

Planning in Incomplete Domains

Daniel Bryce
daniel.bryce@usu.edu
Department of Computer Science
Utah State University
4205 Old Main Hill, Logan, UT 84321

March 9, 2011

Abstract

Engineering complete planning domain descriptions is often very costly because of human-error or lack of domain knowledge. While many have studied knowledge acquisition, relatively few have studied the synthesis of plans when the domain model is incomplete (i.e., actions have incomplete preconditions or effects). Prior work has evaluated the correctness of plans synthesized by disregarding such incomplete features, but not how to synthesize plans by reasoning about the incompleteness. In this work, we describe several techniques for reasoning with the action incompleteness to make plans robust, as measured by the number of incomplete domain interpretations under which a plan succeeds. Among the techniques, we show that representing explanations of plan failure with prime implicants provides a natural approach to comparing plans by counting prime implicants (diagnoses) instead of domain interpretations (i.e., model counting) – leading to better scalability and higher quality plans.

We present and empirically evaluate a forward heuristic search planner, called `DeFAULT`, that synthesizes plans by propagating information about faults due to incompleteness both within the state space and the relaxed planning space by using intuitions from model based diagnosis and assumption-based truth maintenance systems. We also provide a translation from incomplete planning domains to conformant probabilistic planning, where action incompleteness is represented by state incompleteness. We compare `DeFAULT` with a control planner that uses the FF heuristic (measuring plan length and ignoring incompleteness), and with the conformant probabilistic planner `POND`. The results show that `DeFAULT` i) scales better than `POND`, ii) finds better solutions than a planner using the FF heuristic, iii) scales best and finds its best quality solutions when counting prime implicants rather than models.

1 Introduction

The knowledge engineering required to create complete and correct domain descriptions for planning problems is often very costly and difficult [Kambhampati, 2007; Wu *et al.*, 2007]. Machine learning techniques have been applied with some success [Wu *et al.*, 2007; Bertoli *et al.*, 2009], but still suffer from impoverished data and limitations of the algorithms [Kambhampati, 2007]. In particular, we are motivated by applications in instructable computing [Mailler *et al.*, 2009] where a domain expert teaches an intelligent system about a particular domain, but can often leave out whole procedures (plans) and aspects of action descriptions. In such cases, the alternative to making domains complete, is to plan around the incompleteness. That is, given knowledge of the possible action descriptions, we seek out plans that will succeed despite any (or most) incompleteness in the domain formulation.

While prior work [Chang and Amir, 2006] has proposed techniques for learning about the incomplete features of a domain, Garland and Lesh [2002] have categorized faults to a plan’s correctness and described plan quality metrics in terms of the faults (which are essentially single-fault diagnoses of plan failure [de Kleer and Williams, 1987; Reiter, 1987]), no prior work has sought to deliberately synthesize low-fault plans based on incomplete STRIPS-style domains (notable work in Markov decision processes [Nilim and El Ghaoui, 2005] and model-based reinforcement learning [Sutton and Barto, 1998] has explored similar issues). Specifically, Garland and Lesh [2002] (henceforth abbreviated, GL) identify four types of plan faults: open preconditions (due to incomplete preconditions), possible clobberers (due to incomplete delete effects), unlisted effects (due to incomplete add effects), and false preconditions. GL develop an algorithm that steps backward through the plan to identify the “critical faults” – those instances where incomplete domain features can cause plan failure. For example, a possible clobberer is a critical fault when (if it is truly a delete effect) it threatens a precondition or goal. The number of critical faults is an important measure of plan quality/correctness, that, unfortunately, no known planners seek to minimize (aside from our prior work [Robertson and Bryce, 2009] on single-fault planning, upon which this work is based). A preliminary version of this work has also appeared recently [Bryce and Weber, 2011].

Example: Consider the following action, that is taken from a modified version of the International Planning Competition (IPC) [Gerevini *et al.*, 2006] PARC printer domain:

```
(:action HtmOverBlack-Move-A4
:parameters ( ?sheet - sheet_t )
:precondition (and (clear) (Available HtmOverBlack-RSRC)
                  (Sheetsize ?sheet A4)
                  (Location ?sheet HtmOverBlack_Entry-EndCap_Exit))
:effect (and (not (Available HtmOverBlack-RSRC))
             (Location ?sheet HtmOverBlack_Exit-Down_TopEntry)
             (not (Location ?sheet HtmOverBlack_Entry-EndCap_Exit))
             (Available HtmOverBlack-RSRC))
:poss-effect (and (not (clear))))
```

The action models a modular printer component that prints on a sheet of A4-sized paper. The action is incomplete because it has a possible effect that the component will become jammed “(not (clear))”. The intuition behind the action is that the component manufacturer did not provide complete specifications and it is unknown if feeding an A4 sheet will cause a paper jam.

We note that an incomplete action is different from a non-deterministic action because each application of the incomplete action will have the same effect at runtime; however, it is not clear what the effect will be at planning time. The action incompleteness can cause plan failure, as in the case of our example, by threatening the precondition of a later action (e.g., the precondition (clear) is threatened in a second application of the HtmOverBlack-Move-A4 action).

Interpreting Incompleteness: A pessimistic approach to reasoning about incomplete actions might assume that possible delete effects will always occur. Plans found under this pessimistic interpretation will be correct despite any action incompleteness, but are likely to be few or nonexistent. In the PARC printer example, a pessimistic interpretation will likely lead to proving that no plan exists, even though it is possible that the action does not have the delete effect on (clear). Alternatively, an optimistic interpretation might assume that no possible delete effect occurs, in which case the planner can ignore that (clear) may be deleted. The optimistic interpretation is equally flawed because the action may actually delete (clear).

Instead, we adopt a cautiously optimistic interpretation where, like the optimistic interpretation, we assume that possible delete effects do not occur, but we also temper our optimism. We compute an explanation for cases under which each proposition that is optimistically true might be false. For example, after applying the action above, we would assert that `(clear)` is true, subject to the assumption that `(clear)` is not a delete effect of the action. Under this cautiously optimistic semantics, we can determine which interpretations of incomplete actions will result in failed goal achievement by inspecting the assumptions under which the goals are false. Plans that fail under fewer interpretations are preferred.

Failure Explanations and Counting: We take three qualitatively different approaches to recording the failure explanation for each proposition established at different times by a plan. The first, our control, amounts to the optimistic interpretation by recording no explanation for the failure to achieve a proposition. The second and third approaches represent failure explanations with propositional sentences, whose models correspond to interpretations of the incomplete actions. The second approach relies on intuitions from model-based diagnosis to represent each failure explanation by a set of diagnoses (each diagnosis is a conjunction of incomplete action features – i.e., a prime implicant). We also explore bounding the cardinality of the prime implicants to reduce the cost of reasoning; GL amounts to evaluating plans with unit cardinality prime implicants. The third approach represents failure explanations by OBDDs. The second and third approaches provide a representation suitable for counting interpretations of the incomplete action features (i.e., propositional models) under which a state proposition is achieved or not. The primary difference is that model counting with prime implicants is intractable [Roth, 1996], but polynomial in the size of an OBDD [Darwiche and Marquis, 2002]. While we use each of the three approaches during plan synthesis to compare plans (in varying capacities), we use the third to provide a final assessment of a plan’s quality: the number of interpretations of the incomplete actions under which the plan fails. That is, we describe several heuristic techniques to speed-up plan synthesis that are based on a particular representation of the failure explanations, but compare the resulting plans with a single, non-heuristic method.

For example, the first approach is entirely heuristic because it completely ignores failure explanations. In the second approach, we represent the failure explanations by prime implicants and, instead of counting models, we count the number of prime implicants. Counting prime implicants is a computationally inexpensive heuristic that assumes fewer diagnoses means fewer failed interpretations of the incomplete actions. The third method counts the actual number of failed action interpretations by representing them as an OBDD (which can be exponential-sized) and performing OBDD model counting (which is polynomial in the OBDD size). We claim that counting diagnoses (prime implicants) is more computationally feasible than counting OBDD models and the resulting plans are of similar quality, and that ignoring incompleteness altogether leads to poor quality plans.

Our claims are based upon GL’s focus on counting a plan’s critical faults as a measure of its quality. We observe that GL’s definition of critical faults is equivalent to computing single-fault diagnoses, which allows us to generalize their notions to multi-fault diagnoses. Intuitively, with more diagnoses for plan failure, the fewer interpretations of the incomplete domain that will achieve the goal. Naturally, a single-fault diagnosis covers more interpretations than a double- or triple-fault, so we count not just the number of diagnoses, but those of different cardinality. We stress that counting diagnoses is an approximation to counting models, but it can nevertheless lead to more efficient planners that find comparable quality solutions.

Planners: We present a forward heuristic planner, called `DeFAULT`, that propagates failure explanations in the state space and relaxed planning problems. `DeFAULT` associates a set of explanations with each time step (i.e., each state in the search space or each planning graph layer in the relaxed planning problem). `DeFAULT`’s heuristic biases search toward plans that will fail in as few interpretations of the incomplete domain as possible. Because no suitable prior work exists for the purpose of empirical comparisons, we

not only compare DeFAULT with a planner that uses the FF heuristic and ignores domain incompleteness, but also attempt a *more fair* comparison with a conformant probabilistic planner. We translate incomplete actions to incomplete states in conformant probabilistic planning and apply an existing planner, POND [Bryce *et al.*, 2006].

Our results indicate that DeFAULT can find much better quality plans than a planner that ignores incompleteness and scales much better than a CPP planner, POND [Bryce *et al.*, 2008]. Because the CPP planner scales so poorly, detailed comparisons are somewhat uninformative (we compare only the number of problems solved); therefore, we discuss the CPP planner in an appendix. In the following, we provide background on the representation of the planning problems studied, a discussion of languages used to capture incomplete actions, a formulation of failure explanations, a definition of diagnosis and model counting, a planner based on failure propagation, a relaxed planning heuristic for failure propagation, empirical evaluation, related work, and conclusion.

2 Background & Representation

This work concerns three planning models: STRIPS, incomplete STRIPS, and conformant probabilistic planning. In the following, we define the first two models, the related action representations, and plan semantics. The discussion of conformant probabilistic planning is relegated to an appendix.

2.1 STRIPS Planning Domains

STRIPS planning domains [Fikes and Nilsson, 1971] correspond to the classical planning model.

Definition 2.1. A STRIPS planning domain D defines the tuple (P, A, I, G) , where

- P is a set of propositions
- A is a set of complete action descriptions, where each $a \in A$ defines
 - $\text{pre}(a) \subseteq P$, a set of preconditions
 - $\text{add}(a) \subseteq P$, a set of add effects
 - $\text{del}(a) \subseteq P$, a set of delete effects
- $I \subseteq P$ defines a set of initially true propositions
- $G \subseteq P$ defines the goal propositions

For example, consider the following domain, which we will use as a running example:

- $P = \{p, q, r, g\}$
- $A = \{a, b, c\}$
 - $\text{pre}(a) = \{p, q\}$, $\text{add}(a) = \{r\}$, $\text{del}(a) = \{\}$
 - $\text{pre}(b) = \{p\}$, $\text{add}(b) = \{r\}$, $\text{del}(b) = \{p\}$
 - $\text{pre}(c) = \{q, r\}$, $\text{add}(c) = \{g\}$, $\text{del}(c) = \{\}$
- $I = \{p, q\}$
- $G = \{g\}$

A plan π for D is a sequence of actions, that when applied to the initial state, leads to a state where the goal is satisfied.

Definition 2.2. A plan $\pi = (a_{-1}, a_0, \dots, a_{n-1}, a_n)$ in a STRIPS domain D is sequence of actions, that corresponds to a sequence of states (s_0, \dots, s_n) , where

- $s_0 = \text{add}(a_{-1}) = I$
- $\text{pre}(a_t) \subseteq s_t$ for $t = 0, \dots, n$
- $s_{t+1} = s_t \setminus \text{del}(a_t) \cup \text{add}(a_t)$ for $t = 0, \dots, n - 1$
- $\text{pre}(a_n) = G$

We omit a_{-1} and a_n from the plans in our discussion when appropriate, with the understanding that each plan must use these artificial initial and goal actions.

For example, the plan (a, b, c) corresponds to the state sequence $(s_0 = \{p, q\}, s_1 = \{p, q, r\}, s_2 = \{q, r\}, s_3 = \{q, r, g\})$, where the goal is satisfied in s_3 .

2.2 Incomplete STRIPS Domains

Incomplete STRIPS domains are identical to STRIPS domains, with the exception that the actions are incompletely specified. Much like planning with incomplete state information [Bonet and Geffner, 2000], the action incompleteness is not completely unbounded. The preconditions and effects of each action can be any subset of the propositions P ; the incompleteness is with regard to a lack of knowledge about which of the subsets correspond to each precondition and effect. We find it convenient to refer to the *known*, *possible*, and *impossible* preconditions and effects. For example, an action's preconditions must consist of the known preconditions, and it must not contain the impossible preconditions, but we do not know if it contains the possible preconditions. The union of the known, possible, and impossible preconditions must equal P ; therefore, an action can represent any two, and we can infer the third. We choose to represent the known and possible, but note that GL represent the known and impossible; with the trade-off making our representation more appropriate if there are fewer possible action features. We contrast our choice of representation with GL in the following section.

Another aspect of our representation that is worthy of explanation is the way in which the incomplete features are independent. For example, we cannot assert that an action has two possible add effects p and q or it has neither. We can only state that p is a possible add effect and q is a possible add effect. We leave the issue of handling such interacting incomplete features for future work and include a discussion of the issues in our conclusion.

In the following, we extend the STRIPS model with features for possible preconditions and effects. We note that an incomplete domain corresponds to a set of complete domains, each differing in terms of the inclusion of the possible features.

Definition 2.3. An incomplete STRIPS domain \tilde{D} defines the tuple (P, \tilde{A}, I, G) , where

- P is a set of propositions
- \tilde{A} is a set of incomplete action descriptions, where each $\tilde{a} \in \tilde{A}$ defines
 - $\text{pre}(\tilde{a}) \subseteq P$, a set of known preconditions
 - $\widetilde{\text{pre}}(\tilde{a}) \subseteq P$, a set of possible preconditions
 - $\text{add}(\tilde{a}) \subseteq P$, a set of known add effects
 - $\widetilde{\text{add}}(\tilde{a}) \subseteq P$, a set of possible add effects
 - $\text{del}(\tilde{a}) \subseteq P$, a set of known delete effects

- $\widetilde{del}(\tilde{a}) \subseteq P$, a set of possible delete effects
- $I \subseteq P$ defines a set of initially true propositions
- $G \subseteq P$ defines the goal propositions

Consider the following example of an incomplete domain:

- $P = \{p, q, r, g\}$
- $\tilde{A} = \{\tilde{a}, \tilde{b}, \tilde{c}\}$
- $pre(\tilde{a}) = \{p, q\}$, $\widetilde{pre}(\tilde{a}) = \{r\}$, $add(\tilde{a}) = \{\}$, $\widetilde{add}(\tilde{a}) = \{r\}$, $del(\tilde{a}) = \{\}$, $\widetilde{del}(\tilde{a}) = \{p\}$
- $pre(\tilde{b}) = \{p\}$, $\widetilde{pre}(\tilde{b}) = \{\}$, $add(\tilde{b}) = \{r\}$, $\widetilde{add}(\tilde{b}) = \{\}$, $del(\tilde{b}) = \{p\}$, $\widetilde{del}(\tilde{b}) = \{q\}$
- $pre(\tilde{c}) = \{r\}$, $\widetilde{pre}(\tilde{c}) = \{q\}$, $add(\tilde{c}) = \{g\}$, $\widetilde{add}(\tilde{c}) = \{\}$, $del(\tilde{c}) = \{\}$, $\widetilde{del}(\tilde{c}) = \{\}$
- $I = \{p, q\}$
- $G = \{g\}$

A plan $\tilde{\pi}$ for \tilde{D} is a sequence of actions, that when applied, may lead to a state where the goal is satisfied.

Definition 2.4. A plan $\tilde{\pi} = (\tilde{a}_{-1}, \tilde{a}_0, \dots, \tilde{a}_{n-1}, \tilde{a}_n)$ in an incomplete domain \tilde{D} is sequence of actions, that corresponds to a sequence of states (s_0, \dots, s_n) , where

- $s_0 = add(\tilde{a}_{-1}) = I$
- $pre(\tilde{a}_t) \subseteq s_t$ for $t = 0, \dots, n$
- $s_{t+1} = s_t \setminus del(\tilde{a}_t) \cup add(\tilde{a}_t) \cup \widetilde{add}(\tilde{a}_t)$ for $t = 0, \dots, n - 1$
- $G = pre(\tilde{a}_n)$

For example, the plan $(\tilde{a}, \tilde{b}, \tilde{c})$ corresponds to the state sequence $(s_0 = \{p, q\}, s_1 = \{p, q, r\}, s_2 = \{q, r\}, s_3 = \{q, r, g\})$, where the goal is satisfied in s_3 .

Definition 2.5. The set of incomplete domain features F is comprised of the following propositions for each $\tilde{a} \in \tilde{A}$:

- $\widetilde{pre}(\tilde{a}, p)$ if $p \in \widetilde{pre}(\tilde{a})$
- $\widetilde{add}(\tilde{a}, p)$ if $p \in \widetilde{add}(\tilde{a})$
- $\widetilde{del}(\tilde{a}, p)$ if $p \in \widetilde{del}(\tilde{a})$

Each subset $F^i \subseteq F$ corresponds to an interpretation D^i of the incomplete domain \tilde{D} .

Definition 2.6. An interpretation D^i of the incomplete domain \tilde{D} is defined with respect to a subset of the incomplete domain features $F^i \subseteq F$ so that:

- $P^i = P$
- $I^i = I$
- $G^i = G$
- For each $\tilde{a} \in \tilde{A}$ there exists an $\tilde{a}^i \in A^i$ where
 - $pre(\tilde{a}^i) = pre(\tilde{a}) \cup \{p \mid \widetilde{pre}(\tilde{a}, p) \in F^i\}$
 - $add(\tilde{a}^i) = add(\tilde{a}) \cup \{p \mid \widetilde{add}(\tilde{a}, p) \in F^i\}$
 - $del(\tilde{a}^i) = del(\tilde{a}) \cup \{p \mid \widetilde{del}(\tilde{a}, p) \in F^i\}$

For example, the STRIPS domain from the previous section is an interpretation of incomplete domain above, where $F^1 = \{\widetilde{\text{add}}(\tilde{a}, r), \widetilde{\text{pre}}(\tilde{c}, q)\}$.

Theorem 2.7. $\tilde{\pi} = (\tilde{a}_{-1}, \tilde{a}_0, \dots, \tilde{a}_{n-1}, \tilde{a}_n)$ is a plan for \tilde{D} , iff there exists at least one interpretation D^i in which $\tilde{\pi}^i = (\tilde{a}_{-1}^i, \tilde{a}_0^i, \dots, \tilde{a}_{n-1}^i, \tilde{a}_n^i)$ is a plan.

Proof. If $\tilde{\pi} = (\tilde{a}_{-1}, \tilde{a}_0, \dots, \tilde{a}_{n-1}, \tilde{a}_n)$ is a plan for \tilde{D} , then there exists at least one interpretation D^i in which $\tilde{\pi}^i = (\tilde{a}_{-1}^i, \tilde{a}_0^i, \dots, \tilde{a}_{n-1}^i, \tilde{a}_n^i)$ is a plan because there is an interpretation D^0 , where $F^0 = \{\widetilde{\text{add}}(\tilde{a}, p) \mid \widetilde{\text{add}}(\tilde{a}, p) \in F\}$. Each action $\tilde{a}^0 \in A^0$ is defined so that

$$\begin{aligned}\text{pre}(\tilde{a}^0) &= \text{pre}(\tilde{a}) \\ \text{add}(\tilde{a}^0) &= \text{add}(\tilde{a}) \cup \widetilde{\text{add}}(\tilde{a}) \\ \text{del}(\tilde{a}^0) &= \text{del}(\tilde{a})\end{aligned}$$

If we assume that $s_t = s_t^0$ (which is true of s_0 and s_0^0 by definition), then

$$\begin{aligned}s_{t+1} &= s_t \setminus \text{del}(\tilde{a}_t) \cup \text{add}(\tilde{a}_t) \cup \widetilde{\text{add}}(\tilde{a}_t) \\ &= s_t^0 \setminus \text{del}(\tilde{a}_t^0) \cup \text{add}(\tilde{a}_t^0) \\ &= s_{t+1}^0\end{aligned}$$

for $t = 0, \dots, n-1$. If $\text{pre}(\tilde{a}_t^0) = \text{pre}(\tilde{a}_t)$ and $s_t^0 = s_t$ for all $t = 0, \dots, n$ then if $\tilde{\pi}$ is a plan $\tilde{\pi}^0$ is a plan.

If $\tilde{\pi}^i = (\tilde{a}_{-1}^i, \tilde{a}_0^i, \dots, \tilde{a}_{n-1}^i, \tilde{a}_n^i)$ is a plan for D^i , then $\tilde{\pi} = (\tilde{a}_{-1}, \tilde{a}_0, \dots, \tilde{a}_{n-1}, \tilde{a}_n)$ is a plan for \tilde{D} because

$$\begin{aligned}\text{pre}(\tilde{a}) &\subseteq \text{pre}(\tilde{a}^i) \\ \text{add}(\tilde{a}) \cup \widetilde{\text{add}}(\tilde{a}_t) &\supseteq \text{add}(\tilde{a}^i) \\ \text{del}(\tilde{a}) &\subseteq \text{del}(\tilde{a}^i)\end{aligned}$$

If we assume that $s_t^i \subseteq s_t$, then

$$\begin{aligned}s_{t+1}^i &= s_t^i \setminus \text{del}(\tilde{a}_t^i) \cup \text{add}(\tilde{a}_t^i) \\ &\subseteq s_t \setminus \text{del}(\tilde{a}_t) \cup \text{add}(\tilde{a}_t) \cup \widetilde{\text{add}}(\tilde{a}_t) \\ &= s_{t+1}\end{aligned}$$

for $t = 0, \dots, n-1$. If $\text{pre}(\tilde{a}_t^i) \subseteq s_t^i$ then $\text{pre}(\tilde{a}_t) \subseteq s_t$ because $\text{pre}(\tilde{a}_t) \subseteq \text{pre}(\tilde{a}_t^i) \subseteq s_t^i \subseteq s_t$ for all $t = 0, \dots, n$ then if $\tilde{\pi}^i$ is a plan $\tilde{\pi}$ is a plan. \square

Definition 2.4 sets a loose requirement that plans with incomplete actions succeed under the most *optimistic* conditions: possible preconditions need not be satisfied and the possible add effects (but not the possible delete effects) are assumed to occur when computing successor states. This notion of optimism is similar to that of GraphPlan [Blum and Furst, 1995] in that both assert every proposition that could be made true at a particular time even if only a subset of the propositions can actually be made true. In GraphPlan, there *may* exist a plan to establish a proposition if the proposition appears in the planning graph, whereas in our definitions there *does* exist an interpretation of the incomplete domain that will establish a proposition if it appears in a state (as a consequence of Theorem 2.7), and this interpretation *may* correspond to the true domain. In GraphPlan, failing to assert a proposition that may be established could eliminate plans, and in

our case, failing to assert a proposition would prevent us from computing interpretations of the incomplete domain that achieve the goal.

By Theorem 2.7 we ensure that the plan is valid for the least constraining (most optimistic) interpretation of the incomplete domain. If the plan can achieve the goal in the most optimistic interpretation then it may achieve the goal in others, but if the goal is not reachable in this interpretation then it cannot be reached in any interpretation. As we will show, we can efficiently determine the interpretations in which a plan is invalid and use the number of such failed interpretations as a plan quality metric.

3 Comparison of Possible Action Features with Local Closed Worlds

Definition 2.3 defines incomplete actions by sets of respective known and possible preconditions and effects. GL define incomplete actions similar to STRIPS actions (Definition 2.1) with additional local closed world statements of the form `DoesNotRelyOn(\tilde{a}, p)` (p is not a precondition of \tilde{a}) or `CompletePreconditions(\tilde{a})` (the preconditions of \tilde{a} are known).

We note that these representations are equivalent if we consider the set of known, possible, and impossible preconditions (and similarly for effects) of actions. For example, stating `CompletePreconditions(\tilde{a})` is equivalent to stating $\widetilde{\text{pre}}(\tilde{a}) = \{\}$ (i.e., the set of possible preconditions is empty). Likewise, stating `DoesNotRelyOn(\tilde{a}, p)` is equivalent to stating $p \notin \widetilde{\text{pre}}(\tilde{a})$, and that for all $q \in P \setminus \text{pre}(\tilde{a})$ the lack of a statement `DoesNotRelyOn(\tilde{a}, q)` is equivalent to stating $q \in \widetilde{\text{pre}}(\tilde{a})$ (i.e., impossible preconditions are not possible preconditions, and not impossible preconditions are possible preconditions).

While the representations are equivalent, the obvious question is whether one is more succinct than the other. The answer largely depends on the problem being modeled. See Table 1 for examples. Notice that the size of the representations is equivalent when stating, for example, that an action has complete preconditions; we either record the fact that the preconditions are complete or that the set of possible preconditions is empty. The difference is with respect to stating, for example, that an individual proposition is not a precondition of an action. Under our representation (Definition 2.3), the set of possible preconditions would not contain a proposition, and under the GL representation it must be stated that the proposition is not a precondition. However, if a proposition is a possible precondition to an action, we would record it as a possible precondition and GL would record nothing. As such, the issue comes down to whether there are many possible or impossible preconditions and effects. Our representation is smaller with many impossible features, and GL is smaller with many possible features.

Example	Definition 2.3	GL
Action \tilde{a} has only the known preconditions p, q	$\text{pre}(\tilde{a}) = \{p, q\},$ $\widetilde{\text{pre}}(\tilde{a}) = \{\}$	$\text{pre}(\tilde{a}) = \{p, q\},$ <code>CompletePreconditions(\tilde{a})</code>
Action \tilde{a} has possible precondition r , but z is neither a known nor a possible precondition	$\text{pre}(\tilde{a}) = \{\},$ $\widetilde{\text{pre}}(\tilde{a}) = \{r\}$	$\text{pre}(\tilde{a}) = \{\},$ <code>DoesNotRelyOn(\tilde{a}, z)</code>

Table 1: Examples comparing representations.

While we describe actions in the grounded (propositional) form, another practical concern is that we use PDDL [McDermott, 1998] action schemas to encode problems. Under the GL representation, extending PDDL action schemas to state impossible preconditions (or effects) could require additional action schema parameters that refer to constants in predicates that are not preconditions. If there are many impossible preconditions, then the action schemas could mention many additional parameters, which is known to lead

to difficulty when grounding the schemas. Our intuition is that possible action features are likely to share parameters with known action features and extending PDDL to support our representation will lead to fewer additional action schema parameters. Furthermore, if there are many impossible features, our representation does not mention these features and therefore does not need to reference their parameters in the PDDL action schemas.

4 Diagnosing Faults in Plans for Incomplete Domains

An plan $\tilde{\pi}$ for an incomplete domain must achieve the goals under optimistic semantics (i.e., possible preconditions need not be satisfied, possible delete effects can be ignored, and possible add effects will occur), but we prefer that plans succeed under more pessimistic conditions. To quantify the extent to which our optimism is misleading, we introduce and expand upon GL's definitions of risks, which we refer to as faults. A fault is a threat to the plan's causal proof that is introduced because of our optimism/ignorance of the underlying domain description. For example, by assuming that a possible delete effect does not occur, we introduce a fault when a possible delete effect does in fact delete a required subgoal. By assuming the optimistic semantics, we allow plans that we might not otherwise consider, but by computing the faults we quantify the level to which the plan is susceptible to failure. The challenge to computing faults is that incomplete action features may have a delayed impact on the plan or no impact at all, and we must determine if they are faults (i.e., guarantee plan failure if the incompleteness manifests unfavorably).

Instead of reviewing GL's definitions, we take a new approach to develop the definitions of faults. Our intuition is that plans with faults are best analyzed within the framework of model-based diagnosis [Reiter, 1987; de Kleer and Williams, 1987]. Among all of the techniques developed within model based diagnosis [de Kleer and Williams, 1987], the most beneficial to analyzing incomplete domains is a clear characterization of multiple-faults. In contrast, GL discusses only single-faults, that they call risks, which do not explain plan failures that may occur because of multiple, interacting incomplete domain features. For example, GL would consider a subgoal that is established by two different actions, which are subject to disjoint faults, as having no faults. However, by using multiple-faults to explain failure to achieve the proposition, we see that the faults (at least one for each action) interact. Clearly, single-faults are important for identifying a single-point-of-failure, but ignoring multiple-faults could lead to an overly optimistic assessment of a plan. In the following, we generalize GL's notions of faults from singletons to sets, which we call fault diagnoses, or just diagnoses.

4.1 Model Based Diagnosis

In defining the diagnoses of plan failure, we draw upon many well established techniques in model based diagnosis (MBD) [Reiter, 1987; de Kleer and Williams, 1987]. Viewing the plan as a physical system, faults are sets of potentially faulty components that describe anomalous behavior, such as an action not having its preconditions satisfied or a goal not being achieved.

There are two terms from MBD that enable us describe which sets of faults may cause plan failure. The first term, a *conflict* [de Kleer and Williams, 1987], is a disjunction of faults where at least one must occur to explain the anomalous behavior. The second term, a *diagnosis*, is a conjunction of faults that must occur to explain the behavior. There may be multiple diagnoses and each diagnosis is a hypothesis explaining failure. Because of their respective disjunctive and conjunctive semantics, conflicts can be expressed by the prime implicants Σ of a logical sentence capturing knowledge of the faulty system, and diagnoses are the prime implicants of Σ [de Kleer *et al.*, 1992].

Reiter [1987] formulates MBD within a system that is defined by a system description SD and system components COMP, taking the respective form of first-order sentences and a finite set of constants. The system description includes a distinct unary predicate $AB(\cdot)$ that indicates abnormal behavior on the part of a system component. For example, the sentence $ANDG(x) \wedge \neg AB(x) \rightarrow out(x) = and(in1(x), in2(x))$ indicates that an And-gate that is not abnormal will have its output equal to the logical-and of its inputs. Along with the system description, OBS is an observation of the system’s behavior. For example, OBS may contain the facts $out(AND_1) = 0, in1(AND_1) = 1, in2(AND_1) = 1$, which is anomalous.

de Kleer *et al.* [1992] show that if $SD \wedge OBS \models AB(c_1) \vee \dots \vee AB(c_n)$ then $AB(c_1) \vee \dots \vee AB(c_n)$ is a conflict and c_1, \dots, c_n functioning normally does not explain OBS. For example, if $SD \wedge OBS \models AB(AND_1)$, then $AB(AND_1)$ is conflict. de Kleer *et al.* [1992] also show that if $AB(c_1) \vee \dots \vee AB(c_n)$ is a prime implicant (i.e., no disjunction of a subset of the literals is also a conflict), then it is a minimal conflict. If Σ is the conjunction of all minimal conflicts (prime implicants), then it is the case that for each diagnosis $AB(c_1) \wedge \dots \wedge AB(c_n)$, that $AB(c_1) \wedge \dots \wedge AB(c_n) \models \Sigma$. Each prime implicant of Σ is a minimal diagnosis (i.e., there is no conjunction of a subset of the diagnosis literals that is also a diagnosis). In our small example, $AB(AND_1)$ is the only conflict, making $AB(AND_1)$ the only diagnosis. In a more complex scenario where the minimal conflicts are $\Sigma = (AB(c_1) \vee AB(c_2)) \wedge (AB(c_1) \vee AB(c_3))$, the diagnoses are $AB(c_1)$ and $AB(c_2) \wedge AB(c_3)$.

4.2 Diagnosing Plan Faults in Incomplete Domains

We describe a plan $\tilde{\pi}$ with a set of Horn clauses $SD(\tilde{\pi})$ and replace the $AB(\cdot)$ predicates with propositions from F (describing the incomplete domain features). A diagnosis is then a conjunction of literals describing incomplete domain features; the plan will fail in each domain interpretation satisfying the diagnosis. We use the hypothetical observation $OBS = \neg \tilde{a}_n$ (i.e., the goal action is not executed) to signify plan failure. A conflict is a disjunction of literals $C = (f_i \vee \dots \vee \neg f_j \vee \dots)$ where each $f \in F$ and $SD(\tilde{\pi}) \wedge \neg \tilde{a}_n \models C$. The query $SD(\tilde{\pi}) \wedge \neg \tilde{a}_n \models C$ is equivalent to showing that $SD(\tilde{\pi}) \wedge \neg \tilde{a}_n \wedge \neg C$ is inconsistent, or that $SD(\tilde{\pi}) \wedge \neg C \models \tilde{a}_n$. As we will show, because system description is a set of Horn clauses and the query $SD(\tilde{\pi}) \wedge \neg C \models \tilde{a}_n$ can be proven by Horn inference.

The system description $SD(\tilde{\pi})$ consists of clauses that define the semantics of plans in incomplete domains, which includes conditions under which an action will have its preconditions satisfied and its effects will change the current state. This subsection i) presents the system description and maps it to the original definitions of plans for incomplete domains and ii) describes how an assumption-based truth maintenance system (ATMS) [de Kleer and Williams, 1987] can support more efficient diagnosis computation. In a following section, we make use of the intuitions developed here to motivate our forward-chaining state-space planner .

Plan System Description: The system description $SD(\tilde{\pi})$ is comprised of Horn clauses (rules) defined over time-stamped propositions denoting state propositions (p_t), plan actions (\tilde{a}_t), the absence of possible preconditions ($\neg \widetilde{pre}(\tilde{a}, p)$), the absence of possible delete effects ($\neg \widetilde{del}(\tilde{a}, p)$), and the presence of possible add effects ($\widetilde{add}(\tilde{a}, p)$).¹

Definition 4.1. *The system description $SD(\tilde{\pi})$ includes the following Horn clauses.*

¹We can replace $\neg \widetilde{del}(\tilde{a}, p)$ and $\neg \widetilde{pre}(\tilde{a}, p)$ by $\widetilde{ndel}(\tilde{a}, p)$ and $\widetilde{npre}(\tilde{a}, p)$ to emphasize how the system description is comprised of Horn clauses because these literals appear only negatively.

- i)
- ii) $\tilde{a}_{t-1} \wedge \bigwedge_{p \in \text{pre}(\tilde{a}_t)} p_t \wedge \bigwedge_{p \in \widetilde{\text{pre}}(\tilde{a}_t, p)} (p_t \vee \neg \widetilde{\text{pre}}(\tilde{a}_t, p)) \rightarrow \tilde{a}_t \quad t = 0 \dots n$
- iii) $\tilde{a}_t \rightarrow p_{t+1} \quad \text{for all } p \in \text{add}(\tilde{a}_t)$
- iv) $\tilde{a}_t \wedge \widetilde{\text{add}}(\tilde{a}_t, p) \rightarrow p_{t+1} \quad \text{for all } p \in \widetilde{\text{add}}(\tilde{a}_t)$
- v) $p_t \wedge \neg \widetilde{\text{del}}(\tilde{a}_t, p) \rightarrow p_{t+1} \quad \text{for all } p \in \widetilde{\text{del}}(\tilde{a}_t)$
- vi) $p_t \rightarrow p_{t+1} \quad \text{for all } p \in P \setminus (\text{del}(\tilde{a}_t) \cup \widetilde{\text{del}}(\tilde{a}_t))$

The rules can be understood as stating: i) the initial action is always executed, ii) actions require their preconditions to be satisfied *but also require the previous action to be successful*, iii) add effects are proven if the action is proven, iv) possible add effects are proven if the action is executed and the possible add effect is actually an add effect, v) propositions that are possibly deleted will in fact be true if they were previously true and they are in fact not deleted, and vi) all non-deleted propositions are true if they were previously true.

To show that the system description is in correspondence with our definition of the semantics of plans, we prove that for each interpretation of an incomplete domain F^i the plan successfully achieves the goal iff $\text{SD}(\tilde{\pi}) \cup \neg C^i \models a_n$, where the conflict is defined $C^i = \bigvee_{f \in F^i} \neg f \vee \bigvee_{f \notin F^i} f$.

Theorem 4.2. $\text{SD}(\tilde{\pi}) \cup \neg C^i \models a_n$ iff $\tilde{\pi}$ is a plan in interpretation D^i .

Proof. We first show that $\text{SD}(\tilde{\pi}) \cup \neg C^i \models a_n$ iff $\text{SD}^i(\tilde{\pi}) \models a_n$, where $\text{SD}^i(\tilde{\pi})$ is the system description of a complete planning domain corresponding to interpretation D^i , and then that $\text{SD}^i(\tilde{\pi}) \models a_n$ iff $\tilde{\pi}$ is a plan in interpretation D^i .

We define $\text{SD}^i(\tilde{\pi})$ as the simplified form of $\text{SD}(\tilde{\pi}) \cup \neg C^i$ whereby each rule in $\text{SD}^i(\tilde{\pi})$ corresponds to a rule in $\text{SD}(\tilde{\pi})$ and for each f ($\neg f$) if $\neg C^i \models f$ ($\neg C^i \models \neg f$) and f ($\neg f$) is a literal in the antecedent of a rule, then we replace it by \top , or if $\neg f$ (f) is a literal in the antecedent of a rule, then we replace it by \perp and perform Boolean simplification (e.g., $p \wedge \top \equiv p$, $p \wedge \perp \equiv \perp$, etc.) on the rule. Any rule where the antecedent becomes \perp is not added to $\text{SD}^i(\tilde{\pi})$. The system description $\text{SD}^i(\tilde{\pi})$ is equivalent to the system description for interpretation D^i where:

- i)
- ii) $\tilde{a}_{t-1} \wedge \bigwedge_{p \in \text{pre}(a_t^i)} p_t \rightarrow \tilde{a}_t \quad t = 0 \dots n$
- iii) $\tilde{a}_t \rightarrow p_{t+1} \quad \text{for all } p \in \text{add}(a_t^i)$
- iv) $p_t \rightarrow p_{t+1} \quad \text{for all } p \in P \setminus \text{del}(a_t^i)$

It is clear that $\text{SD}(\tilde{\pi}) \cup \neg C^i \models a_n$ iff $\text{SD}^i(\tilde{\pi}) \models a_n$ because $\text{SD}(\tilde{\pi}) \cup \neg C^i$ and $\text{SD}^i(\tilde{\pi})$ are logically equivalent – the latter was derived from the former by substituting logical true and false for the premises entailed by $\neg C^i$.

	$\text{SD}^i(\tilde{\pi}) \models \tilde{a}_n$	$\tilde{\pi}$ is a plan in interpretation D^i
Base Case	$\text{SD}^i(\tilde{\pi}) \models \tilde{a}_{-1}$	\tilde{a}_{-1} is the first action of $\tilde{\pi}$
Action Effects	If $\text{SD}^i(\tilde{\pi}) \models \tilde{a}_t$, and $P_t = \{p \mid \text{SD}^i(\tilde{\pi}) \models p\}$, then $P_{t+1} = \{p \mid p \in P_t \setminus \text{del}(a_t^i) \cup \text{add}(a_t^i)\}$	If \tilde{a}_t is applicable to s_t^i , then $s_{t+1}^i = s_t^i \setminus \text{del}(a_t^i) \cup \text{add}(a_t^i)$
Action Applicability	If $\text{SD}^i(\tilde{\pi}) \models \tilde{a}_{t-1} \wedge \bigwedge_{p \in \text{pre}(a_t^i)} p_t$ then $\text{SD}^i(\tilde{\pi}) \models \tilde{a}_t$	If \tilde{a}_{t-1} is applicable and $\text{pre}(a_t^i) \subseteq s_t^i$ then \tilde{a}_t is applicable to s_t^i .

To show that $SD^i(\tilde{\pi}) \models a_n$ iff $\tilde{\pi}$ is a plan in interpretation D^i , we list both sides of the proof in the table above. It should be clear that the base case holds in both directions. For the action effects, if we assume that an action proposition and a number of state propositions are derived (the action is applicable to a state), then we can derive the propositions denoted by P_{t+1} which is equivalent to s_{t+1}^i . For action applicability, if we assume that the prior action proposition and all precondition propositions are derived (the prior action was applicable and the current action has its preconditions satisfied), then the current action proposition can be derived (the current action is applicable).

Therefore, by simplifying the incomplete plan system description $SD(\tilde{\pi}) \cup \neg C^i$ to the complete plan system description $SD^i(\tilde{\pi})$ for an domain interpretation D^i , and showing its correspondence to the plan semantics of a plan in a domain interpretation, $SD(\tilde{\pi}) \cup \neg C^i \models a_n$ iff $\tilde{\pi}$ is a plan in interpretation D^i . \square

The system description of the example plan $(\tilde{a}, \tilde{b}, \tilde{c})$ from Section 2 is as follows:

$$\begin{array}{lll}
\tilde{a}_{-1} & & \tilde{b}_1 \wedge q_2 \wedge r_2 \rightarrow \tilde{c}_2 \\
\tilde{a}_{-1} \rightarrow p_0 & \tilde{a}_0 \wedge p_1 \rightarrow \tilde{b}_1 & \tilde{b}_1 \wedge \neg \widetilde{\text{pre}}(\tilde{c}, q) \wedge r_2 \rightarrow \tilde{c}_2 \\
\tilde{a}_{-1} \rightarrow q_0 & q_1 \wedge \widetilde{\text{del}}(\tilde{b}, q) \rightarrow q_2 & \tilde{c}_2 \rightarrow g_3 \\
\tilde{a}_{-1} \wedge p_0 \wedge q_0 \wedge r_0 \rightarrow \tilde{a}_0 & \tilde{b}_1 \rightarrow r_2 & q_2 \rightarrow q_3 \\
\tilde{a}_{-1} \wedge p_0 \wedge q_0 \wedge \neg \widetilde{\text{pre}}(\tilde{a}, r) \rightarrow \tilde{a}_0 & r_1 \rightarrow r_2 & r_2 \rightarrow r_3 \\
p_0 \wedge \widetilde{\text{del}}(\tilde{a}, p) \rightarrow p_1 & & \tilde{c}_2 \wedge g_3 \rightarrow \tilde{a}_3 \\
\tilde{a}_0 \wedge \widetilde{\text{add}}(\tilde{a}, r) \rightarrow r_1 & & \\
q_0 \rightarrow q_1 & &
\end{array}$$

We can simplify the system description for the domain interpretation D^1 where $F^1 = \{\widetilde{\text{add}}(\tilde{a}, r), \widetilde{\text{pre}}(\tilde{c}, q)\}$, making $C^1 = \widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee \neg \widetilde{\text{add}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{b}, q) \vee \neg \widetilde{\text{pre}}(\tilde{c}, q)$ to obtain $SD^1(\tilde{\pi})$:

$$\begin{array}{lll}
\tilde{a}_{-1} & & \tilde{b}_1 \wedge q_2 \wedge r_2 \rightarrow \tilde{c}_2 \\
\tilde{a}_{-1} \rightarrow p_0 & \tilde{a}_0 \wedge p_1 \rightarrow \tilde{b}_1 & \tilde{c}_2 \rightarrow g_3 \\
\tilde{a}_{-1} \rightarrow q_0 & q_1 \rightarrow q_2 & q_2 \rightarrow q_3 \\
\tilde{a}_{-1} \wedge p_0 \wedge q_0 \wedge r_0 \rightarrow \tilde{a}_0 & \tilde{b}_1 \rightarrow r_2 & r_2 \rightarrow r_3 \\
\tilde{a}_{-1} \wedge p_0 \wedge q_0 \rightarrow \tilde{a}_0 & r_1 \rightarrow r_2 & \tilde{c}_2 \wedge g_3 \rightarrow \tilde{a}_3 \\
p_0 \rightarrow p_1 & & \\
\tilde{a}_0 \rightarrow r_1 & & \\
q_0 \rightarrow q_1 & &
\end{array}$$

Consider the propositions derived from $SD^1(\tilde{\pi})$ and the state sequence $(s_0 = \{p, q\}, s_1 = \{p, q, r\}, s_2 = \{q, r\}, s_3 = \{q, r, g\})$ corresponding to the plan. The derived propositions are: $\{\tilde{a}_{-1}, p_0, q_0, \tilde{a}_0, p_1, q_1, r_1, b_1, q_2, r_2, \tilde{c}_2, q_3, r_3, g_3, \tilde{a}_3\}$. Each proposition in each state is in correspondence with a time-stamped proposition that is derived. Upon inspection, it is possible to see that both $SD(\tilde{\pi}) \cup \neg C^1 \models \tilde{a}_3$ and $SD^1(\tilde{\pi}) \models \tilde{a}_3$. Therefore C^1 is a conflict. However, as Reiter [1987] describes, only those AB(\cdot) literals (and in our case incomplete feature literals) that appear in a proof are needed in the conflict set; so that by inspecting the incomplete feature literals used in the proof trees for $SD(\tilde{\pi}) \cup \neg C^1 \models \tilde{a}_3$ shown in Figure 1, we find more succinct conflict sets:

$$\begin{aligned}
C^{1'} &= \widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee \widetilde{\text{del}}(\tilde{b}, q) \\
C^{1''} &= \widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee \neg \widetilde{\text{add}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{b}, q)
\end{aligned}$$

where the first subsumes the second. In other words, replacing C^1 with $C^{1'}$ would allow us to conclude $SD(\tilde{\pi}) \cup \neg C^{1'} \models \tilde{a}_3$ and derive a smaller conflict.

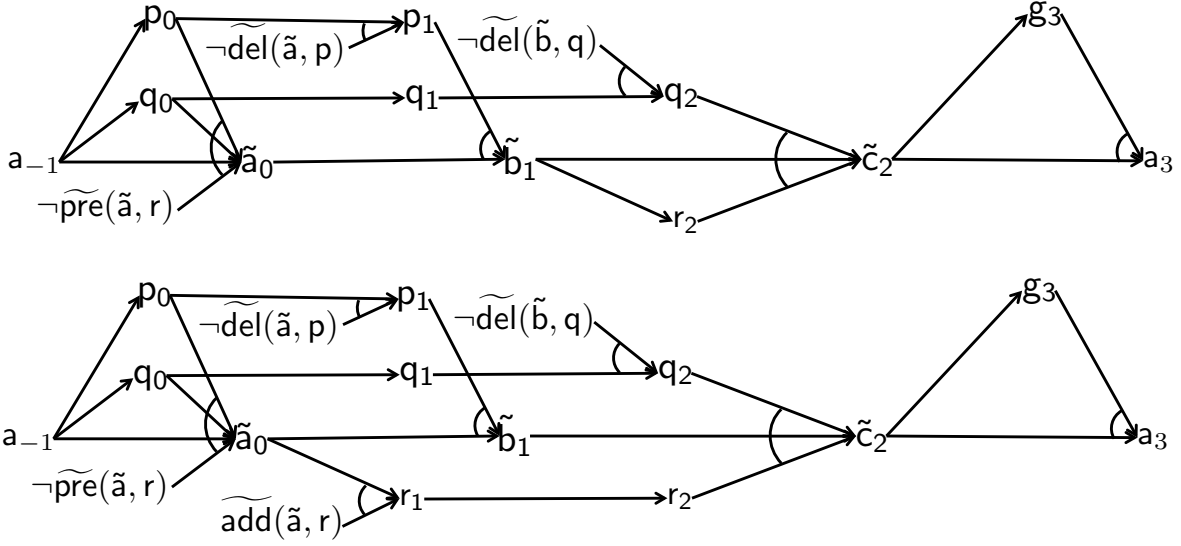


Figure 1: Proof trees for $SD(\tilde{\pi}) \cup \neg C^1 \models \tilde{a}_3$ of plan $\pi = (\tilde{a}, \tilde{b}, \tilde{c})$.

If we examine all subsets of the incomplete features $F^i \subseteq F$ where $SD(\tilde{\pi}) \cup \neg C^i \models \tilde{a}_3$, then we can determine the minimal conflicts. In our example, we can derive the following minimal conflicts:

$$\Sigma = (\widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee \widetilde{\text{del}}(\tilde{b}, q)) \wedge (\widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee \widetilde{\text{pre}}(\tilde{c}, q))$$

We determine the following diagnoses (each a prime implicant of Σ):

$$\begin{aligned} &\widetilde{\text{pre}}(\tilde{a}, r) \\ &\widetilde{\text{del}}(\tilde{a}, p) \\ &\widetilde{\text{del}}(\tilde{b}, q) \wedge \widetilde{\text{pre}}(\tilde{c}, q) \end{aligned}$$

The diagnoses are cases that will guarantee plan failure. The first action \tilde{a} can fail because it might require a precondition r that is not satisfied. The second action \tilde{b} can fail because its precondition p may be deleted by \tilde{a} . The third action \tilde{c} can fail if its possible precondition q is required and the second action \tilde{b} possibly deletes q .

Truth Maintenance Systems: The generate-and-test method of computing conflict sets involves selecting all possible sets $F^i \subseteq F$ and determining if $SD(\tilde{\pi}) \cup \neg C^i \models a_n$. An alternative is to employ an assumption-based truth maintenance system (ATMS) [de Kleer, 1986] so that we can simultaneously compute all possible proofs for all possible sets F^i . The approach is to assert each possible literal that might appear in any sentence $\neg C^i$ as an assumption, and label each by the assumptions under which it is derived. Then, by Modus Ponens, we apply the system description rules to derive and label new propositions. The assumptions labeling \tilde{a}_n indicate which are required to derive \tilde{a}_n , and consequently the conflict sets and diagnoses of plan failure.

An ATMS label for a proposition q that is derived from rules of the form $p_{i1} \wedge \dots \wedge p_{in} \rightarrow q$ is defined by:

$$l(q) = \bigvee_i \bigwedge_j l(p_{ij})$$

In the ATMS, we must label each premise and assumption. The only premise is the initial action \tilde{a}_{-1} , and the assumptions are the literals of F . The labels are defined:

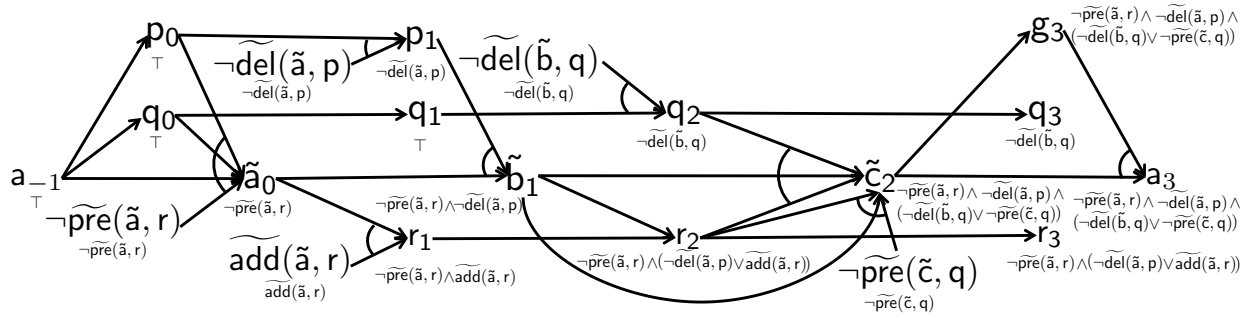


Figure 2: ATMS labels and proof trees for the system description of plan $(\tilde{a}, \tilde{b}, \tilde{c})$.

$$\begin{aligned}
l(a_{-1}) &= \top, \\
l(f) &= f \text{ for each } f \in F, \text{ and} \\
l(\neg f) &= \neg f \text{ for each } f \in F
\end{aligned}$$

The label of the initial action is \top (logical true) to denote that it is a premise. The label of each incomplete feature literal is the literal itself to denote that it can be derived in contexts where it is given. All other propositions are proven by one or more rules, and we define their labels as described above.

Figure 2 depicts the labels associated with each literal by the propositional sentence underneath the literal. Consider the label $l(\tilde{c}_2)$, that is proven by two clauses, $\tilde{b}_1, q_2, r_2 \rightarrow \tilde{c}_2$ and $\tilde{b}_1, \neg \widetilde{\text{pre}}(\tilde{c}, q), r_2 \rightarrow \tilde{c}_2$. The label for each of the antecedents of the clauses are as follows:

$$\begin{aligned}
l(\tilde{b}_1) &= \neg \widetilde{\text{pre}}(\tilde{a}, r) \wedge \neg \widetilde{\text{del}}(\tilde{a}, p) \\
l(q_2) &= \neg \widetilde{\text{del}}(\tilde{b}, q) \\
l(r_2) &= \neg \widetilde{\text{pre}}(\tilde{a}, r) \wedge (\neg \widetilde{\text{del}}(\tilde{a}, p) \vee \widetilde{\text{add}}(\tilde{a}, r)) \\
l(\neg \widetilde{\text{pre}}(\tilde{c}, q)) &= \neg \widetilde{\text{pre}}(\tilde{c}, q)
\end{aligned}$$

allowing us to compute

$$\begin{aligned}
l(\tilde{c}_2) &= (l(\tilde{b}_1) \wedge l(q_2) \wedge l(r_2)) \vee (l(\tilde{b}_1) \wedge l(\neg \widetilde{\text{pre}}(\tilde{c}, q)) \wedge l(r_2)) \\
&= \neg \widetilde{\text{pre}}(\tilde{a}, r) \wedge \neg \widetilde{\text{del}}(\tilde{a}, p) \wedge (\neg \widetilde{\text{del}}(\tilde{b}, q) \vee \neg \widetilde{\text{pre}}(\tilde{c}, q))
\end{aligned}$$

If for some proposition q it is the case that $\neg C^i \models l(q)$, it is the same as saying that $\text{SD}(\tilde{\pi}) \cup \neg C^i \models q$ or that $\neg l(q) \models C^i$, meaning that C^i is a conflict for q . If Σ is the conjunction of all conflicts, then it is also the case that $\neg l(q) \models \Sigma$. If Σ is the conjunction of all implicates of $\neg l(q)$, then the two are also logically equivalent. Therefore, if a minimal diagnosis is a prime implicant of Σ it is also a prime implicant of $\neg l(q)$.

By inspecting the negated goal action label $\neg l(a_3) = \widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee (\widetilde{\text{del}}(\tilde{b}, q) \wedge \widetilde{\text{pre}}(\tilde{c}, q))$, we see that there are three conjunctive sentences that entail it: $\widetilde{\text{pre}}(\tilde{a}, r)$, $\widetilde{\text{del}}(\tilde{a}, p)$, and $\widetilde{\text{del}}(\tilde{b}, q) \wedge \widetilde{\text{pre}}(\tilde{c}, q)$, which correspond to the previously stated diagnoses. By counting the models of the negated label $\neg l(\tilde{a}_3)$, of the goal action, it is possible to determine how many interpretations of the incomplete domain will fail to achieve the goal with the plan. In this example, there are thirty-two interpretations, and twenty-six will fail to achieve the goal.

4.3 Counting Models and Diagnoses

The ATMS labels computed in the previous section identify those combinations of incomplete domain features that permit a plan to satisfy the goals (i.e., the models of the labels correspond to interpretations of the incomplete domain that will succeed). From the labels, it is possible to compute exactly how many interpretations of the incomplete domain features lead to a successful or unsuccessful plan. Thus, counting

the number of domains that will not successfully achieve the goals can be reduced to counting the models of the negated goal action label $\neg l(\tilde{a}_n)$ (a propositional sentence). The planner described in the next section is based on the idea of using an ATMS to represent plans, and many of its subroutines involve comparing propositional sentences (e.g., it compares and selects actions in its relaxed plans that will achieve a proposition in more domain interpretations). In comparing a propositional sentence ϕ with another, we will refer to its set of models $M(\phi)$, its set of prime implicants $PI(\phi)$, and its sets of k -cardinality prime implicants $PI_k(\phi)$.

Counting models requires polynomial time when a propositional sentence is represented by an OBDD, and it requires exponential time when represented by prime implicants. However, we note that the number of prime implicants can be indicative of the number of models, and simply counting the number of prime implicants is a heuristic.

From the example in the previous section, the disjunction of the three prime implicants: $\widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee (\widetilde{\text{del}}(\tilde{b}, q) \wedge \widetilde{\text{pre}}(\tilde{c}, q))$ has twenty-six models, whereas the disjunction of the two prime implicants $\widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p)$ has twenty-four models – the number of prime implicants, in this case, is proportional to the number of models, 3:26 and 2:24. While the relationship between prime implicants and models does not hold in general, we can use the number of prime implicants to heuristically compare propositional sentences (estimating the number of models).

Another observation is that having fewer prime implicants of smaller cardinality can result in fewer models. For example, both $\widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p)$ and $\widetilde{\text{pre}}(\tilde{a}, r) \vee (\widetilde{\text{del}}(\tilde{b}, q) \wedge \widetilde{\text{pre}}(\tilde{c}, q))$ have two prime implicants, but the former has twenty-four models and the latter has twenty models. Thus, when comparing two propositional sentences, we can compare $|PI_1(\phi)|$ and $|PI_1(\psi)|$, and if equal, compare $|PI_2(\phi)|$ and $|PI_2(\psi)|$, and so on, until $|PI_k(\phi)| \neq |PI_k(\psi)|$ for some $k > 0$; if k is the minimum cardinality where $|PI_k(\phi)| < |PI_k(\psi)|$, then we prefer ϕ (assuming ϕ represents interpretations of incomplete actions where a plan fails). Thus, we define two preference relations on propositional formulas representing plan failure:

- Model-based: $\phi \prec_M \psi$ if $|M(\phi)| < |M(\psi)|$
- Diagnosis-based: $\phi \prec_{PI} \psi$ if $|PI_k(\phi)| < |PI_k(\psi)|, k > 0$, and $|PI_j(\phi)| = |PI_j(\psi)|$ for all $j < k$.

In the following, we dispense with the notation for preference relations, assuming that the context dictates whether the propositional sentences are compared by models or diagnoses.

Comparing the prime-implicants is much less expensive than counting and comparing the number of models, but this heuristic may be wrong. Nevertheless, we empirically compare counting OBDD models to counting prime implicants within our planner, and demonstrate improvements in scalability when counting prime implicants. Throughout our discussion, when we refer to counting models of ϕ , we assume that ϕ is represented by an OBDD, and when we refer to counting the prime implicants of ϕ , we assume that ϕ is already represented by prime implicants (i.e., we assume the representation that is most natural for the type of counting in order to ignore any additional cost of normal form conversion).

5 Forward State Space Planning With Faults

We present a forward state space planner called `DeFAULT` that uses the approaches developed in the previous sections to search for plans that have few faults or few interpretations of the incomplete domain features that result in plan failure. Recall that having few faults and few failed interpretations are connected but rely on counting different quantities (prime implicants or models). We employ the optimistic semantics for incomplete domain features and extend our state description with failure explanations to capture which

incomplete domain features can cause failure to achieve each state proposition. We note that computing and representing the prime implicants can be costly; we address this by formulating our approach for any arbitrary, but fixed, bound on the prime implicant cardinality. While the cardinality of each prime implicant is bounded, the number of prime implicants per proposition is indirectly bounded (i.e., there is a finite number of sets with cardinality k or less). The impact of bounding the prime implicant cardinality is that we may under-approximate the number of interpretations of the incomplete domain in which the plan will fail.

We adapt the ATMS label propagation for a plan's system description to the plan's state space representation. We propagate the equivalent of negated labels with the rules described below to capture failure explanations as opposed to success explanations. The negated labels provide us a measure of parsimony within the planner because the planner prefers plans with smaller length and a smaller number of models or prime implicants (i.e., the planner attempts to minimize both plan quality metrics by breaking plan length ties with model or prime implicant counting). As we show, the rules for propagating failure explanations over the plan are derived from the ATMS label semantics.

Fault Propagation: We previously described how to recursively define the failure explanation (negated ATMS label) for a goal proposition; the propositional models of the label reflect which interpretations succeed in achieving the goal. In the following, we discuss rules for the forward propagation of failure explanations to compliment our forward state-space planner.

Initially, we use the explanation $d(\tilde{a}_{-1}) = \perp$ to denote that there are no failures affecting the initial state action and for all $p \in P$, $d(p_{-1}) = \top$ (we assume that the initial action is applied to a state where all propositions are false, meaning their failure explanations indicate they are false in all interpretations). For all times $0 \leq t \leq n$, we define:

$$d(\tilde{a}_t) = d(\tilde{a}_{t-1}) \vee \bigvee_{p \in \text{pre}(\tilde{a}_t)} d(p_t) \vee \bigvee_{p \in \widetilde{\text{pre}}(\tilde{a}_t)} (d(p_t) \wedge \widetilde{\text{pre}}(\tilde{a}_t, p))$$

$$d(p_{t+1}) = \begin{cases} d(p_t) \wedge d(\tilde{a}_t) & : p \in \text{add}(\tilde{a}_t) \\ d(p_t) \wedge (d(\tilde{a}_t) \vee \widetilde{\text{add}}(\tilde{a}_t, p)) & : p \in \widetilde{\text{add}}(\tilde{a}_t) \\ \top & : p \in \text{del}(\tilde{a}_t) \\ d(p_t) \vee \widetilde{\text{del}}(\tilde{a}_t, p) & : p \in \widetilde{\text{del}}(\tilde{a}_t) \\ d(p_t) & : \text{otherwise} \end{cases}$$

where the failure explanation of an action $d(\tilde{a}_t)$ captures cases where the prior action failed or one if its preconditions is not satisfied, and the failure explanation of a proposition $d(p_{t+1})$ captures cases where the previous action's effects can cause failure to achieve the proposition.

The following theorem relates the failure explanations to the ATMS labels for a plan's system description.

Theorem 5.1. $d(p_{t+1}) = \neg l(p_{t+1})$ and $d(\tilde{a}_t) = \neg l(\tilde{a}_t)$ for $t = -1, \dots, n$.

Proof. By definition $d(\tilde{a}_{-1}) = \neg l(\tilde{a}_{-1}) = \perp$. If we assume that $d_t(p) = \neg l(p_t)$ for all $p \in s_t$ and $d(\tilde{a}_{t-1}) = \neg l(\tilde{a}_{t-1})$, then

$$\begin{aligned}
d(\tilde{a}_t) &= d(\tilde{a}_{t-1}) \vee \bigvee_{p \in \text{pre}(\tilde{a}_{t-1})} d(p_t) \vee \bigvee_{p \in \widetilde{\text{pre}}(\tilde{a})} (d(p_t) \wedge \widetilde{\text{pre}}(\tilde{a}_t, p)) \\
&= \neg l(\tilde{a}_{t-1}) \vee \bigvee_{p \in \text{pre}(\tilde{a})} \neg l(\mathbf{p}_t) \vee \bigvee_{p \in \widetilde{\text{pre}}(\tilde{a})} (\neg l(\mathbf{p}_t) \wedge \widetilde{\text{pre}}(\tilde{a}_t, p)) \\
&= \neg \left(l(\tilde{a}_{t-1}) \wedge \bigwedge_{p \in \text{pre}(\tilde{a})} l(\mathbf{p}_t) \wedge \bigwedge_{p \in \widetilde{\text{pre}}(\tilde{a})} (l(\mathbf{p}_t) \vee \neg \widetilde{\text{pre}}(\tilde{a}_t, p)) \right) \\
&= \neg l(\tilde{a}_t)
\end{aligned}$$

and

$$\begin{aligned}
d(p_{t+1}) &= \begin{cases} d(p_t) \wedge d_t(\tilde{a}_t) & : p \in \text{add}(\tilde{a}_t) \\ d(p_t) \wedge (d(\tilde{a}_t) \vee \neg \widetilde{\text{add}}(\tilde{a}_t, p)) & : p \in \widetilde{\text{add}}(\tilde{a}_t) \\ \top & : p \in \text{del}(\tilde{a}_t) \\ d(p_t) \vee \widetilde{\text{del}}(\tilde{a}_t, p) & : p \in \widetilde{\text{del}}(\tilde{a}_t) \\ d(p_t) & : \text{otherwise} \end{cases} \\
&= \begin{cases} \neg l(\mathbf{p}_t) \wedge \neg l(\mathbf{a}_t) & : p \in \text{add}(\tilde{a}_t) \\ \neg l(\mathbf{p}_t) \wedge (\neg l(\mathbf{a}_t) \vee \neg \widetilde{\text{add}}(\tilde{a}_t, p)) & : p \in \widetilde{\text{add}}(\tilde{a}_t) \\ \top & : p \in \text{del}(\tilde{a}_t) \\ \neg l(\mathbf{p}_t) \vee \widetilde{\text{del}}(\tilde{a}_t, p) & : p \in \widetilde{\text{del}}(\tilde{a}_t) \\ \neg l(\mathbf{p}_t) & : \text{otherwise} \end{cases} \\
&= \begin{cases} \neg(l(\mathbf{p}_t) \vee l(\mathbf{a}_t)) & : p \in \text{add}(\tilde{a}_t) \\ \neg(l(\mathbf{p}_t) \vee (l(\mathbf{a}_t) \wedge \widetilde{\text{add}}(\tilde{a}_t, p))) & : p \in \widetilde{\text{add}}(\tilde{a}_t) \\ \neg(\perp) & : p \in \text{del}(\tilde{a}_t) \\ \neg(l(\mathbf{p}_t) \wedge \neg \widetilde{\text{del}}(\tilde{a}_t, p)) & : p \in \widetilde{\text{del}}(\tilde{a}_t) \\ \neg(l(\mathbf{p}_t)) & : \text{otherwise} \end{cases} \\
&= \begin{cases} \neg l(\mathbf{p}_{t+1}) : p \in \text{add}(\tilde{a}_t) \\ \neg l(\mathbf{p}_{t+1}) : p \in \widetilde{\text{add}}(\tilde{a}_t) \\ \neg l(\mathbf{p}_{t+1}) : p \in \text{del}(\tilde{a}_t) \\ \neg l(\mathbf{p}_{t+1}) : p \in \widetilde{\text{del}}(\tilde{a}_t) \\ \neg l(\mathbf{p}_{t+1}) : \text{otherwise} \end{cases} \\
&= \neg l(\mathbf{p}_{t+1}).
\end{aligned}$$

□

To count the number of interpretations under which a plan fails, we count the models of, $d(\tilde{\pi}) = d(\tilde{a}_n)$, which expresses the interpretations where any of the plan actions did not have its preconditions satisfied or the goal was not satisfied.

Consider the fault propagation required for our example plan $(\tilde{a}, \tilde{b}, \tilde{c})$. Initially, the explanation for the initial action is $d(\tilde{a}_{-1}) = \perp$, and state $s_0 = \{p, q\}$ is labeled as follows:

$$\begin{aligned}
d(p_0) &= \perp \\
d(q_0) &= \perp \\
d(r_0) &= \top \\
d(g_0) &= \top
\end{aligned}$$

After applying \tilde{a} to s_0 , we attain the state $s_1 = \{p, q, r\}$ with the following explanations:

$$\begin{aligned}
d(\tilde{a}_0) &= d(p_0) \vee d(q_0) \vee (d(r_0) \wedge \widetilde{\text{pre}}(\tilde{a}, r)) \\
&= \perp \vee \perp \vee (\top \wedge \widetilde{\text{pre}}(\tilde{a}, r)) \\
&= \widetilde{\text{pre}}(\tilde{a}, r) \\
d(p_1) &= d(p_0) \vee \widetilde{\text{del}}(\tilde{a}, p) \\
&= \perp \vee \widetilde{\text{del}}(\tilde{a}, p) \\
&= \widetilde{\text{del}}(\tilde{a}, p) \\
d(q_1) &= d(q_0) = \perp \\
d(r_1) &= d(r_0) \wedge (d(\tilde{a}_0) \vee \neg \widetilde{\text{add}}(\tilde{a}, r)) \\
&= \top \wedge (\widetilde{\text{pre}}(\tilde{a}, r) \vee \neg \widetilde{\text{add}}(\tilde{a}, r)) \\
&= \widetilde{\text{pre}}(\tilde{a}, r) \vee \neg \widetilde{\text{add}}(\tilde{a}, r) \\
d(g_1) &= d(g_0) = \top
\end{aligned}$$

Applying \tilde{b} to s_1 results in the state $s_2 = \{q, r\}$, with the explanations:

$$\begin{aligned}
d(\tilde{b}_1) &= d(\tilde{a}_0) \vee d(p_1) \\
&= \widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \\
d(p_2) &= \top \\
d(q_2) &= d(q_1) \vee \widetilde{\text{del}}(\tilde{b}, q) \\
&= \perp \vee \widetilde{\text{del}}(\tilde{b}, q) \\
&= \widetilde{\text{del}}(\tilde{b}, q) \\
d(r_2) &= d(r_1) \wedge d(\tilde{b}_1) \\
&= (\widetilde{\text{pre}}(\tilde{a}, r) \vee \neg \widetilde{\text{add}}(\tilde{a}, r)) \wedge (\widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p)) \\
&= \widetilde{\text{pre}}(\tilde{a}, r) \vee (\neg \widetilde{\text{add}}(\tilde{a}, r) \wedge \widetilde{\text{del}}(\tilde{a}, p)) \\
d(g_2) &= d(g_1) = \top
\end{aligned}$$

Finally, after applying \tilde{c} to s_2 , we compute $s_3 = \{q, r, g\}$ and the explanations:

$$\begin{aligned}
d(\tilde{c}_2) &= d(\tilde{b}_1) \vee d(r_2) \vee (d(q_2) \wedge \widetilde{\text{pre}}(\tilde{c}, q)) \\
&= (\widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p)) \vee (\widetilde{\text{pre}}(\tilde{a}, r) \vee (\neg \widetilde{\text{add}}(\tilde{a}, r) \wedge \widetilde{\text{del}}(\tilde{a}, p))) \vee ((\widetilde{\text{del}}(\tilde{b}, q)) \wedge \widetilde{\text{pre}}(\tilde{c}, q)) \\
&= \widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee (\widetilde{\text{del}}(\tilde{b}, q) \wedge \widetilde{\text{pre}}(\tilde{c}, q)) \\
d(p_3) &= d(p_2) = \top \\
d(q_3) &= d(q_2) = \widetilde{\text{del}}(\tilde{b}, q) \\
d(r_3) &= d(r_2) = \widetilde{\text{pre}}(\tilde{a}, r) \vee (\neg \widetilde{\text{add}}(\tilde{a}, r) \wedge \widetilde{\text{del}}(\tilde{a}, p)) \\
d(g_3) &= d(g_2) \wedge d(\tilde{c}_2) \\
&= \widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee (\widetilde{\text{del}}(\tilde{b}, q) \wedge \widetilde{\text{pre}}(\tilde{c}, q))
\end{aligned}$$

The plan results in the following failure diagnoses:

$$d(\tilde{\pi}) = d(\tilde{a}_3) = d(g_3) \vee d(\tilde{c}_2) = \widetilde{\text{pre}}(\tilde{a}, r) \vee \widetilde{\text{del}}(\tilde{a}, p) \vee (\widetilde{\text{del}}(\tilde{b}, q) \wedge \widetilde{\text{pre}}(\tilde{c}, q))$$

Forward State-Space Planning: DeFAULT is a forward state-space planner that is based on Downward [Helmert, 2006], and its greedy best first search algorithm. DeFAULT compares partial plans only in terms of their heuristic value (described in the next section). While DeFAULT does not compare the faults introduced by plan prefixes leading to states on the fringe of the search, these faults seed the relaxed planning problem that is used in the heuristic computation.

6 Incomplete Domain Heuristics

Similar to propagating failure explanations in a plan, we can propagate explanations in the relaxed planning problem to compute a search heuristic. The heuristic is the number of actions in a relaxed plan, and, while we do not use the number of failed domain interpretations as the primary heuristic, we use the failure labels to bias the selection of the relaxed plan actions and break ties between search nodes with an equivalent number of actions in their relaxed plans. We solve the relaxed planning problem using a planning graph and we start with a brief description of planning graphs in STRIPS domains.

Planning Graph Heuristics: A relaxed planning graph is a layered graph with sets of vertices $(\mathcal{P}_t, \mathcal{A}_t, \dots, \mathcal{A}_{t+m}, \mathcal{P}_{t+m+1})$. The planning graph built for a state s_t defines $\mathcal{P}_t = \{p_t | p \in s_t\}$, $\mathcal{A}_{t+k} = \{a_t | \forall p \in \text{pre}(a) p_t \in \mathcal{P}_{t+k}, a \in A \cup A(P)\}$, and $\mathcal{P}_{t+k+1} = \{p_{t+k+1} | a_{t+k} \in \mathcal{A}_{t+k}, p \in \text{add}(a)\}$, for $k = 0, \dots, m$. The set $A(P)$ includes noop actions for each proposition, such that $A(P) = \{a(p) | p \in P, \text{pre}(a(p)) = \text{add}(a(p)) = p, \text{del}(a(p)) = \emptyset\}$. The h^{FF} heuristic [Hoffmann and Nebel, 2001] solves this relaxed planning problem by choosing actions from \mathcal{A}_{t+m} to support the goals in \mathcal{P}_{t+m+1} , and recursively for each chosen action's preconditions, counting the number of chosen actions.

Incomplete Domain Planning Graphs: Propagating failure explanations in the planning graph resembles propagating failure explanations within a plan. The primary difference is how we define the failed interpretations for a proposition when the proposition has multiple sources of support; recall that we allow only serial plans and at each time each state proposition is supported by persistence and/or a single action – action choice is handled in the search space. In a level of the relaxed planning graph, there are potentially many actions supporting a proposition, and we select the supporter with the fewest failed interpretations (requiring us to count models or prime implicants). The chosen supporting action, denoted $a_{t+k}(p)$, determines the failed interpretations affecting a proposition p at level $t + k + 1$.

A relaxed planning graph with propagated labels is a layered graph of sets of vertices of the form $(\mathcal{P}_t, \mathcal{A}_t, \dots, \mathcal{A}_{t+m}, \mathcal{P}_{t+m+1})$. The relaxed planning graph built for a state \tilde{s}_t defines $\mathcal{P}_0 = \{p_t | p \in \tilde{s}_t\}$, $\mathcal{A}_{t+k} = \{a_{t+k} | \forall p \in \text{pre}(\tilde{a}) p_{t+k} \in \mathcal{P}_{t+k}, \tilde{a} \in \tilde{A} \cup A(P)\}$, and $\mathcal{P}_{t+k+1} = \{p_{t+k+1} | a_{t+k} \in \mathcal{A}_{t+k}, p \in \text{add}(\tilde{a}) \cup \text{add}(\tilde{a})\}$, for $k = 0, \dots, m$. Much like the successor function used to compute next states, the relaxed planning graph assumes an optimistic semantics for action effects by adding possible add effects to proposition layers, but, as we will explain below, it associates failure explanations with the possible adds.

Each planning graph vertex has an explanation, denoted $d(\cdot)$. The failed interpretations $d(p_t)$ affecting a proposition are defined by the current state, and for $k \geq 0$,

$$d(\tilde{a}_{t+k}) = \bigvee_{p \in \text{pre}(\tilde{a})} d(p_{t+k}) \vee \bigvee_{p \in \widetilde{\text{pre}}(\tilde{a})} (d(p_{t+k}) \wedge \widetilde{\text{pre}}(\tilde{a}, p))$$

$$d(p_{t+k+1}) = \begin{cases} d(a_{t+k}(p)) & : p \in \text{add}(a_{t+k}(p)) \\ d(a_{t+k}(p)) \vee \neg \widetilde{\text{add}}(a_{t+k}(p), p) & : p \in \widetilde{\text{add}}(a_{t+k}(p)) \end{cases}$$

Every action in every level k of the planning graph will fail in any interpretation where their preconditions are not supported. A proposition will fail to be achieved in any interpretation where the chosen supporting action fails to add the proposition.

We note that the rules for propagating failures in the planning graph differ from the rules for propagating failures in the state space. In the state space, the action failures include interpretations where any prior action fails. In the relaxed planning problem, the action failure labels include only interpretations affecting the

action’s preconditions, and not prior actions; it is not clear which actions will be executed prior to achieving a proposition because many actions may be used to achieve other propositions at the same time step.

Heuristic Computation: We terminate the relaxed planning graph expansion at the level $t + k + 1$ where one of the following conditions is met: i) the planning graph reaches a fixed point where the explanations do not change, $d(p_{t+k}) = d(p_{t+k+1})$ for all $p \in P$, or ii) the goals have been reached at $t + k + 1$ and the fixed point has not yet been reached. Our $h^{\sim FF}$ heuristic makes use of the chosen supporting action $a_{t+k}(p)$ for each proposition that requires support in the relaxed plan, and, hence, measures the number of actions used while attempting to minimize failed interpretations (the supporting actions are chosen by comparing failure explanations). The other heuristic $h^{\sim M}$ measures the number of interpretations that fail to reach the goals in the last level (i.e., such that $h^{\sim M} = |M(\bigvee_{p \in G} d(p_{t+m+1}))|$, where $m + 1$ is the last level of the planning graph. DeFAULT uses both heuristics, treating $h^{\sim FF}$ as the primary heuristic and using $h^{\sim M}$ to break ties. While it is likely that swapping the role of the heuristics may lead to better quality plans (fewer failed interpretations), our informal experiments determined that the scalability of DeFAULT is greatly limited in such cases; measuring failed interpretations is not correlated with solution depth in the search graph unlike relaxed plan length. The relaxed plans are informed by the propagated explanations because we use the model count or prime implicant count to bias action selection.

7 Empirical Evaluation

The empirical evaluation is divided into three sections: the domains used for the experiments, the test setup used, and a discussion of the results. We use five configurations of the planner: DeFAULT-*FF*, DeFAULT-*PIk* ($k = 1, 2, 3$), and DeFAULT-*BDD*, that differ in how they reason about domain incompleteness (we refer to each configuration DeFAULT-*X* as *X*). *FF* is the control that does not compute failure explanations and uses the FF heuristic; it is likely to find a plan that will work for only the most optimistic domain interpretation. *PIk*, where k is the bound on the cardinality of the prime implicants, uses only prime implicants to compare failure explanations. *BDD* uses OBDDs to represent and count failure explanations. We omit a thorough comparison with the POND planner using the translated CPP instances because we were not able to draw any meaningful conclusions due to POND’s scalability; however, we do report the number of instances that POND solved. (We also attempted, but do not compare with the PFF [Domshlak and Hoffmann, 2006] planner because of several unresolved stability and implementation issues with the planner.)

The questions that we would like to answer with our evaluation include:

- Q1: Will translating incomplete STRIPS problems to CPP problems and using an off-the-shelf CPP planner scale?
- Q2: Can a classical planner (that ignores action incompleteness) find reasonable quality solutions in incomplete domains?
- Q3: In what cases does a planner that counts models fail to scale?
- Q4: Can a planner that counts prime implicants scale well and find high quality solutions?
- Q5: How does bounding the size of prime implicants affect plan quality and scalability?

7.1 Domains

There are four domains that we use in the evaluation: a modified Pathways, Bridges, a modified PARC Printer, and Barter World. In all domains, we derived multiple instances by randomly (with probabilities 0.25, 0.5, 0.75, and 1.0) injecting incomplete domain features. The manner by which the features were injected varies by domain, as we describe below; we injected incomplete features that made sense for each domain, rather than modifying the domains in a uniform manner. For each probability of injecting incomplete features (except 1.0) and each instance, we derived ten instances with different random seeds and present the results for each seed. The problem instances generators and DeFAULT planner are available at <http://www.cs.usu.edu/~danbryce/supplemental/default.jar>.

The Pathways domain from the international planning competition involves actions that model chemical reactions in signal transduction pathways. Pathways is a naturally incomplete domain where the lack of knowledge of the reactions is quite common because they are an active research topic in biology. We introduced each type of incompleteness to model incomplete knowledge of products required, created, or destroyed by reactions.

The Bridges domains consist of a traversable grid and the task is to find a different treasure at each corner of the grid. There are three versions where each subsequent version has an additional type of incompleteness. In Bridges1, a bridge might be required to cross between some grid locations, modeled as an incomplete precondition. In Bridges2, many of the bridges may have a troll living underneath that will take all the treasure accumulated, and are modeled by a possible delete effect. In Bridges3, some of the corners may give additional treasures, modeled as a possible add effect. The instances involve different size grids (2, 4, 8, 16, and 32)

The PARC Printer domain from the international planning competition involves planning paths for sheets of paper through a modular printer. A source of domain incompleteness is that a module accepts only certain paper sizes, but its documentation is incomplete. Thus, for such modules a possible delete effect models that the module will become jammed.

The Barter World domain involves navigating a grid and bartering items to travel between locations. Items are available at different locations and may be required to travel between other locations. The domain is incomplete because some of the actions that acquire certain items are not always known to be successful (possible add effects) and traveling between some locations may require certain items (possible preconditions) and may result in the loss of an item (possible delete). The instances involve different size grids (2, 4, 8, 16, 32, and 64) and types of items (1, 2, and 4).

7.2 Test Setup

The tests were run on a machine running Linux with a 3 Ghz Xeon processor, a memory limit of 2GB, and a time limit of 20 minutes per run. All DeFAULT code is written in Java and run on the 1.6 JVM. All versions of DeFAULT share the same greedy best first search implementation that uses deferred heuristic evaluation and a dual-queue for preferred and non-preferred operators [Helmert, 2006]. All versions also use the same planning graph implementation based on STAN [Long and Fox, 1999]. The number of failed interpretations for a plan $\tilde{\pi}$ found by any of the planners is reported herein by counting models of an OBDD representing $d(\tilde{\pi})$. The versions of the planner are compared by the number of interpretations of the incomplete domain that achieve the goal and total planning time in seconds. We also report detailed results on the number of solved instances per domain. Each configuration of DeFAULT returns the first solution that it generates, and POND is required to find solutions with the minimum probability of goal satisfaction τ equal to the minimum proportion of successful domain interpretations among plans found by DeFAULT. POND uses a

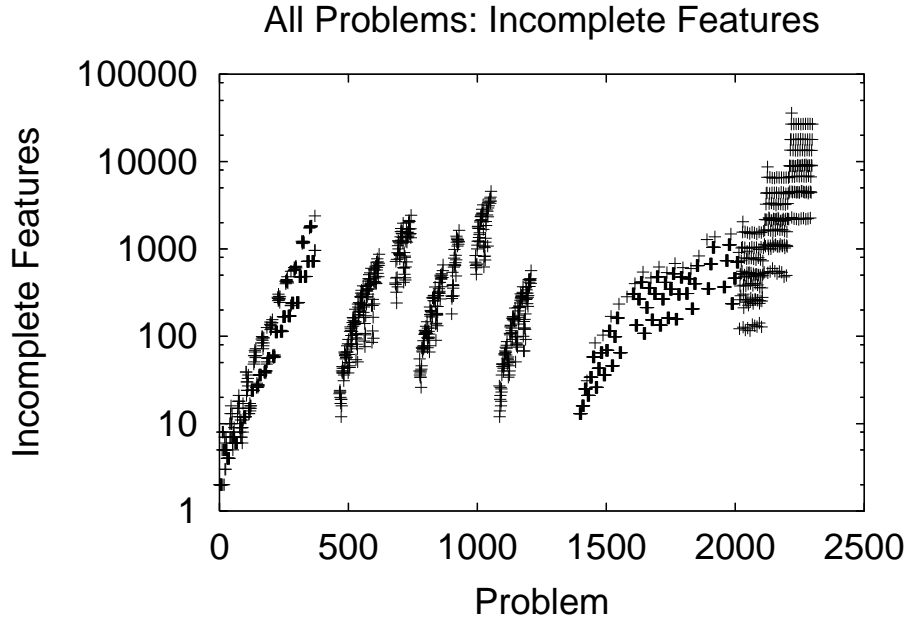


Figure 3: Incomplete Features By Instance.

Monte-Carlo based heuristic and we used ten particles per heuristic computation, as is representative of its best performing settings [Bryce *et al.*, 2008].

7.3 Results

We first present results across all problem instances and domains to assess the overall performance of each version of `DeFAULT`, and then discuss the performance within each domain to understand how the structure of the various domains affects performance. Figures 2 to 6 show results that are relevant to every domain, and the following figures detail each domain.

Overall Comparison: Figure 3 depicts the number of incomplete features for each problem instance. Each point represents an instance that was solved by at least one of the planner configurations. The first cluster of points (between 1 and 500) are from the Bridges domain. The next three clusters (between 500 and 1300) are from the PARC Printer domain. The next cluster (between 1300 and 2000) is from the Pathways domain. The final cluster (between 2000 and 2500) is from the Barter World domain. The number of incomplete features varies from less than 10 to over 25,000, and are a product of increasing the probability of injecting incomplete features and problem size.

Table 2 lists the number of solved instances by domain for each of the `DeFAULT` configurations and `POND`. We delineate the `FF` configuration at the left to indicate that it sets the highest standard for performance as our control, and among the four configurations that reason about incompleteness we bold the maximum number of instances solved for each row. We also delineate `POND` on the right. Among the techniques reasoning about incompleteness, `PI2` solves the most problems overall, but is relatively close to both of the other `PI` configurations. The `BDD` configuration solves the most problems in the Bridges domain, but only by a small margin relative to the other domains. We also see that `PI1` solves the second-most problems and solves significantly more in the Pathways domain. `POND` does not solve a competitive

Domain	FF	PI1	PI2	PI3	BDD	POND
PARCprinter 0.25	130	83	85	86	80	10
PARCprinter 0.5	130	87	88	87	80	0
PARCprinter 0.75	130	82	83	81	80	0
PARCprinter 1.0	13	10	9	9	8	0
Parcprinter	403	262	265	263	248	10
Bridges1 0.25	33	19	19	19	19	2
Bridges1 0.5	33	15	15	15	15	2
Bridges1 0.75	32	18	17	17	18	2
Bridges1 1.0	4	2	2	1	2	1
Bridges2 0.25	29	15	16	16	16	3
Bridges2 0.5	31	16	12	12	19	3
Bridges2 0.75	31	13	14	15	16	2
Bridges2 1.0	4	1	1	0	2	1
Bridges3 0.25	36	25	25	25	26	1
Bridges3 0.5	37	22	22	22	23	2
Bridges3 0.75	38	25	25	24	25	1
Bridges3 1.0	4	3	3	3	2	1
Bridges	312	174	171	169	183	21
Barter 0.25	150	106	128	129	108	60
Barter 0.5	150	134	137	134	118	45
Barter 0.75	150	140	138	137	111	27
Barter 1.0	15	14	14	14	11	2
Barter	465	394	417	414	348	155
Pathways 0.25	160	40	40	40	40	19
Pathways 0.5	160	70	60	50	60	13
Pathways 0.75	170	60	50	40	60	12
Pathways 1.0	19	5	6	6	7	2
Pathways	509	175	156	136	167	46
Total	1689	1005	1009	982	946	232

Table 2: Instances Solved By Domain

number of instances in any of the domains – this is due to a weak search heuristic, as we describe below.

Figure 4 depicts the total cumulative time across all instances in all domains for the solved instances. The figure plots every one hundredth point, but each point reflects the true cumulative time that includes those points not plotted. The `DeFAULT-FF` configuration appears to have a high cumulative time, but only as a consequence of solving several instances in PARC Printer (with an albeit high planning time) that the other configurations could not. Among the configurations that reason about incompleteness, we see that as the bound on the cardinality of the prime implicants increases the total planning time increases. Each of the prime implicant approaches also takes less time overall than the `BDD` configuration. `POND` performs very poorly, and is unable to scale very well, and we do not plot its runtime in the figure.

Figure 6 includes several plots and a table that characterize the relative quality of the solutions found by each configuration of `DeFAULT`. The scatter plots include points that, for example in the upper-left plot,

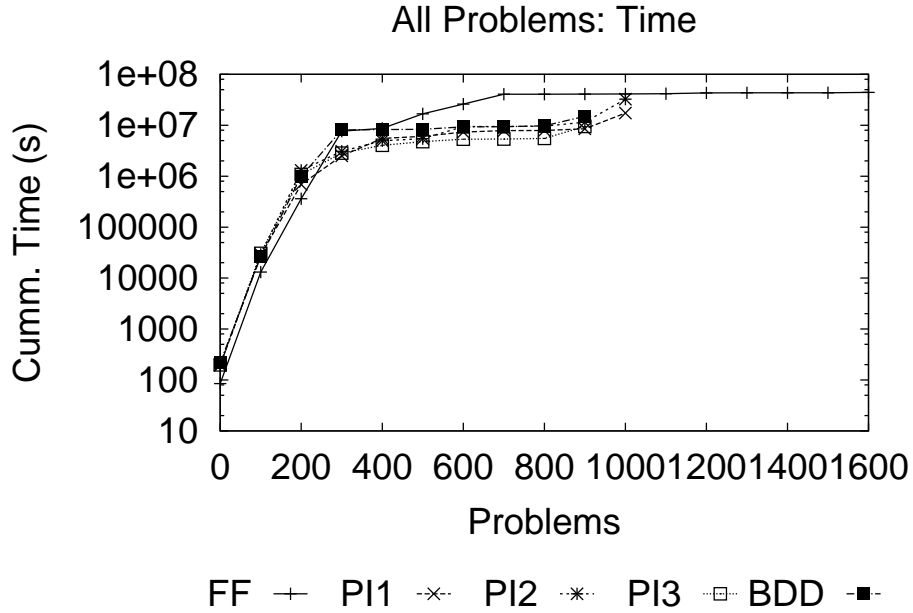


Figure 4: Cumulative Time Comparison in All Domains.

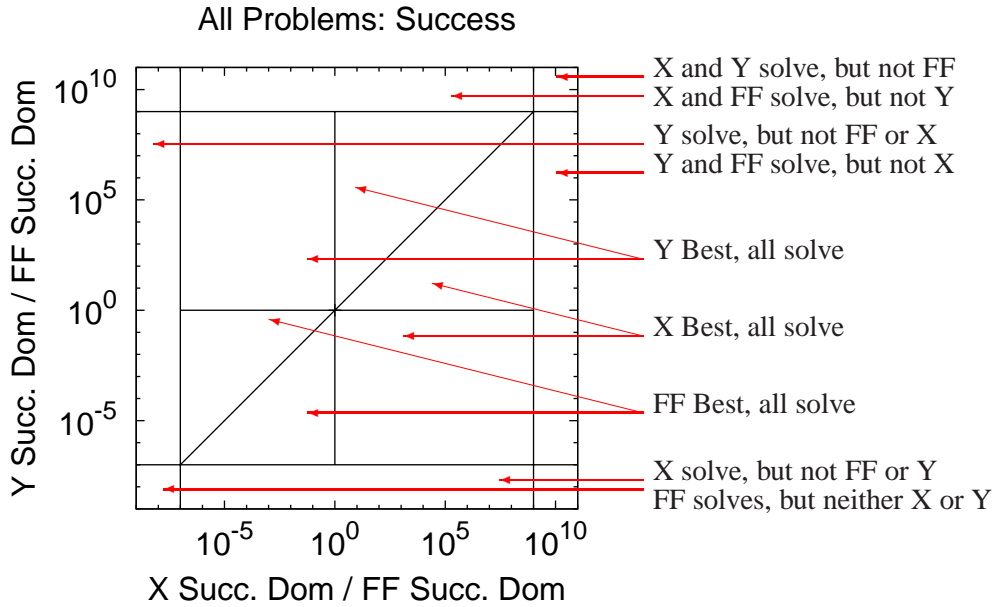


Figure 5: Scatterplot Legend.

correspond to

$$\left(\frac{(2^{|\mathcal{F}|} - |M(d(\tilde{\pi}_{PI1}))|)}{(2^{|\mathcal{F}|} - |M(d(\tilde{\pi}_{FF}))|)}, \frac{(2^{|\mathcal{F}|} - |M(d(\tilde{\pi}_{PI2}))|)}{(2^{|\mathcal{F}|} - |M(d(\tilde{\pi}_{FF}))|)} \right)$$

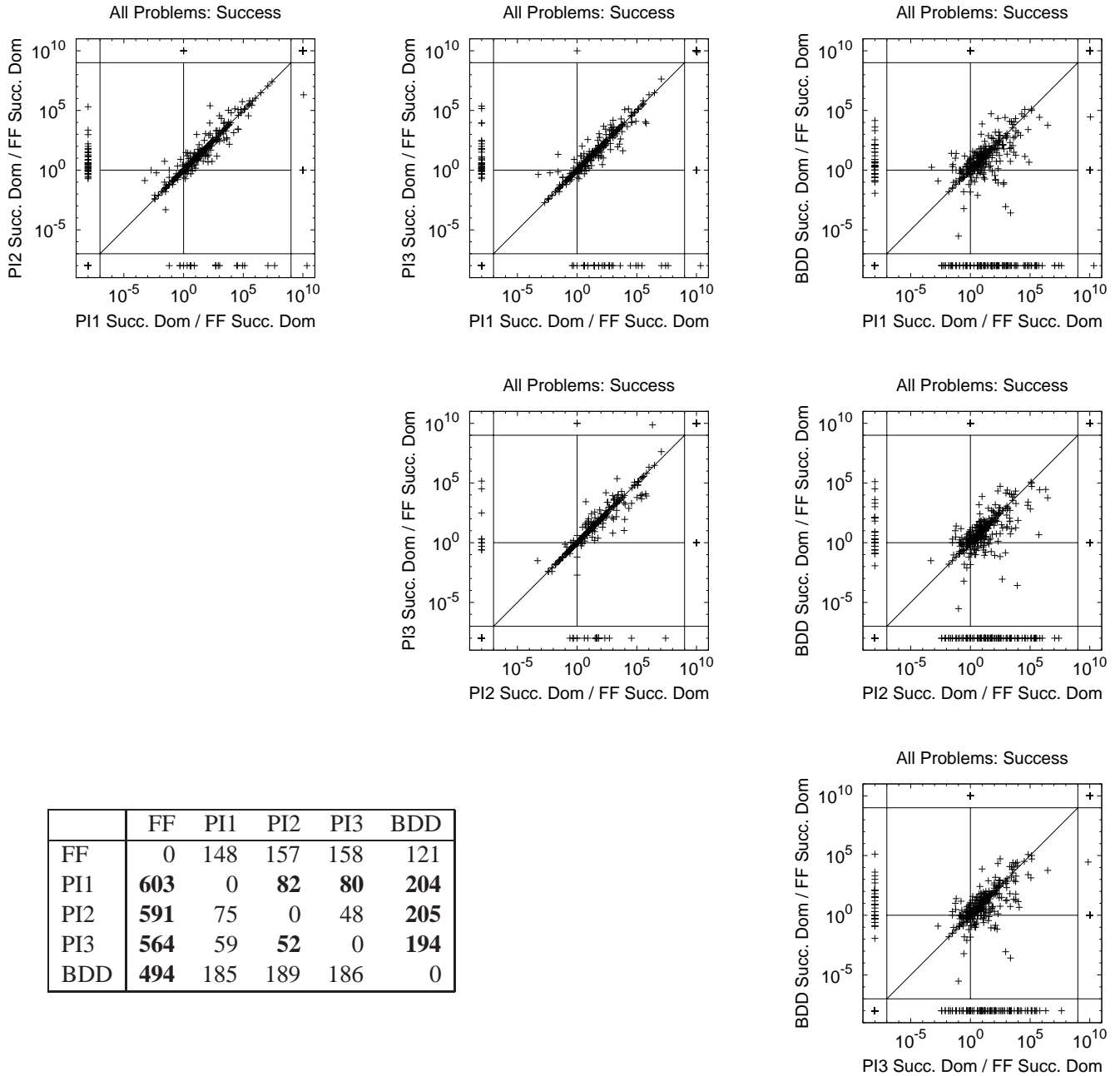


Figure 6: Ratio of solution quality in All Domains

which in words is the ratio of the number of successful interpretations for the plan found by DeFAULT-PI1 with that of DeFAULT-FF along the horizontal axis and the ratio of DeFAULT-PI2 with that of DeFAULT-FF along the vertical axis. As summarized by Figure 5 If DeFAULT-FF solves the problem and DeFAULT-PI1 does not we replace the ratio by $1e-8$ (outside of the bounding box within the plot), if DeFAULT-FF does not solve the problem but DeFAULT-PI1 does then we replace the ratio by $1e10$

(outside the bounding box), and if neither solves the problem we replace the ratio by 1.0. All points to the left of the diagonal line ($x=y$) indicate that `DeFAULT-PI2` found a better solution than `DeFAULT-PI1`, and all points to the right indicate the opposite. Also, all points below the horizontal line (where the ratio is 1.0) indicate that `DeFAULT-FF` found a solution better than `DeFAULT-PI2` and vice-versa above the line. Likewise, the vertical line (where the ratio is 1.0) indicates that `DeFAULT-FF` found a solution better than `DeFAULT-PI1` to the left and vice-versa to the right. Each plot represents a different comparison of the planner configurations. The table in the figure is the count of the instances where the configuration in each row finds a better solution (i.e., more interpretations that succeed) than the configuration in the column. The counts only include instances where both configurations returned a solution. The bolded entries in the table indicate the configuration that found superior solutions more frequently than the other (e.g., `DeFAULT-FF` found 60 superior solutions compared to `DeFAULT-PI1`, but `DeFAULT-PI1` found 95 superior solutions and is therefore bolded).

The results in Figure 6 indicate the following. The table within the figure shows that i) all configurations find superior solutions more frequently than *FF*, ii) each of the *PI* configurations finds superior solutions more frequently than *BDD*, and iii) of the *PI* configurations *PI1* finds superior solutions more frequently. In comparing the magnitude of the difference in plan quality, by examining the scatter plots, we see that among the *PI* configurations, there is not a significant distinction between them in terms of quality; however in comparison to the *BDD* configuration each of the *PI* configurations shows marked improvement. In the most extreme cases, the *PI* configurations find solutions that are several orders of magnitude better than the *BDD* configuration, whereas at its best the *BDD* configuration can find solutions that are one to two orders of magnitude better than any of the *PI* methods. Another interesting observation is that when comparing the cases where the *FF* configuration solves a problem but only one of either the *PI* or *BDD* configurations solves the problem, the *BDD* configuration tends to find fewer solutions, but *PI* tends to find more solutions.

In summary, the answers to our research questions that are supported by these results are as follows.

- Q1: Of the two competitive CPP planners that we used to solve translated incomplete planning domains, POND exhibited very poor scalability in relation to `DeFAULT` and PFF would require further development to address our problems, making it unclear whether PFF would perform better than `DeFAULT`. POND fails to scale very well because it uses a Monte Carlo based heuristic that samples states from its belief state for which to compute a heuristic; as the number of incomplete features grows POND is sampling a diminishing proportion of the possible states, making its heuristic less informed.
- Q2: The table in Figure 6 indicates that, no, a classical planner cannot find reasonable quality solutions when planning in incomplete domains. The number of instances where it is inferior to the other planner configurations is nearly three times the number of cases where it is superior. The majority of the cases where the classical planner finds superior solutions are better by a smaller magnitude than the magnitude of the cases where it is superior. However, the classical planner can find solutions in many cases where a planner that reasons about incompleteness cannot due to scalability.
- Q3: Table 2 indicates that the *BDD* configuration which counts models out-scales the other approaches in the Bridges domain, is comparable in the PARC Printer and Pathways domains, and fails to scale as well in the Barter World domain. Figure 3 provides one possible explanation in that the Barter World domain instances include nearly an order of magnitude more incomplete domain features, making model counting that much more expensive in the worst case. We designed the Barter

World problem to have failure explanations with high cardinality prime implicants, which also impacts the OBDD representation and makes model counting costly if not prohibitive.

- Q4: As shown by Figure 6 and Table 2, counting prime implicants does indeed lead to high quality solutions, and quite unexpectedly, better solutions than counting models. The issue of scalability is linked with the structure of the domain and the relative cost of model counting, per Q3.
- Q5: Figure 6 suggests that as the bound on the size of the prime implicants becomes smaller, scalability improves, but overall quality is not affected. The plan quality being unaffected is somewhat unintuitive, but may be explained in terms our intuition for how we compare two formulas expressed by prime implicants; we compare the numbers of smaller cardinality prime implicants before the larger because the smaller cardinality prime implicants refer to relatively more models. It appears that comparing formulas in terms of their unit cardinality prime implicants is sufficient for most instances that we evaluated, which supports GL’s motivations for studying single-fault plans.

Barter World: Figure 7 depicts results in the Barter World domain in a fashion similar to Figure 6, and Figure 11 includes a plot depicting the cumulative planning time for all solved instances for each of the configurations. Planner scalability and runtime generally worsen as more information about the incompleteness is used within the planner (i.e., *FF* scales best, then *PI_k* is best with lower values of *k*, and *BDD* is worst). In terms of quality, *BDD* finds the most superior solutions, but with relatively lower magnitude better quality than those superior solutions found by the *PI* configurations. There are many instances that are solved by the *PI* configurations that are not solved by the *BDD* configuration and are not counted in the Figure 7 table, but can be seen in the plots.

PARC Printer: Figure 8 depicts the plan quality results and Figure 11 depicts the cumulative planning time for the PARC Printer domain. The total time taken by the *PI* configurations is significantly less than the other configurations, including the *FF* configuration. The *PI* configurations also scale better than the *BDD* configuration. Each of the *PI* configurations has superior quality to that of *FF* and *BDD* (with a slight edge to *PI1*). The magnitudes of the differences in plan quality among the methods are relatively small in this domain.

Bridges: Figure 9 depicts the plan quality results and Figure 11 depicts the cumulative planning time in the Bridges domain, and the configurations show similar performance. Unlike the prior domains, *BDD* finds the most and the best solutions in this domain.

Pathways: Figure 10 depicts the plan quality results and Figure 11 depicts the cumulative time results in the Pathways domain. The planning time and scalability is higher for the *PI* configurations than the *BDD* method, and the plan quality tends to be better as well. The *PI1* configuration appears to find the best quality plans and is followed by *PI2* and *PI3*.

Discussion: Our empirical evaluation has shown that counting prime implicants appears to be a viable alternative to model counting in incomplete domains and that there is an advantage to reasoning about multi-fault diagnoses beyond the single-fault diagnoses studied by GL. We also showed that off-the-shelf CPP planners are not, in their current form, reasonable approaches to solving incomplete domain planning problems. Lastly, while ignoring the domain incompleteness allows a classical planner to scale much better than the approaches that reason with the incompleteness, it returns very poor solutions.

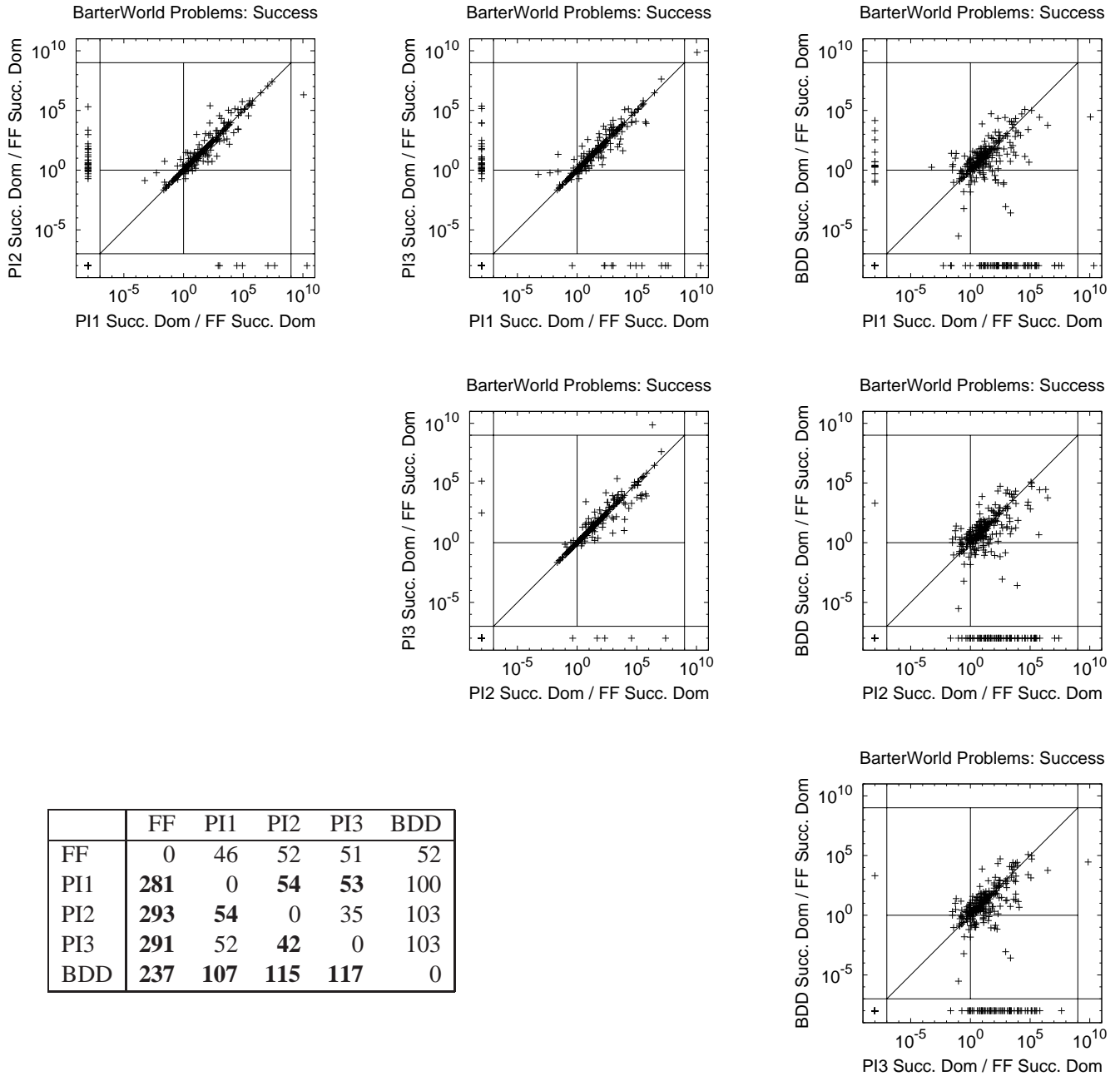


Figure 7: Ratio of solution quality in Barter World Domain

8 Related Work

Planning in incomplete domains is noticeably similar to planning with incomplete state information [Bonet and Geffner, 2000; Bryce *et al.*, 2008; Domshlak and Hoffmann, 2006; Bertoli *et al.*, 2001; Smith and Weld, 1998; Palacios and Geffner, 2006], where action descriptions instead of states are incomplete. As we have shown, incomplete domains

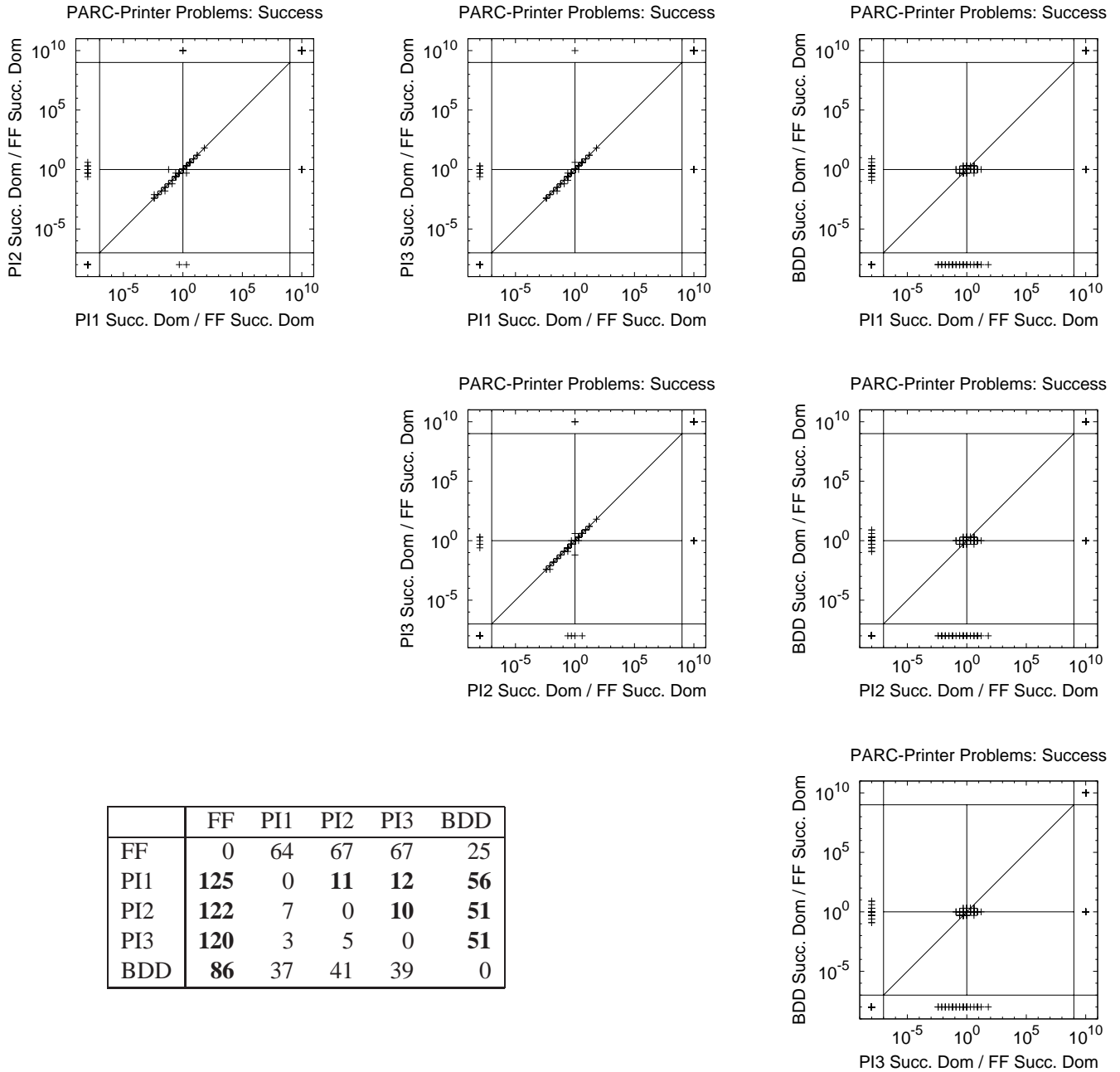


Figure 8: Ratio of solution quality in PARC Printer Domain

can be translated to CPP domains, and planners such as POND [Bryce *et al.*, 2008] and PFF [Domshlak and Hoffmann, 2006] are applicable. However, while the translation is theoretically feasible, the approach does not seem to scale as well as our DeFAULT planner. The connection between DeFAULT and POND is much deeper than their ability to solve incomplete planning problems if we consider their underlying state representations and heuristics. POND represents belief states with OBDDs and DeFAULT annotates its states with sen-

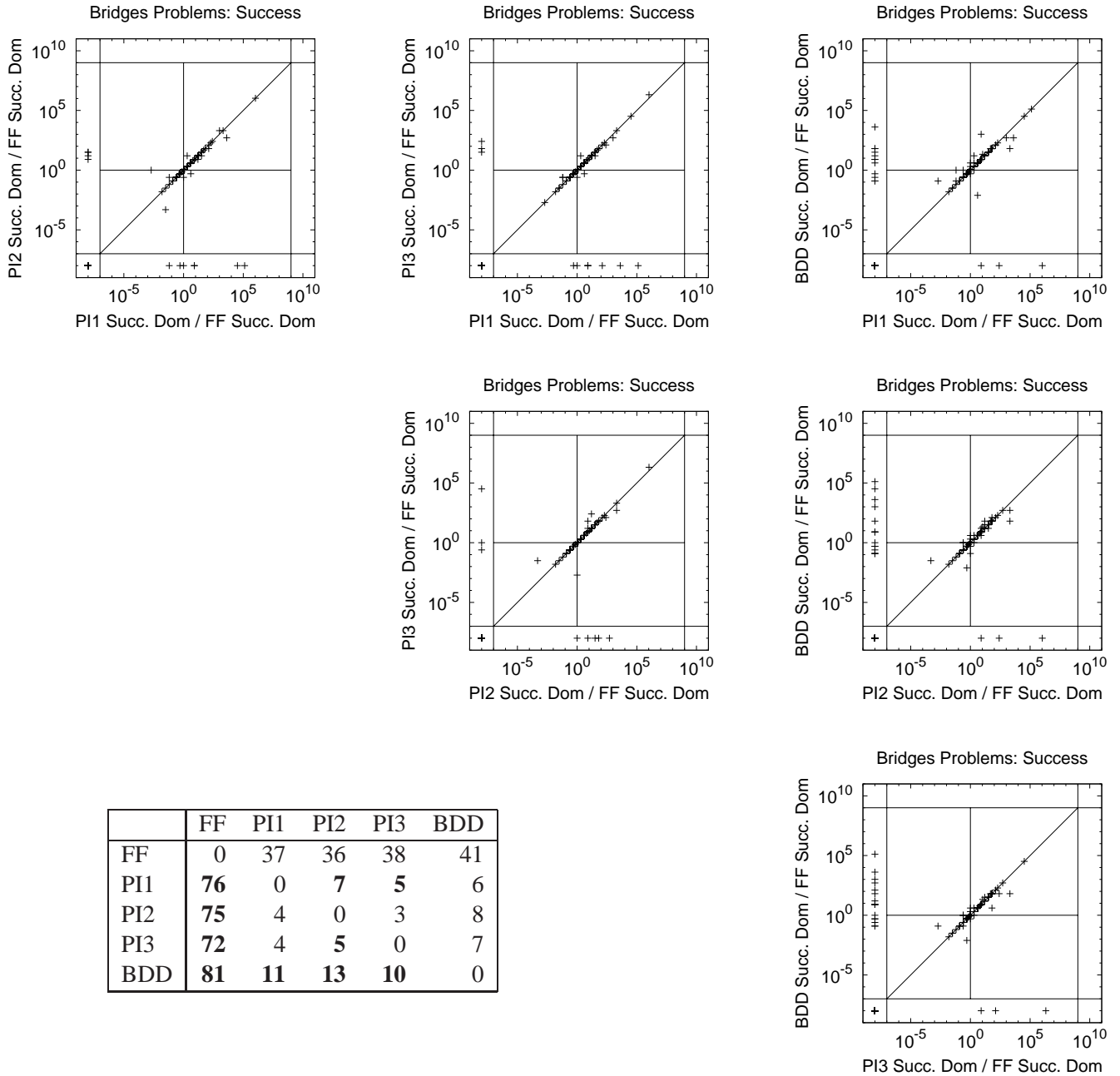


Figure 9: Ratio of solution quality in Bridges Domain

tences represented by OBDDs or prime implicants. DeFAULT could have represented its state with a single propositional sentence of the form $\bigwedge_{p \in st} (p \leftrightarrow \neg d_t(p))$ that would resemble POND's belief state. Both DeFAULT and POND propagate propositional sentences over planning graphs to compute their heuristics; however, DeFAULT selects a single action to support each proposition in the relaxed plan, and POND may select several to increase the probability of goal satisfaction. DeFAULT can be seen as a type of con-

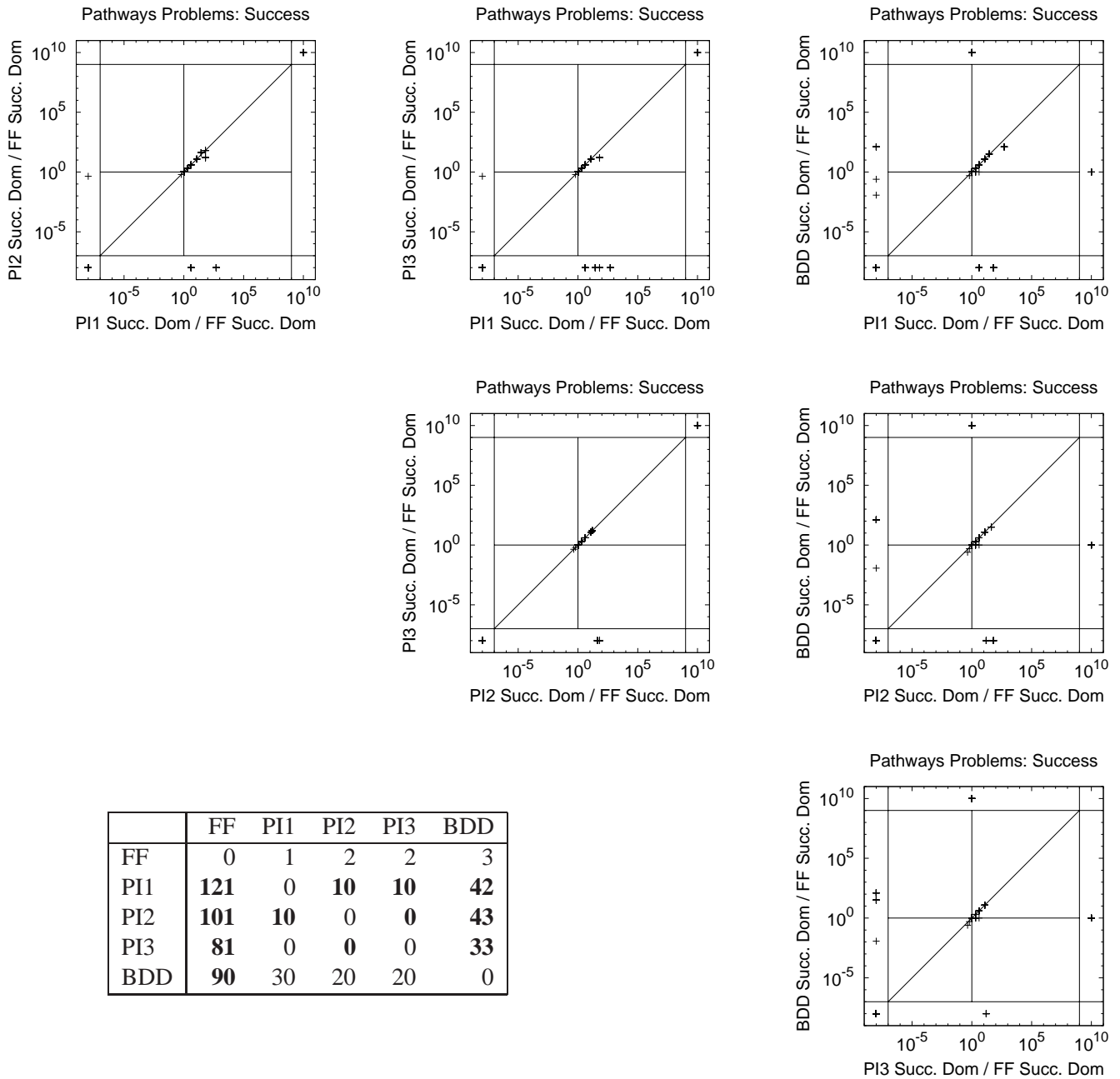


Figure 10: Ratio of solution quality in Pathways Domain

formant planner that factors and approximates the belief state (similar to Tu *et al.* [2011]) and instead of computing the probability of goal satisfaction, counts models or prime implicants.

Our investigation is an instantiation of model-lite planning, motivated by Kambhampati [2007]. Kambhampati [2007] argues that as planning becomes more adopted in applications, such as web service composition or other net-centric tasks, it must be more robust to users that can make mistakes or omissions. Constraint-

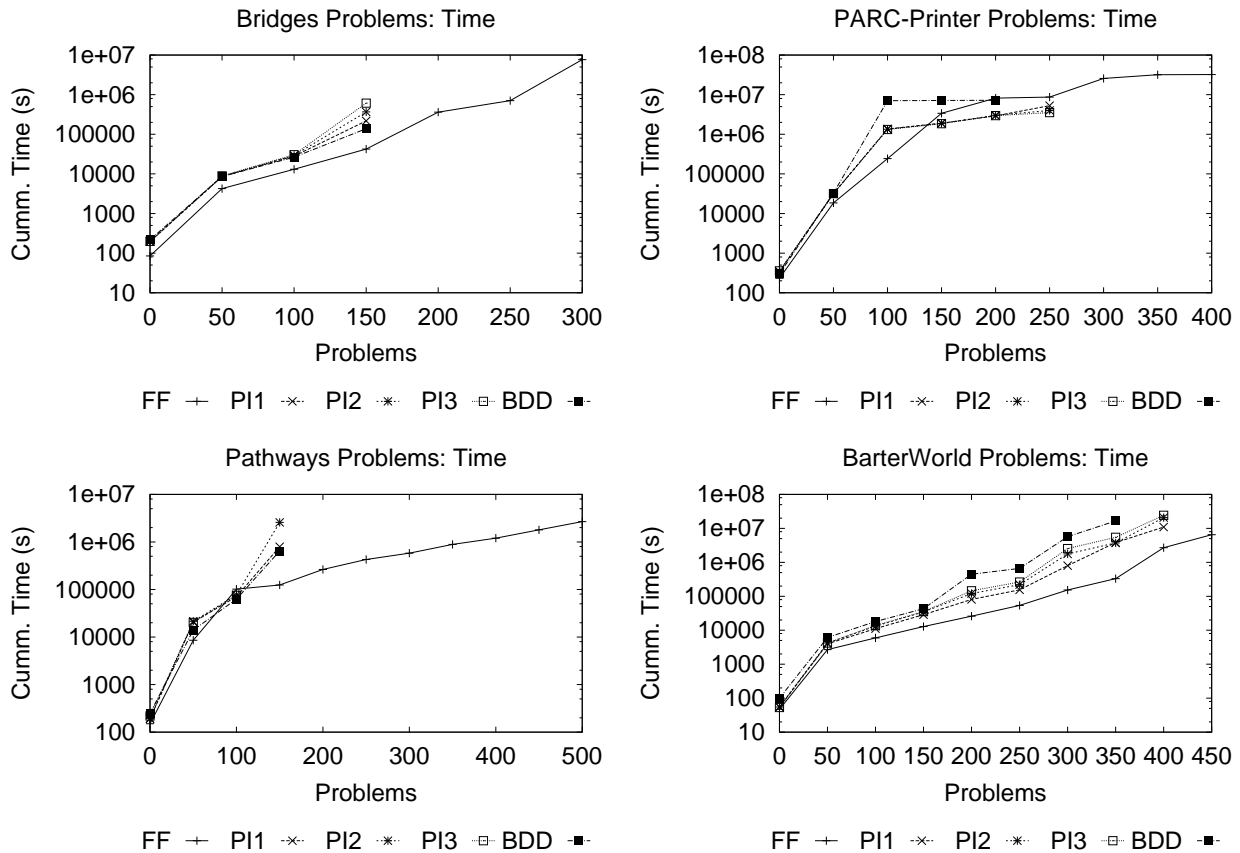


Figure 11: Cumulative Time Comparison in each domain.

based hierarchical task networks are an alternative, pointed out by Kambhampati [2007], which avoid specifying all preconditions and effects through methods and constraints that correspond to underlying, implicit causal links.

Prior work of Chang and Amir [2006] also addresses planning with incomplete models, but focusses on online planning and execution to learn the model, similar to model-based reinforcement learning. We differ in that we assume no feedback from the environment and attempt to find the best plan possible offline. However, the plans found by DeFAULT have the potential to guide either knowledge engineers or experimentation as studied by Chang and Amir [2006]. We have conducted a preliminary study [Bryce and Weber, 2011] into planning and learning in incomplete domains that is similar to that of Chang and Amir [2006], and note that the *FF* configuration (ignoring incompleteness) is most similar to the planner used by Chang and Amir. Our results indicate that using any one of the methods to reason about incompleteness leads to plans that require an agent to re-plan less often, and in many cases take less overall time (including both planning and execution) to achieve their goals.

As previously stated, this work is a natural extension of the Garland and Lesh [2002] model for evaluating plans in incomplete domains. Our methods for computing failure explanations are slightly different in that we can use fault interactions of degree greater than one. We also go beyond the work of GL to synthesize plans with incomplete domains and show that prime implicant counting, in the spirit of their definition of plan faults, can lead to significantly better plans and better scalability than model counting in some cases.

Interestingly, our *PI1* configuration is most similar to the framework considered by GL, and it tends to perform the best across our experiments.

We base our generalization of the work of GL on approaches to model based diagnosis [de Kleer and Williams, 1987]. There are several additional techniques that we can incorporate, such as strategies for probing faults, reconfiguring systems, correlated faults, and many others. Part of our contribution is showing that diagnosis is not only a natural fit for planning in incomplete domains, but that it offers many useful tools to expand the capability of planning systems for such problems.

9 Conclusion & Future Work

We have presented the first work to address planning in incomplete STRIPS domains as heuristic search to find high quality plans. Our planner, *DeFAULT*, performs forward search while maintaining plan failure explanations, and estimates the future faults incurred by propagating faults on planning graphs. We have shown that, compared to a planner that essentially ignores aspects of the incomplete domain, *DeFAULT* is able to scale reasonably well but find much better quality plans. We have also shown that representing explanations of plan failure with prime implicants can lead to better scalability than a complete representation using OBDDs and counting models.

Future work on this topic will focus on additional heuristics for estimating faults that incorporate possible delete effects and other negative interactions. One direction for extending the heuristics will use interaction propagation to measure the cost added by possible or real mutexes [Bryce and Smith, 2006]. Directions for extending our model of incompleteness include i) adding probabilistic measures of various action features existing in the true domain, ii) investigating how incompleteness can be integrated with uncertain/stochastic action effects, and iii) studying how to use correlated and conditional effects. Handling probabilistic incompleteness will amount to weighted model and prime implicant counting. The challenge of handling correlated incompleteness is that we may be forced to include additional propositions into F that state which of the combinations are legal within different interpretations of the incomplete domain. Addressing more expressive action models will require extending the system description and fault propagation semantics. We are confident that many of the extensions will resemble issues addressed by model based diagnosis and represent a unique application to planning.

Acknowledgements: This work was supported by the DARPA contract HR001-07-C-0060.

References

- Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI*, pages 473–486, 2001.
- Piergiorgio Bertoli, Adi Botea, and Simone Fratini, editors. *ICKEPS*, 2009.
- Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal, Québec, Canada, August 20-25 1995*, pages 1636–1642. Morgan Kaufmann, 1995.
- B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proceedings of AIPS'00*, 2000.
- Daniel Bryce and David E. Smith. Using interaction to compute better probability estimates in plan graphs. In *ICAPS 2006 Workshop on Planning Under Uncertainty and Execution Control for Autonomous Systems*, 2006.

- Daniel Bryce and Christopher Weber. Planning and acting in incomplete domains. In *Proceedings of the Twenty First International Conference on Automated Planning and Scheduling*, 2011. To appear.
- D. Bryce, S. Kambhampati, and D.E. Smith. Planning graph heuristics for belief space search. *Journal of AI Research*, 26:35–99, 2006.
- D. Bryce, S. Kambhampati, and D.E. Smith. Sequential monte carlo in probabilistic planning reachability heuristics. *AIJ*, 172(6-7):685–715, 2008.
- Allen Chang and Eyal Amir. Goal achievement in partially known, partially observable domains. In *ICAPS'06*, 2006.
- Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.
- Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.
- Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. *Characterizing diagnoses and systems*, pages 54–65. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.
- Johan de Kleer. An assumption-based tms. *Artificial Intelligence*, 28(2):127–162, 1986.
- C. Domshlak and J. Hoffmann. Fast probabilistic planning through weighted model counting. In *Proceedings of ICAPS'06*, pages 243–251, 2006.
- R. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of AAAI'71*, pages 608–620, 1971.
- Andrew Garland and Neal Lesh. Plan evaluation with incomplete action descriptions. In *Proceedings of AAAI'02*, 2002.
- A. Gerevini, B. Bonet, and R. Givan, editors. *5th International Planning Competition*. Cumbria, UK., 2006.
- Malte Helmert. The fast downward planning system. *JAIR*, 26:191–246, 2006.
- J. Hoffmann and R. Brafman. Conformant planning via heuristic forward search: A new approach. In *Proceedings of ICAPS'04*, pages 355–364, 2004.
- J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.
- S. Kambhampati. Model-lite planning for the web age masses. In *Proceedings of AAAI'07*, 2007.
- Derek Long and Maria Fox. Efficient implementation of the plan graph in stan. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.
- R. Mailler, D. Bryce, J. Shen, and C. Orielly. Mable: A framework for natural instruction. In *AAMAS'09*, 2009.
- D. McDermott. Pddl-the planning domain definition language. Technical report, Available at: www.cs.yale.edu/homes/dvm, 1998.
- Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Oper. Res.*, 53(5):780–798, September-October 2005.
- Héctor Palacios and Hector Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *AAAI*, 2006.
- Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
- Jared Robertson and Daniel Bryce. Reachability heuristics for planning in incomplete domains. In *Proceedings of the ICAPS'09 Workshop on Heuristics for Domain Independent Planning*, 2009.
- D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, Apr 1996.
- David E. Smith and Daniel S. Weld. Conformant Graphplan. In Jack Mostow and Charles Rich, editors,

Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA., pages 889–896. AAAI Press / The MIT Press, 1998.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

Phan Huy Tu, Tran Cao Son, Michael Gelfond, and A. Ricardo Morales. Approximation of action theories and its application to conformant planning. *Artif. Intell.*, 175(1):79–119, 2011.

Kangheng Wu, Quiag Yang, and Yunfei Jiang. Arms: an automatic knowledge engineering tool for learning action models for ai planning. *K. Eng. Rev.*, 22(2):135–152, 2007.

A Conformant Probabilistic Planning for Incomplete Domains

By viewing incomplete domains as a set of complete domains and describing the uncertain features with state propositions, we can formulate problems as conformant planning. Conformant planning captures all state uncertainty in a belief state (or set of possible states), which can be thought of as a version space representing alternative hypotheses of the domain. Conformant plans seek to apply a sequence of actions that will guarantee goal achievement from all possible states – such plans would be without fault. Unfortunately, we are not guaranteed that it is possible to achieve the goal from all states (i.e., with all interpretations of the incomplete domain), and must be satisfied with achieving the goal from as many states as possible.

If we cannot achieve the goal from all initial states, then we must select the appropriate type of conformant planner, of which there are two possibilities: non-deterministic and probabilistic. Non-deterministic conformant planners describe state (and action effect) uncertainty as sets of possibilities, whereas probabilistic conformant planners use probability distributions over the sets of possibilities. This distinction has an important implication when the planner is tasked with achieving the goal from as many states as possible. Existing non-deterministic conformant planners [Bryce *et al.*, 2006; Hoffmann and Brafman, 2004; Palacios and Geffner, 2006; Bertoli *et al.*, 2001] solve the *strong* planning problem, where solutions must achieve the goal from all possible states. Whereas, probabilistic planners find plans that achieve the goal with as much probability as possible; if the distribution over initial states (interpretations of the domain) is a uniform distribution, then the number of initial states achieving the goal is proportional to the probability the goal is achieved.

While using a probabilistic planner is not completely necessary for finding plans that achieve the goal from as many states as possible, existing planners can be applied without modification. While we do not explore the possibility, incomplete domains could be specified with probabilities that incomplete features exist in the ground-truth domain, and a probabilistic planner would be the most likely choice of planner. Non-deterministic planners would be sufficient for the incomplete domains studied in this work if they could find plans that achieve the goals from as many initial states as possible; however, because no such planners are readily available, we choose to employ conformant probabilistic planning (CPP).

In following, we describe how the incomplete actions can be translated into incomplete state information. The resulting CPP problem can be solved by any CPP planner. As we will show, the probability of achieving the goal in the CPP translation is related to the number of failed interpretations of the original incomplete domain.

A.1 Conformant Probabilistic Planning

Conformant planning deals with the problem of having a set of states (a belief state) consistent with an agent's knowledge and attempting to find a plan that will achieve the goals simultaneously from as many of the possible states as possible.

Definition A.1. A conformant probabilistic planning domain D defines the tuple (P, A, b_I, G, τ) , where

- P is a set of propositions
 - A is a set of actions, where each $a \in A$ defines a set of conditional effects $\text{eff}(a) = \{\text{eff}_0(a), \dots, \text{eff}_{m-1}(a)\}$. Each conditional effect $\text{eff}_i(a)$ consists of
 - $\text{pre}_i(a) : 2^P \rightarrow \{\perp, \top\}$, a condition formula
 - $\text{add}_i(a) \subseteq P$, a set of add effects
 - $\text{del}_i(a) \subseteq P$, a set of delete effects
- We require that no two conditional effects conflict (i.e., there are no cases where two conditional effects are applicable in a state s and they disagree on the effect on a proposition, $s \models \text{pre}_i(a)$, $s \models \text{pre}_j(a)$ and $\text{add}_i(a) \cap \text{del}_j(a) \neq \{\}$).
- b_I is the initial belief state, such that $b_I(s) = \text{Pr}(s)$, for all $s \subseteq P$.
 - $G \subseteq P$ is the goal
 - $0 < \tau \leq 1$ is a lower bound on the probability of goal satisfaction

For example, the CPP translation (defined below) of the incomplete domain example from Section 2 defines:

- $P = \{p, q, r, g, \text{valid}\} \cup F$, where $F = \{\widetilde{\text{pre}}(a, r), \widetilde{\text{add}}(a, r), \widetilde{\text{del}}(a, p), \widetilde{\text{del}}(b, q), \widetilde{\text{pre}}(c, q)\}$
- $A = \{a, b, c\}$

$\text{pre}_0(a) = \text{valid} \wedge p \wedge q \wedge (r \vee \neg \widetilde{\text{pre}}(a, r)) \wedge \widetilde{\text{add}}(a, r)$	$\text{add}_0(a) = \{r\}$	$\text{del}_0(a) = \{\}$
$\text{pre}_1(a) = \text{valid} \wedge p \wedge q \wedge (r \vee \neg \widetilde{\text{pre}}(a, r)) \wedge \widetilde{\text{del}}(a, p)$	$\text{add}_1(a) = \{\}$	$\text{del}_1(a) = \{p\}$
$\text{pre}_2(a) = \neg(\text{valid} \wedge p \wedge q \wedge (r \vee \neg \widetilde{\text{pre}}(a, r)))$	$\text{add}_2(a) = \{\}$	$\text{del}_2(a) = \{\text{valid}\}$
$\text{pre}_0(b) = \text{valid} \wedge p \wedge \widetilde{\text{del}}(b, q)$	$\text{add}_0(b) = \{\}$	$\text{del}_0(b) = \{q\}$
$\text{pre}_1(b) = \text{valid} \wedge p$	$\text{add}_1(b) = \{r\}$	$\text{del}_1(b) = \{\}$
$\text{pre}_2(b) = \neg(\text{valid} \wedge p)$	$\text{add}_2(b) = \{\}$	$\text{del}_2(b) = \{\text{valid}\}$
$\text{pre}_0(c) = \text{valid} \wedge r \wedge (q \vee \neg \widetilde{\text{pre}}(a, q))$	$\text{add}_0(c) = \{g\}$	$\text{del}_0(c) = \{\}$
$\text{pre}_1(c) = \neg(\text{valid} \wedge r \wedge (q \vee \neg \widetilde{\text{pre}}(a, q)))$	$\text{add}_1(c) = \{\}$	$\text{del}_1(c) = \{\text{valid}\}$
- $b_I(s) = \begin{cases} 1/2^5 & : s = \{p, q, \text{valid}\} \cup F^i, F^i \subseteq F \\ 0 & : \text{otherwise} \end{cases}$
- $G = \{g, \text{valid}\}$

Without providing in-depth details on the translation (see Section A.2), we use a proposition *valid* to denote if it is not the case that a state (or one of its predecessors) failed to satisfy a required precondition. Each initial state has a different subset of the propositions in F set to true, and corresponds to one interpretation of the incomplete domain. The probability of achieving the goal indicates the number of interpretations of the incomplete domain that both satisfy all required preconditions and satisfy the original goal.

Definition A.2. A plan $\pi = (a_0, \dots, a_{n-1})$ in a CPP domain D is sequence of actions, that corresponds to a sequence of belief states (b_0, \dots, b_n) , where

- $b_0 = b_I$
- $b_{t+1}(s') = \sum_{s \in \mathcal{S}(s')} b_t(s)$, where
 - $\mathcal{S}(s') = \{s \mid s' = s \setminus \text{del}(a_t, s) \cup \text{add}(a_t, s)\}$
 - $\text{del}(a_t, s) = \bigcup_{i:s \models \text{pre}_i(a_t)} \text{del}_i(a_t)$
 - $\text{add}(a_t, s) = \bigcup_{i:s \models \text{pre}_i(a_t)} \text{add}_i(a_t)$
- $\tau \leq \sum_{s:G \subseteq s} b_n(s)$

For example, the plan (a, b, c) corresponds to a belief state sequence (b_0, b_1, b_2, b_3) . Each belief state assigns 2^5 states non-zero probabilities, and to retain a succinct example we list the belief states below as propositional sentences (each model of a sentence corresponds to a state whose probability is $1/2^5$). The belief states are as follows:

$$b_0 = p \wedge q \wedge \neg r \wedge \neg g \wedge \text{valid}$$

$$b_1 = (p \wedge q \wedge \neg r \wedge \neg g \wedge \neg \text{valid} \wedge \widetilde{\text{pre}}(a, r)) \vee \\ (p \wedge q \wedge r \wedge \neg g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r)) \vee \\ (\neg p \wedge q \wedge r \wedge \neg g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r)) \vee \\ (p \wedge q \wedge \neg r \wedge \neg g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r)) \vee \\ (\neg p \wedge q \wedge \neg r \wedge \neg g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r))$$

$$b_2 = (p \wedge q \wedge \neg r \wedge \neg g \wedge \neg \text{valid} \wedge \widetilde{\text{pre}}(a, r)) \vee \\ (\neg p \wedge q \wedge r \wedge \neg g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r) \wedge \neg \widetilde{\text{del}}(b, q)) \vee \\ (\neg p \wedge \neg q \wedge r \wedge \neg g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r) \wedge \widetilde{\text{del}}(b, q)) \vee \\ (\neg p \wedge q \wedge r \wedge \neg g \wedge \neg \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r)) \vee \\ (\neg p \wedge q \wedge r \wedge \neg g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r) \wedge \neg \widetilde{\text{del}}(b, q)) \vee \\ (\neg p \wedge \neg q \wedge r \wedge \neg g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r) \wedge \widetilde{\text{del}}(b, q)) \vee \\ (\neg p \wedge q \wedge \neg r \wedge \neg g \wedge \neg \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r))$$

$$b_3 = (p \wedge q \wedge \neg r \wedge \neg g \wedge \neg \text{valid} \wedge \widetilde{\text{pre}}(a, r)) \vee \\ (\neg p \wedge q \wedge r \wedge g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r) \wedge \neg \widetilde{\text{del}}(b, q)) \vee \\ (\neg p \wedge \neg q \wedge r \wedge g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r) \wedge \widetilde{\text{del}}(b, q) \wedge \neg \widetilde{\text{pre}}(c, q)) \vee \\ (\neg p \wedge \neg q \wedge r \wedge \neg g \wedge \neg \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r) \wedge \widetilde{\text{del}}(b, q) \wedge \widetilde{\text{pre}}(c, q)) \vee \\ (\neg p \wedge q \wedge r \wedge \neg g \wedge \neg \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \neg \widetilde{\text{add}}(a, r)) \vee \\ (\neg p \wedge q \wedge r \wedge g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r) \wedge \neg \widetilde{\text{del}}(b, q)) \vee \\ (\neg p \wedge \neg q \wedge r \wedge g \wedge \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r) \wedge \widetilde{\text{del}}(b, q) \wedge \neg \widetilde{\text{pre}}(c, q)) \vee \\ (\neg p \wedge \neg q \wedge r \wedge \neg g \wedge \neg \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r) \wedge \widetilde{\text{del}}(b, q) \wedge \widetilde{\text{pre}}(c, q)) \vee \\ (\neg p \wedge q \wedge \neg r \wedge \neg g \wedge \neg \text{valid} \wedge \neg \widetilde{\text{pre}}(a, r) \wedge \widetilde{\text{del}}(a, p) \wedge \widetilde{\text{add}}(a, r))$$

The number of models of the logical representation of b_3 where both g and valid are true is six, making the probability that the goal is satisfied $6/2^5 = 0.1875$. Recall that the plan failure explanations included 26 models, which corresponds to six successful models, as in CPP.

A.2 Translation to Conformant Probabilistic Planning

The intuition behind the translation to CPP is to augment the states so that new propositions describe the action incompleteness. For example, if \tilde{a} has a possible precondition p , then the new unobservable proposition $\widetilde{\text{pre}}(\tilde{a}, p)$ is introduced to indicate whether or not p is a precondition. The new propositions describing action incompleteness correspond with F. The actions require modification as well: an action \tilde{a} is translated to an action \bar{a} with conditional effects. The action's effects describe cases (subject to incomplete features) where the action's effects are given or the action fails. For example, the action \tilde{a} with possible precondition p has the add effect q , which corresponds to the \bar{a} effect $\text{pre}_i(\bar{a}) = \neg\widetilde{\text{pre}}(\tilde{a}, p) \vee p$, $\text{add}_i(\bar{a}) = q$ (i.e., q is given as an effect only if p is not a precondition or, if it is a precondition, p is true).

Definition A.3. The CPP translation $\bar{D} = (\bar{P}, \bar{A}, \bar{b}_I, \bar{G}, \tau)$ of the incomplete domain (P, \tilde{A}, I, G) , is as follows:

- $\bar{P} = P \cup \{\text{valid}\} \cup F$
- For each $\tilde{a} \in \tilde{A}$, there is an $\bar{a} \in \bar{A}$, with one effect for each known add or delete effect and each possible add or delete effect and one effect recording if the action has failed. The effects are summarized by the following table.

\tilde{a} Effect	$\text{pre}_i(\bar{a})$	$\text{add}_i(\bar{a})$	$\text{del}_i(\bar{a})$
$p \in \text{add}(\tilde{a})$	$\text{valid} \wedge \widetilde{\text{pre}}(\tilde{a})$	$\{p\}$	$\{\}$
$p \in \widetilde{\text{add}}(\tilde{a})$	$\text{valid} \wedge \widetilde{\text{pre}}(\tilde{a}) \wedge \widetilde{\text{add}}(\tilde{a}, p)$	$\{p\}$	$\{\}$
$p \in \text{del}(\tilde{a})$	$\text{valid} \wedge \widetilde{\text{pre}}(\tilde{a})$	$\{\}$	$\{p\}$
$p \in \widetilde{\text{del}}(\tilde{a})$	$\text{valid} \wedge \widetilde{\text{pre}}(\tilde{a}) \wedge \widetilde{\text{del}}(\tilde{a}, p)$	$\{\}$	$\{p\}$
	$\neg(\text{valid} \wedge \widetilde{\text{pre}}(\tilde{a}))$	$\{\}$	$\{\text{valid}\}$

where $\widetilde{\text{pre}}(\tilde{a}) = \bigwedge_{p \in \text{pre}(\tilde{a})} p \wedge \bigwedge_{p \in \widetilde{\text{pre}}(\tilde{a})} (p \vee \neg\widetilde{\text{pre}}(\tilde{a}, p))$ indicates the preconditions that must be satisfied by the action.

- The initial belief state b_I is defined

$$\bar{b}_I(s) = \begin{cases} 1/2^{|\mathbb{F}|} & : s = I \cup \mathbb{F}_i, \mathbb{F}_i \subseteq \mathbb{F} \\ 0 & : \text{otherwise} \end{cases}$$

Theorem A.4. A plan $\tilde{\pi} = (\tilde{a}_{-1}, \tilde{a}_0, \dots, \tilde{a}_{n-1}, \tilde{a}_n)$ for an incomplete domain \tilde{D} fails in $|M(d(\pi))|$ interpretations iff in the translated CPP domain \bar{D} the corresponding plan $\bar{\pi} = (\bar{a}_0, \dots, \bar{a}_{n-1})$ achieves its goals with probability $1 - \frac{|M(d(\pi))|}{2^{|\mathbb{F}|}}$.

Proof. By construction, there is a state $s_0^i = I$ in D^i that corresponds to a state $\bar{s}_0^i \in \bar{b}_0$ (i.e., $\bar{b}_0(\bar{s}_0^i) > 0$) where $\bar{s}_0^i = I \cup \mathbb{F}^i \cup \{\text{valid}\}$.

We consider two cases for action applicability: i) the plan has not failed because of the current or prior actions, or ii) the current or prior actions cause plan failure.

If we assume that the plan has not failed (i.e., all prior actions were applicable and $\text{valid} \in \bar{s}_t^i$) and $\bar{s}_t^i = s_t^i \cup \mathbb{F}^i \cup \{\text{valid}\}$, then \tilde{a}_t^i is applicable to s_t^i iff $\bar{s}_t^i \models \text{valid} \wedge \widetilde{\text{pre}}(\tilde{a}_t^i)$. If \tilde{a}_t^i is applicable to s_t^i then $\text{pre}(\tilde{a}_t^i) \subseteq s_t^i$, which also means that $\{\text{valid}\} \cup \text{pre}(\tilde{a}_t^i) \subseteq \bar{s}_t^i$. If $\{\text{valid}\} \cup \text{pre}(\tilde{a}_t^i) \subseteq \bar{s}_t^i$, then

$\bar{s}_t^i \models \text{valid} \wedge \overline{\text{pre}}(\tilde{a}_t)$ because by contradiction

$$\begin{aligned}
& \bar{s}_t^i \wedge \neg(\text{valid} \wedge \overline{\text{pre}}(\tilde{a}_t)) \\
& s \wedge \text{valid} \wedge \text{pre}(\tilde{a}_t^i) \wedge F^i \wedge \neg \left(\text{valid} \wedge \bigwedge_{p \in \text{pre}(\tilde{a}_t)} p \wedge \bigwedge_{p \in \widetilde{\text{pre}}(\tilde{a}_t)} (p \vee \neg \widetilde{\text{pre}}(\tilde{a}, p)) \right) \\
& s \wedge \text{valid} \wedge \text{pre}(\tilde{a}_t^i) \wedge F^i \wedge \left(\neg \text{valid} \vee \bigvee_{p \in \text{pre}(\tilde{a}_t)} \neg p \vee \bigvee_{p \in \widetilde{\text{pre}}(\tilde{a}_t)} (\neg p \wedge \widetilde{\text{pre}}(\tilde{a}, p)) \right) \\
& s \wedge \text{valid} \wedge \text{pre}(\tilde{a}_t^i) \wedge F^i \wedge \left(\bigvee_{\substack{p \in \widetilde{\text{pre}}(\tilde{a}_t) \\ \widetilde{\text{pre}}(\tilde{a}, p) \in F^i}} (\neg p \wedge \top) \vee \bigvee_{\substack{p \in \widetilde{\text{pre}}(\tilde{a}_t) \\ \widetilde{\text{pre}}(\tilde{a}, p) \notin F^i}} (\neg p \wedge \perp) \right) \\
& s \wedge \text{valid} \wedge \text{pre}(\tilde{a}_t^i) \wedge F^i \wedge \left(\bigvee_{\substack{p \in \widetilde{\text{pre}}(\tilde{a}_t) \\ \widetilde{\text{pre}}(\tilde{a}, p) \in F^i}} \neg p \right) \\
& \perp
\end{aligned}$$

where $s = \bar{s}_t^i \setminus (\{\text{valid}\} \cup \text{pre}(\tilde{a}_t^i) \cup F^i)$. Likewise, if $\bar{s}_t^i \models \text{valid} \wedge \overline{\text{pre}}(\tilde{a}_t)$ then \tilde{a}_t^i is applicable to s_t^i because no prior action failed (otherwise valid would not be satisfied) and, per the contradiction proof above, it must be the case that $\text{pre}(\tilde{a}_t^i) \subseteq \bar{s}_t^i$. If $\text{pre}(\tilde{a}_t^i) \subseteq \bar{s}_t^i$ then $\text{pre}(\tilde{a}_t^i) \subseteq s_t^i$ because $s_t^i = \bar{s}_t^i \setminus (\{\text{valid}\} \cup F^i)$, meaning that \tilde{a}_t^i is applicable to s_t^i .

If we assume that a prior or current action failed, then \tilde{a}_t^i is not applicable to s_t^i iff $\bar{s}_t^i \models \neg(\text{valid} \wedge \overline{\text{pre}}(\tilde{a}_t))$. If \tilde{a}_t^i is not applicable to s_t^i then $\bar{s}_t^i \models \neg(\text{valid} \wedge \overline{\text{pre}}(\tilde{a}_t))$ because by the contradiction

$$\begin{aligned}
& \bar{s}_t^i \wedge \text{valid} \wedge \overline{\text{pre}}(\tilde{a}_t) \\
& \bar{s}_t^i \wedge \text{valid} \wedge \bigwedge_{p \in \text{pre}(\tilde{a}_t)} p \wedge \bigwedge_{p \in \widetilde{\text{pre}}(\tilde{a}_t)} (p \vee \neg \widetilde{\text{pre}}(\tilde{a}, p)) \\
& \bar{s}_t^i \wedge \text{valid} \wedge \bigwedge_{p \in \text{pre}(\tilde{a}_t)} p \wedge \bigwedge_{\substack{p \in \widetilde{\text{pre}}(\tilde{a}_t) \\ \widetilde{\text{pre}}(\tilde{a}, p) \in F^i}} (p \vee \perp) \wedge \bigwedge_{\substack{p \in \widetilde{\text{pre}}(\tilde{a}_t) \\ \widetilde{\text{pre}}(\tilde{a}, p) \notin F^i}} (p \vee \top) \\
& \bar{s}_t^i \wedge \text{valid} \wedge \bigwedge_{p \in \text{pre}(\tilde{a}_t)} p \wedge \bigwedge_{\substack{p \in \widetilde{\text{pre}}(\tilde{a}_t) \\ \widetilde{\text{pre}}(\tilde{a}, p) \in F^i}} p \\
& \bar{s}_t^i \wedge \text{valid} \wedge \bigwedge_{p \in \text{pre}(\tilde{a}_t^i)} p
\end{aligned}$$

and if either $\text{valid} \notin \bar{s}_t^i$ or $\text{pre}(\tilde{a}_t^i) \not\subseteq \bar{s}_t^i$ then $\bar{s}_t^i \wedge \text{valid} \wedge \bigwedge_{p \in \text{pre}(\tilde{a}_t^i)} p$ is unsatisfiable. If $\bar{s}_t^i \models \neg(\text{valid} \wedge \overline{\text{pre}}(\tilde{a}_t))$ then \tilde{a}_t^i is not applicable to s_t^i because if valid is satisfied a previous action failed or per the derivation above if $\text{pre}(\tilde{a}_t^i) \not\subseteq \bar{s}_t^i$ then $\text{pre}(\tilde{a}_t^i) \not\subseteq s_t^i$.

If we assume that $\bar{s}_t^i = s_t^i \cup F^i \cup \{valid\}$ and plan has not failed, then $\bar{s}_{t+1}^i = s_{t+1}^i \cup F^i \cup \{valid\}$ because:

$$\begin{aligned}
\bar{s}_{t+1}^i &= \bar{s}_t^i \setminus del(\bar{a}_t, \bar{s}_t^i) \cup add(\bar{a}_t, \bar{s}_t^i) \\
&= (s_t^i \cup F^i \cup \{valid\}) \setminus del(\bar{a}_t, \bar{s}_t^i) \cup add(\bar{a}_t, \bar{s}_t^i) \\
&= (s_t^i \cup F^i \cup \{valid\}) \setminus del(\tilde{a}_t^i) \cup add(\tilde{a}_t^i) \\
&= s_{t+1}^i \cup F^i \cup \{valid\}
\end{aligned}$$

where the add effects are related so that:

$$\begin{aligned}
add(\bar{a}_t, \bar{s}_t^i) &= \bigcup_{j: \bar{s}_t^i \models pre_j(a_t)} add_j(a_t) \\
&= \bigcup_{\substack{j: \bar{s}_t^i \models valid \wedge \overline{pre}(\bar{a}) \\ p \in add(\bar{a}_t)}} p \cup \bigcup_{\substack{j: \bar{s}_t^i \models valid \wedge \overline{pre}(\bar{a}) \wedge \widetilde{add}(\tilde{a}, p) \\ p \in add(\tilde{a}_t)}} p \\
&= add(\tilde{a}_t^i)
\end{aligned}$$

and similarly for the delete effects, $del(\bar{a}_t, \bar{s}_t^i) = del(\tilde{a}_t^i)$.

If the plan has failed then s_t^i is undefined iff $valid \notin s_t^i$. If s_t^i is undefined then an action \tilde{a}_k^i at time $0 \leq k \leq t-1$ failed because its precondition was not satisfied by s_k^i . As shown above, if \tilde{a}_k^i is not applicable to s_k^i , then $\bar{s}_k^i \models \neg(valid \wedge \overline{pre}(\tilde{a}_k))$, and $valid \notin \bar{s}_{k+1}^i$, meaning that $valid \notin \bar{s}_j^i$ for all $j \geq k+1$. If $valid \notin s_t^i$ then s_t^i is undefined because as shown above, for some prior time k a failure occurred $\bar{s}_k^i \models \neg(valid \wedge \overline{pre}(\tilde{a}_k))$. If a failure occurred at time k , then all actions a_j^i , $j \geq k$, are not applicable and the corresponding state s_j^i is undefined.

Each interpretation F^i corresponds to an initial state $\bar{s}_0^i \in b_0$, and per the above, applying π^i satisfies $pre(\tilde{a}_n^i) = G$ iff applying $\bar{\pi}$ to \bar{s}_0^i achieves the goal $G \cup \{valid\}$. If the number of interpretations that fail to achieve the goal is equal to the number of models of the failure explanation $|M(d(\pi))|$, and each state of the initial belief state is given probability $1/2^{|F|}$, then the probability that $\bar{\pi}$ achieves the goal is $1 - \frac{|M(d(\pi))|}{2^{|F|}}$. \square