On the Use of Prime Implicates in Conformant Planning*

Son Thanh To  
New Mexico State University  
Department of Computer Science  
sto@cs.nmsu.edu

Tran Cao Son  
New Mexico State University  
Department of Computer Science  
tson@cs.nmsu.edu

Enrico Pontelli  
New Mexico State University  
Department of Computer Science  
epontell@cs.nmsu.edu

Abstract

The paper presents an investigation of the use of two alternative forms of CNF formulae—prime implicates and minimal CNF—to compactly represent belief states in the context of conformant planning. For each representation, we define a transition function for computing the successor belief state resulting from the execution of an action in a belief state; results concerning soundness and completeness are provided. The paper describes a system (PIP) which dynamically selects either of these two forms to represent belief states, and an experimental evaluation of PIP against state-of-the-art conformant planners. The results show that PIP has the potential of scaling up better than other planners in problems rich in disjunctive information about the initial state.

Introduction and Motivation

Conformant planning (Smith and Weld 1998) is the problem of planning in presence of incomplete information about the initial state. One of the most important questions in conformant planning is how to represent the information about the initial situation, which is often referred as the initial belief state. In a domain with \( n \) propositions, the size of the initial belief state can be \( 2^n \). In the literature, the description of the initial belief state is often given as a CNF formula with some additional constructs (as discussed later).

The representation method used to encode belief states affects the performance of a conformant planner in several ways. It can quickly increase the memory usage of the planner, leading to undesirable out-of-memory situations, if the size of the belief state is large. It also directly affects the time complexity in computing the successor belief states, since this task often requires the planner to test for the satisfaction of a conjunction of literals in a belief state, which is a NP-hard problem.

In the past, several representations have been developed. An indirect representation of belief states is used in CFF (Brafman and Hoffmann 2004), while ordered binary decision diagram (OBDD) is employed in POND (Bryce, Kambhampati, and Smith 2006); approximation states have been introduced in CPA (Tran et al. 2009); the work presented in DNF (To, Pontelli, and Son 2009) relies on explicit disjunctive formulae. The investigation presented in (To, Pontelli, and Son 2009) also discusses in more details the advantages and disadvantages of each of the aforementioned representations. It is worth mentioning that, regardless of the representation of belief states, all planners have difficulties scaling up when the size of the initial belief state is large. For example, all planners fail to find a solution of a modified version of the coins-21 problem instance from the IPC-2006 competition, either because they run out of memory or the plan computation exceeds reasonable time limit. The initial belief state of this problem contains \( 10^{10} \) states, compared to the “easy instances” in the same domain—e.g., the coins-20 instance has an initial belief state containing less than \( 10^9 \) possible states, and can be solved by all planners in less than two minutes.

The above issues motivated us to investigate alternative representations of belief states in conformant planning. Inspired by recent developments in other areas (e.g., the d-DNNF representation of SAT (Darwiche 2001)) and aware of the difficulties involved in using OBDD in computing the successor state, our goal is to identify a belief state representation with two desirable properties. First, the size of the representation should be minimal (as defined later). Second, the representation should facilitate a simple and efficient way for determining the satisfaction of a set of literals given a belief state.

In this paper, we investigate two different CNF-based belief state representations. The first representation employs prime implicates, called pi-formula. In this representation, a formula is replaced by its set of prime implicates or its pi-form. This representation exhibits a number of desirable properties. The complexity of checking tautological in a pi-form of a formula is linear for a literal (and polynomial for a clause) in the number of the propositions. Second, the representation is unique among the set of equivalent CNF formulae, making tractable the problem of checking for repetitions of belief states in a search tree. Third, this representation is, in many cases, very compact. Finally, the computation of the successor belief states under this representation can be done efficiently. The main disadvantage of this representation is that the size of the pi-form of a formula is sometimes very large. To this end, we investigate an alternative representa-
tion, called minimal CNF, whose size is often significantly smaller than its equivalent pi-form and can compensate for the pi-form representation in many cases.

In this paper, we provide a formal description of the two representations and design a best-first search conformant planner (PIP) that adopts them. We provide an experimental comparison of PIP against other planners. Our experiments show that PIP is comparable in speed with state-of-the-art planners and scales up better in domains rich in disjunctive information about the initial belief state.

Background: Conformant Planning

A planning problem is a tuple \( P = \langle F, O, I, G \rangle \), where \( F \) is a set of propositions, \( O \) is a set of actions, \( I \) describes the initial state of the world, and \( G \) describes the goal. A literal is either a proposition \( p \in F \) or its negation \( \neg p \). \( \ell \) denotes the complement of a literal \( \ell \). For a set of literals \( L \), \( L = \{ \ell \mid \ell \in L \} \). A conjunction of literals is often represented as the set of its literals.

A set of literals \( X \) is consistent if there is no \( p \in F \) such that \( \{ p, \neg p \} \subseteq X \), and complete if, for each \( p \in F \), either \( p \in X \) or \( \neg p \in F \). A state \( s \) is a consistent and complete set of literals. A belief state is a set of states. We will often use lowercase (resp. uppercase) letter, possibly with indices, to represent a state (resp. a belief state).

Each action \( a \) in \( O \) is associated with a precondition \( \phi \) (denoted by \( \text{pre}(a) \)) and a set of conditional effects of the form \( \psi \rightarrow \ell \) (also denoted by \( a : \psi \rightarrow \ell \)), where \( \phi \) and \( \psi \) are sets of literals and \( \ell \) is a literal.

A state \( s \) satisfies a literal \( \ell \), denoted by \( s \models \ell \), if \( \ell \in s \). A state \( s \) satisfies a conjunction of literals \( X \), denoted by \( s \models X \), if it satisfies every literal belonging to \( X \). The satisfaction of a formula in a state is defined in the usual way. Likewise, a belief state \( S \) satisfies a literal \( \ell \), denoted by \( S \models \ell \), if \( s \models \ell \) for every \( s \in S \).

Given a state \( s \), an action \( a \) is executable in \( s \) if \( s \models \text{pre}(a) \). The effects of executing \( a \) in \( s \) is

\[
e(a, s) = \{ \ell \mid \exists (a : \psi \rightarrow \ell), s \models \psi \}
\]

The transition function, denoted by \( \Phi \), in the planning domain of \( P \) is defined by

\[
\Phi(a, s) = \begin{cases} 
  s \cup e(a, s) & \text{if } s \models \text{pre}(a) \\
  \bot & \text{otherwise}
\end{cases}
\]

where \( \bot \) denotes a failed state.

We can extend the function \( \Phi \) to define \( \hat{\Phi} \), a transition function which maps sequences of actions and belief states to belief states. \( \hat{\Phi} \) is used to reason about the effects of plans. Let \( S \) be a belief state. We say that an action \( a \) is executable in a belief state \( S \) if it is executable in every state belonging to \( S \). Let \( \{ a_1, \ldots, a_n \} \) be a sequence of actions:

- If \( n = 0 \) then \( \hat{\Phi}([], S) = S \);
- If \( n > 0 \) then
  - if \( \hat{\Phi}(\{ a_1, \ldots, a_{n-1} \}, S) = \bot \) or if \( a_n \) is not executable in \( \hat{\Phi}(\{ a_1, \ldots, a_{n-1} \}, S) \), then \( \hat{\Phi}(\{ a_1, \ldots, a_n \}, S) = \bot \);
  - if \( \hat{\Phi}(\{ a_1, \ldots, a_{n-1} \}, S) \neq \bot \) and \( a_n \) is executable in \( \hat{\Phi}(\{ a_1, \ldots, a_{n-1} \}, S) \), then
    \[
    \hat{\Phi}(\{ a_1, \ldots, a_n \}, S) = \{ \Phi(a_n, s') \mid s' \in \hat{\Phi}(\{ a_1, \ldots, a_{n-1} \}, S) \}.
    \]

The initial state of the world \( (I) \) is a belief state and is represented by a formula. In all benchmarks, \( I \) consists of a conjunction of a set of literals, a set of oneof-\( I \)-statements—representing an exclusive-or of its components—and a set of or-\( I \)-statements—representing the logical or of its components. By \( S_f \) we denote the set of all states satisfying \( I \). The goal description \( G \) can contain literals and or-\( I \)-clauses.

A sequence of actions \( [a_1, \ldots, a_n] \) is a solution of \( P \) if \( \hat{\Phi}(\{ a_1, \ldots, a_n \}, S_f) \) satisfies the goal \( G \). In this paper, for an action \( a \), we will denote with \( \mathcal{C}_a \) the set of conditional effects of \( a \).

The function \( \hat{\Phi} \) can be used in the implementation of a best-first search based conformant planner. As we have discussed earlier, one of the important factors to this effort lies in the representation of belief states.

CNF Representations for Belief States

In this section, we explore the use of prime implicates in representing belief states for the development of a conformant planner. Observe that the development of a new representation of belief states needs to come with a set of operations such as checking for the satisfaction of a condition and/or updating a belief state with a set of effects.

A clause \( \alpha \) is a set of fluent literals. \( \alpha \) is tautological if \( \{ f, \neg f \} \subseteq \alpha \) for some \( f \in F \) and it is a unit clause if \( |\alpha| = 1 \). A CNF formula is a set of clauses. A literal \( l \) is in a CNF formula \( \varphi \), denoted by \( l \in \varphi \), if there exists \( \alpha \in \varphi \) such that \( l \in \alpha \). By \( \varphi_1 \) (resp. \( \varphi_2 \)) we denote the set of clauses in \( \varphi \) which contain \( l \) (resp. \( \bot \)).

A clause \( \alpha \) subsumes a clause \( \beta \) (or \( \beta \) is subsumed by \( \alpha \)) if \( \alpha \subseteq \beta \). Given a CNF formula \( \varphi \), a clause \( \alpha \) in \( \varphi \) is said to be trivially redundant for \( \varphi \) if it is tautological or it is subsumed by another clause in \( \varphi \). The technique of simplifying a CNF formula by removing subsumed clause(s) from that formula is called subsumption.

A clause \( \alpha \) is said to be resolvable with another clause \( \beta \) if there exists a literal \( \ell \) such that \( \ell \in \alpha \cap \beta \), and their resolvent \( \alpha \cup \beta \), defined by \( \alpha \cup \beta = (\alpha \setminus \{ \ell \}) \cup (\beta \setminus \{ \ell \}) \), is a non-tautological clause. In this case, we say that \( \alpha \) is resolvable with \( \beta \) on \( \ell \). Observe that, if \( \alpha \) and \( \beta \) are two clauses in a CNF formula \( \varphi \) and there exists a clause in \( \varphi \) which is subsumed by \( \alpha \cup \beta \), then \( \varphi \) can be simplified to an equivalent smaller formula by replacing all the clauses subsumed by \( \alpha \cup \beta \) with \( \varphi \) with this resolvent. This technique is referred to as subsumable resolution and \( \alpha \cup \beta \) is called a subsumable resolvent. For example, applying subsumable resolution to the set \( \{ \{ f, g \}, \{ f, h, \neg g \} \} \) results in \( \{ \{ f, g \}, \{ f, h \} \} \), whereas applying this technique to \( \{ \{ f, g \}, \{ f, g \}, \{ f, h \} \} \) returns \( \{ \{ f \} \} \). For two sets of clauses \( \varphi \) and \( \psi \), we denote with \( \varphi \psi = \alpha \beta \) is resolvable with \( \beta \psi \) with this resolvent.

For two CNF formulæ \( \varphi = \{ \{ \alpha_1, \ldots, \alpha_n \} \} \) and \( \psi = \{ \beta_1, \ldots, \beta_m \} \), the cross-product of \( \varphi \) and \( \psi \), denoted by \( \varphi \times \psi \), is the CNF-formula defined by \( \{ \alpha \cup \beta \mid \alpha \in \varphi, \beta \in \psi \} \). If either \( \varphi \) or \( \psi \) is empty then \( \varphi \times \psi = \emptyset \). The reduced-cross-product of \( \varphi \) and \( \psi \), denoted by \( \varphi \odot \psi \), is the CNF-formula obtained from \( \varphi \times \psi \) by removing all trivially redundant clauses in \( \varphi \times \psi \).

For a set of CNF formulæ \( \Psi = \{ \varphi_1, \ldots, \varphi_n \} \), \( \times[\Psi] \)
(resp. $\otimes[\Psi]$) denotes $\varphi_1 \times \varphi_2 \times \ldots \times \varphi_n$ (resp. $\varphi_1 \otimes \varphi_2 \otimes \ldots \otimes \varphi_n$). It is easy to see that both $\times[\Psi]$ and $\otimes[\Psi]$ are a CNF-formula equivalent to $\bigvee_{i=1}^n \varphi_i$.

A clause $\alpha$ is said to be an implicate of a formula $\varphi$ if $\varphi \models \alpha$. It is a prime implicate of $\varphi$ if there is no other implicate $\beta$ of $\varphi$ such that $\beta$ subsumes $\alpha$. Obviously, if a unit clause is an implicate of a formula then it is also a prime implicate of that formula. We denote the set of prime implicates of a formula $\varphi$ by $PI(\varphi)$. Clearly, a CNF formula $\varphi$ is in prime implicate form (pi-formula, for short) if $\varphi = PI(\varphi)$. One can prove the following proposition

**Proposition 1.** The reduced-cross-product of a set of pi-formulae is a pi-formula.

Thus, the pi-formula of the disjunction of a small set of pi-formulae can be computed in polynomial time.

**Prime Implicate Representation**

**Definition 1.** A PI-state is a pi-formula. A set of PI-states is called a PI-belief state.

It is easy to see that if $\varphi$ is a pi-formula, checking whether a literal $\ell$ is satisfied by $\varphi$ can be done in linear time in the size of $\varphi$ (the set of propositions).

We will now specify how the function $\Phi$ can be computed given that a belief state is represented by a PI-state.

**Definition 2.** Let $\varphi$ be a PI-state and $\ell$ be a literal. The update of $\varphi$ by $\ell$, denoted by $upd_{\varphi}(\ell)$, is defined as follows:

$$ upd_{\varphi}(\ell) = (\varphi \setminus (\varphi \cup \ell)) \land \ell $$

Intuitively, $upd_{\varphi}(\ell)$ encodes the PI-state after execution of an action, that causes $\ell$ to be true, in $\varphi$. For example,

- $upd_{\varphi}(\varphi, \ell_1)$ is a PI-state; and
- $upd_{\varphi}(\varphi, \ell_1, \ell_2) = upd_{\varphi}( upd_{\varphi}(\varphi, \ell_1), \ell_2) , \ell_1 \neq \ell_2$.

The above proposition shows the result of updating a PI-state $\varphi$ using a consistent set of literals $L$ is independent from the order in which the various literals of $L$ are introduced. For a consistent set of literals $L$, we define $upd_{\varphi}(\varphi, L) = upd_{\varphi}( upd_{\varphi}(\varphi, \ell), L \setminus \{\ell\} )$ for any $\ell \in L$ if $L \neq \emptyset$ and $upd_{\varphi}(\varphi, \emptyset) = \varphi$.

Let us now define the transition function $\Phi_{PI}$ for PI-states. Given an action $a$ with the precondition $pre(a)$, its set of conditional effects $C_a$, and a PI-state $\varphi$, we need to define the successor PI-state $\Phi_{PI}(a, \varphi)$. Note that, when computing $\Phi_{PI}(a, \varphi)$, for each $\psi \rightarrow \ell$ in $C_a$, there are three cases that need to be considered:

- $\varphi \models \psi$
- $\varphi \models \neg \psi$
- $\varphi \not\models \psi$ and $\varphi \not\models \neg \psi$

As an example, if $\varphi = p \land q$ and $\psi = r$, then we have $\varphi \not\models \psi$ and $\varphi \not\models \neg \psi$. In order to define $\Phi_{PI}$, we need the following definition.

**Definition 3.** Let $\varphi$ be a PI-state and $\gamma$ a consistent set of literals. The enabling form of $\varphi$ w.r.t. $\gamma$, denoted by $\varphi \oplus \gamma$, is a PI-belief state defined by

$$ \varphi \oplus \gamma = \{ \{ \varphi \} \} $$

$$ PI(\varphi \land \gamma), PI(\varphi \land \neg \gamma) \} $$

where $\neg \gamma$ is the clause $\{ \ell \mid \ell \in \gamma \}$. It is easy to see that $\varphi \oplus \gamma$ is a set of (at most two) PI-states such that for every $\delta \in \varphi \oplus \gamma$, $\delta \models \gamma$ or $\delta \models \neg \gamma$. We can prove the following:

**Proposition 3.** Let $\varphi$ be a PI-state and let $\gamma$ be a consistent set of literals. Let $\sigma$ be the union set of all literals in unit clauses in $\varphi$:

- $\varphi \oplus \gamma = \varphi \oplus (\gamma \setminus \sigma)$;
- If $\gamma$ is a consistent set of literals, then $\varphi \oplus \gamma$ can be computed in polynomial time; and
- If $\gamma$ is a consistent set of literals in $\varphi \setminus \gamma$ is bounded by a constant, then $\varphi \oplus \gamma$ can be computed in polynomial time.

For a PI-belief state $\Psi$, let $\Psi + \gamma = \bigcup_{\varphi \in \Psi} (\varphi \oplus \gamma)$.

**Proposition 4.** Let $\varphi$ be a PI-state (resp. PI-belief state). If $\gamma$ is a consistent set of literals, then $\varphi \oplus \gamma$ (resp. $\Psi + \gamma$) is a PI-belief state equivalent to $\varphi$ (resp. $\Psi$). If $\gamma_1$ and $\gamma_2$ are two consistent sets of literals, then

$$ (\varphi \oplus \gamma_1) \oplus \gamma_2 = (\varphi \oplus \gamma_2) \oplus \gamma_1. $$

**Definition 4.** Let $a$ be an action with the set of conditional effects $C_a$. A PI-state $\varphi$ is said to be enabling for $a$ if for every conditional effect $\psi \rightarrow \ell$ in $C_a$, either $\varphi \models \psi$ or $\varphi \models \neg \psi$. A PI-belief state $\Psi$ is enabling for $a$ if every PI-state in $\Psi$ is enabling for $a$.

For an action $a$ and a PI-state $\varphi$, let $enb_{\varphi}(a, \varphi) = ((\varphi \oplus \psi_1) \oplus \ldots) \oplus \psi_k$ where $C_a = \{ \psi_1 \rightarrow \ell_1, \ldots, \psi_k \rightarrow \ell_k \}$.

**Proposition 5.** For every PI-state $\varphi$ and action $a$, $enb_{\varphi}(a, \varphi)$ is a PI-belief state which is equivalent to $\varphi$ and enabling for $a$.

For an action $a$ and a PI-state $\varphi$, the effect of $a$ in $\varphi$, denoted $e(a, \varphi)$, is defined as follows:

$$ e(a, \varphi) = \{ \ell \mid \psi \models \ell \in C_a, \varphi \models \psi \}. $$

We are now ready to define the function $\Phi_{PI}$.

**Definition 5.** Let $\varphi$ be a PI-state and let $a$ be an action. $\Phi_{PI}(a, \varphi)$ denotes the transition function for PI-states:

$$ \Phi_{PI}(a, \varphi) = \begin{cases} \ominus\{upd_{\varphi}(\phi, e(a, \phi)) \mid \phi \in enb_{\varphi}(a, \varphi)\} & \text{if } \varphi \models pre(a) \\ \perp & \text{otherwise} \end{cases} $$

The computation of $\Phi_{PI}$ can be done efficiently under reasonable assumption, as stated next.

**Proposition 6.** Let $\varphi$ be a PI-state and let $a$ be an action whose conditional effects are $C_a = \{ \psi_i \rightarrow \ell_i \mid i = 1, \ldots, k \}$. The computation of $\Phi_{PI}(a, \varphi)$ can be done in polynomial time in the size of $\varphi$ if $k$ and $|\psi_i|$, $\forall i = 1, \ldots, k$, are bounded by a constant.

Observe that, since $k$ and the size of every $\psi_i$ is usually small, the assumption can be acceptable and computing
\( \Phi_{PI}(a, \varphi) \) largely depends on the size of \( \varphi \). \( \Phi_{PI} \) can be extended to define \( \hat{\Phi}_{PI} \), which allows us to reason about the effects of sequences of actions, in the same manner \( \hat{\Phi} \) is defined. The next theorem shows that \( \hat{\Phi}_{PI} \) is equivalent to the complete semantics defined by \( \hat{\Phi} \). Thus, any planner using \( \hat{\Phi}_{PI} \) in its search for solutions will be sound and complete.

**Theorem 1.** Let \( \varphi \) be a PI-state and \([a_1, \ldots, a_n]\) be an action sequence. Then, \( \hat{\Phi}_{PI}([a_1, \ldots, a_n], \varphi) \equiv \hat{\Phi}(([a_1, \ldots, a_n], BS(\varphi)) \) where \( BS(\varphi) \) denotes the set of states satisfying \( \varphi \).

The prime implicature representation has nice properties, e.g., checking entailment of a set of literals or a clause can be done efficiently and the reduced-cross-product of a set of PI-states will result in a PI-state. Moreover, since the PI-state equivalent to a given CNF formula is unique, it is easy to ensure that no PI-state is explored more than once in an implementation.

The critical problem with this representation lies in the fact that the complexity of the function \( \Phi_{PI} \) depends on the size of the PI-state which, unfortunately, can be much larger than the size of an equivalent CNF formula. For example, for \( \varphi = \{\{f, \neg g\}, \{g, \neg h\}, \{\neg f, h\}\} \), we have that \( PI(\varphi) = \{\{f, \neg g\}, \{g, \neg h\}, \{\neg f, h\}, \{\neg f, g\}, \{\neg g, h\}\} \) which is twice as big as \( \varphi \). For this reason, we investigate the second CNF-representation.

**Minimal CNF Representation**

**Definition 6.** A CNF formula \( \varphi \) is minimal if

- \( \varphi \) does not contain a trivially redundant clause; and
- \( \varphi \) does not contain two clauses \( \gamma \) and \( \delta \) such that \( \gamma \) is resolvable with \( \delta \) and \( \gamma | \delta \) subsumes a clause in \( \varphi \).

A CNF-state is a minimal CNF formula. A set of CNF-states is called a CNF-belief state.

Intuitively, a CNF-state \( \varphi \) is minimal in the sense that it does not contain trivially redundant clauses and it cannot be simplified by subsumable resolution. Observe that a PI-state is also a CNF-state, but the converse is not necessarily true. Furthermore, for one CNF formula \( \varphi \), the PI-state equivalent to \( \varphi \) is unique but there can be more than one CNF-state equivalent to \( \varphi \). In the following, we write \( \text{min}(\cdot) \) to denote an idempotent function that converts an arbitrary CNF formula \( \varphi \) to an equivalent CNF-state. For CNF-states, updating a state with a literal is defined as follows.

**Definition 7.** Let \( \varphi \) be a CNF-state and \( l \) be a literal. The update of \( \varphi \) by \( l \), denoted by \( \text{upd}(\varphi, l) \), is defined as follows:

\[
\text{upd}(\varphi, l) = \text{min}(\varphi \setminus (\varphi \cup \{l\})) \land l \land \varphi(l)
\]

Similar to Proposition 2, we can extend \( \text{upd} \) to define the updating of a CNF-state by a consistent set of literals by \( \text{upd}(\varphi, L) = \text{upd}(\varphi \setminus \{l\}) \cup \{l\} \) for some \( l \in L \). Using this definition, most of the operations on CNF-states and CNF-belief states can be defined similarly to the operations on PI-states and PI-belief states (Definitions 3-5). Due to lack of space, we omit their precise formulation. Let us note that this allows us to define a function, \( \hat{\Phi}_{CNF} \), for computing the result of a sequence of actions applied on a CNF-state. Furthermore, this function is equivalent to \( \hat{\Phi}_{PI} \).

**PIP—a Planner with Dynamic Representation Selection**

In this section, we describe a conformance planner, called PIP, that employs both the PI-state and the CNF-state in its representation of belief states. PIP is a heuristic best-first search, progression-based planner. We develop PIP from the source code of Dnf (To, Pontelli, and Son 2009).

Since preprocessing a CNF formula to a compact CNF form makes the computation of its prime implicates more efficient, first we start computing the minimal CNF formula encoding \( I \) using a fixed-point algorithm of subsumption, subsumable resolution, and unit propagation (Piette et al. 2008) whose running time is polynomial in the size of the set of propositions. The resulting CNF-state is fed to a test phase (TestPhase). The function TestPhase decides whether the CNF-state representation or the PI-state representation should be used as follows.

- The minimal CNF representation will be selected if the computation of the initial PI-state takes too long. More precisely, if it is greater than \( |F|^3 \times T_I \), where \( F \) is the set of propositions and \( T_I \) is the time spent for computing CNF-state from the CNF-formula encoding \( I \).

- The minimal CNF representation will be used if, within the exploration of a small number of belief states \( (N) \) (in our experiment, we set \( N = \min(10, |F|) \)), a pilot search for a plan using PI representation takes longer time than that using CNF representation. In addition, if the total size of the PI-states generated by the pilot search using PI representation is not smaller than twice of the total size of the CNF-states generated by the other pilot search then the minimal CNF representation is used.

- Otherwise, the PI-state representation will be chosen.

**Algorithm 1** Search(\( F,O,I,G \))

1: **Input:** Problem \((F,O,I,G)\)
2: **Output:** A plan if exists
3: Compute CNF-state \( \varphi_I \) from \( I \)
4: Create an empty queue \( Q \) and let \( P = \text{null} \)
5: Set \((P, Q, PI) = \text{TestPhase}(F, O, \varphi_I, G)\)
6: if \( P \neq \text{null} \) then
7: \hspace{1em} return \( P \)
8: end if
9: if \( PI = \text{true} \) then
10: \hspace{1em} return SearchPI(F, O, Q, G)
11: else
12: \hspace{1em} return SearchCNF(F, O, Q, G)
13: end if

The PIP will commit to a representation based on the result of the test phase, then will use the corresponding transition function to search for a plan. The overall algorithm is given in Algorithm 1 where SearchPI(\( F,O,Q,G \)) and SearchCNF(\( F,O,Q,G \)) implement the best-first search engine using the PI-state and the CNF-state representations. For computing the PI-states, we use an incremental algorithm for computing prime implicates, called IPIA (de Kleer 1992).
Proposition 7. Given two PI-states \(\phi\) and \(\psi\) and two clauses \(c_1\) and \(c_2\) such that \(c_1 \in \phi\) and \(c_2 \in \psi\), it holds that:

- If \(c_1 \subseteq c_2\) then \(c_2\) is a prime implicite of \(\phi \otimes \psi\).
- If \(c_1\) is a unit clause, \(c_1 \cup c_2\) is not tautological, and \(c_2\) does not belong to the previous case then \(c_1 \cup c_2\) is a prime implicite of \(\phi \otimes \psi\).

Proposition 7 allows us to reduce subsumption checking and avoid the creation of redundant clauses.

**Table 1: Benchmarks from Literature**

<table>
<thead>
<tr>
<th>Problem</th>
<th>PIP</th>
<th>DNF</th>
<th>CPA</th>
<th>CFF</th>
<th>POND</th>
</tr>
</thead>
<tbody>
<tr>
<td>block-1</td>
<td>0.587*</td>
<td>0.677</td>
<td>0.6844</td>
<td>0.085</td>
<td>0.0226</td>
</tr>
<tr>
<td>block-2</td>
<td>0.8318*</td>
<td>0.7238</td>
<td>0.7614</td>
<td>0.223</td>
<td>TO</td>
</tr>
<tr>
<td>block-3</td>
<td>TO*</td>
<td>216.1/331</td>
<td>OM</td>
<td>48/80</td>
<td>TO</td>
</tr>
<tr>
<td>bomb-50-10</td>
<td>1.249/0</td>
<td>1.289/0</td>
<td>21/58</td>
<td>F</td>
<td>1.16/90</td>
</tr>
<tr>
<td>bomb-100-20</td>
<td>2.481/90</td>
<td>2.691/90</td>
<td>110/110</td>
<td>F</td>
<td>34/190</td>
</tr>
<tr>
<td>bomb-100-20</td>
<td>4.65/180</td>
<td>5.15/180</td>
<td>244/118</td>
<td>F</td>
<td>28/180</td>
</tr>
<tr>
<td>cc-40</td>
<td>1.11/1075</td>
<td>7.39/153</td>
<td>87/18</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>cc-64-3</td>
<td>1.53/283</td>
<td>21.24/87</td>
<td>F</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>cc-87-43</td>
<td>2.19/387</td>
<td>40.4/124</td>
<td>F</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>coins-15</td>
<td>1.2997*</td>
<td>1.07/67</td>
<td>7.562</td>
<td>0.1279</td>
<td>2.689</td>
</tr>
<tr>
<td>coins-20</td>
<td>2.46/146*</td>
<td>1.44/99</td>
<td>17/105</td>
<td>0.15/107</td>
<td>16/143</td>
</tr>
<tr>
<td>coins-21</td>
<td>OM*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>ds-6-5</td>
<td>26.8/464</td>
<td>135/70</td>
<td>235/1971</td>
<td>98.5/347</td>
<td>TO</td>
</tr>
<tr>
<td>ds-8-5</td>
<td>211/105</td>
<td>658/878</td>
<td>2152/541</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>ds-10-3</td>
<td>209/110</td>
<td>1933/880</td>
<td>4694/648</td>
<td>2388/1360</td>
<td>TO</td>
</tr>
<tr>
<td>1d-4-2</td>
<td>9.81/10*</td>
<td>2.18/64</td>
<td>2.24/52</td>
<td>12.44/72</td>
<td>TO</td>
</tr>
<tr>
<td>1d-5-2</td>
<td>81.6/312*</td>
<td>5.85/122</td>
<td>11.24/88</td>
<td>188/126</td>
<td>TO</td>
</tr>
<tr>
<td>1d-6-2</td>
<td>434/492*</td>
<td>15.4/186</td>
<td>33.7/124</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>1d-6-3</td>
<td>OM*</td>
<td>508/180</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>gripper-50</td>
<td>3.62/298</td>
<td>8.71/234</td>
<td>106.7/100</td>
<td>52.1/198</td>
<td>3.2/294</td>
</tr>
<tr>
<td>gripper-60</td>
<td>5.28/358</td>
<td>13.7/278</td>
<td>189/3246</td>
<td>91.6/238</td>
<td>6.4/334</td>
</tr>
<tr>
<td>gripper-70</td>
<td>7.17/418</td>
<td>16.5/316</td>
<td>278/650</td>
<td>151/278</td>
<td>11/144</td>
</tr>
<tr>
<td>gripper-80</td>
<td>10.36/478</td>
<td>22.58/350</td>
<td>432/166</td>
<td>234/318</td>
<td>18/474</td>
</tr>
<tr>
<td>lag-4-3-1</td>
<td>2.73/4</td>
<td>3.434</td>
<td>4.848</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>lag-5-3-3</td>
<td>10.9/14</td>
<td>13.9/6</td>
<td>206</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>lag-6-3-3</td>
<td>46/6</td>
<td>52/6</td>
<td>82/6</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>lag-7-3-3</td>
<td>171/52</td>
<td>178/14</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>push-5-6</td>
<td>23.6/312</td>
<td>24.1/783</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>push-6-5</td>
<td>51/226</td>
<td>58/143</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>push-7-5</td>
<td>163/956</td>
<td>130/1767</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>push-7-7</td>
<td>391/774</td>
<td>494/2565</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>raok-2</td>
<td>1.65/27</td>
<td>0.57/26</td>
<td>1.1/32</td>
<td>0.04/21</td>
<td>0.07/34</td>
</tr>
<tr>
<td>raok-3</td>
<td>2.7177*</td>
<td>1.65/153</td>
<td>3.8/152</td>
<td>0.25/66</td>
<td>12/102</td>
</tr>
<tr>
<td>raok-4</td>
<td>TO*</td>
<td>TO</td>
<td>TO</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>sortnet-5</td>
<td>1.25/5*</td>
<td>0.94/15</td>
<td>0.94/12</td>
<td>0.26/15</td>
<td>NA</td>
</tr>
<tr>
<td>sortnet-10</td>
<td>22.15/4*</td>
<td>1.68/31</td>
<td>3.18/39</td>
<td>OM</td>
<td>NA</td>
</tr>
<tr>
<td>sortnet-15</td>
<td>255/119*</td>
<td>35.9/181</td>
<td>244/65</td>
<td>NA</td>
<td>0.14/65</td>
</tr>
<tr>
<td>sortnet-5</td>
<td>1.58/10*</td>
<td>2.57/10</td>
<td>OM</td>
<td>1.92/10</td>
<td>2.9/10</td>
</tr>
<tr>
<td>sortnet-6</td>
<td>22.9/15*</td>
<td>397/15</td>
<td>OM</td>
<td>18.1/15</td>
<td>TO</td>
</tr>
<tr>
<td>sortnet-7</td>
<td>537/21*</td>
<td>OM</td>
<td>OM</td>
<td>91/21</td>
<td>TO</td>
</tr>
<tr>
<td>sortnet-8</td>
<td>69/702/10</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
</tr>
<tr>
<td>ucts-c-3</td>
<td>4.22/3*</td>
<td>0.54/3</td>
<td>1.19/3</td>
<td>0.15/3</td>
<td>NA</td>
</tr>
<tr>
<td>ucts-c-4</td>
<td>TO*</td>
<td>0.56/6</td>
<td>18.3/6</td>
<td>0.47/7</td>
<td>NA</td>
</tr>
<tr>
<td>ucts-c-5</td>
<td>TO*</td>
<td>0.73/10</td>
<td>1.8/10</td>
<td>NA</td>
<td>F</td>
</tr>
</tbody>
</table>

**Heuristic**

PIP uses two search heuristics: the number of satisfied subgoals and the size of the CNF-state, i.e., the sum of the size of all clauses in the CNF formula. This heuristic may seem naive, nevertheless it is very close to that of DNF, a planner similar to PIP except for the belief state representation. This choice allows us to use DNF in our comparisons. The comparison of PIP with other planners is for proving that pi-formulæ can be used to build a competitive conformant planner.

**Experimental Evaluation**

We compare PIP with the following conformant planners: DNF (To, Pontelli, and Son 2009), CPA (Tran et al. 2009), CFF (Brafman and Hoffmann 2004), POND (Bryce, Kambhampati, and Smith 2006), and \(\tau_0\) (Palacios and Gefner 2007) using conformant planning benchmarks from literature. To the best of our knowledge, these planners currently yield the best performance in these domains. We also use a set of new benchmarks modified from those in literature by replacing one-of-clause(s) with or-clause(s) in the initial state description, to generate cases that are rich in disjunctive information. All the experiments have been carried out using a dedicated Intel Core 2 Dual 9400 2.66GHz 4GB Linux workstation. The time-out limit is set to 2 hours. The experimental results are reported in tables 1 and 2. We report the time and plan length for each planner. ‘OM’, ‘TO’, and ‘F’ denote out-of-memory, time-out, and abnormal termination of the planner. Due to space limitation, we only report the results of our experiments in a few large instances of each benchmark. In the following, we discuss each table and evaluate the strengths and weaknesses of PIP against other planners.

**Benchmarks From Literature**

Table 1 contains the results obtained from experiments on the domains block, dispose (ds-n-m), raokkeys (raok-n), and ucts-cycle (uts-c-n) used in IPC-2008, coins and sortnet are from IPC-2006, and bomb and gripper are from the authors of CFF. The remaining problems, including corner-cube (cc-n-m), dispose (1d-n-m), look-and-grab (lng-n-m), push (push-n-m), and sort-number (sort-n) are included in the package of \(\tau_0\).

The results show that PIP is the best in five (bomb, corner-cube, gripper, look-and-grab, and push) out of thirteen domains. DNF outperforms the others in three domains (dispose, dispose, and ucts-cycle). The overall performance of PIP is comparable to that of DNF for the benchmarks. Note that, DNF and CPA take advantage of the oneof-combination technique, aimed at reducing the disjunctive form of the initial belief state. Without this technique, these planners would have trouble dealing with several benchmark, such as coins, dispose, dispose, look-and-grab, and push. Although this technique does not harm the soundness and completeness of the planners, the initial data is not equivalent to the original. Thus, it is somewhat “unfair” to compare the effectiveness of the belief state representations under these conditions.
Our experiments reveal that the search trees of PIP and DNF for the instance problems of bomb and the first three instance problems of look-and-grab are the same but PIP performs better. Hence, we can conclude that the prime implicate representation is better than the DNF representation for these cases. For sort-number, t 0 is the best on most instances but only PIP is able to deal with the largest instance. t 0 is also the best on coins and raookay. POND outperforms the others on block and sortnet but its performance is poor on the other domains. Note that, on the PIP column, "*" indicates that the minimal CNF representation has been selected. Observe that most of the best solutions are found by PIP using prime implicate representation.

### Challenging Problems

In this subsection, we introduce a new set of problems aimed at demonstrating situations where PIP can show its full potential. They are variants of problems in Table 1, obtained by replacing oneof statements with or statements of the same set of literals. This modification is carried out only for the cases where the problem description remains consistent. These variants are renamed by adding the prefix "or-" to their original name and they are shown in Table 2. Due to the large size of the disjunctive normal form formulae representing belief states in these problems, a DNF belief state representation based planner, e.g., DNF and CPA, provide much poorer performance compared to that on the original problems. Moreover, in these problems, the oneof-combination technique is not applicable, making the performance of DNF and CPA even worse. On the contrary, due to the capability of maintaining a compact size of CNF formulae representing belief states, PIP outperforms impressively not only DNF belief state representation based planners, but also all other competitive state-of-the-art conformant planners. It is worth mentioning that there is no significant difference between the performance of any other planner on these problems in comparison with that on the original problems.

### Conclusion and Future Work

In the paper, we showed that prime implicate formulae have the potential of offering highly desirable properties in the representation of belief states for conformant planning. We developed an effective complete transition function for computing successor belief states using the prime implicate representation. We also proposed an alternative compact CNF form, minimal CNF, for representing belief states in the case the prime implicate form is not suitable for them. Another complete transition function for this representation has also been provided. A planner that uses alternative representations, like PIP, appears to be a valid approach to combine strengths of different representations and to reduce their disadvantages. This has been validated by the experimental results. We also identified a set of problems where a planner based on conjunctive representations of belief states offers better results. The theoretical results shown in Propositions 1 and 7 may also be very useful in solving other problems which require computation of prime implicates.

Using different types of formulae to represent belief states in a conformant planner is a promising research direction.

### Table 2: Challenging Domains

<table>
<thead>
<tr>
<th>Problem</th>
<th>PIP</th>
<th>DNF</th>
<th>CPA</th>
<th>T0</th>
<th>CFF</th>
<th>POND</th>
</tr>
</thead>
<tbody>
<tr>
<td>or-coins-15</td>
<td>0.91/81*</td>
<td>OM</td>
<td>OM</td>
<td>0.112/27</td>
<td>2.36/89</td>
<td>10.6/124</td>
</tr>
<tr>
<td>or-coins-21</td>
<td>1478/7321*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-coins-23</td>
<td>4079/3185*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-coins-26</td>
<td>4482/3321*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-coins-29</td>
<td>9991/5368*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-coins-30</td>
<td>7026/11213*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-ds-6-5</td>
<td>4.83/358</td>
<td>OM</td>
<td>OM</td>
<td>208/6347</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-ds-8-3</td>
<td>27.85/392</td>
<td>OM</td>
<td>OM</td>
<td>278/761</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-ds-8-5</td>
<td>30.84/357</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-ds-10-5</td>
<td>171.1/2486</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-ds-10-9</td>
<td>198/2485</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-1d-5-2</td>
<td>4.56/344</td>
<td>34.1/94</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-1d-5-4</td>
<td>14.45/312*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-1d-6-2</td>
<td>10.41/492*</td>
<td>190.1/150</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-1d-6-4</td>
<td>33.24/492*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-1d-10-4</td>
<td>667.12/3806*</td>
<td>OM</td>
<td>OM</td>
<td>F</td>
<td>TO</td>
<td>OM</td>
</tr>
<tr>
<td>or-push-4-4</td>
<td>1.23/132</td>
<td>OM</td>
<td>OM</td>
<td>15.45/210</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-push-5-6</td>
<td>3.42/312</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-push-6-5</td>
<td>6.16/216</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-push-10-5</td>
<td>203.7/841</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
</tr>
<tr>
<td>or-push-10-8</td>
<td>255.3/1218</td>
<td>OM</td>
<td>OM</td>
<td>TO</td>
<td>TO</td>
<td>TO</td>
</tr>
</tbody>
</table>

For example, the performance of CNF and DNF suggests that a combination of the two techniques into one planner might yield interesting result. Finally, an improved heuristic is also desirable.

### References


Palacios, H., and Geffner, H. 2007. From Conformant into Classical Planning: Efficient Translations that may be Complete Too. In ICAPS.


