomic_state "walked")) (c_and (c_atomic_state "couple") (c_not (
    c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true))))) (c_at_least 2 (
    r_inverse (r_atomic_state "at_car")) (c_atomic_state "car"))))))
(numerical cn "cn" "cars at next walked places" (n_count (c_and (c_atomic_state "car") (
    c_some (r_atomic_state "at_car") (c_some (r_inverse (r_atomic_state "next")) (c_some (
    r_inverse (r_atomic_state "walked")) (c_and (c_atomic_state "couple") (c_not (
    c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true)))))))))))
(boolean cn2 "cn2" "two cars at next walked place" (b_nonempty (c_and (c_some (r_inverse (
    r_atomic_state "next")) (c_some (r_inverse (r_atomic_state "walked")) (c_and (
    c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
    walked" true)))))) (c_at_least 2 (r_inverse (r_atomic_state "at_car")) (c_atomic_state
     "car")))))
(numerical spc "spc" "complete source couples count" (n_count (c_and (c_and (
    c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
    walked" true)))) (c_all (r_atomic_state "partner_of") (c_and (c_atomic_state "person")
     (c_and (c_some (r_inverse (r_atomic_state "partner_of")) (c_and (c_atomic_state "
    couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true)))))
     (c_same_as (r_atomic_state "at_person") (r_composition (r_inverse (r_atomic_state "
    partner_of")) (r_atomic_state "walked"))))))))))
(boolean sct "sct" "some complete source couple present" (b_nonempty (c_and (c_and (
    c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
    walked" true)))) (c_all (r_atomic_state "partner_of") (c_and (c_atomic_state "person")
     (c_and (c_some (r_inverse (r_atomic_state "partner_of")) (c_and (c_atomic_state "
    couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true)))))
     (c_same_as (r_atomic_state "at_person") (r_composition (r_inverse (r_atomic_state "
    partner_of")) (r_atomic_state "walked"))))))))))
(numerical au "au" "up tents at active current places" (n_count (c_and (c_atomic_state "
    tent") (c_and (c_atomic_state "up") (c_some (r_atomic_state "at_tent") (c_some (
    r_inverse (r_atomic_state "walked")) (c_and (c_atomic_state "couple") (c_not (
    c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true)))))))))))
(numerical ad "ad" "down tents at active current places" (n_count (c_and (c_atomic_state "
    tent") (c_and (c_atomic_state "down") (c_some (r_atomic_state "at_tent") (c_some (
    r_inverse (r_atomic_state "walked")) (c_and (c_atomic_state "couple") (c_not (
    c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true)))))))))))
(numerical ndp "ndp" "down tents at next places with an away person" (n_count (c_and (
    c_atomic_state "tent") (c_and (c_atomic_state "down") (c_some (r_atomic_state "at_tent
    ") (c_and (c_some (r_inverse (r_atomic_state "next")) (c_some (r_inverse (
    r_atomic_state "walked")) (c_and (c_atomic_state "couple") (c_not (c_same_as (
    r_atomic_state "walked") (r_atomic_goal "walked" true)))))) (c_some (r_inverse (
    r_atomic_state "at_person")) (c_and (c_atomic_state "person") (c_and (c_some (
    r_inverse (r_atomic_state "partner_of")) (c_and (c_atomic_state "couple") (c_not (
    c_same_as (r_atomic_state "walked") (r_atomic_goal "walked" true))))) (c_not (
    c_same_as (r_atomic_state "at_person") (r_composition (r_inverse (r_atomic_state "
    partner_of")) (r_atomic_state "walked"))))))))))))))
(numerical nu "nu" "up tents at next places" (n_count (c_and (c_atomic_state "tent") (
    c_and (c_atomic_state "up") (c_some (r_atomic_state "at_tent") (c_some (r_inverse (
```

```
    r_atomic_state "next")) (c_some (r_inverse (r_atomic_state "walked")) (c_and (
    c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
    walked" true))))))))))))
  (numerical nd "nd" "down tents at next places" (n_count (c_and (c_atomic_state "tent") (
    c_and (c_atomic_state "down") (c_some (r_atomic_state "at_tent") (c_some (r_inverse (
    r_atomic_state "next")) (c_some (r_inverse (r_atomic_state "walked")) (c_and (
    c_atomic_state "couple") (c_not (c_same_as (r_atomic_state "walked") (r_atomic_goal "
    walked" true))))))))))))
)
(:rules
  ((:conditions (greater_zero r) (greater_zero fl) (equal_zero a) (positive st) (
    greater_zero nu)) (:effects (decreases r)))
  ((:conditions (greater_zero r) (equal_zero fl) (equal_zero a) (positive st) (greater_zero
    nu) (greater_zero cn)) (:effects (decreases r)))
  ((:conditions (greater_zero r) (greater_zero ch) (greater_zero a) (greater_zero nu) (
    positive s) (negative st) (equal_zero fl) (negative at)) (:effects (increases a) (
    increases an) (positive at) (decreases ch) (increases cn) (negative s) (unchanged spc)
     (unchanged nu)))
  ((:conditions (greater_zero r) (greater_zero ch) (greater_zero a) (greater_zero nu) (
    positive s) (positive st) (equal_zero fl) (negative at)) (:effects (increases a) (
    increases an) (positive at) (decreases ch) (increases cn) (positive st) (unchanged spc)
     (unchanged nu)))
  ((:conditions (greater_zero r) (equal_zero ch) (greater_zero a) (greater_zero nu) (
    positive s) (equal_zero fl) (positive cn2) (negative at)) (:effects (decreases a) (
    decreases an) (positive s) (positive st) (decreases cn) (unchanged nu)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero nu) (positive s) (
    greater_zero fl) (positive apt) (equal_zero cn)) (:effects (decreases a) (decreases an)
     (positive s) (positive sct) (increases spc) (negative at) (unchanged nu)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero nu) (positive s) (
    greater_zero fl) (positive apt) (greater_zero cn)) (:effects (decreases a) (decreases
    an) (decreases cn) (increases ch) (positive s) (positive sct) (increases spc) (
    negative at) (unchanged nu)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero nu) (positive s) (
    greater_zero fl) (negative apt) (equal_zero cn)) (:effects (decreases a) (decreases an)
     (negative at) (positive s) (positive st) (unchanged nu)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero nu) (positive s) (
    greater_zero fl) (negative apt) (greater_zero cn)) (:effects (decreases a) (decreases
    an) (decreases cn) (increases ch) (negative at) (positive s) (positive st) (unchanged
    nu)))
  ((:conditions (greater_zero r) (equal_zero fl) (greater_zero a) (greater_zero nu) (
    positive s) (positive st) (positive at) (positive cn2)) (:effects (decreases a) (
    decreases an) (positive s) (positive st) (positive sct) (increases spc) (negative at)
    (decreases cn) (increases ch) (unchanged nu)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero nu) (negative s) (positive
    at)) (:effects (decreases a) (decreases an) (negative at) (positive s) (positive st) (
    unchanged nu)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero nd) (equal_zero nu) (
    equal_zero fl) (greater_zero ch) (equal_zero cn)) (:effects (decreases nd) (increases
    nu) (unchanged a) (unchanged an) (unchanged ap) (unchanged ab) (unchanged s) (
    unchanged st) (unchanged sct) (unchanged spc) (unchanged ch)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero ndp) (equal_zero nu) (
    equal_zero fl) (greater_zero cn)) (:effects (decreases ndp) (decreases nd) (increases
    nu) (unchanged a) (unchanged an) (unchanged ap) (unchanged ab) (unchanged s) (
    unchanged st) (unchanged sct) (unchanged spc) (unchanged cn)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero nd) (equal_zero nu) (
    greater_zero fl) (equal_zero cn)) (:effects (decreases nd) (increases nu) (unchanged a)
     (unchanged an) (unchanged ap) (unchanged ab) (unchanged s) (unchanged st) (unchanged
    sct) (unchanged spc)))
  ((:conditions (greater_zero r) (greater_zero a) (greater_zero ndp) (equal_zero nu) (
    greater_zero fl) (greater_zero cn)) (:effects (decreases ndp) (decreases nd) (
    increases nu) (unchanged a) (unchanged an) (unchanged ap) (unchanged ab) (unchanged s)
     (unchanged st) (unchanged sct) (unchanged spc) (unchanged cn)))
  ((:conditions (greater_zero r) (greater_zero ch) (positive ch2) (equal_zero fl) (
    equal_zero a) (equal_zero nd) (equal_zero nu) (greater_zero ad) (positive s)) (:
    effects (decreases ad) (increases nd) (increases a) (increases an) (decreases ch) (
```

```
      increases cn) (negative apt) (negative at) (unchanged s)))
   ((:conditions (greater_zero r) (greater_zero ch) (greater_zero fl) (equal_zero a) (
      equal_zero nd) (equal_zero nu) (greater_zero ad) (positive s)) (:effects (decreases ad)
       (increases nd) (increases a) (increases an) (increases cn) (negative apt) (negative
      at) (unchanged s)))
   ((:conditions (greater_zero r) (equal_zero fl) (equal_zero a) (equal_zero nd) (equal_zero
      nu) (greater_zero au) (greater_zero ch) (negative ch2) (positive st) (negative sct)) (:
      effects (increases a) (increases ap) (decreases ch) (negative s) (negative st) (
      unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (equal_zero a) (equal_zero nd) (equal_zero
      nu) (greater_zero au) (greater_zero ch) (negative ch2) (positive st) (positive sct)) (:
      effects (increases a) (increases ap) (decreases ch) (negative s) (negative st) (
      positive apt) (decreases spc) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (equal_zero a) (equal_zero nd) (equal_zero
      nu) (greater_zero au) (greater_zero ch) (negative ch2) (positive st) (positive sct)) (:
      effects (increases a) (increases ap) (decreases ch) (positive s) (positive st) (
      positive apt) (decreases spc) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (equal_zero ch) (negative s)) (:effects (decreases a) (decreases ap)
      (increases ch) (positive s) (negative st) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (equal_zero ch) (positive s) (positive st) (positive sct) (positive
      apt)) (:effects (decreases a) (decreases ap) (increases ch) (positive s) (positive st)
       (positive sct) (negative apt) (positive ab) (unchanged spc) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (equal_zero ch) (positive s) (positive st) (positive sct) (positive
      apt)) (:effects (decreases a) (decreases ap) (increases ch) (positive s) (positive st)
       (positive sct) (negative apt) (positive ab) (increases spc) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (equal_zero ch) (positive s) (positive st) (positive sct) (negative
      apt)) (:effects (decreases a) (decreases ap) (increases ch) (positive s) (positive st)
       (positive sct) (increases spc) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (greater_zero ch) (positive s) (positive st) (positive sct) (negative
       ch2)) (:effects (decreases a) (decreases ap) (increases ch) (positive s) (positive st)
       (positive sct) (positive ch2) (increases spc) (negative apt) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (equal_zero ch) (positive s) (positive st) (negative sct) (positive
      apt)) (:effects (decreases a) (decreases ap) (increases ch) (positive s) (positive st)
       (negative sct) (negative apt) (positive ab) (unchanged spc) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (equal_zero ch) (positive s) (positive st) (negative sct) (positive
      apt)) (:effects (decreases a) (decreases ap) (increases ch) (positive s) (positive st)
       (positive sct) (negative apt) (positive ab) (increases spc) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (greater_zero ch) (positive s) (positive st) (negative sct) (negative
       ch2)) (:effects (decreases a) (decreases ap) (increases ch) (positive st) (positive
      sct) (positive ch2) (increases spc) (negative apt) (unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero au) (greater_zero a) (
      greater_zero ap) (greater_zero ch) (positive s) (negative st) (negative ch2)) (:
      effects (decreases a) (decreases ap) (increases ch) (positive st) (positive ch2) (
      unchanged au)))
   ((:conditions (greater_zero r) (equal_zero fl) (greater_zero ch) (positive ch2) (
      equal_zero a) (equal_zero nd) (equal_zero nu) (equal_zero ad) (greater_zero au)) (:
      effects (decreases au) (increases ad) (unchanged ch)))
   ((:conditions (greater_zero r) (greater_zero fl) (greater_zero ch) (equal_zero a) (
      equal_zero nd) (equal_zero nu) (equal_zero ad) (greater_zero au)) (:effects (decreases
       au) (increases ad)))
  )
)
```

B.13 Logistics

```
(policy
```

```
(:features
 (numerical remaining "r" "packages not at their goal location" (n_count (c_and (c_and (
    c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "location")))
    (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true))))))
 (numerical carried "c" "packages in any carrier" (n_count (c_and (c_and (c_atomic_state "
    obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "location"))) (c_some (
    r_atomic_state "in") (c_or (c_atomic_state "truck") (c_atomic_state "airplane"))))))
 (numerical ready_unload "ru" "carried packages whose carrier is at their goal location" (
    n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_and (c_some (r_atomic_state "in") (c_or (
    c_atomic_state "truck") (c_atomic_state "airplane"))) (c_same_as (r_composition (
    r_atomic_state "in") (r_atomic_state "at")) (r_atomic_goal "at" true))))))
 (numerical pkg_same_city "psc" "unshipped packages at a location in their goal city" (
    n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_and (c_some (r_atomic_state "at") (c_atomic_state "
    location")) (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))
     (c_same_as (r_composition (r_atomic_state "at") (r_atomic_state "in-city")) (
    r_composition (r_atomic_goal "at" true) (r_atomic_state "in-city")))))))))
 (numerical pkg_diff_city "pdc" "unshipped packages at a location outside their goal city"
    (n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_and (c_some (r_atomic_state "at") (c_atomic_state "
    location")) (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))
     (c_not (c_same_as (r_composition (r_atomic_state "at") (r_atomic_state "in-city")) (
    r_composition (r_atomic_goal "at" true) (r_atomic_state "in-city"))))))))))
 (numerical truck_at_pkg_same "tps" "same-city packages with a truck at package location" (
    n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_and (c_some (r_atomic_state "at") (c_atomic_state "
    location")) (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))
     (c_and (c_same_as (r_composition (r_atomic_state "at") (r_atomic_state "in-city")) (
    r_composition (r_atomic_goal "at" true) (r_atomic_state "in-city"))) (c_some (
    r_atomic_state "at") (c_some (r_inverse (r_atomic_state "at")) (c_atomic_state "truck")
    ))))))))
 (numerical truck_at_pkg_diff "tpd" "different-city packages with a truck at package
    location" (n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at"
    true) (c_atomic_state "location"))) (c_and (c_some (r_atomic_state "at") (
    c_atomic_state "location")) (c_and (c_not (c_same_as (r_composition (r_atomic_state "
    at") (r_atomic_state "in-city")) (r_composition (r_atomic_goal "at" true) (
    r_atomic_state "in-city")))) (c_some (r_atomic_state "at") (c_some (r_inverse (
    r_atomic_state "at")) (c_atomic_state "truck")))))))))
 (numerical in_truck_same "its" "packages in trucks in their goal city" (n_count (c_and (
    c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "
    location"))) (c_and (c_some (r_atomic_state "in") (c_atomic_state "truck")) (c_same_as
     (r_composition (r_composition (r_atomic_state "in") (r_atomic_state "at")) (
    r_atomic_state "in-city")) (r_composition (r_atomic_goal "at" true) (r_atomic_state "
    in-city")))))))
 (numerical in_truck_diff "itd" "packages in trucks outside their goal city" (n_count (
    c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "
    location"))) (c_and (c_some (r_atomic_state "in") (c_atomic_state "truck")) (c_not (
    c_same_as (r_composition (r_composition (r_atomic_state "in") (r_atomic_state "at")) (
    r_atomic_state "in-city")) (r_composition (r_atomic_goal "at" true) (r_atomic_state "
    in-city")))))))))
 (numerical truck_airport_diff "tad" "different-city packages in trucks at airports" (
    n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_and (c_some (r_atomic_state "in") (c_atomic_state "
    truck")) (c_and (c_not (c_same_as (r_composition (r_composition (r_atomic_state "in")
    (r_atomic_state "at")) (r_atomic_state "in-city")) (r_composition (r_atomic_goal "at"
    true) (r_atomic_state "in-city")))) (c_some (r_composition (r_atomic_state "in") (
    r_atomic_state "at")) (c_atomic_state "airport")))))))
 (numerical pkg_airport_diff "pad" "different-city packages at airports" (n_count (c_and (
    c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "
    location"))) (c_and (c_some (r_atomic_state "at") (c_atomic_state "airport")) (c_not (
    c_same_as (r_composition (r_atomic_state "at") (r_atomic_state "in-city")) (
    r_composition (r_atomic_goal "at" true) (r_atomic_state "in-city")))))))))
 (numerical plane_at_pkg_airport "ppa" "different-city airport packages with airplane at
    same airport" (n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at"
```

```
     true) (c_atomic_state "location"))) (c_and (c_some (r_atomic_state "at") (
    c_atomic_state "airport")) (c_and (c_not (c_same_as (r_composition (r_atomic_state "at
    ") (r_atomic_state "in-city")) (r_composition (r_atomic_goal "at" true) (
    r_atomic_state "in-city")))) (c_some (r_atomic_state "at") (c_some (r_inverse (
    r_atomic_state "at")) (c_atomic_state "airplane")))))))))
  (numerical in_plane_notdest "ipn" "packages in airplanes outside goal city" (n_count (
    c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "
    location"))) (c_and (c_some (r_atomic_state "in") (c_atomic_state "airplane")) (c_not
    (c_same_as (r_composition (r_composition (r_atomic_state "in") (r_atomic_state "at"))
    (r_atomic_state "in-city")) (r_composition (r_atomic_goal "at" true) (r_atomic_state "
    in-city")))))))))
  (numerical in_plane_dest "ipd" "packages in airplanes in their goal city" (n_count (c_and
    (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (c_atomic_state "
    location"))) (c_and (c_some (r_atomic_state "in") (c_atomic_state "airplane")) (
    c_same_as (r_composition (r_composition (r_atomic_state "in") (r_atomic_state "at")) (
    r_atomic_state "in-city")) (r_composition (r_atomic_goal "at" true) (r_atomic_state "
    in-city")))))))
  (numerical pkg_dest_airport "pda" "packages at airport in their goal city but not at goal"
     (n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at" true) (
    c_atomic_state "location"))) (c_and (c_some (r_atomic_state "at") (c_atomic_state "
    airport")) (c_and (c_not (c_same_as (r_atomic_state "at") (r_atomic_goal "at" true)))
    (c_same_as (r_composition (r_atomic_state "at") (r_atomic_state "in-city")) (
    r_composition (r_atomic_goal "at" true) (r_atomic_state "in-city")))))))))
  (numerical truck_at_dest_airport "tda" "destination-city airport packages with truck at
    same airport" (n_count (c_and (c_and (c_atomic_state "obj") (c_some (r_atomic_goal "at"
     true) (c_atomic_state "location"))) (c_and (c_some (r_atomic_state "at") (
    c_atomic_state "airport")) (c_and (c_not (c_same_as (r_atomic_state "at") (
    r_atomic_goal "at" true))) (c_and (c_same_as (r_composition (r_atomic_state "at") (
    r_atomic_state "in-city")) (r_composition (r_atomic_goal "at" true) (r_atomic_state "
    in-city"))) (c_some (r_atomic_state "at") (c_some (r_inverse (r_atomic_state "at")) (
    c_atomic_state "truck")))))))))
)
(:rules
  ((:conditions (greater_zero ready_unload)) (:effects (decreases remaining) (decreases
    ready_unload) (decreases carried)))
  ((:conditions (equal_zero carried) (equal_zero truck_at_pkg_same) (equal_zero
    truck_at_pkg_diff) (equal_zero truck_at_dest_airport) (equal_zero pkg_airport_diff) (
    equal_zero pkg_dest_airport) (greater_zero pkg_same_city)) (:effects (increases
    truck_at_pkg_same) (unchanged pkg_same_city)))
  ((:conditions (equal_zero carried) (equal_zero truck_at_pkg_same) (equal_zero
    truck_at_pkg_diff) (equal_zero truck_at_dest_airport) (equal_zero pkg_airport_diff) (
    greater_zero pkg_diff_city)) (:effects (increases truck_at_pkg_diff) (unchanged
    pkg_diff_city)))
  ((:conditions (equal_zero carried) (greater_zero pkg_airport_diff) (equal_zero
    plane_at_pkg_airport)) (:effects (increases plane_at_pkg_airport) (unchanged
    pkg_airport_diff)))
  ((:conditions (equal_zero carried) (equal_zero truck_at_pkg_same) (equal_zero
    truck_at_pkg_diff) (equal_zero truck_at_dest_airport) (greater_zero pkg_dest_airport))
     (:effects (increases truck_at_dest_airport) (unchanged pkg_dest_airport)))
  ((:conditions (equal_zero carried) (equal_zero pkg_airport_diff) (greater_zero
    pkg_same_city) (greater_zero truck_at_pkg_same)) (:effects (decreases pkg_same_city) (
    increases carried) (increases in_truck_same) (unchanged remaining)))
  ((:conditions (greater_zero in_truck_same) (equal_zero ready_unload)) (:effects (increases
     ready_unload) (unchanged carried) (unchanged in_truck_same)))
  ((:conditions (equal_zero carried) (equal_zero pkg_airport_diff) (greater_zero
    pkg_diff_city) (greater_zero truck_at_pkg_diff)) (:effects (decreases pkg_diff_city) (
    increases carried) (increases in_truck_diff) (unchanged remaining)))
  ((:conditions (greater_zero in_truck_diff) (equal_zero truck_airport_diff)) (:effects (
    increases truck_airport_diff) (unchanged carried) (unchanged in_truck_diff)))
  ((:conditions (greater_zero truck_airport_diff)) (:effects (decreases carried) (decreases
    in_truck_diff) (increases pkg_airport_diff) (unchanged remaining)))
  ((:conditions (equal_zero carried) (greater_zero pkg_airport_diff) (greater_zero
    plane_at_pkg_airport)) (:effects (decreases pkg_airport_diff) (increases carried) (
    increases in_plane_notdest) (unchanged remaining)))
  ((:conditions (greater_zero in_plane_notdest)) (:effects (decreases in_plane_notdest) (
```

```
      increases in_plane_dest) (unchanged carried)))
   ((:conditions (greater_zero in_plane_dest) (equal_zero ready_unload)) (:effects (decreases
       carried) (decreases in_plane_dest) (increases pkg_dest_airport) (unchanged remaining))
      )
   ((:conditions (equal_zero carried) (greater_zero pkg_dest_airport) (greater_zero
      truck_at_dest_airport)) (:effects (decreases pkg_dest_airport) (increases carried) (
      increases in_truck_same) (unchanged remaining)))
  )
)
```

B.14 Miconic

```
(policy
  (:features
    (numerical remaining "r" "passengers not yet served" (n_count (c_and (c_atomic_state "
       passenger") (c_not (c_atomic_state "served")))))
    (numerical boarded "b" "passengers currently in the lift" (n_count (c_atomic_state "
       boarded")))
    (numerical ready_depart "rd" "boarded passengers whose destination is the current lift
       floor" (n_count (c_and (c_atomic_state "boarded") (c_some (r_atomic_state "destin") (
       c_atomic_state "lift-at")))))
    (numerical ready_board "rb" "waiting passengers whose origin is the current lift floor" (
       n_count (c_some (r_atomic_state "origin") (c_atomic_state "lift-at"))))
    (numerical boarded_dest_above "bda" "boarded passengers with destination above the current
        lift floor" (n_count (c_and (c_atomic_state "boarded") (c_some (r_atomic_state "
       destin") (c_some (r_inverse (r_atomic_state "above")) (c_atomic_state "lift-at"))))))
    (numerical boarded_dest_below "bdb" "boarded passengers with destination below the current
        lift floor" (n_count (c_and (c_atomic_state "boarded") (c_some (r_atomic_state "
       destin") (c_some (r_atomic_state "above") (c_atomic_state "lift-at"))))))
    (numerical origin_above "oa" "waiting passengers whose origin is above the current lift
       floor" (n_count (c_some (r_atomic_state "origin") (c_some (r_inverse (r_atomic_state "
       above")) (c_atomic_state "lift-at")))))
    (numerical origin_below "ob" "waiting passengers whose origin is below the current lift
       floor" (n_count (c_some (r_atomic_state "origin") (c_some (r_atomic_state "above") (
       c_atomic_state "lift-at")))))
    (numerical dist_boarded_dest_above "dbda" "upward distance from lift to a boarded
       passenger destination" (n_distance (c_atomic_state "lift-at") (r_atomic_state "above")
        (c_some (r_inverse (r_atomic_state "destin")) (c_atomic_state "boarded"))))
    (numerical dist_boarded_dest_below "dbdb" "downward distance from lift to a boarded
       passenger destination" (n_distance (c_atomic_state "lift-at") (r_inverse (
       r_atomic_state "above")) (c_some (r_inverse (r_atomic_state "destin")) (c_atomic_state
        "boarded"))))
    (numerical dist_origin_above "doa" "upward distance from lift to a waiting passenger
       origin" (n_distance (c_atomic_state "lift-at") (r_atomic_state "above") (c_some (
       r_inverse (r_atomic_state "origin")) (c_atomic_state "passenger"))))
    (numerical dist_origin_below "dob" "downward distance from lift to a waiting passenger
       origin" (n_distance (c_atomic_state "lift-at") (r_inverse (r_atomic_state "above")) (
       c_some (r_inverse (r_atomic_state "origin")) (c_atomic_state "passenger"))))
  )
  (:rules
    ((:conditions (greater_zero ready_depart)) (:effects (decreases remaining) (decreases
       boarded) (decreases ready_depart)))
    ((:conditions (greater_zero ready_board)) (:effects (increases boarded) (decreases
       ready_board) (unchanged remaining)))
    ((:conditions (greater_zero boarded) (equal_zero ready_depart) (greater_zero
       boarded_dest_above)) (:effects (decreases dist_boarded_dest_above) (unchanged boarded)
        (unchanged remaining)))
    ((:conditions (greater_zero boarded) (equal_zero ready_depart) (equal_zero
       boarded_dest_above) (greater_zero boarded_dest_below)) (:effects (decreases
       dist_boarded_dest_below) (unchanged boarded) (unchanged remaining)))
    ((:conditions (equal_zero boarded) (equal_zero ready_board) (greater_zero origin_above)) (:
       effects (decreases dist_origin_above) (unchanged remaining)))
    ((:conditions (equal_zero boarded) (equal_zero ready_board) (equal_zero origin_above) (
       greater_zero origin_below)) (:effects (decreases dist_origin_below) (unchanged
       remaining)))
```

```
  )
)
```

B.15 Satellite

```
(policy
  (:features
    (numerical remaining "r" "missing goal image pairs" (n_count (r_and (r_atomic_goal "
        have_image" true) (r_complement (r_atomic_state "have_image")))))
    (numerical useful_off "uo" "off instruments on powered satellites supporting a remaining
        goal mode" (n_count (c_and (c_atomic_state "instrument") (c_and (c_not (c_atomic_state
         "power_on")) (c_and (c_some (r_atomic_state "on_board") (c_atomic_state "power_avail")
        ) (c_some (r_atomic_state "supports") (c_some (r_inverse (r_atomic_goal "have_image"
        true)) (c_and (c_atomic_state "direction") (c_not (c_same_as (r_atomic_state "
        have_image") (r_atomic_goal "have_image" true)))))))))))
    (numerical powered_uncalibrated "pu" "powered uncalibrated instruments supporting a
        remaining goal mode" (n_count (c_and (c_atomic_state "instrument") (c_and (
        c_atomic_state "power_on") (c_and (c_not (c_atomic_state "calibrated")) (c_some (
        r_atomic_state "supports") (c_some (r_inverse (r_atomic_goal "have_image" true)) (
        c_and (c_atomic_state "direction") (c_not (c_same_as (r_atomic_state "have_image") (
        r_atomic_goal "have_image" true)))))))))))
    (numerical calibration_ready "cr" "powered uncalibrated useful instruments whose satellite
         points at their calibration target" (n_count (c_and (c_atomic_state "instrument") (
        c_and (c_atomic_state "power_on") (c_and (c_not (c_atomic_state "calibrated")) (c_and
        (c_same_as (r_atomic_state "calibration_target") (r_composition (r_atomic_state "
        on_board") (r_atomic_state "pointing"))) (c_some (r_atomic_state "supports") (c_some (
        r_inverse (r_atomic_goal "have_image" true)) (c_and (c_atomic_state "direction") (
        c_not (c_same_as (r_atomic_state "have_image") (r_atomic_goal "have_image" true)))))))))
        ))))
    (numerical useful_calibrated "uc" "powered calibrated instruments supporting a remaining
        goal mode" (n_count (c_and (c_atomic_state "instrument") (c_and (c_atomic_state "
        power_on") (c_and (c_atomic_state "calibrated") (c_some (r_atomic_state "supports") (
        c_some (r_inverse (r_atomic_goal "have_image" true)) (c_and (c_atomic_state "direction
        ") (c_not (c_same_as (r_atomic_state "have_image") (r_atomic_goal "have_image" true))))
        )))))))
    (numerical image_ready "ir" "missing image pairs supported by a powered calibrated
        instrument on a satellite pointing at the direction" (n_count (r_and (r_and (
        r_atomic_goal "have_image" true) (r_complement (r_atomic_state "have_image"))) (
        r_composition (r_restriction (r_composition (r_inverse (r_atomic_state "pointing")) (
        r_inverse (r_atomic_state "on_board"))) (c_and (c_atomic_state "instrument") (c_and (
        c_atomic_state "power_on") (c_atomic_state "calibrated")))) (r_atomic_state "supports")
        ))))
    (numerical powered_irrelevant "pi" "powered instruments that do not support any remaining
        goal mode" (n_count (c_and (c_atomic_state "instrument") (c_and (c_atomic_state "
        power_on") (c_not (c_some (r_atomic_state "supports") (c_some (r_inverse (
        r_atomic_goal "have_image" true)) (c_and (c_atomic_state "direction") (c_not (
        c_same_as (r_atomic_state "have_image") (r_atomic_goal "have_image" true)))))))))))
    (numerical bad_images "bi" "acquired image pairs that are not goals" (n_count (r_and (
        r_atomic_state "have_image") (r_complement (r_atomic_goal "have_image" true)))))
    (numerical remaining_pointing "rp" "missing goal pointing pairs" (n_count (r_and (
        r_atomic_goal "pointing" true) (r_complement (r_atomic_state "pointing")))))
  )
  (:rules
    ((:conditions (greater_zero image_ready)) (:effects (decreases remaining) (unchanged
        bad_images)))
    ((:conditions (greater_zero calibration_ready) (equal_zero image_ready)) (:effects (
        decreases powered_uncalibrated) (increases useful_calibrated) (unchanged remaining) (
        unchanged bad_images)))
    ((:conditions (greater_zero powered_uncalibrated) (equal_zero calibration_ready) (
        equal_zero image_ready)) (:effects (increases calibration_ready) (unchanged
        powered_uncalibrated) (unchanged remaining) (unchanged bad_images)))
    ((:conditions (greater_zero useful_calibrated) (equal_zero image_ready)) (:effects (
        increases image_ready) (unchanged useful_calibrated) (unchanged remaining) (unchanged
        bad_images)))
```

```
   ((:conditions (greater_zero useful_off) (equal_zero useful_calibrated) (equal_zero
      powered_uncalibrated) (equal_zero image_ready)) (:effects (decreases useful_off) (
      increases powered_uncalibrated) (unchanged remaining) (unchanged bad_images)))
   ((:conditions (greater_zero powered_irrelevant) (greater_zero remaining) (equal_zero
      useful_off) (equal_zero useful_calibrated) (equal_zero powered_uncalibrated) (
      equal_zero image_ready)) (:effects (decreases powered_irrelevant) (unchanged remaining)
       (unchanged bad_images)))
   ((:conditions (equal_zero remaining) (greater_zero remaining_pointing)) (:effects (
      decreases remaining_pointing) (unchanged remaining) (unchanged bad_images)))
 )
)
```