On Belief State Representation and Its Application in Conformant Planning

Son Thanh To and Tran Cao Son and Enrico Pontelli

Department of Computer Science
New Mexico State University
{sto | tson | epontell}@cs.nmsu.edu

Abstract

This paper proposes a general methodology for developing a complete transition function that can compute exact successor belief states under an arbitrary representation after execution of actions, and its application to conformant planning. The approach is built on an abstract notion of belief state representation, called $\mathcal{R}$-*state*, and a transition function $\Phi_{\mathcal{R}}$ for computing successor $\mathcal{R}$-states after an action is performed. The paper introduces a set of basic operations over $\mathcal{R}$-states as components of $\Phi_{\mathcal{R}}$, along with the corresponding abstract algorithms for the implementation of $\Phi_{\mathcal{R}}$. The paper then proposes the use of several compact logical formula forms, including *minimal-DNF*, *minimal-CNF*, and *prime implicates*, as possible representations of belief states. For each of these representations, the paper investigates the instantiation of the abstract function $\Phi_{\mathcal{R}}$ into a precise transition function, along with a discussion of the corresponding computational properties. Each of these representation method is deployed in a new separate conformant planner. The paper provides an evaluation of the effectiveness of this approach, by comparing one of the resulting planners—DNF, that employs minimal-DNF representation—against other state-of-the-art planners. The experiments show that DNF is highly competitive with other planners, thus providing a validation of the proposed approach. To emphasize the effectiveness of belief state representation, the paper proposes a new set of problems, that are beyond the capability of DNF and other planners, yet they can be solved by some of our planners using different representations, like minimal-CNF and prime implicates.

Keywords: Belief State Representation, Conformant Planning, Disjunctive Normal Form Formula, Conjunctive Normal Form Formula, Prime Implicates, Transition Function.

1. Introduction

Conformant planning [9, 20] is the problem of generating a sequence of actions that can achieve a given goal regardless of the fact that the knowledge about the initial state of the world is incomplete. As an example, consider a robot that needs to clean two rooms r_1 and r_2. Initially, the robot knows that it is in room r_1 but it does not know whether its trash bag is empty or not. The robot can *move* between the rooms, *vacuum* the room if its trash bag is empty, and *empty* the trash bag. Vacuuming a room results in the trash bag being full. An obvious solution for the robot is to empty the bag, vacuum room r_1, empty the bag, move to room r_2, and vacuum room r_2.

Conformant planning has been known to be one of the most challenging problems in automated planning [2, 10] due to conditional action effects in presence of incomplete information. Since its introduction, conformant planning has attracted the attention of several researchers—leading to the development of sophisticated state-of-the-art conformant planners, such as Conformant-FF (CFF) [12], KACMBP [7], POND [5],

T0 [18, 19], T1 [1], CPA[1] [29, 28], CPLS [15], and GC[LAMA] [16]. It is important to observe that most of the best performing planners are *best-first search* and *progression-based* planners, whose development starts with the selection of a representation language and the definition of a progression function that, given a state and an action, computes the next state(s) of the world.

To deal with incomplete information about the world, the notion of a *belief state*—defined as the set of possible states—has been introduced and is widely used in the literature for planning in presence of incomplete information [20, 3]. An advantage of this notion lies in its simplicity in representing and reasoning about effects of actions in the presence of incomplete information about the initial state. Indeed, any formalization of *Reasoning about Action and Change (RAC)* with complete information about the world state can be straightforwardly generalized to deal with the incompleteness of the state by dealing with each individual possible state in the belief state separately and maintaining the set of resulting states (belief state). More precisely, given a transition function Φ that computes the resulting state after the execution of an action a in a state s, denoted by $\Phi(a, s)$, the function $\widehat{\Phi}(a, S) = \{\Phi(a, s) \mid s \in S\}$ characterizes the transitions between belief states and can be used for reasoning with incomplete information.

A challenging aspect in the use of belief states in conformant planning is their size—which is exponential in the number of propositions with unknown truth value. Different solutions for this problem have been proposed. For instance:

- Conformant-FF (CFF) [12] uses an implicit representation of a belief state as a sequence of actions from the initial state to the belief state;
- KACMBP [7] and POND [5] employ a BDD-based representation [4];
- CPA [29, 28] approximates a belief state by a set of subsets of states (*partial states*);
- T0 [18, 19] transforms the conformant planning problem to classical planning problem;
- Similar to CFF, T1 [1] represents a belief state by a subset of it with a set of literals, called *tentative literals*, whose satisfaction in the belief state is verified in the same manner as CFF does.

This paper presents a novel approach to the aforementioned problem, by proposing different compact propositional logical formulae as alternative belief state representations; we show that the use of a compact belief state representation can significantly improve the planner's performance and scalability. For each representation, this work develops a transition function for efficiently computing the exact successor belief states after the execution of actions. The paper presents a *general methodology* for developing a complete transition function for an arbitrary propositional representation. It is worth noting that defining such a transition function for even a concrete representation other than belief states is particularly challenging and not explored by other works. It is inherently challenging due to context-dependent action effects in presence of incomplete information, i.e., the effect of actions depends on the true state while the true state is uncertain among the set of states represented in a formula.

We start with the development of a *generic transition function*, that can be used with different representations, for computing successor belief states in the form of each representation. The new transition function is defined over classes of formulae—called $\mathcal{R}$*-states*, representing belief states—and deals with the incomplete information on a *need-to-know basis*. We devise an algorithm for the implementation of the transition function that can be modularly instantiated, depending on the concrete belief state representation. We identify different features that affect the computation of the transition function and prove that the algorithm is *optimal* in term of the *minimum* number of intermediate formulae necessarily defined in the function, a

[1] Different versions of CPA have been developed. In this paper, whenever we refer to CPA, we will imply CPA(H), the version used in IPC 2008, http://ippc-2008.loria.fr/wiki/index.php/Main_Page.

factor that impacts the complexity of the function.

We illustrate the new proposal by instantiating it in three different concrete representations of belief states:

- *minimal-DNF*, a compact form of disjunctive normal form formulae,
- *minimal-CNF*, a compact form of conjunctive normal form formulae, and
- *prime implicates*, a well-known, special form of minimal-CNF.

For each of these representations, we present the definition of the transition function. Furthermore, the complexity of the transition function for each of these representations will also be investigated in this work. We implement three heuristic conformant planning systems, called DNF, CNF, and PIP that employs the minimal-DNF, minimal-CNF, and prime implicates representations, respectively. Each of these planners use a heuristic function mostly based on the number of satisfied subgoals and the number of known literals. Let us note that these representations have been preliminarily and independently introduced in the context of conformant planning in our previous works [25, 26, 27].

To validate our approach, we compare our first planner, DNF, with other state-of-the-art conformant planners on a large set of diverse benchmarks that contains most benchmarks available in the literature. For a better evaluation of different approaches, we diversify the set of benchmarks with a new set of problems, obtained by extending several problems from the literature. The new constraints incorporated in these problems capture some real-world needs, and make the problems harder. The experiments show that DNF is very competitive with other planners.[2]

To emphasize the effectiveness of belief state representations, we also propose a new set of problems that are beyond the capability of DNF and most other previously developed planners. We show that CNF and PIP offer a superior performance on this set of problems because of representations based on conjunctive formulae. We discuss the reason for the dramatic difference between the performance of DNF and the performance of CNF and PIP. These results support the thesis that planners need to adapt the representation of the belief state depending on the specific class of problems being considered.

The paper is organized as follows. Section 2 presents key notions used in planning and in the development of the approach of this paper. Section 3 presents a novel methodology for development of a complete transition function for an arbitrary belief state representation. Section 4 investigates the minimal-DNF representation, the use of the methodology developed in Section 3 for a precise definition of the transition function for this representation, and implements it in the conformant planner DNF. Similarly to Section 3, section 5 and 6 investigate the minimal-CNF and prime implicate representations, respectively. Section 7 discusses state-of-the-art approaches to conformant planning. Section 8 discusses the experimental results, the criteria for evaluation of a representation, and some guidances for selection of a representation among the three representations investigated by this paper. Section 9 summarizes the main results of the paper. Finally, the proofs for most results, except several trivial ones, are presented in the Appendix.

2. Background: Conformant Planning, Propositional Logic, and Belief State Representation

In this section, we review the key notions in conformant planning and belief state representation.
A *conformant planning problem* P is a tuple of the form $P = \langle F, O, I, G\rangle$ where:[3]

[2] The systems and the benchmarks used in our experiments can be downloaded from `http://www.cs.nmsu.edu/~sto`.

[3] Propositions and actions with variables are viewed as shorthands of their ground instantiations.

- F, referred to as the *domain* of P, is a finite set of propositions; a *literal* is either an element $p \in F$ or its negation $\neg p$.
- O is a finite set of actions. Each action a in O is associated with a precondition, denoted by $pre(a)$, and a set of conditional effects C_a of the form $\psi \rightarrow \ell$ (also written as $a : \psi \rightarrow \ell$), where $pre(a)$ is a set of literals that can be viewed as the conjunctions of the literals in the set, ψ is another set of literals, and ℓ is a literal.

  We refer to ψ as the *condition* of the effect ℓ of action a (or simply a *e-condition* of a). The set of e-conditions of a is denoted by $\Psi(a)$, i.e.,

  $$\Psi(a) = \{\psi \mid \exists(a : \psi \rightarrow \ell)\}$$

  We often write $a : \psi \rightarrow \eta$ to denote the set of conditional effects of a with the same e-condition ψ—from now on referred to as a *combined conditional effect*.
- I is a formula over F describing the initial situation. I is a finite conjunction of literals, `oneof`-clauses, and `or`-clauses. Formally, a `oneof`-clause (resp. `or`-clause) is of the form $\texttt{oneof}(\varphi_1, \ldots, \varphi_k)$ (resp. $\texttt{or}(\varphi_1, \ldots, \varphi_k)$), where φ_i is a conjunction of literals for $i = 1, \ldots, k$.
- G is a propositional formula over F defining the goal.

The complement of a literal ℓ, denoted by $\bar{\ell}$, is the negation of ℓ—i.e., $\bar{\ell} = \neg\ell$, where $\overline{\neg p} = p$ for every p in F. For a set of literals L, $\overline{L} = \{\bar{\ell} \mid \ell \in L\}$. In this paper, we often identified the conjunction $\ell_1 \wedge \ldots \wedge \ell_k$ with set of literals $\{\ell_1, \ldots, \ell_k\}$. A set of literals X is said to be *consistent* if X does not contain a pair of complementary literals, i.e., for every literal ℓ in X, $\bar{\ell} \notin X$. A set of literals X is said to be *complete* if for each proposition p in F, $\{p, \neg p\} \cap X \neq \emptyset$.

A set of literals X satisfies a literal ℓ, denoted by $X \models \ell$, if $\ell \in X$. X satisfies a conjunction of literals $Y = \ell_1 \wedge \ldots \wedge \ell_k$, denoted by $X \models Y$, if $X \models \ell_i$ for all $1 \leq i \leq k$. X satisfies a `oneof`-clause $\texttt{oneof}(\varphi_1, \ldots, \varphi_k)$ if there exists j, $1 \leq j \leq k$, such that $X \models \varphi_j$ and $X \not\models \varphi_i$ for every $i \neq j$. X satisfies an `or`-clause $\texttt{or}(\varphi_1, \ldots, \varphi_k)$ if there exists some j, $1 \leq j \leq k$, such that $X \models \varphi_j$. A set of literals X satisfies the initial situation I if it satisfies every literal, `oneof`-clause, and `or`-clause in I.

The satisfaction of a propositional formula φ w.r.t. a set of literals X, denoted by $X \models \varphi$, is defined in the usual way. When $X \models \varphi$, we say that φ is true in X; when $X \models \neg\varphi$, we say that φ is false in X; otherwise, we say that φ is unknown in X.

A *state* s is a consistent and complete set of literals. A *belief state* is a set of states. A belief state S satisfies a formula φ, denoted by $S \models \varphi$ if $s \models \varphi$ for every $s \in S$. By S_I we denote the set of all states satisfying I; we will refer to S_I as the *initial belief state* of P.

Given a state s, an action a is *executable* in s if $s \models pre(a)$. The effect of executing a in s is

$$e(a, s) = \{\ell \mid \exists(a : \psi \rightarrow \ell).\ s \models \psi\} \tag{1}$$

$e(a, s)$ is the set of literals that will become true in the successor state after executing a in s. The transition function that returns the result of the execution of a in s, denoted by $\Phi(a, s)$, is defined by:

$$\Phi(a, s) = \begin{cases} s \setminus \overline{e(a, s)} \cup e(a, s) & s \models pre(a) \text{ and } e(a, s) \text{ is consistent} \\ \textit{undefined} & \text{otherwise} \end{cases} \tag{2}$$

Let S be a belief state and a be an action. We say that a is executable in S if it is executable in every state belonging to S.

Let P be a planning problem. The transition function of P that maps each pair composed of an action a and a belief state S to a belief state is defined as follows:

$$\Phi(a, S) = \begin{cases} \{\Phi(a, s) \mid s \in S\} & a \text{ is executable in } S \text{ and } \Phi(a, s) \text{ is defined for every } s \in S \\ \mathit{undefined} & \text{otherwise} \end{cases} \tag{3}$$

We can extend the function Φ to define $\widehat{\Phi}$, an extended transition function which maps each pair composed of a sequence of actions and a belief state to a belief state—necessary for reasoning about effects of plans. Let $\alpha_n = [a_1, \ldots, a_n]$ be a sequence of actions:

- If $n = 0$ then $\widehat{\Phi}([\,], S) = S$;
- If $n > 0$ then
  - if $\widehat{\Phi}(\alpha_{n-1}, S)$ is undefined then $\widehat{\Phi}(\alpha_n, S)$ is undefined;
  - if $\widehat{\Phi}(\alpha_{n-1}, S)$ is defined then

$$\widehat{\Phi}(\alpha_n, S) = \Phi(a_n, \widehat{\Phi}(\alpha_{n-1}, S))$$

  where $\alpha_{n-1} = [a_1, \ldots, a_{n-1}]$.

A sequence of actions $[a_1, \ldots, a_n]$ is a *solution* of P if $\widehat{\Phi}([a_1, \ldots, a_n], S_I)$ satisfies the goal G. A conformant planning problem P is said to be *consistent* if

1. The initial belief state S_I is non-empty, and
2. For every pair of an action a and a state s such that a is executable in s, $\Phi(a, s)$ is defined.

In this work, we assume that any given problem is consistent.

Example 1. Consider a slightly different version of the planning problem discussed in the introduction. The planning problem $P = \langle F, O, I, G \rangle$ is represented as follows:

- *Domain* F: F can be described using the following propositions:
  - $clean_i$: room r_i is clean ($i = 1, 2$).
  - at_i: the robot is at room r_i ($i = 1, 2$).
  - bag_is_empty: the vacuum bag is empty.

  Thus,

$$F = \{clean_1, clean_2, at_1, at_2, bag\_is\_empty\}$$

- *Initial belief state* I: Suppose that initially no room is clean, the vacuum bag is empty, and the robot is in one of the two rooms but it does not know exactly which one. I is described by the following set:

$$\{\neg clean_1, \neg clean_2, bag\_is\_empty, \texttt{oneof}(at_1, at_2)\}$$

  Observe that the uncertainty of the initial state lies in the location of the robot. Thus, the initial belief state can be easily computed as $S_I = \{s_1, s_2\}$, where

$$s_1 = \{\neg clean_1, \neg clean_2, at_1, \neg at_2, bag\_is\_empty\}$$
$$s_2 = \{\neg clean_1, \neg clean_2, \neg at_1, at_2, bag\_is\_empty\}$$

- *Actions* O: The robot can vacuum, move from a room to the other, and empty the vacuum bag. Then O can be described by

$$O = \{vacuum, move, empty\_the\_bag\}$$

where

  - $pre(vacuum) = bag\_is\_empty$
    $vacuum : at_1 \rightarrow \{clean_1, \neg bag\_is\_empty\}$
    $vacuum : at_2 \rightarrow \{clean_2, \neg bag\_is\_empty\}$
  - $pre(move) = \textbf{\textit{True}}$
    $move : at_1 \rightarrow \{at_2, \neg at_1\}$
    $move : at_2 \rightarrow \{at_1, \neg at_2\}$
  - $pre(empty\_the\_bag) = \textbf{\textit{True}}$
    $empty\_the\_bag : \textbf{\textit{True}} \rightarrow bag\_is\_empty$

One can easily compute that:

$$\begin{array}{lcl} e(vacuum, s_1) & = & \{clean_1\ \neg bag\_is\_empty\} \\ \Phi(vacuum, s_1) & = & \{clean_1, \neg clean_2, at_1, \neg at_2, \neg bag\_is\_empty\} \\ e(vacuum, s_2) & = & \{clean_2\ \neg bag\_is\_empty\} \\ \Phi(vacuum, s_2) & = & \{\neg clean_1, clean_2, \neg at_1, at_2, \neg bag\_is\_empty\} \end{array}$$

Hence,

$$\Phi(vacuum, S_I) = \left\{ \begin{array}{l} \{clean_1, \neg clean_2, at_1, \neg at_2, \neg bag\_is\_empty\}, \\ \{\neg clean_1, clean_2, \neg at_1, at_2, \neg bag\_is\_empty\} \end{array} \right\}$$

- *Goal* G: The goal is that both rooms are clean. Thus, $G = clean_1 \wedge clean_2$.

It is easy to see that $[vacuum, move, empty\_the\_bag, vacuum]$ is a solution of P.

The function Φ has been employed in the implementation of several heuristic conformant planners. Algorithm 1 illustrates a generic heuristic forward search-based conformant planner. As evident in Algorithm 1, a conformant planning system needs to adopt a representation for belief states. In the rest of this paper, we will propose a general methodology for the development of planners based on Algorithm 1 by using formulae for belief state representation. We will need the following notations.

Let φ be a formula. A state that satisfies φ is called a *model* of φ. The set of models of φ, denoted by $BS(\varphi)$, is the *belief state form of* φ, or *belief state of* φ for short. The belief state of a formula is unique. A formula φ is *satisfiable* iff φ has a model ($BS(\varphi) \neq \emptyset$). A formula ψ is true (or false) in φ, denoted by $\varphi \models \psi$ (resp. $\varphi \models \neg\psi$), iff ψ is true (resp. false) in every model of φ. Note that $\varphi \models \psi$ iff $BS(\varphi) \subseteq BS(\psi)$. φ and ψ are *equivalent*, denoted by $\varphi \equiv \psi$, iff $BS(\varphi) = BS(\psi)$ iff $\varphi \models \psi$ and $\psi \models \varphi$. ψ is *tautological* if it is true in every formula. A tautological formula is equivalent to $true$ and its belief state is the set of all possible states of the world. We have the following basic propositions that will be needed in this paper.

Proposition 1. *For a non-empty set of satisfiable formulae* $\{\varphi_1, \ldots, \varphi_n\}$ *and a formula* φ*, we have the following results:*

1. $BS(\varphi_1 \vee \ldots \vee \varphi_n) = BS(\varphi_1) \cup \ldots \cup BS(\varphi_n)$

2. $\varphi_1 \vee \ldots \vee \varphi_n \models \varphi$ *iff* $\forall \varphi_i \in \{\varphi_1, \ldots, \varphi_n\}.\ \varphi_i \models \varphi$

Proposition 2. *For a non-empty set of satisfiable formulae* $\{\varphi_1, \ldots, \varphi_n\}$*, the following holds*

$$BS(\varphi_1 \wedge \ldots \wedge \varphi_n) = BS(\varphi_1) \cap \ldots \cap BS(\varphi_n)$$

The above propositions are used regularly in this paper in the development of our theory for belief state representation.

Algorithm 1 Plan(F, O, I, G) {Best First Search Planner}

Input: A planning problem $\langle F, O, I, G \rangle$
Output: Solution if exists; No solution otherwise
Create priority queue Q
Initialize Q with start node $(S_I, [])$
Let $Exist = \{S_I\}$ {Set of belief states that have been generated}
while Q is not empty **do**
  Extract the first element $N = (S, CP)$ of Q {CP is the plan from S_I to S}
  if S satisfies G **then**
    return CP {the plan reaching the goal G}
  else
    for each action a such that $S \models pre(a)$ **do**
      Compute $S' = \Phi(a, S)$ {successor of S using a}
      if $S' \notin Exist$ **then**
        Insert $(S', CP \circ [a])$ in Q {heuristic of S' as priority in Q}
        $Exist = Exist \cup \{S'\}$
      end if
    end for
  end if
end while
return no solution

ψ is said to be *known* in φ if ψ is either true or false in φ. Otherwise, we say that ψ is *unknown* in φ. Observe that, ψ is unknown in φ iff there exist $s, s' \in BS(\varphi)$ such that ψ is true in s and ψ is false in s'.

Let φ be a propositional formula over F. $lit(\varphi)$ denotes the set of literals that appear in φ. $prop(\varphi) = \{p \in F \mid \{p, \neg p\} \cap lit(\varphi) \neq \emptyset\}$. Two formulae φ and ψ are said to be dependent if iff $prop(\varphi) \cap prop(\psi) \neq \emptyset$. Two formulae are said to be *independent* if they are not dependent. We also have the following propositions.

Proposition 3. *Let* φ *be a satisfiable formula and* s *be a model of* φ*. Let* δ *be the subset of* s *such that* $prop(\delta) = prop(\varphi)$*. Then,*

$$\delta \models \varphi$$

Proposition 4. *Let* φ *be a satisfiable formula and* φ *be a satisfiable non-tautological formula. If* φ *and* ψ *are independent then* $\varphi \not\models \psi$*.*

Definition 1. *Let* α *and* β *be two formulae and* ψ *be a satisfiable formula. We say that*

- *α and β agree on ψ, denoted by $\alpha =_{\models\psi} \beta$, if either $\alpha \models \psi \wedge \beta \models \psi$ or $\alpha \models \neg\psi \wedge \beta \models \neg\psi$ holds.*
- *α and β disagree on ψ, denoted by $\alpha \neq_{\models\psi} \beta$, if either $\alpha \models \psi \wedge \beta \models \neg\psi$ or $\alpha \models \neg\psi \wedge \beta \models \psi$ holds.*

Observe that α and β agree or disagree on ψ iff ψ is known in both α and β. In this paper, we will use the following lemma, that generalizes the commutativity of a function, in several proofs.

Lemma 1. *Let $\mathcal{D}$ and $\mathcal{D}'$ be two domains. Let $f : \mathcal{D} \times \mathcal{D}' \mapsto \mathcal{D}$ be a function such that*

$$\forall x \in \mathcal{D}. \forall y, z \in \mathcal{D}'.\ f(f(x, y), z) = f(f(x, z), y)$$

For every sequence $Y = (y_1, \ldots, y_n),\ y_1, \ldots,\ y_n \in \mathcal{D}'$ and for every permutation $Z = (z_1, \ldots, z_n)$ of Y:

$$\forall x \in \mathcal{D}.\ f(\ldots(f(x, y_1), \ldots), y_n) = f(\ldots(f(x, z_1), \ldots), z_n)$$

3. An Abstract Transition Function for Arbitrary Belief State Representations

In this section, we will present a novel methodology for the development of a complete transition function for an arbitrary representation of belief states. To begin, we consider the following example.

Example 2. *Let us consider a domain $F = \{f, g, h\}$, an action a with three conditional effects $C_a = \{f \rightarrow \neg g, f \wedge \neg g \rightarrow \neg f, \neg f \rightarrow f\}$ and $pre(a) =$ True, and a belief state S that contains all possible states of the world but $\{\neg f, \neg g, \neg h\}$.*

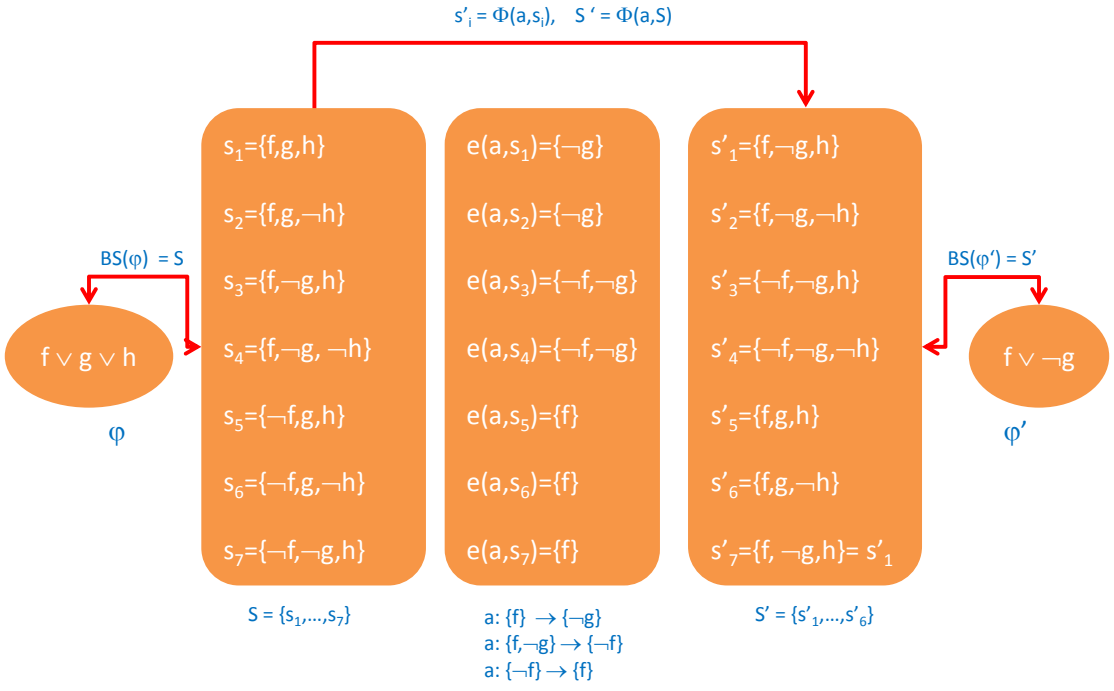

Figure 1: An example of computing successor belief states by using the function Φ

Figure 1 shows the belief state $S = \{s_1, \ldots, s_7\}$ and the computation of the successor belief state S' after executing a in S using the function Φ (Equation 2).

Observe that S and S' are the belief state forms of the CNF-formulae $\varphi = f \vee g \vee h$ and $\varphi' = f \vee \neg g$ respectively. Since each formula has a unique belief state form, we want to use their compact encoding φ and φ' instead of the belief states S and S' of exponential size.

Definition 2. *A collection $\mathcal{R}$ of formulae is a* representation of belief states *(or* representation*, for short) if for every belief state S there exists a formula $\varphi \in \mathcal{R}$ such that $BS(\varphi) = S$. $\mathcal{R}$ is said to be* unique *if for every pair of α and β in $\mathcal{R}$ such that $\alpha \equiv \beta$ then $\alpha = \beta$.*

Given a representation $\mathcal{R}$, a formula φ in $\mathcal{R}$ is called an *$\mathcal{R}$-state*. We say that *φ represents a belief state S* if $S = BS(\varphi)$. For a set of $\mathcal{R}$-states Γ and a set of belief states Λ, we say that Γ represents Λ if $\Lambda = \{BS(\varphi) \mid \varphi \in \Gamma\}$.

Intuitively, a representation specifies the type of formulae that will be used in the encoding of belief states. For example:

- The set of all formulae is trivially a representation;
- The class of binary decision diagram (BDD) [4] is a representation, since each belief state S can be represented by the formula $\bigvee_{s \in S}(\bigwedge_{\ell \in s} \ell)$ and for each formula φ there exists a BDD equivalent to φ;
- The class of prime implicate formulae is a representation.

On the other hand, the set of all conjunctions between literals is not a representation since it is not expressive enough for the encoding of disjunctive information.

We will next define a transition function $\Phi_{\mathcal{R}}$ between $\mathcal{R}$-states that captures the function Φ (Equation 2) for belief states.

Definition 3. *Let $\mathcal{R}$ be a representation. A function $\Phi_{\mathcal{R}}$ that maps pairs composed of an action and a $\mathcal{R}$-state into $\mathcal{R}$-states is said to be a* transition function for $\mathcal{R}$ *if for every $\mathcal{R}$-state φ and for every action a, $\Phi_{\mathcal{R}}(a, \varphi)$ represents the belief state $\Phi(a, BS(\varphi))$, i.e.,*

$$BS(\Phi_{\mathcal{R}}(a, \varphi)) = \Phi(a, BS(\varphi))$$

In this section, we will denote with $\mathcal{R}$ an arbitrary representation, with φ an arbitrary satisfiable $\mathcal{R}$-state, and with a an arbitrary action. Definition 3 indicates that if $\Phi_{\mathcal{R}}$ is a transition function for $\mathcal{R}$ and φ represents a belief state S, then $\Phi_{\mathcal{R}}(a, \varphi)$ is a $\mathcal{R}$-state φ' that represents the belief state $\Phi(a, S)$. In this case, φ' is said to be a result of the execution of a in φ, or a successor of φ. Next, we will show how to construct such a function $\Phi_{\mathcal{R}}$ for efficiently computing a successor $\mathcal{R}$-state φ' after executing a in φ without operating over all the states in $BS(\varphi)$ or $BS(\varphi')$, as the function Φ does.

Observe from Figure 1 that, to compute the result of execution of action a in state s_1, for example, we need to compute $e(a, s_1)$. This is the set of literals (in this case, $e(a, s_1)$ is the singleton set $\{\neg g\}$) that must be true in the successor state s'_1. The other literals in s_1, independent from $e(a, s_1)$, will remain the same in s'_1. This means that s'_1 is obtained by making the *minimum change* to s_1 such that the literals in $e(a, s_1)$ become true in the new state s'_1. We define the effect of executing an action in a formula as follows.

Definition 4. *The* effect of executing a in φ, *denoted by* $e(a, \varphi)$, *is the set of literals defined as*

$$e(a, \varphi) = \{\ell \mid \exists(a : \psi \rightarrow \ell).\ \varphi \models \psi\}$$

In general, the effect of executing a in φ does not capture the effects of executing a in all states in $BS(\varphi)$ (e.g., φ in Example 2). This is because the effects of executing an action in different states can be different. For example, the effects of executing a in s_1, s_3, and s_5 are all different as shown in Figure 1. However, one may observe that $e(a, s_1) = e(a, s_2) = \{\neg g\}$, $e(a, s_3) = e(a, s_4) = \{\neg f, \neg g\}$, and $e(a, s_5) = e(a, s_6) = e(a, s_7) = \{f\}$. Thus, if we divide $BS(\varphi)$ into three sets of states $S_1 = \{s_1, s_2\}$, $S_2 = \{s_3, s_4\}$, and $S_3 = \{s_5, s_6, s_7\}$ (Figure 2) then the effects of executing a in the states in each of these sets are identical. Observe also that S_1, S_2, and S_3 can be represented by the CNF-states $\varphi_1 = f \wedge g$, $\varphi_2 = f \wedge \neg g$, and $\varphi_3 = \neg f \wedge (g \vee h)$ respectively. Moreover, one can check from Figure 2 that $\forall s \in BS(\varphi_i).\ e(a, \varphi_i) = e(a, s)$, for $i = 1, 2, 3$. Let $s'_j = \Phi(a, s_j)$ (for $j = 1, \ldots, 7$) and $S'_i = \Phi(a, S_i)$. Let φ'_i be the formula obtained by updating φ_i with the minimum change such that the literals in $e(a, \varphi_i)$ become true in the new formula φ'_i. Intuitively, $BS(\varphi'_i) = S'_i$ and hence $BS(\varphi'_1) \cup BS(\varphi'_2) \cup BS(\varphi'_3) = BS(\varphi')$ or $\varphi'_1 \vee \varphi'_2 \vee \varphi'_3 \equiv \varphi'$. Indeed, it is easy to see that $\varphi'_1 = f \wedge \neg g$, $\varphi'_2 = \neg f \wedge \neg g$, $\varphi_3 = f \wedge (g \vee h)$ and the disjunction $\varphi'_1 \vee \varphi'_2 \vee \varphi'_3$ can be simplified to the CNF-state φ'. On the other hand, since $S = S_1 \cup S_2 \cup S_3$, we have $\varphi \equiv \varphi_1 \vee \varphi_2 \vee \varphi_3$. This result can also be easily verified . Thus, the computation of φ' includes three major steps as described in Figure 2.

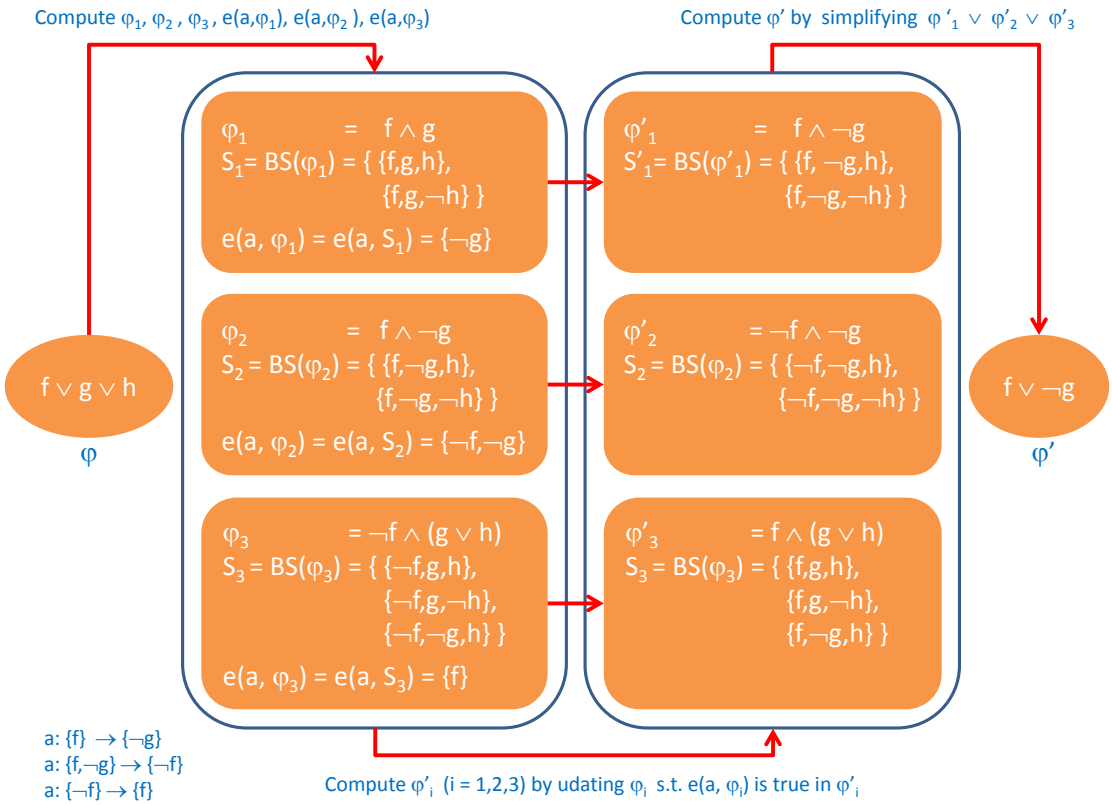

Figure 2: Computing successor CNF-state φ' after executing action a in CNF-state φ for Example 2

The transition function $\Phi_{\mathcal{R}}$ for $\mathcal{R}$ will be developed based on three similar computation steps as follows

- Step 1: Compute the set of $\mathcal{R}$-states $\{\varphi_i \mid i = 1, \ldots, n\}$ such that

$$BS(\varphi_1) \cup \ldots \cup BS(\varphi_n) = BS(\varphi) \text{ and } \forall s, s' \in BS(\varphi_i).\ e(a, s) = e(a, s') \tag{4}$$

  Compute $e(a, \varphi_i)$ for $i = 1, \ldots, n$;

- Step 2: For each $\mathcal{R}$-state φ_i, compute φ'_i by updating φ_i such that every literal in $e(a, \varphi_i)$ is true in φ'_i;

- Step 3: Convert the disjunction $\varphi'_1 \vee \ldots \vee \varphi'_n$ into an equivalent $\mathcal{R}$-state φ', i.e., $BS(\varphi') = BS(\varphi'_1 \vee \ldots \vee \varphi'_n)$

We assume the existence of the following operations:

1. $conv_{\mathcal{R}}$: is a function that maps a formula φ of the form $\varphi' \wedge \psi$ or $\varphi' \wedge \neg\psi$, where φ' is a $\mathcal{R}$-state and ψ is a consistent set of literals, into an equivalent $\mathcal{R}$-state, i.e.

$$BS(\varphi) = BS(conv_{\mathcal{R}}(\varphi)) \tag{5}$$

2. $update_{\mathcal{R}}$: $\mathcal{R} \times 2^F \to \mathcal{R}$ is a function that maps a pair of a $\mathcal{R}$-state φ and a consistent set of literals e to a $\mathcal{R}$-state such that

$$BS(update_{\mathcal{R}}(\varphi, e)) = \{s \setminus \overline{e} \cup e \mid s \in BS(\varphi)\} \tag{6}$$

3. $merge_{\mathcal{R}} : 2^{\mathcal{R}} \setminus \emptyset \to \mathcal{R}$ is a function that maps non-empty sets of $\mathcal{R}$-states into equivalent $\mathcal{R}$-states satisfying the following properties: for all $\Gamma \in 2^{\mathcal{R}}$, $\Gamma \neq \emptyset$ we have that:

$$BS(merge_{\mathcal{R}}(\Gamma)) = \bigcup_{\gamma \in \Gamma} BS(\gamma) \tag{7}$$

Intuitively, the functions $update_{\mathcal{R}}$ and $merge_{\mathcal{R}}$ are introduced for the computation in the second and third steps respectively. A precise definition of these two functions will be determined whenever $\mathcal{R}$ is given as a concrete representation.

We need to show how to compute the set of $\mathcal{R}$-states $\{\varphi_i \mid i = 1, \ldots, n\}$ (Step 1) that satisfies (4). Observe that, for each $\mathcal{R}$-state φ_i, the effect of executing a in every state in $BS(\varphi_i)$ is the same if the truth value of each e-condition ψ of a in every state in $BS(\varphi_i)$ is the same, i.e., ψ is known in φ_i. For example, in Figure 2, the effects of a in the two states $s_1 = \{f, g, h\}$ and $s_2 = \{f, g, \neg h\}$ of $BS(\varphi_1)$ are the same ($e(a, s_1) = e(a, s_2) = \neg g$) because the first e-condition of a (i.e., f) is true in both s_1 and s_2, while the other two e-conditions of a (i.e., $f \wedge \neg g$ and $\neg f$) are false in the both states. Similarly, all the three e-conditions of a are known in φ_2 as well as in φ_3. On the other hand, $e(a, s_1) \neq e(a, s_3)$ because the second e-condition ($f \wedge \neg g$) is false in s_1 but it is true in s_3, making the literal $\neg f$ included in $e(a, s_3)$ but not in $e(a, s_1)$. We have the following definition.

Definition 5.

- *We say that* φ is enabling for a *if every e-condition of* a *is known in* φ*, i.e.,* $\forall \psi \in \Psi(a)$ *either* $\varphi \models \psi$ *or* $\varphi \models \neg\psi$ *holds.*

- *A set of $\mathcal{R}$-states Γ is said to be an* enabling form *of φ for a if*
  1. *Every formula in Γ is enabling for a;*
  2. $\bigcup_{\varphi' \in \Gamma} BS(\varphi') = BS(\varphi)$.

It is easy to see that the following proposition holds.

Proposition 5. *If φ is enabling for a then*

$$\forall s \in BS(\varphi).\, e(a, s) = e(a, \varphi)$$

Proposition 5 shows that an enabling form of φ for a satisfies (4). Thus, our goal now is to compute an enabling form of φ for a. The following proposition helps us to achieve this goal.

Proposition 6. *Let φ and ψ be two satisfiable formulae such that ψ is unknown in φ. Then,*

1. *$\varphi \wedge \psi$ is satisfiable and $BS(\varphi \wedge \psi) = \{s \mid s \in BS(\varphi), s \models \psi\}$,*
2. *$\varphi \wedge \neg\psi$ is satisfiable and $BS(\varphi \wedge \neg\psi) = \{s \mid s \in BS(\varphi), s \models \neg\psi\}$, and*
3. $BS(\varphi \wedge \psi) \cup BS(\varphi \wedge \neg\psi) = BS(\varphi)$.

Following Proposition 6, if ψ is an e-condition of a unknown in φ, we can replace φ with two formulae $\varphi \wedge \psi$ and $\varphi \wedge \neg\psi$, in which ψ is known. Since these two formulae may not be in $\mathcal{R}$ and we would like to remain in the representation $\mathcal{R}$, we need to convert them into an $\mathcal{R}$-state. We call this process an extension of φ on ψ under $\mathcal{R}$ and define it formally as follows.

Definition 6. *Let ψ be a consistent set of literals. The* extension of φ on ψ under $\mathcal{R}$*, denoted by $\varphi \oplus_{\mathcal{R}} \psi$, is defined by*

$$\varphi \oplus_{\mathcal{R}} \psi = \begin{cases} \{\varphi\} & \text{if } \varphi \models \psi \text{ or } \varphi \models \neg\psi \\ \{conv_{\mathcal{R}}(\varphi \wedge \psi), conv_{\mathcal{R}}(\varphi \wedge \neg\psi)\} & \text{otherwise} \end{cases}$$

It is easy to see that if ψ is the only e-condition of action a, then the extension of φ on ψ is an enabling form of φ for a. Since an action may have multiple e-conditions, we need to extend φ recursively on every e-condition of the action. We generalize Definition 6 for a sequence of consistent sets of literals as follows.

Definition 7. *Let $\Psi = \langle \psi_1, \ldots, \psi_n \rangle$ be a sequence of consistent sets of literals. The* extension of φ on Ψ under $\mathcal{R}$*, denoted by $\varphi \oplus_{\mathcal{R}} \Psi$, is defined by*

$$\varphi \oplus_{\mathcal{R}} \Psi = \begin{cases} \{\varphi\} & \text{if } \Psi \text{ is empty } (n = 0) \\ \bigcup_{\gamma \in \varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_{n-1} \rangle} (\gamma \oplus_{\mathcal{R}} \psi_n) & \text{otherwise} \end{cases} \tag{8}$$

Intuitively, if Ψ is an enumeration of $\Psi(a)$ then $\varphi \oplus_{\mathcal{R}} \Psi$ is an enabling form of φ for a. This is proved in the next proposition.

Proposition 7. *Let $\langle \psi_1, \ldots, \psi_n \rangle$ be an enumeration of $\Psi(a)$. If φ be a satisfiable then $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle$ is an enabling form of φ for a and every $\mathcal{R}$-state in this set is satisfiable.*

Proposition 7 allows us to compute an enabling form of an arbitrary $\mathcal{R}$-state φ for an arbitrary action a. Observe that the enabling form of a $\mathcal{R}$-state φ is a set of $\mathcal{R}$-states each of which represents a set of states that agree on every e-condition of a and the union of those sets is equal to the belief state of φ. However, an $\mathcal{R}$-state may have multiple enabling forms of different sizes. For example, if we replace φ_3 in Figure 2 with two CNF-states $\varphi_4 = \neg f \wedge g$, that represents the set of the first two states in S_3, and $\varphi_5 = \neg f \wedge \neg g \wedge h$, that represents the singleton set of the last state in S_3, then the set $\{\varphi_1, \varphi_2, \varphi_4, \varphi_5\}$ is an enabling form of φ for a. As another example, let $\varphi_6 = \neg f \wedge h$, that represents the set of the first state and the last state in S_3, then $\{\varphi_1, \varphi_2, \varphi_4, \varphi_5, \varphi_6\}$ is another enabling form of φ for a.

However, it is easy to see that there does not exist an enabling form of φ for a in Example 2 that contains less than three formulae; if an enabling form of φ for a contains three formulae, then each of those formulae will represent a different belief state among S_1, S_2, and S_3. This means that any enabling form of φ for a that contains a minimum number of formulae represents the same set of belief state $\{S_1, S_2, S_3\}$. Next we will show that, among all enabling forms of φ for a, $\varphi \oplus_{\mathcal{R}} \Psi$ (where Ψ is an arbitrary enumeration of $\Psi(a)$) contains the minimum number of $\mathcal{R}$-states.

Lemma 2. *Let $\Psi = \langle \psi_1, \ldots, \psi_n \rangle$ be a sequence of consistent sets of literals. For every pair of two different $\mathcal{R}$-states (α, β) in $\varphi \oplus_{\mathcal{R}} \Psi$, there exists ψ in Ψ such that α and β disagree on ψ.*

Theorem 1. *Let $\Psi = \langle \psi_1, \ldots, \psi_n \rangle$ be an enumeration of $\Psi(a)$. Then, $\varphi \oplus_{\mathcal{R}} \Psi$ contains the minimum number of $\mathcal{R}$-states among all the enabling forms of φ for a. If Γ is an enabling form of φ for a and $|\Gamma| = |\varphi \oplus_{\mathcal{R}} \Psi|$, then $\varphi \oplus_{\mathcal{R}} \Psi$ and Γ represent the same set of belief states, i.e., $\{BS(\varphi') \mid \varphi' \in \varphi \oplus_{\mathcal{R}} \Psi\} = \{BS(\gamma) \mid \gamma \in \Gamma\}$.*

Theorem 1 reveals that whatever enumeration Ψ of $\Psi(a)$ is selected, $\varphi \oplus_{\mathcal{R}} \Psi$ is a set of the minimum number of $\mathcal{R}$-states in all enabling forms of φ for a, and it represents the same set of belief states. We use $enb_{\mathcal{R}}(a, \varphi)$ to denote $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle$ for some enumeration $\langle \psi_1, \ldots, \psi_n \rangle$ of $\Psi(a)$, where the order ψ_i in the enumeration is immaterial. We are now ready to define $\Phi_{\mathcal{R}}$ that will be proved to be a transition function for $\mathcal{R}$ representation as follows.

Definition 8. *The execution of a in φ results in a $\mathcal{R}$-state, denoted by $\Phi_{\mathcal{R}}(a, \varphi)$, is defined as follows.*

$$\Phi_{\mathcal{R}}(a, \varphi) = \begin{cases} merge_{\mathcal{R}}(\{update_{\mathcal{R}}(\gamma, e(a, \gamma)) \mid \gamma \in enb_{\mathcal{R}}(a, \varphi)\}) & \text{if } \varphi \models pre(a) \\ undefined & \text{otherwise} \end{cases}$$

Theorem 2. *$\Phi_{\mathcal{R}}$ defined in Definition 8 is a transition function for $\mathcal{R}$.*

Similar to the manner in which the function $\widehat{\Phi}$ is defined by extending the function Φ, we are going to define an extension of $\Phi_{\mathcal{R}}$ to reason about the results of plans as follows.

Definition 9. *The* extension of $\Phi_{\mathcal{R}}$ *(Definition 8), denoted by $\widehat{\Phi_{\mathcal{R}}}$, is a function that maps pairs of an action sequence and a $\mathcal{R}$-state into $\mathcal{R}$-states, defined as follows.*

For every $\mathcal{R}$-state φ and for every sequence of actions $\alpha_n = [a_1, \ldots, a_n]$:

- *If $n = 0$ then $\widehat{\Phi_{\mathcal{R}}}([\,], \varphi) = \varphi$;*
- *If $n > 0$ then*

$$\widehat{\Phi_{\mathcal{R}}}(\alpha_n, \varphi) = \Phi_{\mathcal{R}}(a_n, \widehat{\Phi_{\mathcal{R}}}(\alpha_{n-1}, \varphi))$$

where $\alpha_{n-1} = [a_1, \ldots, a_{n-1}]$ and $\Phi_{\mathcal{R}}(a, undefined) = undefined$ for every action a.

We have the following theorem.

Theorem 3. *Let $\alpha_n = [a_1, \ldots, a_n]$ be a sequence of actions. Then $\widehat{\Phi_{\mathcal{R}}}(\alpha_n, \varphi)$ represents the belief state $\widehat{\Phi}(\alpha_n, BS(\varphi))$:*

$$BS(\widehat{\Phi_{\mathcal{R}}}(\alpha_n, \varphi)) = \widehat{\Phi}(\alpha_n, BS(\varphi))$$

Theorems 2 and 3 show that $\Phi_{\mathcal{R}}$ and $\widehat{\Phi_{\mathcal{R}}}$ capture correctly the functions Φ and $\widehat{\Phi}$ respectively. Thus, $\Phi_{\mathcal{R}}$ and $\widehat{\Phi_{\mathcal{R}}}$ can be used in the implementation of a conformant planner following Algorithm 1 over the representation $\mathcal{R}$. Observe that Algorithm 2 is built on Algorithm 1, where Φ is replaced with $\Phi_{\mathcal{R}}$ and each belief state is replaced by a $\mathcal{R}$-state representing it.

Algorithm 2 Plan(F, O, I, G) {Best First Search Planner Using Representation $\mathcal{R}$}

Input: A planning problem $\langle F, O, I, G\rangle$
Output: Solution if exists; No solution otherwise
Compute $\mathcal{R}$-state φ_I such that $BS(\varphi_I) = S_I$
Create priority queue Q
Initialize Q with start node $(\varphi_I, [])$
Let $Exist = \{\varphi_I\}$ {Set of $\mathcal{R}$-states that have been generated}
while Q is not empty **do**
  Extract the first element $N = (\varphi, CP)$ of Q {CP is the plan from φ_I to φ}
  if φ satisfies G **then**
    return CP {the plan reaching the goal G}
  else
    for each action a such that $S \models pre(a)$ **do**
      Compute $\varphi' = \Phi_{\mathcal{R}}(a, \varphi)$ {successor of φ using a}
      if $\varphi' \notin Exist$ **then**
        Insert $(\varphi', CP \circ [a])$ in Q {heuristic of φ' as priority in Q}
        $Exist = Exist \cup \{\varphi'\}$
      end if
    end for
  end if
end while
return no solution

We next present a generic way for computing $\Phi_{\mathcal{R}}$ for an arbitrary representation $\mathcal{R}$. By Definition 8 and Theorem 2, if a is executable in φ then $\Phi_{\mathcal{R}}(a, \varphi)$ can be computed using the following steps:

1. Compute $enb_{\mathcal{R}}(a, \varphi)$ and $e(a, \gamma)$ for each $\gamma \in enb_{\mathcal{R}}(a, \varphi)$.
2. Compute $update_{\mathcal{R}}(\gamma, e(a, \gamma))$ for each $\gamma \in enb_{\mathcal{R}}(a, \varphi)$.
3. Merge $update_{\mathcal{R}}(\gamma, e(a, \gamma))$ where $\gamma \in enb_{\mathcal{R}}(a, \varphi)$ to create $\Phi_{\mathcal{R}}(a, \varphi)$.

The computations of $update_{\mathcal{R}}$ and $merge_{\mathcal{R}}$ operators depend on the representation and thus need to be developed given a specific representation. The computation of $enb_{\mathcal{R}}(a, \varphi)$ requires the computation of $\oplus_{\mathcal{R}}$ which is implemented in the procedure $\texttt{extending}_{\mathcal{R}}(\varphi, \mathit{effect}, \psi \rightarrow \eta)$ (Algorithm 3). Specifically, the procedure takes as input a $\mathcal{R}$-state φ, a set of literals *effect*, and a combined conditional effect $a : \psi \rightarrow \eta$. It returns a set of pairs of the form $\langle \gamma, e(\gamma)\rangle$ where $\gamma \in \varphi \oplus_{\mathcal{R}} \psi$ and $e(\gamma)$ is a set of literals associated with γ.

Intuitively, $e(\gamma)$ is used to compute $e(a,\gamma)$ for $\gamma \in enb(a,\varphi)$. Observe that this procedure uses the operator $conv_{\mathcal{R}}$, that also needs to be developed when a concrete representation is given.

It is easy to see that the following proposition holds.

Proposition 8. *For a $\mathcal{R}$-state φ and a combined conditional effect $a : \psi \to \eta$,* $\texttt{extending}_{\mathcal{R}}(\varphi, \mathit{effect}, \psi \to \eta)$ *returns* $\{\langle \gamma, \mathit{effect} \cup \{\ell \mid a : \psi \to \ell \wedge \gamma \models \psi\}\rangle \mid \gamma \in \varphi \oplus_{\mathcal{R}} \psi\}$.

Algorithm 3 $\texttt{extending}_{\mathcal{R}}(\varphi, \mathit{effect}, \psi \to \eta)$ Computing $\varphi \oplus_{\mathcal{R}} \psi$

Input: $\mathcal{R}$-state φ, set of literals *effect*, combined conditional effect $\psi \to \eta$
Output: $\{\langle \gamma, e(\gamma)\rangle \mid \gamma \in \varphi \oplus_{\mathcal{R}} \psi\}$
if $\varphi \models \psi$ or $\varphi \models \neg\psi$ **then**
  if $\varphi \models \psi$ **then**
    $e(\varphi) = e(\varphi) \cup \eta$ {η is added to $e(\varphi)$ only if $\varphi \models \psi$}
  end if
  return $\{\langle \varphi, e(\varphi)\rangle\}$
else
  Let $\varphi_1 = conv_{\mathcal{R}}(\varphi \wedge \psi)$
  Set $e(\varphi_1) = e(\varphi) \cup \eta$ {$\varphi_1 \models \psi$ so η is added to $e(\varphi_1)$}
  Let $\varphi_2 = conv_{\mathcal{R}}(\varphi \wedge \neg\psi)$
  Set $e(\varphi_2) = e(\varphi)$ {$\varphi_2 \models \neg\psi$ so η is not added to $e(\varphi_2)$}
  return $\{\langle \varphi_1, e(\varphi_1)\rangle, \langle \varphi_2, e(\varphi_2)\rangle\}$
end if

Algorithm 4 $\texttt{enabling}(a, \varphi)$ Computing $enb_{\mathcal{R}}(a,\varphi)$ and the effects of a in these $\mathcal{R}$-states

Input: $\mathcal{R}$-state φ, action a
Output: $\{\langle \gamma, e(a,\gamma)\rangle \mid \gamma \in enb_{\mathcal{R}}(a,\varphi)\}$
Set $e(\varphi) = \emptyset$
Let $Result = \{\langle \varphi, e(\varphi)\rangle\}$
Let $\{\psi_i \to \eta_i \mid i = 1, \ldots, n\}$ be the set of combined conditional effects of a
for $i = 1$ **to** n **do**
  Let $X = \emptyset$
  for each $\langle \gamma, e(\gamma)\rangle \in Result$ **do**
    $X = X \cup \texttt{extending}_{\mathcal{R}}(\gamma, e(\gamma), \psi_i \to \eta_i)$ {Algorithm 3}
  end for
  Set $Result = X$
end for
return $Result$

One can observe that the procedure $\texttt{enabling}(a,\varphi)$ (Algorithm 4) is used to compute $enb_{\mathcal{R}}(a,\varphi)$ and $e(a,\gamma)$ for $\gamma \in enb_{\mathcal{R}}(a,\varphi)$ as shown in the following proposition.

Proposition 9. *For a $\mathcal{R}$-state φ and an action a,* $\texttt{enabling}(a,\varphi)$ *returns* $\{\langle \gamma, e(a,\gamma)\rangle \mid \gamma \in enb_{\mathcal{R}}(a,\varphi)\}$.

The transition function $\Phi_{\mathcal{R}}(a,\varphi)$ is implemented in the procedure $\texttt{phi}_{\mathcal{R}}(a,\varphi)$ (Algorithm 5). The correctness of the algorithm is a natural consequence of Proposition 9 and Definition 8.

Algorithm 5 $\texttt{phi}_{\mathcal{R}}(a, \varphi)$ Computing $\Phi_{\mathcal{R}}(a, \varphi)$

1: **Input**: $\mathcal{R}$-state φ, action a.
2: **Output**: Successor $\mathcal{R}$-state $\Phi_{\mathcal{R}}(a, \varphi)$
3: **if** $\varphi \not\models pre(a)$ **then**
4: **return** $undefined$
5: **end if**
6: Let $X = \texttt{enabling}(a, \varphi)$ {Algorithm 4}
7: Let $Y = \emptyset$
8: **for** each $\langle \gamma, \mathit{effect} \rangle$ in X **do**
9: Compute $\gamma' = update_{\mathcal{R}}(\gamma, \mathit{effect})$ {update $\mathcal{R}$-states in the computed enabling form}
10: Let $Y = Y \cup \{\gamma'\}$
11: **end for**
12: Compute $\varphi_{succ} = merge_{\mathcal{R}}(Y)$
13: **return** φ_{succ}

Proposition 10. *For a $\mathcal{R}$-state φ and an action a, the procedure $\texttt{phi}_{\mathcal{R}}(a, \varphi)$ returns $\varphi_{succ} = \Phi_{\mathcal{R}}(a, \varphi)$.*

Observe from Algorithm 5 that the number of calls to the $update_{\mathcal{R}}$ function, and hence the size of the set Y fed to $merge_{\mathcal{R}}$ (line 12), depends on the number of elements in the enabling form of φ for a. After the application of $\oplus_{\mathcal{R}}$ on a $\mathcal{R}$-state γ and an e-condition ψ of a, we obtain a set of at most two $\mathcal{R}$-states if ψ is unknown in γ. Hence, $enb_{\mathcal{R}}(a, \varphi)$ contains at most 2^k $\mathcal{R}$-states if k is the number of e-conditions of a that are unknown in φ. However, the number of $\mathcal{R}$-states in $enb_{\mathcal{R}}(a, \varphi)$ can be much less that 2^k. The reason is that an e-condition ψ_i of a unknown in φ can be known in some element(s) of $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_j \rangle$ even thought ψ_i is not in the sequence $\langle \psi_1, \ldots, \psi_j \rangle$.

For example, consider $\varphi = f \vee g$, $\psi_1 = \neg f$, and $\psi_2 = g$. Clearly, ψ_1 and ψ_2 are both unknown in φ. We have $\varphi \oplus_{\mathcal{R}} \psi_1 = \{conv_{\mathcal{R}}((f \vee g) \wedge \neg f), conv_{\mathcal{R}}((f \vee g) \wedge f)\}$. Observe that $(f \vee g) \wedge \neg f \equiv \neg f \wedge g$ and $(f \vee g) \wedge f \equiv f$. Since $\psi_2 = g$ is known (true) in $\neg f \wedge g$, it is known in one of the two elements in $\varphi \oplus_{\mathcal{R}} \psi_1$. This implies that $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \psi_2 \rangle$ contains only three elements instead of 2^2 elements. As another example, in Figure 2, there are three e-conditions of a unknown in φ but an enabling form of φ for a contains only three elements instead of 2^3 elements. In the worst case, nevertheless, $enb_{\mathcal{R}}(a, \varphi)$ can have size 2^k, although it is the smallest in all enabling forms of φ for a as proved in Theorem 1. This means that the cost of computing $\Phi_{\mathcal{R}}(a, \varphi)$ can be exponential in the number of unknown e-conditions of a (in the worst case). This is understandable as the problem of checking whether a proposition holds after the execution of an action in presence of incomplete information is a co-NP complete problem [2].

In the next sections, we will propose three different representations of belief states, that are collections of DNF and CNF formulae satisfying certain properties, and the application of these representations in conformant planning. We will show how the theory developed in this section can be instantiated in the development of a complete transition function for each of the proposed representations. In particular, we will provide a precise definition of each of the operators $conv_{\mathcal{R}}$, $update_{\mathcal{R}}$, and $merge_{\mathcal{R}}$ for each $\mathcal{R}$ representation being considered.

4. Minimal-DNF Representation

In this section, we investigate the *minimal-DNF* representation, denoted by μDNF, which is a collection of *disjunctive normal form (DNF)* formulae satisfying some minimality criteria. We introduce a conformant

planner, called DNF, that implements the μDNF representation and its transition function $\Phi_{\mu DNF}$ for the best-first search in belief state space for solutions.

4.1. The μDNF Representation and Its Operators

In this subsection, we define the μDNF representation and the necessary functions for computing $\Phi_{\mu DNF}$. We start with some auxiliary notations. A set of literals δ is called a *partial state* if δ is consistent. A set Δ of sets of literals represents the DNF formula $\bigvee_{\delta\in\Delta}(\bigwedge_{\ell\in\delta}\ell)$. We often use upper and lower case Greek letters to denote a DNF formula and a set of literals, respectively.

Definition 10. *A DNF formula Δ is said to be* minimal *if for every $\delta \in \Delta$, δ is a partial state (consistent) and there exists no $\delta' \in \Delta$ such that $\delta' \subsetneq \delta$.*

It is easy to see that every belief state can be viewed as a minimal-DNF formula. This allows for the following definition of the minimal-DNF representation.

Definition 11. *The* minimal-DNF representation*, denoted by μDNF-*representation*, is the collection of minimal-DNF formulae. A μDNF-state is a minimal-DNF formula.*

The next example illustrates the above definitions.

Example 3. *Let us consider a domain with three propositions f, g, and h. We consider the following.*

- *$\{f, \neg g\}$ is a partial state but $\{f, g, \neg g\}$ is not since $\{f, g, \neg g\}$ is inconsistent.*
- *$\Delta_1 = \{\{f, \neg g\}, \{f, g, \neg g\}\}$ is a DNF-formula but it is not a μDNF-state since it contains the element $\{f, g, \neg g\}$ which is not a partial state. Observe that, if we remove the inconsistent set from Δ_1, then we obtain the μDNF-state $\{\{f, \neg g\}\}$ which is equivalent to Δ_1.*
- *$\{\{f\}\}$ is a μDNF-state but $\Delta_2 = \{\{f\}, \{f, \neg g\}\}$ is not as $\{f\} \subset \{f, \neg g\}$. If we remove $\{f, \neg g\}$ from Δ_2, then we obtain the first μDNF-state equivalent to $\{\{f\}\}$.*

The following function converts a DNF formula into an equivalent μDNF-state.

Definition 12. *Let Δ be a DNF formula. We define*

$$\mu(\Delta) = min(refine(\Delta))$$

where $refine(\Delta) = \{\delta \mid \delta \in \Delta, \delta$ is consistent$\}$ and $min(\Delta) = \{\delta \mid \delta \in \Delta \wedge \nexists \delta' \in \Delta.\ \delta' \subsetneq \delta\}$.

Proposition 11. *For every DNF formula Δ, $\mu(\Delta)$ is a μDNF-state equivalent to Δ.*

Example 4. *For $\Delta = \{\{f\}, \{f, \neg g\}, \{f, g, \neg g\}\}$, we have that*

$$refine(\Delta) = \{\{f\}, \{f, \neg g\}\}$$

and

$$\mu(\Delta) = min(refine(\Delta)) = min(\{\{f\}, \{f, \neg g\}\}) = \{\{f\}\}.$$

Clearly, $\{\{f\}\}$ is a μDNF-state equivalent to Δ but has a significantly smaller size.

In what follows we will show the relation of a partial state to its belief state.

Proposition 12. *Let δ be a partial state. Then,*

1. $BS(\delta) = \{s \mid s \text{ is a state, and } \delta \subseteq s\}$.
2. *δ contains the smallest number of literals among those that represent the belief state $BS(\delta)$.*

To complete the instantiation of the representation and define $\Phi_{\mu DNF}$, we will need to define the functions $conv_{\mu DNF}$, $update_{\mu DNF}$, and $merge_{\mu DNF}$.

For a μDNF-state Δ and a set of literals ψ, $conv_{\mu DNF}$ should map $\Delta \wedge \gamma$, where γ is either ψ or $\neg\psi$, to an equivalent μDNF-state (Definition 6). The idea is to transform $\Delta \wedge \gamma$ into a DNF formula, then use the function μ to convert it into a μDNF-state. Due to the distributivity of $\wedge$ over $\vee$, $\Delta \wedge \gamma \equiv \{\delta \wedge \gamma \mid \delta \in \Delta\}$. Hence, we need to convert each $\delta \wedge \gamma$ to some set(s) of literals. When $\gamma = \psi$, we have that $\delta \wedge \psi$ is equivalent to $\delta \cup \psi$. When $\gamma = \neg\psi$, we observe that $\delta \wedge \neg\psi$ is equivalent to δ if $\delta \models \neg\psi$, i.e., $\delta \cap \overline{\psi} \neq \emptyset$, and $\delta \wedge \neg\psi$ is equivalent to the DNF formula $\{\delta \wedge \overline{\ell} \mid \ell \in \psi\}$ otherwise. Note that, if $\delta \models \psi$, then $\psi \subseteq \delta$ and $\delta \wedge \neg\psi$ is unsatisfiable but still equivalent to $\{\delta \wedge \overline{\ell} \mid \ell \in \psi\}$, as every $\delta \wedge \overline{\ell}$ in this DNF formula is inconsistent.

Definition 13. *Let Δ be a μDNF-state and ψ be a consistent set of literals. Then,*

$$conv_{\mu DNF}(\Delta \wedge \psi) = min(\{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset\})$$

$$conv_{\mu DNF}(\Delta \wedge \neg\psi) = min(\{\delta \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} \neq \emptyset\} \cup \bigcup_{\delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\})$$

Proposition 13. *Let Δ be a μDNF-state and ψ be a consistent set of literals. Then,*

1. *$conv_{\mu DNF}(\Delta \wedge \psi)$ is a μDNF-state equivalent to $\Delta \wedge \psi$.*
2. *$conv_{\mu DNF}(\Delta \wedge \neg\psi)$ is a μDNF-state equivalent to $\Delta \wedge \neg\psi$.*

Thus, $conv_{\mu DNF}$ satisfies the conditions required in Equation (5).

Our next task is to define the function $update_{\mu DNF}$ that satisfies the Equation (6). Intuitively, updating a μDNF-state can be done by updating every partial state in it, as a μDNF-state is a disjunctive set of partial states.

Definition 14. *Let Δ be a μDNF-state and e be a consistent set of literals. $update_{\mu DNF}(\Delta, e)$ is defined as follows:*

$$update_{\mu DNF}(\Delta, e) = min(\{\delta \setminus \overline{e} \cup e \mid \delta \in \Delta\})$$

As expected, we have the following proposition which confirms that $update_{\mu DNF}$ satisfies the condition (6).

Proposition 14. *Let Δ be a μDNF-state and e be a consistent set of literals. Then, $update_{\mu DNF}(\Delta, e)$ is a μDNF-state and*

$$BS(update_{\mu DNF}(\Delta, e)) = \{s \setminus \overline{e} \cup e \mid s \in BS(\Delta)\}$$

Thanks to Proposition 1, merging a set of μDNF-states into a single μDNF-state can be easily defined as follows.

Definition 15. *Let Γ be a set of μDNF-states. Then $merge_{\mu DNF}$ is define as*

$$merge_{\mu DNF}(\Gamma) = min(\bigcup_{\Delta \in \Gamma} \Delta)$$

The following proposition shows that $merge_{\mu DNF}$ satisfies condition (7).

Proposition 15. *Let Γ be a set of μDNF-states. Then $merge_{\mu DNF}(\Gamma)$ is a μDNF-state and*

$$BS(merge_{\mu DNF}(\Gamma)) = \bigcup_{\Delta \in \Gamma} BS(\Delta)$$

With the introduction of Definitions 13, 14, and 15 along with the respective Propositions 13, 14, and 15, we now have a complete precise definition of $\Phi_{\mu DNF}$ whose correctness has been proved. For a better understanding of how $\Phi_{\mu DNF}$ works, we consider the following example.

Example 5. *Let us consider a domain $F = \{f, g, h, k\}$, a μDNF-state $\Delta = \{\{f, g\}, \{g, \neg h\}\}$, and an action $a = \langle \emptyset, \{f \rightarrow \neg g, h \rightarrow f, f \wedge h \rightarrow k\}\rangle$. Let us compute $\Phi_{\mu DNF}(a, \Delta)$.*

First, we need to compute $enb_{\mu DNF}(a, \Delta)$ $e(a, \Delta')$ for $\Delta' \in enb_{\mu DNF}(a, \Delta)$ using Algorithm 4. We have that $\Psi(a) = \{f, h, f \wedge h\}$. It is easy to see that none of the e-conditions in $\Psi(a)$ is known in Δ and the set of combined conditional effects of a is the same as the set of conditional effects of a. Suppose that the (combined) conditional effects of a will be introduced in computing $enb_{\mu DNF}(a, \Delta)$ in the given order. We start with the conditional effect $a : f \rightarrow \neg g$ and $e(\Delta)$ is initialized with the empty set. Since f is unknown in Δ, the first application of $\oplus_{\mu DNF}$ returns two μDNF-states $\Delta_1 = conv_{\mu DNF}(\Delta \wedge f)$ and $\Delta_2 = conv_{\mu DNF}(\Delta \wedge \neg f)$. Using Definition 13, we have

$$\begin{aligned}
\Delta_1 &= conv_{\mu DNF}(\Delta \wedge f) \\
&= min(\{\delta \cup \{f\} \mid \delta \in \Delta \wedge \delta \cap \{\neg f\} = \emptyset\}) \\
&= min(\{\{f, g\} \cup \{f\}, \{g, \neg h\} \cup \{f\}\}) \\
&= min(\{\{f, g\}, \{f, g, \neg h\}\}) \\
&= \{\{f, g\}\} \\
e(a, \Delta_1) &= e(a, \Delta) \cup \{\neg g\} = \{\neg g\}
\end{aligned}$$

$$\begin{aligned}
\Delta_2 &= conv_{\mu DNF}(\Delta \wedge \neg f) \\
&= min(\{\delta \mid \delta \in \Delta \wedge \delta \cap \{\neg f\} \neq \emptyset\}) \cup \bigcup_{\delta \in \Delta \wedge \delta \cap \{\neg f\} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \{f\} \setminus \delta\}) \\
&= min(\emptyset \cup \{\{f, g\} \cup \{\overline{\ell}\} \mid \ell \in \{f\} \setminus \{f, g\}\} \cup \{\{g, \neg h\} \cup \{\overline{\ell}\} \mid \ell \in \{f\} \setminus \{g, \neg h\}\}) \\
&= min\{\emptyset \cup \emptyset \cup \{\{\neg f, g, \neg h\}\} \\
&= \{\{\neg f, g, \neg h\}\} \\
e(a, \Delta_2) &= e(a, \Delta) = \emptyset \qquad \text{(Note that, } \Delta_2 \text{ does not satisfy } f \text{ so } \neg g \text{ is not added to } e(a, \Delta_2))
\end{aligned}$$

Now we obtain the set of two μDNF-states $\Delta_1 = \{\{f, g\}\}$ and $\Delta_2 = \{\{\neg f, g, \neg h\}\}$. We continue with the second conditional effect $a : h \rightarrow f$. Since h is unknown in Δ_1, $\Delta_1 \oplus_{\mu DNF} h$ results in the following:

$$\begin{aligned}
\Delta_3 &= conv_{\mu DNF}(\Delta_1 \wedge h) = \{\{f, g, h\}\} \\
e(a, \Delta_3) &= e(a, \Delta_1) \cup \{f\} = \{f, \neg g\} \\
\Delta_4 &= conv_{\mu DNF}(\Delta_1 \wedge \neg h) = \{\{f, g, \neg h\}\} \\
e(a, \Delta_4) &= e(a, \Delta_1) = \{\neg g\}
\end{aligned}$$

Since $\Delta_2 \models \neg h$, $\Delta_2 \oplus_{\mu DNF} \neg h = \{\Delta_2\}$ and we obtain the set $\{\Delta_2, \Delta_3, \Delta_4\}$.

We are left with the conditional effect $f \wedge h \rightarrow k$. Observe that $f \wedge h$ is known in all three μDNF-states Δ_2, Δ_3, and Δ_4. As such, the application of $\oplus_{\mu DNF}$ does not change the set of μDNF-state and we have that $enb_{\mu DNF}(a, \Delta) = \{\Delta_2, \Delta_3, \Delta_4\}$. Observe that among these μDNF-state, only Δ_3 satisfies $f \wedge h$. Hence, $e(a, \Delta_3) = \{f, \neg g\} \cup \{k\} = \{f, \neg g, k\}$ and $e(a, \Delta_2)$ and $e(a, \Delta_4)$ do not change.

Updating the obtained set of μDNF-states w.r.t. the corresponding effect of executing a, we have

$$update_{\mu DNF}(\Delta_2, e(a, \Delta_2)) = \Delta_2 = \{\{\neg f, g, \neg h\}\} \text{ (because } e(a, \Delta_2) = \emptyset)$$
$$update_{\mu DNF}(\Delta_3, e(a, \Delta_3)) = \{\{f, g, h\} \setminus \{\neg f, g, \neg k\} \cup \{f, \neg g, k\}\} = \{\{f, \neg g, h, k\}\}$$
$$update_{\mu DNF}(\Delta_4, e(a, \Delta_4)) = \{\{f, g, \neg h\} \setminus \{g\} \cup \{\neg g\}\} = \{\{f, \neg g, \neg h\}\}$$

Now, we are ready to compute $\Phi_{\mu DNF}(a, \Delta)$ as

$$\begin{aligned}\Phi_{\mu DNF}(a, \Delta) &= merge_{\mu DNF}(update_{\mu DNF}(\Delta_2, e(a, \Delta_2)) \cup update_{\mu DNF}(\Delta_3, e(a, \Delta_3)) \cup \\ &\qquad update_{\mu DNF}(\Delta_4, e(a, \Delta_4))) \\ &= min(update_{\mu DNF}(\Delta_2, e(a, \Delta_2)) \cup update_{\mu DNF}(\Delta_3, e(a, \Delta_3) \cup \\ &\qquad update_{\mu DNF}(\Delta_4, e(a, \Delta_4))) \\ &= \{\{\neg f, g, \neg h\}, \{f, \neg g, h, k\}, \{f, \neg g, \neg h\}\}\end{aligned}$$

In Section 3, we proved that $enb_{\mu DNF}(a, \Delta)$ contains the minimum number of μDNF-states among the enabling forms of Δ for a. Thus, Algorithm 5 only needs to make a minimal number of calls to $update_{\mu DNF}$ as well as to pass the minimum number of μDNF-states to the $merge_{\mu DNF}$ function. In turn, we observe that $update_{\mu DNF}$ and $merge_{\mu DNF}$ also consider each partial state in each μDNF-state. Hence, the performance of $\Phi_{\mu DNF}$ also depends on the total number of partial states of all μDNF-states in $enb_{\mu DNF}(a, \Delta)$. We have the following proposition.

Proposition 16. *For a μDNF-state Δ and an action a, $\bigcup_{\Delta_i \in enb_{\mu DNF}(a,\Delta)} \Delta_i$ is also a μDNF-state and every δ in $\bigcup_{\Delta_i \in enb_{\mu DNF}(a,\Delta)} \Delta_i$ belongs to one and only one μDNF-state in $enb_{\mu DNF}(a, \Delta)$.*

The above proposition shows that the union set of $enb_{\mu DNF}(a, \Delta)$ is also minimal and there is no partial state that belongs to more than one μDNF-state in $enb_{\mu DNF}(a, \Delta)$. Next, we will analyze the computational properties of $\Phi_{\mu DNF}$ as well as the running time of the algorithms for computing this function.

4.2. Computing Successor μDNF-States Using $\Phi_{\mu DNF}$

We will now present an instantiation of the procedure $\texttt{extending}_{\mathcal{R}}$ (Algorithm 3), $\texttt{extending}_{\mu\texttt{DNF}}$ (Algorithm 6), and an instantiation of the procedure $\texttt{phi}_{\mathcal{R}}$ (Algorithm 5), $\texttt{phi}_{\mu\texttt{DNF}}$ (Algorithm 8). This is because these procedures involve the computation of the operators $conv_{\mu DNF}$, $update_{\mu DNF}$, and $merge_{\mu DNF}$. For ease of following, we also present a slight variation of `enabling` (Algorithm 4), $\texttt{enabling}_{\mu\texttt{DNF}}$ (Algorithm 7), where the procedure $\texttt{extending}_{\mu\texttt{DNF}}$ is used in place of $\texttt{extending}_{\mathcal{R}}$.

The procedure $\texttt{extending}_{\mu\texttt{DNF}}(\Delta, e(a, \Delta), \phi \rightarrow \eta)$ computes $\Delta \oplus_{\mu DNF} \psi$ by computing

$$\Delta_1 = conv_{\mu DNF}(\Delta \wedge \psi) \text{ and } \Delta_2 = conv_{\mu DNF}(\Delta \wedge \overline{\psi})$$

according to Definition 13 (Lines 4-5, 8-11). It starts by initializing Δ_1 and Δ_2 with the set of partial states satisfying ψ and $\neg\psi$ in Δ, respectively (Lines 4-5). The computation in the loop (Lines 8-11) updates Δ_1

Algorithm 6 $\texttt{extending}_{\mu\texttt{DNF}}(\Delta, \mathit{effect}, \psi \to \eta)$ Computing $\Delta \oplus_{\mu DNF} \psi$ and the effects of a

Input: μDNF-state Δ, set of known effects effect, conditional effect $\psi \to \eta$
Output: $\{\langle \Delta_i, e(a, \Delta_i)\rangle \mid \Delta_i \in \Delta \oplus_{\mu DNF} \psi\}$
Initialize: Δ_0, Δ_1, Δ_2, and $Result$
  $\Delta_1 = \{\delta \mid \delta \in \Delta, \psi \subseteq \delta\}$ $\{\delta \models \psi\}$
  $\Delta_2 = \{\delta \mid \delta \in \Delta, \delta \cap \overline{\psi} \neq \emptyset\}$ $\{\delta \models \neg\psi\}$
  $\Delta_0 = \Delta \setminus (\Delta_1 \cup \Delta_2)$ $\{\Delta_0 = \{\delta \mid \delta \in \Delta, \psi \text{ is unknown in } \delta\}\}$
  $Result = \emptyset$
for each δ in Δ_0 **do**
  $\Delta_1 = min(\Delta_1 \cup \{\delta \cup \psi\})$
  $\Delta_2 = min(\Delta_2 \cup \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\})$
end for
if $\Delta_1 \neq \emptyset$ **then**
  Set $e(\Delta_1) = \mathit{effect} \cup \eta$ $\{\Delta_1 \models \psi$ so η is added to $e(a, \Delta_1)\}$
  Set $Result = Result \cup \{\langle \Delta_1, e(\Delta_1)\rangle\}$
end if
if $\Delta_2 \neq \emptyset$ **then**
  Set $e(\Delta_2) = \mathit{effect}$ $\{\Delta_2 \models \neg\psi$ so η is not added to $e(a, \Delta_2)\}$
  Set $Result = Result \cup \{\langle \Delta_2, e(\Delta_2)\rangle\}$
end if
return $Result$

(resp. Δ_2) by adding to it $\delta \cup \psi$ (resp. $\{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\}$) for each partial state $\delta \in \Delta$ such that ψ is unknown in δ and maintaining its minimality. η is added to $e(\Delta_1)$ but not added to $e(\Delta_2)$ ($e(\Delta_i)$ is used to compute $e(a, \Delta_i)$ for $\Delta_i \in enb_{\mu DNF}(a, \Delta)$) since $\Delta_1 \models \psi$ and $\Delta_2 \models \neg\psi$. It is worth mentioning that $\psi \models \delta$ is implemented by the test $\psi \subseteq \delta$ because both ψ and δ are conjunction of literals. For the same reason, $\psi \models \neg\delta$ is implemented by the test $\delta \cap \psi \neq \emptyset$.

Algorithm 7 $\texttt{enabling}_{\mu\texttt{DNF}}(a, \varphi)$ Computing $enb_{\mu DNF}(a, \varphi)$ and the effects of a in these μDNF-states

Input: μDNF-state Δ, action a
Output: $\{\langle \Delta', e(a, \Delta')\rangle \mid \Delta' \in enb_{\mu DNF}(a, \Delta)\}$
Set $e(\Delta) = \emptyset$
Let $Result = \{\langle \Delta, e(\Delta)\rangle\}$
Let $\{\psi_i \to \eta_i \mid i = 1, \ldots, p\}$ be the set of combined conditional effects of a
for $i = 1$ **to** p **do**
  Let $X = \emptyset$
  for each $\langle \Delta', e(\Delta')\rangle \in Result$ **do**
    $X = X \cup \texttt{extending}_{\mu\texttt{DNF}}(\Delta', e(\Delta'), \psi_i \to \eta_i)$ {Algorithm 6}
  end for
  Set $Result = X$
end for
return $Result$

Proposition 17. *For a μDNF-state Δ and a combined conditional effect $a : \psi \to \eta$,*

$\texttt{extending}_{\mu\texttt{DNF}}(\Delta, \mathit{effects}, \psi \rightarrow \eta)$
returns $\{\langle \Delta_i, \mathit{effects} \cup \{\ell \mid a : \psi \rightarrow \ell \wedge \Delta_i \models \psi\}\rangle \mid \Delta_i \in \Delta \oplus_{\mu DNF} \psi\}$.

Since $update_{\mu DNF}$ and $merge_{\mu DNF}$ functions are simple and require only set operations, to avoid duplicate checking for non-minimal partial states by both functions in Definitions 14 and 15, we implement the two functions interspersedly in Algorithm 8 (Lines 7-13).

Algorithm 8 $\texttt{phi}_{\mu\texttt{DNF}}(a, \Delta)$ Computing $\Phi_{\mu DNF}(a, \Delta)$

1: **Input**: μDNF-state Δ, action a
2: **Output**: μDNF-state $\Phi_{\mu DNF}(a, \Delta)$
3: **if** $\Delta \not\models pre(a)$ **then**
4: **return** $undefined$
5: **end if**
6: Let $X = \texttt{enabling}(a, \Delta)$ {Algorithm 4}
7: Let $\Delta_{succ} = \emptyset$
8: **for** each $\langle \Delta', \mathit{effect}\rangle$ in X **do**
9: **for** each δ in Δ' **do**
10: Compute $\delta' = \delta \setminus \overline{\mathit{effect}} \cup \mathit{effect}$
11: Set $\Delta_{succ} = min(\Delta_{succ} \cup \{\delta'\})$
12: **end for**
13: **end for**
14: **return** Δ_{succ}

One can easily see the correctness of Algorithm 8 for computing $\Phi_{\mu DNF}(a, \Delta)$ as follows.

Proposition 18. *For a μDNF-state Δ and an action a,* $\texttt{phi}_{\mu\texttt{DNF}}(a, \Delta)$ *returns* $\Phi_{\mu DNF}(a, \Delta)$.

The computational cost of $\Phi_{\mu DNF}(a, \Delta)$, denoted by $T(\Phi_{\mu DNF}(a, \Delta))$, is shown in the following theorem.

Theorem 4. *Let n be the number of propositions in the domain, Δ be a μDNF-state, and a be an action. Let k be the number of e-conditions of a, that are unknown in Δ. Let r be the size of the largest e-condition of a. Then the computational complexity of $\Phi_{\mu DNF}(a, \Delta)$ is*

$$T(\Phi_{\mu DNF}(a, \Delta)) = O(n|\Delta|^2(1 + r)^{2k})$$

It is shown in the analysis (the proof of Theorem 4) that the cost of computing $\Phi_{\mu DNF}(a, \Delta)$ lies mostly in the cost of computing $enb_{\mu DNF}(a, \Delta)$ and the cost of merging, most of which, in turn, lie in the cost of computing the min function—which is quadratic in the size of the μDNF-states.

Theorem 4 shows that the computation of a successor μDNF-state is exponential in k, the number of e-conditions of the action that are unknown in the μDNF-state, with the base of $r + 1$, where r is the number of literals in the largest e-condition of the action. This is, however, true only in the worst case, where every literals in every unknown e-condition introduced to the extension process is unknown in every partial state of every μDNF-states in the set X and every unknown e-condition has the largest size r. In fact, the actual cost of computing $\Phi_{\mu DNF}$ and the size of μDNF-states in $Result$ can be much smaller as each e-condition ψ is usually unknown in a small portion of partial states in the μDNF-states in X and for each partial state

δ in which ψ is unknown, not all literals in ψ are always unknown in δ. On the other hand, as observed in most benchmarks, the majority of e-conditions of an action have size 1 or 2, and very few of them have size 3 or greater—as a matter of fact, we did not encounter any e-condition with more than 3 literals. Hence, $\Phi_{\mu DNF}$ is polynomial for problems where the number of unknown e-conditions of the actions is small.

4.3. DNF *Conformant Planner*

In this subsection, we will introduce a progression-based conformant planner, called DNF, that employs the μDNF representation along with the transition function $\Phi_{\mu DNF}$ for computing successor μDNF-states in a best-first search in the belief space for conformant solutions.

We implement the DNF planner by modifying the CPA system [22, 28]. This choice was motivated by the fact that both CPA and DNF rely on disjunctive normal form formulae for representing belief states. This also allows DNF to make use of several preprocessing techniques developed in the recent CPA [28], including backward-chaining, forward-chaining, and one-of combination for reducing the size of the DNF formula representing the initial belief state.

In this paper, we use two heuristic functions in parallel as follows. Given a μDNF-state Δ, the heuristic function used in DNF is built on the following values:

- $h_{goal}(\Delta)$: the number of subgoals satisfied by Δ.
- $h_{card}(\Delta)$: the number of partial states contained in Δ (cardinality of Δ).
- $h_{lit}(\Delta)$: the number of known literals in Δ.
- $h_{dis}(\Delta)$: the *square distance* from Δ to the goal, defined by

$$h_{dis}(\Delta) = \Sigma_{\delta \in \Delta}(|G| - h_{goal}(\delta))^2$$

  where G is the goal of the problem.

The first heuristic function used in DNF is defined by triples of the form $\langle h_{goal}(\Delta), h_{card}(\Delta), h_{dis}(\Delta)\rangle$ in the lexicographical order. The second heuristic function has the form $\langle h_{goal}(\Delta), h_{lit}(\Delta), h_{dis}(\Delta)\rangle$.

While the first two values are inspired by CPA and other planners, the number of known literals helps in reducing the uncertainty and the size of the μDNF-state. The last feature aims at promoting μDNF-states in which the satisfaction of goals among its partial states is uniform.

5. Minimal-CNF Representation

In this section, we present our second representation, called *minimal-CNF* and denoted by μCNF, which is a set of conjunctive normal (CNF) formulae that satisfy certain compactness criteria. We will show how to instantiate the function $\Phi_{\mathcal{R}}$ in the development of the transition function $\Phi_{\mu CNF}$ for this representation. We then discuss the computational aspects of $\Phi_{\mu CNF}$. We start with some basic notions and preprocessing techniques for CNF formula.

5.1. Background: CNF Formulae and Preprocessing Techniques

A *clause* is a disjunction of literals. A *CNF formula* is a conjunction of clauses. A CNF formula is often viewed as a set of clauses, while a clause can be viewed as a set of literals. A clause is said to be *trivial* if it contains the set $\{\ell, \bar{\ell}\}$ for some literal ℓ. Removing the trivial clause(s) from a CNF formula φ results in a CNF formula equivalent to φ.

A *unit clause* is a singleton set, i.e., it contains only one literal. The literal in a unit clause is called a *unit literal*.

Let α and β be two clauses. We say that α *subsumes* β if $\alpha \subset \beta$. A clause is said to be subsumed by a CNF formula if it is subsumed by some clause in the CNF formula. A CNF formula φ can be simplified into an equivalent CNF formula by removing from φ the clause(s) subsumed by another clause in φ. This technique is called *subsumption*. Given a CNF formula φ, by $r(\varphi)$ we denote the CNF formula obtained by removing from φ every clause that is trivial or subsumed by another clause in φ. We have $\varphi \equiv r(\varphi)$. If $\varphi = r(\varphi)$ then φ is said to be a *reduced* CNF formula.

Proposition 19. *Let φ be a CNF formula, α and β be two clauses. Then*

1. $r(\varphi \cup \{\alpha\}) = r(r(\varphi) \cup \{\alpha\})$
2. $r(r(r(\varphi) \cup \{\alpha\}) \cup \{\beta\}) = r(r(r(\varphi) \cup \{\beta\}) \cup \{\alpha\})$

Let φ be a CNF formula and let α be a clause such that $\varphi \models \alpha$. α is called an *implicate of* φ. If there exists in φ some clause(s) subsumed by α then φ can be simplified into an equivalent CNF formula by replacing the set $\{\beta \mid \beta \in \varphi \wedge \alpha \subset \beta\}$ in φ with α. Observe that if $\{\ell\}$ is a unit clause of φ, then φ can be simplified by removing from φ every clause α that contains ℓ since α is subsumed by ℓ. On the other hand, if a clause β in φ contains $\bar{\ell}$ then β can be shortened to $\beta' = \beta \setminus \{\bar{\ell}\}$ because $\bar{\ell}$ is false in φ. φ then is simplified by replacing β with β'. If $|\beta| = 2$ then β' is a unit clause. The formula now can be simplified w.r.t. the new unit clause β'. This process is referred to as *unit propagation*.

Let α and β be two clauses. α is said to be *resolvable with* β if there exists a literal ℓ such that $\ell \in \alpha$, $\bar{\ell} \in \beta$, and $(\alpha \setminus \{\ell\}) \cup (\beta \setminus \{\bar{\ell}\})$ is a nontrivial clause. In this case, we say that α is *resolvable with* β *on* ℓ and $(\alpha \setminus \{\ell\}) \cup (\beta \setminus \{\bar{\ell}\})$ is called the *resolvent of* α *and* β, denoted by $\alpha|\beta$. One can prove that $\alpha \wedge \beta \models \alpha|\beta$. If α and β belong to a CNF formula φ then $\alpha|\beta$ is called a *resolvent of* φ. It is easy to see that $\alpha|\beta$ is also an implicate of φ. If there exist in φ a clause that is subsumed by $\alpha|\beta$ then φ can be simplified to an equivalent CNF formula by replacing from φ every clause subsumed by $\alpha|\beta$. This technique is referred to as *subsumable resolution*.

Given two CNF formulae $\varphi = \{\alpha_1, \ldots, \alpha_n\}$ and $\psi = \{\beta_1, \ldots, \beta_m\}$. The cross-product of φ and ψ, denoted by $\varphi \times \psi$, is the CNF formula defined by

$$\varphi \times \psi = \{\alpha \cup \beta \mid \alpha \in \varphi, \beta \in \psi\}$$

Observe that $\times$ is nothing but a standard transformation of disjunction of CNF formulae into an equivalent CNF formula. Since $\times$ is commutative and associative, we generalize the definition of $\times$ for a set of CNF formulae $\Psi = \{\varphi_1, \ldots, \varphi_n\}$, called cross-product of Ψ and denoted by $\times[\Psi]$, as follows

$$\times[\Psi] = \varphi_1 \times \varphi_2 \times \cdots \times \varphi_n$$

We have that $\times[\Psi]$ is a CNF formula and $\times[\Psi] \equiv \bigvee_{\psi_i \in \Psi} \psi_i$.

The reduced-cross-product of φ and ψ, denoted by $\varphi \otimes \psi$, is the reduced CNF formula defined by

$$\varphi \otimes \psi = r(\varphi \times \psi)$$

One can see that $\otimes$ is commutative as shown in the following proposition.

Proposition 20. *Let φ, ψ_1, and ψ_2 be three CNF formulae. Then,*

$$(\varphi \otimes \psi_1) \otimes \psi_2 = (\varphi \otimes \psi_2) \otimes \psi_1$$

Proposition 20 allows us to define the reduced-cross-product of a set of CNF formulae $\Psi = \{\varphi_1, \ldots, \varphi_n\}$, denoted by $\otimes[\Psi]$, as follows

$$\otimes[\Psi] = \varphi_1 \otimes \varphi_2 \otimes \cdots \otimes \varphi_n$$

One can easily prove the correctness of the following proposition.

Proposition 21. *Let $\Psi = \{\varphi_1, \ldots, \varphi_n\}$ be a set of CNF formulae. Then $\otimes[\Psi]$ is a reduced CNF formula and*

$$\otimes[\Psi] \equiv \varphi_1 \vee \ldots \vee \varphi_n$$

5.2. The μCNF Representation and Its Operators

Definition 16. *A CNF formula φ is said to be* minimal *if φ is a reduced CNF formula and there is no resolvent of φ that subsumes a clause in φ.*

A minimal-CNF formula is a reduced CNF formula that cannot be simplified using the subsumable resolution technique. The commutativity of the function $r()$ (Proposition 19) allows us to define a recursive function, denoted by μ_{CNF}, that maps a CNF formula to a minimal-CNF formula as follows.

Definition 17. *Let φ be a CNF formula. By $\mu_{CNF}(\varphi)$ we denote the CNF formula defined as follows.*

$$\mu_{CNF}(\varphi) = \begin{cases} r(\varphi) & \text{if } r(\varphi) \text{ is minimal} \\ r(r(\varphi) \cup \{\rho\}) & \text{for some resolvent } \rho \text{ of } r(\varphi) \text{ that subsumes at least a clause in } r(\varphi) \end{cases}$$

Example 6. *Given a CNF formula*

$$\varphi = \{\{f, g, \neg g\}, \{f, \neg k, h\}, \{\neg g, h, k\}, \{f, \neg k\}, \{\neg g, h\}, \{g, k\}, \{\neg h, k\}, \{f, h, k\}\}$$

After removing trivial clauses from φ, the first clause $\{f, g, \neg g\}$, we obtain

$$\varphi_1 = \{\{f, \neg k, h\}, \{\neg g, h, k\}, \{f, \neg k\}, \{\neg g, h\}, \{g, k\}, \{\neg h, k\}, \{f, h, k\}\}$$

After removing every subsumed clause from φ_1, the first two clauses in φ_1 are gone as they are subsumed by the third and fourth clauses respectively. Hence,

$$\varphi_2 = r(\varphi) = \{\{f, \neg k\}, \{\neg g, h\}, \{g, k\}, \{\neg h, k\}, \{f, h, k\}\}$$

Observe that the second and the third clauses in φ_2 are resolvable on g and their resolvent is $\{h, k\}$, which subsumes the last clause in φ_2. Hence, φ_2 can be simplified to

$$\{\{f, \neg k\}, \{\neg g, h\}, \{g, k\}, \{\neg h, k\}, \{h, k\}\}$$

by replacing $\{f, h, k\}$ with $\{h, k\}$.

The last two clauses in the updated φ_2 are resolvable on h and their resolvent is the unit clause $\{k\}$ which subsumes the last three clauses. Thus, the formula can be simplified further to

$$\{\{f, \neg k\}, \{\neg g, h\}, \{k\}\}$$

In this new CNF formula, $\{f, \neg k\}$ and $\{k\}$ are resolvable on k and their resolvent is the unit clause $\{f\}$ which subsumes the first clause. Hence, the first clause is replaced with $\{f\}$ to obtain

$$\{\{f\}, \{\neg g, h\}, \{k\}\}$$

This formula does not contain a pair of resolvable clauses so the last formula is returned as the result of $\mu_{CNF}(\varphi)$.

One can observe from the above example that unit propagation is a special case of the subsumable resolution technique. It is easy to see that, after applying the aforementioned techniques for preprocessing a CNF formula, we obtain a CNF formula satisfying the following properties.

Proposition 22. *Let φ be a satisfiable CNF formula:*

1. *$\mu_{CNF}(\varphi)$ is a minimal-CNF formula equivalent to φ.*
2. *μ_{CNF} is idempotent. That is, if φ is minimal then $\mu_{CNF}(\varphi) = \varphi$.*

Proposition 22 shows that for each satisfiable CNF formula φ there exists a minimal-CNF formula $\mu_{CNF}(\varphi)$ equivalent to φ. This allows the definition of minimal-CNF representation as follows.

Definition 18. *The* minimal-CNF representation*, denoted by μCNF, is the set of minimal-CNF formulae. A μCNF-*state *is a minimal-CNF formula.*

To complete the definition of the function $\Phi_{\mu CNF}$ for μCNF, we need to define the operators $conv_{\mu CNF}$, $update_{\mu CNF}$, and $merge_{\mu CNF}$. Recall that $conv_{\mu CNF}$ converts the formulae of the forms $\varphi \wedge \psi$ and $\varphi \wedge \neg\psi$ into equivalent μCNF-states, where φ is a μCNF-state and ψ is a consistent set of literals. Observe that $\varphi \wedge \psi$ is a CNF formula, where ψ is the set of unit clauses $\{\{\ell\} \mid \ell \in \psi\}$; and $\varphi \wedge \neg\psi$ is also a CNF formula where $\neg\psi$ is the (nontrivial) clause $\overline{\psi} = \{\overline{\ell} \mid \ell \in \psi\}$. Thus, we can simply use μ_{CNF} function for transforming those CNF formulae into equivalent μCNF-states.

Definition 19. *Let φ be a μCNF-state and ψ be a consistent set of literals. Then*

$$conv_{\mu CNF}(\varphi \wedge \psi) = \mu_{CNF}(\varphi \cup \{\{\ell\} \mid \ell \in \psi\})$$

$$conv_{\mu CNF}(\varphi \wedge \neg\psi) = \mu_{CNF}(\varphi \cup \{\overline{\psi}\})$$

Proposition 23. *Let φ be a μCNF-state and ψ be a consistent set of literals. Then,*

1. *$conv_{\mu CNF}(\varphi \wedge \psi)$ is a μCNF-state equivalent to $\varphi \wedge \psi$*
2. *$conv_{\mu CNF}(\varphi \wedge \neg\psi)$ is a μCNF-state equivalent to $\varphi \wedge \neg\psi$*

For the definition of $update_{\mu CNF}$ and $merge_{\mu CNF}$, we need the following notions. Let φ be a CNF formula and ℓ be a literal.

- φ_ℓ denotes the set of clauses in φ which contain ℓ.
- $\varphi - \ell$ denotes the CNF formula obtained from φ by removing every occurrence of literal ℓ.

Example 7. *Let* $\varphi = \{\{p, \neg q\}, \{q, \neg r\}, \{p, r\}\}$*, then*
$\varphi_p = \{\{p, \neg q\}, \{p, r\}\}$
$\varphi_{\neg q} = \{\{p, \neg q\}\}$
$\varphi - p = \{\{\neg q\}, \{q, \neg r\}, \{r\}\}$

To define the update function $update_{\mu CNF}$ for μCNF, first we need to define the update of a reduced CNF formula by a literal as follows.

Definition 20. *Let* φ *be a reduced CNF-formula and* ℓ *be a literal. The update of* φ *by* ℓ*, denoted by* $update_r(\varphi, \ell)$*, is the reduced CNF-formula defined as follows:*

$$update_r(\varphi, \ell) = \begin{cases} \varphi & \text{if } \{\ell\} \in \varphi \\ r(\varphi \setminus \{\{\bar{\ell}\}\} \cup \{\{\ell\}\}) & \text{if } \{\bar{\ell}\} \in \varphi \\ r(\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \cup \{\{\ell\}\} \cup (\varphi_\ell - \ell) \times (\varphi_{\bar{\ell}} - \bar{\ell})) & \text{otherwise} \end{cases}$$

By this definition, $update_r(\varphi, \ell)$ represents the set of states obtained by updating the set of states $BS(\varphi)$ so that ℓ becomes true in the new states as shown in the following.

Proposition 24. *Let* φ *be a CNF formula and* ℓ *be a literal. Then,*

$$update_r(\varphi, \ell) \equiv \{s \setminus \{\bar{\ell}\} \cup \{\ell\} \mid s \in BS(\varphi)\}$$

The definition is illustrated in the next few examples:
- $update_r(\{\{f\}, \{g, \neg p\}\}, p) = \{\{f\}, \{p\}\}$
- $update_r(\{\{f\}, \{h, p\}\}, p) = \{\{f\}, \{p\}\}$
- $update_r(\{\{f\}, \{g, \neg p\}, \{h, p\}\}, p) = \{\{f\}, \{p\}, \{g, h\}\}$

The following proposition shows that $update_r$ is commutative over the family of reduced CNF formulae.

Proposition 25. *Let* φ *be a reduced CNF formula and* ℓ_1 *and* ℓ_2 *be two literals such that* $\ell_1 \neq \bar{\ell}_2$*, then*

$$update_r(update_r(\varphi, \ell_1), \ell_2) = update_r(update_r(\varphi, \ell_2), \ell_1)$$

This proposition (together with Lemma 1) shows that the result of updating a reduced CNF-formula φ by a consistent set of literals is independent from the order in which the literals are introduced. This allows us to define the function $update_{\mu CNF}$ as follows.

Definition 21. *Let* φ *be a* μCNF*-state and* η *be a consistent set of literals. Then* $update_{\mu CNF}(\varphi, \eta)$ *is the* μCNF*-state defined as*

$$update_{\mu CNF}(\varphi, \eta) = \mu_{CNF}(update_r(\varphi, \eta))$$

where

$$update_r(\varphi, \eta) = \begin{cases} \varphi & \text{if } \eta = \emptyset \\ update_r(update_r(\varphi, \eta \setminus \{\ell\}), \ell) & \text{if } \exists \ell \in \eta \end{cases}$$

Proposition 26. *Let* φ *be a* μCNF*-state and* η *be a consistent set of literals. Then*

$$BS(update_{\mu CNF}(\varphi, \eta)) = \{s \setminus \overline{\eta} \cup \eta \mid s \in BS(\varphi)\}$$

The $merge_{\mu CNF}$ function for μCNF-states can be defined easily as follows.

Definition 22. *Let Λ be a set of μCNF-states. Then, $merge_{\mu CNF}(\Lambda)$ is defined as*

$$merge_{\mu CNF}(\Lambda) = \mu_{CNF}(\otimes[\Lambda])$$

The next proposition is derived directly from the above definition and Proposition 21.

Proposition 27. *Let Λ be a set of μCNF-states. Then, $merge_{\mu CNF}(\Lambda)$ is a μCNF-state and*

$$BS(merge_{\mu CNF}(\Lambda)) = \bigcup_{\varphi \in \Lambda} BS(\varphi)$$

Thus, the function $\Phi_{\mu CNF}$ has been completely defined and its correctness proved. We next present the computational aspects of this function.

5.3. Computational Aspects and Complexity of $\Phi_{\mu CNF}$

In this subsection we will discuss the aspects of computing $\Phi_{\mu CNF}$, the cost of computing its component functions, and then the total cost of computing this function.

Let φ be a μCNF-state and a be an action executable in φ. Recall that, to compute $\Phi_{\mu CNF}(a, \varphi)$, we need to compute $enb_{\mu CNF}(a, \varphi)$ (Algorithm 4), updating the μCNF-states in this set, and then merging them into the successor μCNF-state. All these three stages will generate intermediate CNF-formulae. For simplicity of the investigation of this computational cost, we assume that the size and the number of clauses of each intermediate CNF formula generated during the computation of $\Phi_{\mu CNF}(a, \varphi)$ do not exceed those of φ, where the size of a CNF formula φ' is defined as $\Sigma_{\alpha \in \varphi'} |\alpha|$. We will show during the discussion of the computational cost of these stages later that, unlike the $\Phi_{\mu DNF}$ function, this assumption, called *size limit*, is reasonable.

For a complete computation of the μCNF-state $\Phi_{\mu CNF}(a, \varphi)$, we need to instantiate Algorithm 5 by providing an algorithm for μ_{CNF}, $update_{\mu CNF}$ and $merge_{\mu CNF}$. The computation of $r(.)$ is quite straightforward so we omit the algorithm for computing this function.

We will start with the problem of checking satisfaction of a formula in a μCNF-state. This problem, unlike that for the minimal-DNF representation, has been known to be NP-hard and, hence, is worth to be discussed.

5.3.1. Satisfaction Checking

In the computation of $\Phi_{\mu CNF}(a, \varphi)$, we need to check whether a set of literals (e.g., $pre(a)$ or an e-condition of a) or a clause (e.g., the negation of an e-condition of a) is satisfied in the μCNF-state φ. Let ψ be a satisfiable non-tautological formula. In general, to check whether ψ is satisfied in φ, we need to check whether $\varphi \wedge \neg\psi$ is unsatisfiable by using a SAT-solver. In the worst case, this computation is exponential in the number of propositions in $prop(\varphi \wedge \neg\psi)$. Thus, reducing the number of calls to the SAT-solver is important. In what follows we will consider several cases where satisfaction checking can be performed in an easier manner. To check whether φ satisfies ψ, we can avoid calling a SAT-solver if any of the following *quick checking cases* is applicable:

- If ψ is independent from φ then $\varphi \not\models \psi$ (by Proposition 4).
- If ψ is a consistent set of literals and there is a literal ℓ in ψ such that ℓ does not belong to any clause in φ, then $\varphi \not\models \psi$. This is because $\varphi \not\models \ell$ (by Proposition 4) and $\psi \models \ell$.

- If ψ is a consistent set of literals and there exists ℓ in ψ such that $\bar{\ell}$ is a unit literal of φ then $\varphi \models \neg\psi$.
- If ψ is a non-trivial clause and there exists a clause β in φ such that $\beta \subseteq \psi$ then $\varphi \models \psi$.

If no quick checking case is applicable, we need to call a SAT-solver ($\varphi \models \psi$ iff $\varphi \wedge \neg\psi$ is unsatisfiable). To reduce the workload of the SAT-solver, we consider the relation between the literals in φ and ψ (excluding the quick checking cases) as follows.

For every unit literal ℓ in φ, every occurrence of ℓ or $\bar{\ell}$ in ψ can be replaced with $true$ or $false$ respectively. For example, if ψ is a set of literals, to check whether ψ is satisfied in φ, we only need to reason about the satisfiability of $\varphi \wedge \neg\psi'$ where ψ' is obtained by removing every literal which is a unit literal in φ. On the other hand, to check whether a clause ψ is satisfied in φ, we can check the satisfiability of $\varphi' \wedge \overline{\psi'}$ where φ' obtained by removing every unit clause $\{\ell\}$ from φ such that $\bar{\ell} \in \psi$ and ψ' is obtained by removing every such unit literal ℓ from ψ.

For further simplification, we use the following technique. Let φ be a μCNF-state and ψ be a formula. A clause α in φ is said to be *relating to* ψ if

- α is dependent on ψ, or
- α is dependent on a clause β in φ and β is relating to ψ.

The set of clauses in φ that are relating to ψ is denoted by $Rel(\varphi, \psi)$. We have the following proposition.

Proposition 28. *Let φ be a μCNF-state and ψ be a satisfiable non-tautological formula dependent on φ.*

1. *$\varphi \models \psi$ iff $Rel(\varphi, \psi) \models \psi$.*
2. *$\varphi \wedge \psi$ is satisfiable iff $Rel(\varphi, \psi) \wedge \psi$ is satisfiable.*

Proposition 28 shows that to check whether ψ is (not) true in a μCNF-state φ, we can check whether ψ is (not) true in its subset $Rel(\varphi, \psi)$. It is easy to see that the procedure $Relation(\varphi, \psi)$ (Algorithm 9) computes this set.

Algorithm 9 first computes the set of clauses in φ that depend on ψ (Lines 3-9). For each clause α in X (where X represents the set of clauses newly added to R, i.e., $Rel(\varphi, \psi)$), it computes the set Y of clauses in V ($V = \varphi \setminus R$) that depends on α (Line 14), and updates R, X, and V according to Y (Line 16). Whenever a clause α in X is taken to compute Y—the set of clauses in φ that are not (yet) in R and depend on α—it is immediately removed from X (Line 13), since every possible clause in φ dependent on α is already in R. The running time of Algorithm 9 is described in the following proposition.

Proposition 29. *Let φ be a μCNF-state and ψ be a formula. Let m be the number of clauses in φ and n be the number of propositions in the domain. Then $Rel(\varphi, \psi)$ can be computed in $O(nm^2)$ time.*

Thus, $Rel(\varphi, \psi)$ can be computed quite quickly. For a literal ℓ dependent on a μCNF-state φ, we say that ℓ is *possibly unknown* in φ if neither $\{\ell\}$ nor $\{\bar{\ell}\}$ are unit clauses of φ; in this case, a call to a SAT-solver is required to determine whether ℓ is known in φ. In conformant planning, ψ is usually a literal or a small set of literals, among which even fewer (or no) literals are possibly unknown in φ. Hence, after applying the simplification techniques presented earlier, the formula to be passed to the SAT-solver ($Rel(\varphi', \psi') \wedge \neg\psi'$) is usually small—and, in particular, it can be significantly smaller than $\varphi \wedge \neg\psi$, the formula passed to the SAT-solver if the above simplification techniques are not used.

Algorithm 9 $Relation(\varphi, \psi)$ Computing $Rel(\varphi, \psi)$

1: **Input**: μCNF-state φ, a formula ψ
2: **Output**: $Rel(\varphi, \psi)$
3: Compute $\pi = prop(\psi) \cup \overline{prop(\psi)}$
4: Let $R = \emptyset$ {Initialize $Rel(\varphi, \psi)$ with the empty set}
5: **for** each α in φ **do**
6: **if** $\alpha \cap \pi \neq \emptyset$ **then**
7: Set $R = R \cup \{\alpha\}$
8: **end if**
9: **end for**
10: Let $X = R$ {X is the set of clauses in φ newly added to R}
11: Let $V = \varphi \setminus R$ {V is the set of clauses in φ that are not (yet) added to R}
12: **while** $X \neq \emptyset$ **and** $V \neq \emptyset$ **do**
13: Let α be a clause in X
14: Set $X = X \setminus \{\alpha\}$
15: Compute $Y = \{\beta \mid \beta \in V$ and β depends on $\alpha\}$
16: **if** $Y \neq \emptyset$ **then**
17: Set $R = R \cup Y$, $X = X \cup Y$, $V = V \setminus Y$
18: **end if**
19: **end while**
20: **return** R

5.3.2. Computing μ_{CNF}

Let φ be a CNF formula. The μCNF-state $\mu_{CNF}(\varphi)$ is computed by the procedure `CNFstate`(φ) (Algorithm 10). To avoid repeating computation of resolvents and subsumption checking, we use the set $treated$ initialized with the empty set (Line 4). Whenever a clause α of φ is taken to compute the resolvent with every other clause in φ, α is added to $treated$. The algorithm terminates and returns the result when every clause in the CNF formula being considered has been treated. One can verify the correctness of Algorithm 10, i.e., the procedure `CNFstate`(φ) returns $\mu_{CNF}(\varphi)$.

The cost of computing $\mu_{CNF}(\varphi)$ using Algorithm 10 is analyzed in the following proposition.

Proposition 30. *Let φ be a CNF formula, N be the size of φ, m be the number of clauses in φ, and n be the number of propositions in the domain. Then $\mu_{CNF}(\varphi)$ can be computed in $O(nm^2N)$ time.*

5.3.3. Computational Complexity of $enb_{\mu CNF}(a, \varphi)$ (Algorithm 4)

To investigate the cost of computing $enb_{\mu CNF}(a, \varphi)$, we first evaluate the running time of `extending`$_{\mathcal{R}}$ (Algorithm 3) for computing $\gamma \oplus_{\mu CNF} \psi$ and the effect $e(\gamma)$, where γ is a μCNF-state stored in the set X in Algorithm 4 and ψ is an e-condition of a. We have the following proposition.

Proposition 31. *Let φ be a μCNF-state, N be the size of φ, m be the number of clauses in φ, and n be the number of propositions in the domain. Let ψ' be the set of literals in ψ that are possibly unknown in φ and $u = |prop(Rel(\varphi, \psi'))|$. Computing $\varphi \oplus_{\mu CNF} \psi$ (Algorithm 3) has time complexity $O(2^u N + nm^2N)$, if ψ is unknown in φ, and it requires $O(2^u N + n)$, otherwise.*

One can observe (see the proof of Proposition 31) that most of the running time of Algorithm 4 for computing $enb_{\mu CNF}(a, \varphi)$ lies in the execution of procedure `extending`$_{\mathcal{R}}$ (Line 9). After the execution

Algorithm 10 `CNFstate`(φ) Computing $\mu_{CNF}(\varphi)$

Input: CNF-formula φ
Output: μCNF-state $\mu_{CNF}(\varphi)$
Compute $\varphi = r(\varphi)$
Set $treated = \emptyset$
while $\varphi \setminus treated \neq \emptyset$ **do**
  Let α be a clause in $(\varphi \setminus treated)$
  Compute $\varphi_0 = \{\alpha | \beta \mid \beta \in \varphi, \beta$ is resolvable with $\alpha\}$
  for each β in φ_0 **do**
    Compute $\varphi' = \{\beta' \mid \beta' \in \varphi \wedge \beta \subset \beta'\}$ {φ' is the set of clauses in φ subsumed by the resolvent β}
    if $\varphi' \neq \emptyset$ **then**
      Set $\varphi = \varphi \setminus \varphi' \cup \{\beta\}$ {replaces the non-empty set of clauses subsumed by β in φ with β}
    end if
  end for
  Set $treated = treated \cup \{\alpha\}$
end while
return φ

of this procedure for each μCNF-state φ' in X, we obtain either the same μCNF-state φ' (if ψ is known in φ') or the pair of two formulae $conv_{\mu CNF}(\varphi' \wedge \psi)$ and $conv_{\mu CNF}(\varphi' \wedge \neg\psi)$ (if ψ is unknown in φ'). Observe from the second case that, in the former formula we add a set of unit clauses to φ', and then use the function μ_{CNF} to convert it into a μCNF-state. This new μCNF-state is usually smaller than φ'—since the new unit clauses usually subsume or shorten a number of clauses in φ'. In the latter formula, we add only a small clause, which is usually a clause with at most two literals; thus, there is a great chance that the new clause (or its resolvent with a clause in φ') will subsume some clause(s) in φ' when applying μ_{CNF}. This makes the assumption about the size limit for the formulae stored in the set X reasonable—i.e., their size is not greater than that of φ.

In Proposition 31, we use the term $u = |prop(Rel(\varphi', \psi'))|$. We need to determine an upper bound for this value as follows:

$$maxprops(a, \varphi) = \max_{\psi \in \Psi(a)} (|prop(Rel(\varphi \cup \Psi'(a), \psi'))|)$$

where

$$\psi' = \{\ell \in \psi \mid \ell \text{ is possibly unknown in } \varphi\} \quad \text{and} \quad \Psi'(a) = \{\psi' \mid \psi \in (\Psi(a))\}$$

Intuitively, for each μCNF-state φ' in X (Algorithm 4), $|prop(Rel(\varphi', \psi''))| \leq maxprops(a, \varphi)$, where $\psi'' = \{\ell \in \psi \mid \ell$ is possibly unknown in $\varphi'\}$, and ψ is an e-condition of a. The cost of computing $enb_{\mu CNF}(a, \varphi)$ is shown in the following proposition.

Proposition 32. *Let φ be a μCNF-state and a be an action. Let m be the number of clauses in φ, N be the size of φ, p be the number of combined conditional effects of a, and k be the number of e-conditions of a that are unknown in φ. Let $u = maxprops(a, \varphi)$. If for each intermediate CNF formula φ' generated during the computation of $enb_{\mu CNF}(a, \varphi)$, the size of φ' and the number of clauses in φ' do not exceed the size of φ and m, respectively, then*

$$T(enb_{\mu CNF}(a, \varphi)) = O(2^{k+u}Np + 2^k Nnm^2)$$

5.3.4. Computing $update_{\mu CNF}$

The computation of $update_{\mu CNF}$ is implemented by the procedure updateCNF(φ, η) (Algorithm 11).

Algorithm 11 updateCNF(φ, η) Computing $update_{\mu CNF}(\varphi, \eta)$

1: **Input**: μCNF-state φ, set of literals η
2: **Output**: μCNF-state $update_{\mu CNF}(\varphi, \eta)$
3: **for** each ℓ in η **do**
4: **if** $\{\ell\} \notin \varphi$ **then**
5: **if** $\{\overline{\ell}\} \in \varphi$ **then**
6: Compute $\varphi = r(\varphi \setminus \{\{\overline{\ell}\}\} \cup \{\{\ell\}\})$ $\{update_r(\varphi, \ell) = r(\varphi \setminus \{\{\overline{\ell}\}\} \cup \{\{\ell\}\})\}$
7: **else**
8: Set $\varphi_0 = \{\alpha \mid \alpha \in \varphi \wedge \alpha \cap \{\ell, \overline{\ell}\} = \emptyset\}$ {compute $\varphi_0 = \varphi \setminus (\varphi_\ell \cup \varphi_{\overline{\ell}})$}
9: Set $\varphi^+ = \{\alpha \setminus \{\ell\} \mid \alpha \in \varphi \wedge \ell \in \alpha\}$ {compute $\varphi^+ = \varphi_\ell - \ell$}
10: Set $\varphi^- = \{\alpha \setminus \{\overline{\ell}\} \mid \alpha \in \varphi \wedge \overline{\ell} \in \alpha\}$ {compute $\varphi^- = \varphi_\ell - \ell$}
11: Compute $\varphi = r(\varphi_0 \cup \{\{\ell\}\} \cup (\varphi^+ \times \varphi^-))$
12: $\{update_r(\varphi, \ell) = r(\varphi \setminus (\varphi_\ell \cup \varphi_{\overline{\ell}}) \cup \{\{\ell\}\} \cup (\varphi_\ell - \ell) \times (\varphi_{\overline{\ell}} - \overline{\ell}))\}$
13: **end if**
14: **end if**
15: **end for**
16: **return** CNFstate(φ) {Algorithm 10}

Proposition 33. *For a μCNF-state φ and a consistent set of literals η, the procedure* updateCNF(φ, η) *returns $update_{\mu CNF}(\varphi, \eta)$.*

The running time of Algorithm 11 for computing $update_{\mu CNF}$ in the worst case is shown in the following proposition.

Proposition 34. *Let φ be a μCNF-state and a be an action. Let m be the number of clauses in φ, k be the number of e-conditions of a that are unknown in φ, and n be the number of propositions in the domain. If for each intermediate CNF formula φ' generated during the computation of $enb_{\mu CNF}(a, \varphi)$, the size of φ' and the number of clauses in φ' do not exceed the size of φ and m, respectively, then the total running time for updating every μCNF-state in $enb_{\mu CNF}(a, \varphi)$ is $O(2^k n^2 m^3)$.*

5.3.5. Computing $merge_{\mu CNF}$

For a set of μCNF-states $\Lambda = \{\varphi_1, \ldots, \varphi_n\}$, by definition, we have that

$$merge_{\mu CNF}(\Lambda) = \mu_{CNF}(\otimes[\Lambda]) = \mu_{CNF}(r(\ldots r(\varphi_1 \times \varphi_2) \times \ldots \varphi_{n-1}) \times \varphi_n)$$

Since for every CNF formula φ', $\mu_{CNF}(\varphi') \equiv r(\varphi')$, if we replace $r()$ with $\mu_{CNF}()$ in $merge_{\mu CNF}(\Lambda)$ then we will obtain a μCNF-state equivalent to $merge_{\mu CNF}(\Lambda)$. Specifically,

$$\mu_{CNF}(\mu_{CNF}(\ldots \mu_{CNF}(\varphi_1 \times \varphi_2) \times \ldots \varphi_{n-1}) \times \varphi_n) \equiv merge_{\mu CNF}(\Lambda)$$

This allows us to apply $\mu_{CNF}()$ in place of $r()$ in the implementation of $merge_{\mu CNF}(\Lambda)$, in order to maintain the size of the obtained CNF formulae as small as possible, as shown in Algorithm 12. The result is a μCNF-state equivalent to $merge_{\mu CNF}(\Lambda)$ so the condition 7 is still satisfied.

Algorithm 12 Computing $merge_{\mu CNF}(\Lambda)$

1: **Input**: Set of μCNF-states Λ
2: **Output**: (A μCNF-state equivalent to) $merge_{\mu CNF}(\Lambda)$
3: Let X be a μCNF-state in Λ. Remove X from Λ.
4: **for** each μCNF-state φ in Λ **do**
5: Compute $X = \mu_{CNF}(X \times \varphi)$
6: **end for**
7: **return** X

The next proposition shows how the size of cross-product of CNF-formulae and, hence, the time complexity for this computation can be reduced.

Proposition 35. *Let φ and ψ be two CNF formulae, α be a clause in φ, and β be a clause in ψ. It holds that*

- *If $\alpha = \beta$ then $r(\varphi \times \psi) = r(((\varphi \setminus \alpha) \times (\psi \setminus \beta)) \cup \alpha)$*
- *If $\alpha \subset \beta$ then $r(\varphi \times \psi) = r((\varphi \times (\psi \setminus \beta)) \cup \beta)$*

Observe that if $|\varphi| = n$ (φ has n clauses) and $|\psi| = m$ then the first item says that the size of the cross-product can be reduced by $n + m - 1$ clauses. The second item indicates that it can be reduced by $n - 1$ clauses. Since the μCNF-states that need to be merged are originated from the same μCNF-state, they usually share a great number of common clauses. Hence, the use of Proposition 35 leads to a significant reduction in computation time in most cases.

We are now going to evaluate the running time of Algorithm 12. Let Γ be the set of μCNF-states obtained by updating every μCNF-state in the set $enb_{\mu CNF}(a, \varphi)$. As presented earlier, this set contains at most 2^k μCNF-states, where k is the number of e-conditions of a that are unknown in φ. Since the μCNF-states in Γ are originated from the same μCNF-state φ, they usually share a number of common clauses that help reduce the size of their cross-product (Proposition 35). In addition, the unit clauses added to the formulae by the $update_{\mu CNF}$ function usually reduce the size of the formulae significantly, through the application of the μ_{CNF} function. Thus, the size limit assumption in this computation, i.e., at the end of each iteration of the **for** loop X contains at most m clauses and its size is bounded by N, is reasonable. Therefore, in each iteration, computing the cross-product $X \times \varphi$ requires $O(nm^2)$ time, and applying μ_{CNF} to the result requires (Onm^2N) time (Line 5). Hence, the overall cost for each iteration is $O(nm^2N)$. There are $|\Gamma| - 1$ iterations, and this number is bounded by $2^k - 1$; thus, the total time for computing $merge_{\mu CNF}(\Gamma)$ is $O(2^k nm^2 N)$.

Proposition 36. *Let φ be a μCNF-state and a be an action. Let m be the number of clauses in φ, N be the size of φ, k be the number of e-conditions of a that are unknown in φ, and n be the number of propositions in the domain. Let Γ be the set of μCNF-states obtained by updating every μCNF-state in $enb_{\mu CNF}(a, \varphi)$: $\Gamma = \{update_{\mu CNF}(\varphi', e(a, \varphi')) \mid \varphi' \in enb_{\mu CNF}(a, \varphi)\}$. If for each intermediate CNF formula φ' generated during the computation of $enb_{\mu CNF}(a, \varphi)$ or the cross-product X in Algorithm 12, the size of φ' and the number of clauses in φ' do not exceed the size of φ and m, respectively, then*

$$T(merge_{\mu CNF}(\Gamma)) = O(2^k nm^2 N)$$

Now we are ready to evaluate the cost of computing $\Phi_{\mu CNF}(a, \varphi)$ as shown the following theorem.

Theorem 5. *Let φ be a μCNF-state and a be an action. Let m be the number of clauses in φ, N be the size of φ, p be the number of combined conditional effects of a, k be the number of e-conditions of a that are unknown in φ, and n be the number of propositions in the domain. Let $u = maxprops(a, \varphi)$. If for each intermediate CNF formula φ' generated during the computation of $enb_{\mu CNF}(a, \varphi)$ or the cross-product X in Algorithm 12, the size of φ' and the number of clauses in φ' do not exceed the size of φ and m, respectively, then*

$$T(\Phi_{\mu CNF}(a, \varphi)) = O(2^{k+u}Np + 2^k n^2 m^3)$$

Thus, besides k, the number of e-conditions of a that are unknown in φ, $\Phi_{\mu CNF}(a, \varphi)$ is also exponential in u. This is because of the complexity of satisfaction checking in this representation.

5.4. CNF *Conformant Planner*

Like DNF, CNF is a progression-based conformant planner, that is built on top of DNF. However, this planner employs the μCNF representation and it does not use the one-of combination technique as this technique works only for disjunctive representations.

The heuristic function used in CNF is similar to that used in DNF—except for the the cardinality and square distance heuristics, that are not applicable to μCNF. The heuristic function for CNF, hence, is defined by the pair $\langle h_{goal}(\varphi), h_{lit}(\varphi)\rangle$ in the lexicographical order, where

- $h_{goal}(\varphi)$: the number of subgoals satisfied by the μCNF-state φ.
- $h_{lit}(\varphi)$: the number of known literals in φ.

The second component of this heuristic function, again, prioritizes expansion of nodes that contain less uncertainty and have a smaller size.

6. Prime Implicate Representation

This section investigates another conjunctive representation, which is the set of prime implicate formulae. As we will see next, this representation has several desirable properties: it is a unique representation that eliminates the possibility of multiple nodes in the search tree representing a same belief state, the satisfaction of a clause or a set of literals in a prime implicate formula can be checked easily, and the update and merge functions are simpler than those for the minimal-CNF representation.

Let φ be a satisfiable formula and α be a nontrivial clause. α is said to be a *prime implicate of* φ if α is an implicate of φ and there does not exist another implicate β of φ that subsumes α. If a unit clause is an implicate of φ then it is a prime implicate of φ. The set of prime implicates of φ is said to be the *prime implicate form of* φ, also called PI-form of φ and denoted by $PI(\varphi)$. A prime implicate formula is a prime implicate form of some formula. The following proposition shows that checking whether a set of literals or a clause is satisfied in a prime implicate formula is easy.

Proposition 37. *Let φ be a prime implicate formula, m be the number of clauses in φ, and n be the number of propositions in the domain. Then,*

1. *Checking whether a literal ℓ is satisfied in φ requires $O(m)$ time.*
2. *Checking whether a nontrivial clause α is satisfied in φ requires $O(nm)$ time.*

Each satisfiable formula has a unique PI-form which is equivalent to the formula. This allows us to use prime implicate formulae as unique representations of belief states.

Definition 23. *The* prime implicate representation*, denoted by* PI*, is the set of prime implicate formulae. A* PI-state *is a prime implicate formula.*

One can observe that a PI-state is also a μCNF-state but the converse is not necessarily true.

Proposition 38. *Every PI-state is a μCNF-state.*

The above proposition suggests us to define the operators $conv_{PI}$, $update_{PI}$, and $merge_{PI}$ for the PI representation based on the definition of the respective operators for the minimal-CNF representation.

As we know from the definition of $conv_{\mu CNF}$, $\varphi \wedge \psi \equiv \varphi \cup \{\{\ell\} \mid \ell \in \psi\}$ and $\varphi \wedge \neg\psi \equiv \varphi \cup \{\overline{\psi}\}$. Hence, $conv_{PI}$ is defined as follows.

Definition 24. *Let φ be a PI-state and ψ be a consistent set of literals. Then,*

$$conv_{PI}(\varphi \wedge \psi) = PI(\varphi \cup \{\{\ell\} \mid \ell \in \psi\})$$

$$conv_{PI}(\varphi \wedge \neg\psi) = PI(\varphi \cup \{\overline{\psi}\})$$

It is easy to see that $conv_{PI}$ satisfies Equation (5) as shown in the following proposition.

Proposition 39. *Let φ be a PI-state and ψ be a consistent set of literals. Then,*

1. *$conv_{PI}(\varphi \wedge \psi)$ is a PI-state equivalent to $\varphi \wedge \psi$*
2. *$conv_{PI}(\varphi \wedge \neg\psi)$ is a PI-state equivalent to $\varphi \wedge \neg\psi$*

Similarly to the definition of $update_{\mu CNF}$, in order to define the update of a PI-state φ by a set of literal ψ $update_{PI}(\varphi, \psi)$, first we need to define the update of φ by a literal ℓ $update_{PI}(\varphi, \ell)$. Intuitively, $update_{PI}(\varphi, \ell)$ should be equivalent to $update_r(\varphi, \ell)$. Observe that, from the third case of Definition 20, each clause α in the cross-product $(\varphi_\ell - \ell) \times (\varphi_{\overline{\ell}} - \overline{\ell})$ is either a trivial clause or a resolvent (and thereby an implicate) of φ. If α is non-trivial then either α is a prime implicate in φ or it is subsumed by a prime implicate of φ. Moreover, $\alpha \notin \varphi_\ell \cup \varphi_{\overline{\ell}}$ since every clause in $(\varphi_\ell - \ell) \times (\varphi_{\overline{\ell}} - \overline{\ell})$ (including α) does not contain either ℓ or $\overline{\ell}$. This implies that, for every α in $(\varphi_\ell - \ell) \times (\varphi_{\overline{\ell}} - \overline{\ell})$, α is trivial, α is subsumed by a clause in $\varphi \setminus (\varphi_\ell \cup \varphi_{\overline{\ell}})$, or α already exists in this set. Thus, adding $(\varphi_\ell - \ell) \times (\varphi_{\overline{\ell}} - \overline{\ell})$ to the set in the third case when φ is a PI-state becomes redundant. Definition 20 can be reduced to the definition of $update_{PI}$ as follows.

Definition 25. *Let φ be a PI-state and ℓ be a literal. The update of φ by ℓ, denoted by $update_{PI}(\varphi, \ell)$, is defined as*

$$update_{PI}(\varphi, \ell) = \begin{cases} \varphi & \text{if } \{\ell\} \in \varphi \\ \varphi \setminus (\varphi_\ell \cup \varphi_{\overline{\ell}}) \cup \{\{\ell\}\} & \text{otherwise} \end{cases}$$

The following proposition shows that $update_{PI}$ satisfies Equation (6).

Proposition 40. *Let φ be a PI-state and ℓ be a literal. Then, $update_{PI}(\varphi, \ell)$ is a PI-state and*

$$BS(update_{PI}(\varphi, \ell)) = \{s \setminus \{\overline{\ell}\} \cup \{\ell\} \mid s \in BS(\varphi)\}$$

Like $update_r$, $update_{PI}$ is commutative as shown in the following proposition.

Proposition 41. *Let φ be a PI-state and ℓ_1 and ℓ_2 be two literals such that $\ell_1 \neq \bar{\ell_2}$, then*

$$update_{PI}(update_{PI}(\varphi, \ell_1), \ell_2) = update_{PI}(update_{PI}(\varphi, \ell_2), \ell_1)$$

Proposition 41 along with Lemma 1 allow us to define the function $update_{PI}$ as follows.

Definition 26. *Let φ be a PI-state and η be a consistent set of literals. Then $update_{PI}(\varphi, \eta)$ is the PI-state defined as*

$$update_{PI}(\varphi, \eta) = \begin{cases} \varphi & \text{if } \eta = \emptyset \\ update_{PI}(update_{PI}(\varphi, \eta \setminus \{\ell\}), \ell) & \text{if } \exists \ell \in \eta \end{cases}$$

The following proposition confirms that $update_{PI}$ satisfies Equation (6).

Proposition 42. *Let φ be a PI-state and η be a consistent set of literals. Then*

$$BS(update_{PI}(\varphi, \eta)) = \{s \setminus \overline{\eta} \cup \eta \mid s \in BS(\varphi)\}$$

To define $merge_{PI}$, we use the following proposition.

Proposition 43. *Let φ and φ' be two prime implicate formulae. Then,*

$$PI(\varphi \times \varphi') = \varphi \otimes \varphi'$$

Thus, the prime implicate form of the cross-product of two PI-states can be computed in polynomial time by removing every trivial or subsumed clause from their cross-product. The $merge_{PI}$ operator for PI-states then is defined as follows.

Definition 27. *Let Λ be a set of PI-states. Then, $merge_{PI}(\Lambda)$ is defined as*

$$merge_{PI}(\Lambda) = \otimes[\Lambda]$$

The following proposition, that follows directly the above definition and Proposition 27, shows that $merge_{PI}$ satisfies Equation (7).

Proposition 44. *Let Λ be a set of PI-states. Then, $merge_{PI}(\Lambda)$ is a PI-state and*

$$BS(merge_{PI}(\Lambda)) = \bigcup_{\varphi \in \Lambda} BS(\varphi)$$

We now have a complete definition of Φ_{PI}, a transition function for PI-states, whose correctness has been proved. Next, we will present the computational aspects of this function.

6.1. Computing PI-states

Let us start by discussing how to compute the PI-form of a formula. Let us consider the problem of computing the PI-form of a formula ϕ. We can start by first converting ϕ into an equivalent CNF formula φ, followed by the computation of the PI-form of φ—which is also the PI-form of ϕ. In principle, $PI(\varphi)$ is computed by repeatedly resolving pairs of clauses in φ, adding to φ the resulting resolvents that are not subsumed by φ, and removing from φ clauses subsumed by the newly added clauses—until no new clauses can be added to φ. In order to avoid generating the same resolvents in distinct ways, in 1967 Tison [24] introduced a technique to resolve clauses over the literals in order: for each literal ℓ, we can generate every possible resolvent of pairs of clauses which are resolvable on ℓ and never consider ℓ again even for the new resulting resolvents until a new clause is added to the theory. In 1990, Kean and Tsiknis [14] proposed an incremental algorithm, called IPIA, for updating a prime implicant/implicate formula when adding new implicants/implicates to the formula. Later, de Kleer [8] improved this method by using a novel data structure, called *trie*, that represents the set of clauses in an order of literals to facilitate subsumption checking. In this paper, we will use the IPIA algorithm for computing PI formulae (PI-states).

The procedure $\texttt{IPIA}(\varphi, \alpha)$ (Algorithm 13) computes the PI-form of the conjunction of a PI-state (φ) and a clause (α). The procedure $\texttt{IPIA}(\varphi)$ (Algorithm 14) computes the PI-form of a CNF formula φ. First, it removes every trivial clause from φ and then incrementally computes the PI-form of φ based on the procedure $\texttt{IPIA}(\varphi, \alpha)$.

Algorithm 13 $\texttt{IPIA}(\varphi, \alpha)$ Computing $PI(\varphi \wedge \alpha)$, where φ is a PI-state, α is a clause

Input: A PI-state φ, a clause α
Output: The PI-state $PI(\varphi \cup \{\alpha\})$
if α is trivial in φ or subsumed by a clause in φ **then**
  return φ
end if
Remove every clause from φ that is subsumed by α
Let $X = \{\alpha\}$
for each literal ℓ in α **do**
  Compute $Y = \{\beta|\gamma \mid \beta \in X_\ell, \gamma \in \varphi_{\bar{\ell}}$, and γ is resolvable with $\beta\}$
  Set $\varphi = \{\alpha \in \varphi \mid \alpha$ is not subsumed by $Y\}$
  Set $X = \{\alpha \in X \mid \alpha$ is not subsumed by $Y\}$
  Set $Y = \{\alpha \in Y \mid \alpha$ is not subsumed by $(\varphi \cup X)\}$
  Set $X = X \cup Y$
end for
return $\varphi \cup X$

We have following proposition about the running time of Algorithm 14 and the size of the resulting PI-state.

Proposition 45. *Let φ be a PI-state and α be a nontrivial clause. Let n be the number of propositions in the domain, m be the number of clauses in φ, r be the number of literals in α, and u be the number of literals in α that are unknown in φ. Then,*

1. *$PI(\varphi \cup \{\alpha\})$ can be computed in $O(nm^{2r})$ time and $PI(\varphi \cup \{\alpha\})$ contains $O(m^r)$ clauses.*
2. *$PI(\varphi \cup \{\alpha\})$ can be computed in $O(nm^{2u})$ time and $PI(\varphi \cup \{\alpha\})$ contains $O(m^u)$ clauses.*

According to Proposition 45, updating a PI-state to keep it in PI-form, when adding to it a clause α, requires polynomial time if $|\alpha|$ or the number of literals in $|\alpha|$ that are unknown in the PI-state is bounded by a constant.

Algorithm 14 `IPIA`(φ) Computing PI-form of a CNF formula φ

1: **Input**: A CNF formula φ
2: **Output**: The PI-state $PI(\varphi)$
3: Remove every trivial clause from φ
4: **if** $\varphi = \emptyset$ **then**
5: **return** φ
6: **end if**
7: Let $result = \{\alpha\}$, where α is a clause in φ
8: Remove α from φ
9: **for** each β in φ **do**
10: Compute $result = IPIA(result, \beta)$ {Algorithm 13}
11: **end for**
12: **return** $result$

In conformant planning, the procedure `IPIA`(φ) is needed only for the computation of the PI-form of the initial belief state. During the search for conformant solutions, only the incremental algorithm (Algorithm 13) is needed for the computation of the $conv_{PI}$ operator. Specifically, given a PI-state φ and an e-condition of an action ψ, according to Definition 24, $conv_{PI}(\varphi \wedge \psi)$ is obtained by repeatedly applying the procedure `IPIA`$(\varphi, \{\ell\})$ for each $\ell \in \psi$ (Algorithm 15), while $conv_{PI}(\varphi \wedge \neg\psi)$ is returned directly by `IPIA`$(\varphi, \overline{\psi})$.

Algorithm 15 `convPI`(φ, ψ) Computing $conv_{PI}(\varphi \wedge \psi) = PI(\varphi \cup \{\{\ell\} \mid \ell \in \psi\})$

1: **Input**: PI-state φ, consistent set of literals ψ
2: **Output**: The PI-state $conv_{PI}(\varphi \wedge \psi)$
3: **for** each ℓ in ψ **do**
4: Compute $\varphi = IPIA(\varphi, \{\ell\})$ {Algorithm 13}
5: **end for**
6: **return** φ

Given the correctness of the IPIA algorithm (Algorithm 13), that for a PI-state φ and a clause α the procedure `IPIA`(φ, α) returns $PI(\varphi \cup \{\alpha\})$, it is easy to see that the following proposition holds.

Proposition 46. *For a PI-state φ and a consistent set of literals, the procedure* `convPI`(φ, ψ) *returns* $conv_{PI}(\varphi \wedge \psi)$ *and the procedure* `IPIA`$(\varphi, \overline{\psi})$ *returns* $conv_{PI}(\varphi \wedge \neg\psi)$.

The $update_{PI}$ can be easily implemented by the procedure `updatePI`(φ, η) (Algorithm 16) according to Definitions 25 and 26. Clearly, the following proposition holds.

Proposition 47. *For a PI-state φ and a consistent set of literals η, the procedure* `updatePI`(φ, η) *returns* $update_{PI}(\varphi, \eta)$.

Due to the simplicity of the $merge_{PI}$ operator (Definition **??**), we omit its implementation in the paper. Let us note that, like $merge_{\mu CNF}$, the computation of $merge_{PI}$ also makes use of Proposition 35 to reduce the computational cost.

Algorithm 16 `updatePI`(φ, η) Computing $update_{PI}(\varphi, \eta)$

1: **Input**: PI-state φ, set of literals η
2: **Output**: PI-state $update_{PI}(\varphi, \eta)$
3: **for** each ℓ in η **do**
4: **if** $\{\ell\} \in \varphi$ **then**
5: **do** nothing $\{update_{PI}(\varphi, \ell) = \varphi\}$
6: **else**
7: Compute $\varphi = \varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \cup \{\{\ell\}\}$
8: **end if**
9: **end for**
10: **return** φ

Theorem 6. *Let φ be a PI-state and a be an action. Let m be the number of clauses in φ, p be the number of combined conditional effects of a, k be the number of e-conditions of a that are unknown in φ, and n be the number of propositions in the domain. Let r be the maximum number of literals in an e-condition of a and u be the maximum number of literals in an e-condition of a that are unknown in φ. Let $e = max\{e(a, s) \mid s \in BS(\varphi)\}$. If the number of clauses in each intermediate formula φ' generated during the computation of $\Phi_{PI}(a, \varphi)$ does not exceed the number of clauses m in φ, then*

$$T(\Phi_{PI}(a, \varphi)) = O(2^k(nm^2r + nm^{2u} + nmp + nme + nm^3))$$

As mentioned earlier, u is very small (usually 0 or 1) in most problem so $T(\Phi_{PI}(a, \varphi))$ is exponential only in k, the number of e-conditions of a unknown in φ, in the worst case. In turn, k is also rather small in most problems so $T(\Phi_{PI}(a, \varphi))$ is polynomial in most problems. However, this is true only if the size of the PI-states generated during the computation of $T(\Phi_{PI}(a, \varphi))$ is not greater than that of φ. This is not always true. Moreover, even though $T(\Phi_{PI}(a, \varphi))$ is polynomial, the number of clauses in the PI-state φ (m) can be very large, compared to an equivalent μCNF-state, so $T(\Phi_{PI}(a, \varphi))$ can be very large too. Our experiments show that in many problems, the set of PI-states generated by the search is identical to the set of μCNF-states. In those problems, since Φ_{PI} is faster than $\Phi_{\mu CNF}$, the use of the PI representation appears to be more efficient than the μCNF representation.

6.2. PIP *Conformant Planner*

The PI representation is employed in a conformant planner, called PIP, that is built on top of CNF and uses the same heuristic scheme of CNF. It is worth noting that the implementation of PIP does not need a SAT-solver software system. Instead, PIP implements the incremental IPIA algorithm (Algorithm 13) for computing PI-states during the search, while Algorithm 14 is used to compute the initial PI-state from the initial μCNF-state, as presented in Subsection 6.1.

7. Related Work

For a better understanding of the empirical study in the next section, in this section we will discuss the approaches adopted in the state-of-the-art conformant planners, including KACMBP [7], POND [5], Conformant-FF [12], `T0` [18, 19], `T1` [1], CPA [29, 28], and GC[LAMA] [16]. The performance of these planners, except KACMBP, will be compared to the performance of our planners in the following section.

Note that all of these planners are heuristic search and progression-based. We will discuss the advantages and disadvantages of the method each of these planners uses to deal with incomplete information.

The approach proposed by Cimatti, Roveri, and Bertoli [6, 7] represents belief states as *binary decision diagrams (BDDs)* [4] and uses the model checking techniques for expanding the search space. Later, Bryce, Kambhampati, and Smith employed BDDs to represent literals and actions in the planning graph for computation of heuristics used to search for solutions in their planner POND[5]. The use of BDDs is advantageous for the following reasons. First, a BDD formula is usually more compact than the belief state it represents. Second, checking whether a literal holds in a BDD formula representing a belief state can be done easily. Finally, there are available libraries for creating and manipulating BDDs, that allow the developer to focus only on the development of a heuristic function in the development of a planner. This approach, however, has the main disadvantage that the size of the BDD representation is still very large and sensitive to ordering of the variables. Furthermore, the manipulation of BDDs is very expensive as it requires intermediate formulae of exponential size. This explains why KACMBP and POND do not scale well as shown in several works [12, 19, 25] and in this paper (with POND).

At the other extreme of the use of belief states (that explicitly enumerates all possible states of the world), Hoffmann and Brafman represent belief states indirectly through the action sequences that lead to them from the initial belief state, and use forward search in the belief state space for solutions [12]. In this approach, the resulting planner CFF does not store the knowledge about the state of the world in memory, except the known literals and the corresponding action sequence. To determine whether a proposition holds in the successor belief state after the execution of an action in a belief state, the planner has to reason about a CNF formula that captures the semantic of the entire action sequence leading to the successor belief state from the initial belief state. The advantage of this representation is that it requires very little memory, scaling up pretty well on a number of problems. The trade-off is that it incurs a great amount of repeated computations. Furthermore, checking whether a proposition holds after the execution of even one single action in the presence of incomplete information is co-NP complete. We believe that this is one of the main reasons why CFF has difficulties in finding a solution for even small instances of harder problems, where the structure of the actions is complex or there are unknown propositions in the conditions of the conditional effects as shown in several works [19, 28, 25].

An alternative, indirect, heuristic search and progression-based approach consists of transforming a conformant planning problem into a *classical planning problem*, and then using the off-the-shelf classical planner FF [13] to search in the corresponding state space for solutions. This method has been employed by the planner `T0` [18, 19] and its predecessor `cf2cs(ff)` [17]. This approach demonstrates a great improvement, i.e., the `T0` planner can solve hard problems of large size, compared to the previous approaches. However, the complete translation is exponential in the conformant width of the problem, and the number of literals in the resulting problem can be exponential in the number of literals in the original problem, making the state space extremely large and prohibiting the planners to scale up. To reduce the complexity of the translation, `T0` sacrifices the completeness of the translation in certain problems. This is the reason why `T0` is unable to return a solution in several problems that have solutions (e.g., large instances of Corners-Square, Push-to, and Look-and-Grab as shown in the experimental results later). The experiments also show that `T0` fails during translation for large instances of several domains, e.g., Adder, Rao's key, and Push-to.

Albore, Ramirez, and Geffner recently introduced a new translation-based approach to conformant planning [1]. This approach translates each conformant planning problem P into a new problem P' by simply replacing the initial belief state in P by a subset of it. For each belief state in the original problem P, the approach maintains a set of *tentative literals* that are satisfied in the corresponding belief state in P'. The translation is used for deriving heuristics and tentative literals for each belief state rather than for employing

a classical planner to find solutions. This translation is complete but not always sound. This means that a solution for P is also a solution for P' and any literal satisfied in a belief state of P must belong to the set of tentative literals for that belief state—but the converse is not necessarily true. As discussed in their paper, the condition for the method to be sound is that the conformant width of the problem is bounded. In their resulting planner `T1`, this condition is instantiated as being not greater than 1—which is satisfied in many benchmarks. Therefore, `T1` performs very well in a large number of domains. However, `T1` still encounters difficulties in other domains, especially those that have conformant width greater than 1, where tentative literals need to be verified and `T1` does that in the same manner as CFF does. Like CFF, `T1` does not handle problems with disjunctions in the goal.

The most recent conformant planner, that uses a classical planner to find conformant solutions, is GC[LAMA] [16]. The approach in GC[LAMA] is based on the fact that, given a conformant problem $P = \langle F, O, I, G\rangle$, every solution of P is also a solution for the classical planning problems of the form $P' = \langle F, O, s, G\rangle$, where s is a state in the initial belief state S_I of P. To solve P, GC[LAMA] tries to find a solution for the sub-problem $P = \langle F, O, s_0, G\rangle$, for some initial state $s_0 \in S_I$, using the competitive classical planner LAMA [11], the winner of the Sequential Satisficing Track of the IPC 2008 and IPC 2011. For each solution α of P' found by LAMA, if α is verified to be also a solution for P then GC[LAMA] returns α as a solution for P. Otherwise, for each state $s \in S_I$ and $s \neq s_0$, GC[LAMA] attempts to introduce additional sequences of actions in α to ensure maintenance of preconditions and effects of the actions in α—using smaller classical planning problems to determine the necessary sequence of actions to insert. Observe that, after insertion of a sequence of action β into α, the new sequence may be no longer a solution for P', as the execution of β can destroy some literals that are useful for the achievement of the goal for P'; this may require backtracking and repeating the process with different solutions of P'. One can see that this method is efficient on problems where the action effects are monotonic, i.e., useful actions create useful literals without destroying other useful literals, so that the addition of supplementary sequences of useful actions into a solution for a sub-problem results in another solution for the sub-problem and it brings us closer to a solution for the sub-problem being considered. The method has also some limitations; for example, the method requires testing each subproblem derived from the initial belief state—thus, a large initial belief state can prove challenging. This method is also potentially incomplete unless the underlying classical planner can find all the solutions for any classical planning problem. This explains why GC[LAMA] is efficient only on the problems in the first experimental test suite in the next section, while it falls short of expectations in the other test suites.

The introduction of CPA [23] brought a different perspective to deal with incomplete information in conformant planning. Instead of using the complete transition function in the search, CPA uses an *approximation* technique, first proposed by Son and Baral [21], which approximates a belief state by the intersection of the states in it. The advantage of this approach lies in the low-complexity of the approximation: the successor (approximated) belief state can be computed in polynomial time. The approximation is, however, incomplete, and so are planners which employ this approximation. To address this issue, a complete condition for the approximation has been identified along with corresponding techniques developed in CPA to make the planner complete [22, 29]. These techniques require the system to deal with *sets* of approximated states, which—in the worst case—are the same as the belief states. The advantage of this approach is that the computation of successor belief states is very simple—i.e., can be performed in the same manner as that for belief states—since the approximated formulae are in disjunctive normal form and they contain sufficient information needed for the state computation. The drawback of this approach is that the approximated formula explodes in many problems as observed in the experiments, preventing the planner to scale up or even hardly to start the search. To address this issue, the authors of the recent CPA [28] developed preprocess-

ing techniques which help reduce the size of the initial disjunctive formula in a certain classes of problems. These techniques enabled the planner to perform very well in several benchmarks and to win the conformant planning category in the IPC-08 (`http://ippc-2008.loria.fr/wiki/index.php/Main_Page`).

8. Experimental Evaluation

8.1. Experimental Setup

In this paper, we compare our planners with seven state-of-the-art conformant planners: CPA, CFF, GC[LAMA], POND, `T0`, and `T1`. These planners have been known as the most competitive conformant planning systems. We do not use KACMBP [7] and MBP[6], whose performance is comparable to that of some of the other planners discussed in this paper—due to the lack of translation of the input theory, as these planners do not recognize theories encoded in the PDDL language. Moreover, they employs analogous state representations as other systems being discussed—e.g., BDD as in POND. The comparison with KACMBP and MBP, nonetheless, can be derived indirectly from the experimental results reported in [12, 19, 29].

In this paper, we use the most recent release of CPA [28]. This is an approximation-based planner, like its predecessor [22, 29]. The version of CPA used in this paper outperforms its predecessor [22, 29] on most benchmarks.

The version of GC[LAMA] used in this paper is newer than the one discussed in [16]. This newer version performs differently than its predecessor on several problems.

We use the current public release of the CFF system [12].[4] We apply CFF using the default setting on most domains, except $cornerscube$ and its variants, where the second heuristic option (-h 2) is used—as this setting offers better results on these domains. The CFF planner does not handle disjunctions in the goal (e.g., as required by the Sortnet and Adder problems) or conjunctions in one-of clauses in the description of the initial belief state (e.g., as needed by the UTS-cycle problem).

There are several available versions of POND which, in conjunction with different parameter options, return very different performance results. Nevertheless, no single version/parameter combination appears to dominate the others; in the experiments reported in this paper we select POND version 2.0 with the default execution settings, since this version of POND outperforms its latest version 2.2 on most problems. Yet, we also use the results of POND version 2.2 on several domains, where its predecessor offers poor performance (e.g., UTS-cycle and Corners-Cube and its variants).

We use the most recent version of `T0`, released on May, 11 2009.[5] Let us recall that `T0` was the winner of the Conformant Track of the 2006 International Planning Competition (IPC5).

The `T1` system [1] we use in our experiment has been obtained directly from the developers (emailed by Alexander Albore on February, 14 2012) and it was the most recent version of the system at that time. `T1` is built on top of CFF and, hence, it does not support problems that contain disjunctive goals or conjunctions inside one-of clauses in the description of the initial belief state.

We perform all the experiments using a dedicated Linux Intel Core 2 Dual 9400 2.66GHz workstation with 4GB of memory. In our empirical study, we are interested in comparing the planners according to *performance*—i.e., speed of computation—and *scalability*—ability to solve increasingly larger problem instances. The execution times we report represent the average of two runs, expressed in seconds and showing minimal variance. We use the Linux function `time` to measure the overall (elapsed) execution

[4] `http://www.loria.fr/~hoffmanj/cff.html`

[5] `http://ldc.usb.ve/~hlp/`

time for each experiment; we observed that the execution time reported by a planner sometimes significantly differs from the actual execution time. This reported time can be significantly greater than the computation time for some planners, e.g., T0, T1, CPA, GC[LAMA], and DNF, especially when the search time is small—this is often due to the time spent by the planners in writing to/reading from files. For this reason, we believe the results we report (especially for small instances) are more meaningful for evaluation of the scalability of each planners. Due to the significant variations of execution times of a planner on the same problem instance, the execution times are rounded up to the closest decimal if the total execution time is less than 100 seconds and to the closest integer otherwise.

The results of the experiments are reported in terms of execution time in seconds (columns **time**) and the length of the plan (columns **len**) (Tables 1-5); the execution time represents the overall execution time for the problem. In particular, for DNF, CNF, PIP, CPA, GC[LAMA], and T0, the execution time includes the translation time and the search time for the plan. In these tables, we use the following terminology:

- **OM**: Out-of-memory (if the execution exceeds 4 Gigabytes)
- **TO**: Time-out (if the execution time exceeds two hours)
- **NA**: The benchmark is non-applicable to the planner. This happens to CFF and T1, as these planners do not support problems with a conjunction in an one-of clause or problems with a disjunction in the goal
- **NOP**: The planner does not return a solution for the problem, due to its incompleteness.
- **E**: The planner reports an incorrect solution.
- **AB**: The planner terminates abnormally, e.g., T0 fails to translate the problem into classical planning, CFF generates *too many clauses* to handle, or the length of the action sequence exceeds the maximum length set by CFF.
- **"-"**: The planner cannot solve the problem instance due to the same reason as indicated for a smaller instance.

In our experiments, we use a large collection of conformant benchmarks, collected from the distributions of CFF, KACMBP, POND, and T0 and from the Conformant Track of the International Planning Competitions in 2006 and in 2008[6]. Due to the similarity and simplicity of several conformant planning benchmarks used in the literature, we further diversify the benchmark pool with a set of new problems, obtained from our modifications of several problems in the literature.

For a better evaluation of the different approaches, we divide the set of benchmarks used in the experiments into four distinct test suites. The first two test suites contain conformant problems available in the literature based on their conformant width [19, 1]. The conformant width of a problem is related to the maximum number of unknown literals in the initial belief state relevant to the precondition or e-conditions of an action or the goal of the problem. This measure is critical to the performance of several systems, such as T0 and T1. Specifically, the translation and the size of the resulting classical problem in T0 are exponential in the width of the original problem. For T1, when the conformant width is greater than 1, the planner has to verify whether a tentative literal holds in a belief state in the same manner as CFF does, i.e.,

[6]There was no Conformant Track in the last International Planning Competition in 2011

by reasoning on a CNF theory constructed from the initial belief state, the sequence of actions leading to the current belief state from the initial belief state, and the action theory of the problem. The experiments reveal that this feature also affects the performance of other planners as well. The last two test suites are the sets of new problems created by our modifications of several problems from the literature. Due to the similarity in the performance of DNF, CNF, and PIP on most problems in the first three test suites, we omit the results of CNF and PIP on those test suites. However, we will report the results of CNF and PIP in the last test suite, where the description of the initial belief state of each problem contains a large set of or-clauses, making the size of the initial belief state, or the disjunctive formula representing it, particularly large.

8.2. Problems from Literature with a Constant Conformant Width

This test suite consists of conformant planning problems from the literature with a constant conformant width (usually 1). The performance results of the planners on these problems are shown in Tables 1 and 2.

The first four domains in Table 1 come from the distribution of CFF: Bomb-b-t is the Bomb-in-the-toilet domain, where b indicates the number of bombs and t denotes the number of toilets. Logistics-u-c-p is a variant of the classical Logistics domain, where u denotes the uncertainty about the initial location of each of p packages, i.e., each package can be initially in one of u different locations. Ring-n is the problem of closing and locking windows in a ring of n rooms, given that the room where the agent is initially in and the state of each window are unknown. Safe-n is the problem of opening a safe with n possible combinations.

The following four problems in the table belong to four grid domains included in the `T0` distribution. Cube-Center-n refers to the problem of reaching the center of a cube of n^3 blocks from an unknown block. Square-Center-n is a similar problem, that involves a square of n^2 cells. Corners-Cube and Corners-Square are variants of Cube-Center and Square-Center, respectively. In this variant problems, the initial location is restricted to the corners of the cube or the square.

In Table 2, we report the results from the Coins, Comm, and the UTS domains, drawn from the 2006 International Planning Competition (IPC5), and the Forest and Dispose domains, used in the 2008 International Planning Competition (IPC6). Coins-e-f-p-c is the problem of collecting c coins from different locations on f different floors using e different elevators, where the initial location of each coin is unknown among p locations on a given floor and the initial position (floor) of each elevator is unknown as well. UTS-k-n is the problem of visiting $2n$ nodes from a completely unknown location, where every pair of nodes is connected by two directed edges. UTS-l-n is a similar problem, where only each pair of two adjacent nodes is connected by two directed edges. In UTS-r-n, from each node there are n directed edges to n other random nodes. Dispose-n-m is a grid problem [17], concerned with moving around and picking up m objects from n^2 different locations, where the initial location of each object is unknown, and dropping them in a designated location. Push-to is a variant of the Dispose domain, where objects need to be picked up only at two designated locations to which all objects have to be pushed to. Pushing an object from a cell to an adjacent cell moves it to the adjacent cell if the object is in the current cell.

The performance of DNF on most problems in this test suite is highly competitive with most of the other planners, except for GC[LAMA], that is exceptionally efficient on most problems in this test suite. This is because of the relative monotonicity of the action effects in these problems, as discussed in Section 7. However, GC[LAMA] does not perform well on the Coins and Ring domains, since it needs to consider all the state in the initial belief state of each problem—and the initial belief state of each instance contains an exponential number of states. Several domains where DNF does not scale up well include Logistics, Ring, Comm, and Forest. We believe that the reason for the poor performance of DNF on Logistics and Forest is that the simple heuristic function used in this planner does not perform well in these two domains, because of the large number of nodes generated and expanded by DNF. For example, on Logistics-4-3-10, DNF

Domain	**DNF**		**GC[LAMA]**		**CPA**		**T0**		**T1**		**CFF**		POND	
Instance	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len
Bomb														
50-10	2.5	90	0.7	90	21	90	0.9	90	**0.6**	90	1.2	90	1.5	90
50-50	3.5	50	**1.2**	50	OM		1.4	50	2.8	50	1.3	50	10.3	50
100-50	9.1	150	**1.8**	150	-		3.4	150	6.7	150	15.9	150	140	150
100-100	25.5	100	**3.1**	100	-		8.1	100	20.5	100	**3.1**	100	184	100
200-200	280	200	**12.7**	200	-		68.5	200	OM		**12.7**	200	OM	
Logistics														
4-2-2	1.1	43	0.6	31	1.3	39	**0.0**	19	0.4	25	**0.0**	18	0.3	21
4-2-4	1.8	123	0.7	106	3.4	65	0.1	40	0.4	58	**0.0**	40	1.8	45
4-3-3	3.5	160	0.7	87	7.1	201	0.1	36	0.5	41	**0.0**	37	21.7	37
4-3-10	1442	1055	1.3	504	OM		**1.2**	150	8.5	216	7.5	150	OM	
4-10-10	TO		**2.4**	247	-		3.8	125	122	167	3.9	121	-	
Ring														
5	1	19	1.6	18	1.2	14	0.5	17	**0.4**	18	25.4	45	3.2	20
6	2.3	23	3.8	22	2.4	17	**0.5**	20	**0.5**	27	367	71	26.3	31
7	8.2	27	13.1	26	7.8	20	0.6	30	0.7	32	4251	105	257	34
8	32.2	31	50	28	30.2	23	0.6	39	0.9	38	TO		2138	39
30	OM		OM		OM		**9.17**	121	382	133	-		TO	
Safe														
10	0.6	10	0.5	10	0.6	10	0.5	10	0.4	10	**0.0**	10	0.1	10
30	0.7	30	0.6	30	2.7	30	0.7	30	**0.5**	30	1.2	30	TO	
50	1	50	**0.7**	50	19.8	50	1.0	50	1.0	50	27.1	50	-	
100	3.2	100	**1.3**	100	387	100	1.8	100	12.3	100	934	100	-	
Square-Center														
16	0.8	191	0.6	62	0.8	118	0.9	44	**0.4**	46	121	136	2118	45
24	1.9	351	0.7	94	0.9	68	1.0	69	**0.6**	70	5943	300	-	
56	39.2	1358	**1.6**	222	2.2	168	6.5	165	2.7	166	-		-	
96	295	2093	**3.6**	382	8.7	288	37.8	285	15.6	288	-		-	
100	353	2854	**3.4**	398	9.8	302	OM		17.7	298	-		-	
120	584	2813	**5.7**	478	17.6	388	-		34.0	358	-		-	
Corners-Square														
16	0.7	91	**0.6**	46	0.7	60	0.7	102	1.05	102	13.9	136	1885	32
28	0.8	156	**0.6**	82	0.9	83	1.1	264	4.6	264	AB		TO	
40	1.7	277	**0.6**	118	1	117	2.4	498	19.3	498	-		-	
100	9.6	1145	**2**	298	4.3	315	77.6	2748	1099	2748	TO		-	
140	25.2	2162	**3.4**	418	9.9	431	257	3907	5306	4233	-		-	
142	21.8	1500	**5.2**	424	10.4	457	NOP		6245	5532	-		-	
148	22.2	1473	**5.9**	442	12	479	-		TO		-		-	
150	21.7	1207	**3.9**	448	12.9	538	-		-		-		-	
Cube-Center														
13	1.3	198	**0.6**	75	1.3	115	0.8	54	0.7	54	4069	209	99.6	54
15	1.5	243	**0.6**	90	2.3	296	0.9	63	0.8	63	TO		1273	63
19	3.6	332	**0.6**	120	2.6	259	1.1	81	**0.5**	81	-		TO	
43	76.8	1156	**1.3**	300	59.5	490	6.0	189	2.3	189	-		-	
87	543	1939	**4.7**	630	1472	1095	85.1	387	24.5	387	-		-	
91	1644	2502	**4.9**	660	1655	1359	OM		28.6	405	-		-	
119	1617	4160	**7.9**	870	TO		-		76.1	532	-		-	
Corners-Cube														
15	0.8	117	**0.5**	63	0.9	103	1.2	147	3.0	159	435	279	2680	182
20	1.2	217	**0.5**	87	1.0	159	2.8	258	8.9	248	2492	332	OM	
24	1.2	193	**0.5**	105	1.1	115	6.3	358	21.4	358	TO		-	
52	5.0	647	**0.7**	231	3.6	301	372	1506	651	1554	-		-	
54	8.5	655	**0.7**	240	3.7	298	OM		725	1591	-		-	
87	13.2	1249	**1.8**	387	12.8	613	-		6632	3999	-		-	
99	64.5	1142	**2.5**	441	22.5	669	-		TO		-		-	
199	136	4741	**10.2**	891	319	2343	-		-		-		-	
299	374	8833	**23.2**	1341	OM		-		-		-		-	

Table 1: Execution times and plan lengths found by the planners for problems from literature with a constant conformant width

Domain	**DNF**		**GC[LAMA]**		**CPA**		**T0**		**T1**		**CFF**		POND	
Instance	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len
Coins														
10 (2-2-4-4)	0.7	27	1.9	62	0.9	67	0.7	26	0.6	34	**0.1**	38	0.5	46
15(2-2-8-6)	1.0	67	2449	150	6.1	362	0.8	79	0.8	78	2.8	89	10.5	124
20(2-3-8-6)	1.3	99	3430	146	17.5	586	0.8	107	1.9	197	17.5	143	97.3	153
21(5-10-10-10)	OM		TO		OM		OM		192	904	TO		OM	
Comm														
10	1.3	80	0.6	77	1.9	65	0.7	75	1.1	77	**0.1**	65	0.7	65
15	3.2	125	0.7	107	4.6	95	0.8	110	1.6	114	**0.2**	95	11.7	95
20	117	296	**1**	247	169	239	1.2	278	43.9	295	5.0	239	-	
25	1270	501	**1.5**	410	1762	389	2.3	453	340	478	39.9	389	-	
Dispose														
4-4	1.5	187	**0.6**	514	7.0	421	1.2	120	1.7	177	1.4	90	178	126
4-5	1.6	180	**0.7**	642	10.3	554	1.5	145	2.5	236	2.6	107	OM	
7-3	14.9	393	**2**	1642	284	1749	35.6	486	38.6	491	2713	328	-	
7-7	57	952	**4.8**	3826	1580	4997	OM		202	933	-		-	
7-10	82.3	1305	**7.5**	5464	2850	7433	-		491	1285	-		-	
10-1	150	218	**2.6**	1444	155	213	**7.5**	716	80	443	6410	449	-	
10-3	209	680	**9.6**	4324	4633	5071	OM		746	1103	-		-	
10-5	261	1286	**17.4**	7204	OM		-		2310	1601	-		-	
10-10	1101	2530	**49.7**	14404	-		-		-		-		-	
Forest														
2	2.3	55	0.7	22	251	39	0.7	16	0.6	12	**0.1**	18	1.2	13
3	566	16343	**1.1**	296	TO		1.2	45	4.4	223	TO		TO	
4	459	8207	**1.2**	212	-		1.5	78	14.0	74	-		OM	
5	TO		**2.1**	834	-		3.6	129	26.8	119	-		-	
Push-to														
4_3	1.4	166	**0.7**	248	113	143	1.8	118	1.1	63	1.3	48	6682	120
5_3	2.6	523	**0.9**	367	2456	248	150	251	**2.2**	114	513	105	TO	
5_4	4.9	314	**1**	480	TO		NOP		3.2	139	TO		-	
5_5	9.2	472	**1.2**	591	-		OM		2.93	108	TO		-	
8_1	27.6	163	**1.4**	463	30.1	184	82.8	464	253	538	-		-	
8_2	36.4	638	**2.6**	904	4210	2903	OM		47.2	336	-		-	
8_3	51.4	1060	**4**	1305	OM		-		54.0	300	-		-	
8_10	1575	5948	**24.2**	4088	-		-		282	430	-		-	
10_1	159	270	**3.3**	734	178	333	3518	1159	2289	908	-		-	
10_2	205	1722	**7.6**	1446	OM		TO		1176	805	-		-	
10_5	1233	6146	**31.5**	3410	-		-		2162	772	-		-	
12_5	3137	7744	**95**	5031	-		-		TO		-		-	
UTS-k														
10	1.9	66	**1**	91	16.5	80	1.5	59	1.4	72	14.6	58	14.2	68
20	15.9	136	**2.6**	151	1445	197	12.3	119	31.8	143	1458.9	118	OM	
30	75.1	206	**7.1**	269	-		58.4	179	OM		-		-	
40	233	276	**26.7**	415	-		215	239	-		-		-	
50	565	346	**43.9**	533	-		563	299	-		-		-	
55	855	381	**47.9**	453	-		OM		-		-		-	
UTS-l														
10	1.9	109	**0.7**	244	36.6	125	1.7	97	1.0	59	1.2	59	52.7	88
20	15.7	209	**1.2**	894	TO		39.0	214	2.6	156	106	119	TO	
30	74.2	309	**2.6**	1944	-		OM		7.6	178	1731	179	OM	
40	242	409	**6**	3394	-		-		26.4	239	TO		-	
50	584	509	**10.8**	5244	-		-		70.5	299	-		-	
UTS-r														
9	1.5	66	**0.9**	103	20.9	80	1.3	58	24.2	67	2.7	62	9.8	64
10	1.7	73	**1.2**	115	33.3	90	1.7	65	OM		7.8	66	19.7	68
20	14.5	139	**2**	187	1054	174	25.0	138	16.8	145	397	131	OM	
30	60.2	214	**7**	355	TO		OM		80.0	239	TO		-	
40	184	284	**18**	535	-		-		OM		-		-	
50	443	352	**34.2**	549	-		-		-		-		-	

Table 2: Results on more problems from literature with a constant conformant width

generates $9,049,609$ nodes and expands $445,326$ nodes, while T1 generates $112,100$ nodes and expands $1,092$ nodes. T0 and CFF generate and expand even less nodes for this problem instance. For the Ring problem, the initial μDNF-state is exponential in the size of the problem—$n3^n$ partial states in the initial μDNF-state for the instance Ring-n. With the Comm domain, DNF (and CPA) spend most of the execution time on translating and simplifying the input theory; e.g., DNF spends only 2.625 seconds on the search for the solution of Comm-25, but after spending more than $1,267$ seconds on the translation phase. In terms of overall performance on this test suite, CPA, T0, and T1 are comparable. These planners scale better than CFF and POND, but they are not as good as DNF.

Observe that the quality of the solutions found by DNF, in terms of length of the plans, is comparable with the solutions found by other planners on most problems, except for the Logistics and Forest domains; in these two domains, DNF finds solutions that are much longer than those found by the other planners. There are several domains where the solutions found by GC[LAMA] are much longer than those found by other planners, e.g., Dispose, Push-to, and UTS-l. While the reason for the long solutions found by DNF is due to the simplistic heuristics used, the reason for GC[LAMA] is due to the approach of this planner, that inserts action sequences into the solution for a sub-problem, as discussed earlier.

8.3. Harder Problems from Literature with Conformant Width Increasing on the Problem's Size

This test suite contains problems where the conformant width increases with the problem's size. The experimental results for these problems are summarized in Table 3. The two Adder domains used in IPC5 and IPC6 encode circuits for a given logical formula—the IPC6 version includes more propositions and more complex goals. The Adder problems are difficult due to the complex action effects and the large number of actions, most of them frequently executable, making the search space extremely large. These problems are difficult for all the planners. CFF and T1 cannot handle these problems due to disjunctions in the goal of the problems. We will discuss the reason a disjunctive goal makes a problem more difficult for GC[LAMA] later. Only DNF, CPA, and POND can solve the smallest instances of these two domains.

The Blocks domain, used in the both IPC5 and IPC6 competitions, is the problem of stacking blocks in a certain order, where the initial positions are unknown. Rao's key and UTS-cycle were used in IPC6. The problem UTS-cycle-n is to follow two labels for visiting n nodes from an unknown node, where each edge is assigned either (but not both) of the two labels, and each node is connected to other two nodes by two directed edges that are uncertain among four given directed edges. Sortnet-n [3], used in IPC5, is the problem of sorting n bits in a network. The last two domains in this table are variants of a family of grid problems [17]. Look-and-Grab-n-r-m is the problem of picking up objects from sufficiently close locations and, after each pickup, deposit the objects being held into a designated location before performing any other pickup. In this problem, the initial location of each of m objects is unknown among n^2 locations, and the second parameter r indicates the radius of the influential extent of the actions. For example, when a pickup action is executed, all the objects in the 8 surrounding locations will be picked up if $r = 1$; the number of such locations increases to 15 if $r = 2$. The 1-Dispose domain is a variant of Dispose where the e-condition for the pickup actions requires the robot's hand to be empty. A solution for 1-Dispose must scan the grid, perform pickups in every cells and deposit into the trash can. These problems are hard not only for T0 and T1, due to their conformant width that increases with the size of the problem instance, but they are also hard for all the other planners, as confirmed by the experimental results in Table 3.

It is easy to see that DNF outperforms all of the other planners on most domains of this test suite, with the exception of $Sortnet$, where POND is the fastest planner in all the instances, and UTS-cycle, where GC[LAMA] scales up better with the problem's size. DNF is the only planner that is able to solve all of the instances, including the largest instances of each domain.

Domain	**DNF**		**GC[LAMA]**		**CPA**		**T0**		**T1**		**CFF**		POND	
Instance	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len
Adder (IPC5)														
1	**6.3**	4	AB		6.6	8	AB		NA		NA		555	3
Adder (IPC6)														
1	**7.6**	3	AB		10.5	3	AB		NA		NA		839	3
Blocks														
1	0.5	7	0.6	8	0.7	7	1.1	5	0.8	6	**0.0**	6	**0.0**	4
2	0.6	29	0.7	35	0.8	41	1.1	23	1.1	21	TO		**0.1**	30
3	**1.6**	146	1.8	158	OM		49.4	80	TO		-		3.2	78
4	OM		OM		-		AB		-		-		TO	
Rao's key														
2	0.5	26	0.7	41	0.5	11	0.8	16	3.94	28	**0.1**	34	E	
3	**0.8**	125	0.9	472	2.3	78	1.1	53	TO		11.9	102	-	
4	TO		OM		TO		AB		TO		TO		-	
Sortnet														
8	1.5	36	0.9	36	0.9	26	32.2	36	NA		NA		**0.1**	26
9	1.0	45	2.6	45	1.7	31	276	45	-		-		**0.1**	31
10	1.0	55	3.4	55	3.1	39	OM		-		-		**0.1**	38
13	2.3	91	31.5	90	40.3	57	-		-		-		**0.1**	55
15	8.1	120	166	119	245	65	-		-		-		**0.2**	65
UTS-cycle														
5	**0.6**	10	0.7	14	1.2	12	3.3	10	NA		NA		17.5	10
6	**0.7**	17	1	22	1.2	22	10.4	19	-		-		3208	15
7	**1.3**	32	2.8	29	2.0	37	58.2	26	-		-		TO	
8	**3.2**	41	6.4	47	4.7	47	OM		-		-		-	
9	**7.5**	58	20.2	70	13.5	77	-		-		-		-	
10	29.2	89	**26.6**	93	33.7	77	-		-		-		-	
12	250	134	**234**	152	6798	127	-		-		-		-	
13	604	156	**421**	211	OM		-		-		-		-	
14	2241	225	**825**	285	-		-		-		-		-	
1-Dispose														
2_1	0.5	14	OM		0.5	12	0.7	16	0.8	14	6.7	113	**0.1**	14
2_3	**0.5**	14	-		**0.5**	12	1.6	14	0.9	16	TO		0.6	14
2_5	**0.9**	14	-		**0.9**	12	OM		1.0	16	-		48	14
2_7	10.4	14	-		13.8	12	-		**1.0**	16	-		2583	10
2_8	42.0	14	-		44.8	12	-		**1.0**		-		OM	
2_9	188	14	-		OM		-		**1.1**	16	-		-	
2_10	1835	14	-		-		-		**1.1**	16	-		-	
5_1	**1.4**	98	-		1.6	98	1209	304	1.9	167	-		-	
5_2	**4.2**	122	-		11.1	102	210	126	TO		-		-	
5_3	**81.1**	122	-		263	102	OM		-		-		-	
5_4	**1732**	536	-		OM		-		-		-		-	
7_2	**34.7**	286	-		138	240	-		TO		-		-	
7_3	**1351**	286	-		OM		-		-		-		-	
10_2	**457**	536	-		-		-		-		-		-	
12_1	**693**	654	-		735	658	-		3056	1463	-		-	
12_2	**1942**	502	-		-		-		-		-		-	
Look&Grab														
4-1-1	0.8	16	1.32	52	1.1	22	1.1	14	0.7	18	**0.4**	37	670	26
4-2-2	1.3	4	**1**	18	1.4	4	32.2	4	1.0	4	AB		-	
4-1-3	**1.8**	18	5.1	46	3.9	32	OM		TO		TO		-	
4-2-3	9.1	4	5.2	18	4.2	4	137	4	**1.1**	4	-		-	
4-3-3	9.7	4	11.1	4	4.6	4	OM		**1.0**	4	-		-	
8-1-1	27.4	99	**2.1**	793	28.5	94	305	140	8.3	118	-		-	
8-1-2	47.3	144	**22.4**	356	86.1	125	NOP		TO		-		-	
8-1-3	**1018**	141	1133.28	426	OM		-		-		-		-	
8-2-3	**363**	73	647.27	250	-		-		-		-		-	
8-3-3	**167**	32	393.15	98	-		-		-		-		-	
9-2-3	**1178**	69	2240.36	364	-		-		-		-		-	
9-3-3	**327**	53	1366.27	190	-		-		-		-		-	

Table 3: Results on hard problems from the literature with conformant width increasing on the problem's size.

For this test suite, GC[LAMA] does not perform as well as in the previously discussed benchmarks. This planner is not able to solve any instance of 1-Dispose and the two Adder problems. The only domain that GC[LAMA] outperforms DNF and other planners is UTS-cycle. It is worth nothing that the UTS-cycle problem is harder than its predecessors, due to the uncertainty about the edges connecting the nodes. Yet, like its predecessors and other problems in the first test suite, this problem still satisfies the monotonic property discussed previously for GC[LAMA]. More specifically, following more edges in order to reach a desired node can only increase the number of nodes visited but cannot make any visited node become unvisited. This is, however, not the case for most of other problems in this test suite. Let us consider, for example, the 1-Dispose domain. The hardness of 1-Dispose lies in the $pickup$ action, that contains a number of conditional effects linear in the number of objects in the problem, and their e-conditions are all unknown. Moreover, some e-conditions are contradictory (the hand is empty and an object is being held) and the action can causes opposite effects depending on the satisfaction of the e-conditions. For example, the execution of $pickup$ may result in an object being held and no longer at the location, if the hand is empty and the object is at the location, but may also drop the object at the location to empty the hand if the object is held by the hand. We suspect that the contradictory e-conditions of the $pickup$ action result in contradictory goals for the supplementary classical planning problems generated by GC[LAMA], making it unable to solve even the smallest instance of this domain.

8.4. New Hard Problems, Extensions of Traditional Problems

In this test suite, we introduce a new set of problems, including New-Ring, New-UTS-k, New-UTS-cycle, New-Push, and New-Dispose obtained respectively as extensions of the Ring, UTS-k, UTS-cycle, Push, and Dispose problems; the extensions incorporate new features, that describe the real-life applications better but also make the problems harder. We will observe that these new problems worsen the performance of most existing planners, but they do not seem to affect the performance of our approach in DNF. This is confirmed by the experimental results in Table 4.

In the New-Ring problem, a lock can be damaged; the status of each window is unknown (either "open" or "closed") and the goal is to have every window closed, and if the lock is not damaged then the window should be locked. The goal contains a set of or-clauses, of the form $or(locked(i), lock\_damaged(i))$, where the predicates $locked(i)$ and $lock\_damaged(i)$ indicate that the window i is locked and the lock of the window i is damaged, respectively.

In New-UTS-k, each node needs to be visited exactly once, except for the root node that can be encountered several times. We introduce a new action, $return$, to allow the agent to return to the root node from a node that is connected to it. A plan for this new problem is also a plan for the original problem, where the $return$ action is replaced with the corresponding $travel$ action, but with the added constraint preventing repeated visits of non-root nodes. In the New-UTS-cycle domain, half of edges that have been followed will be dropped in order to reduce the repetition of following the same edges.

In the New-Push domain, for each cell i we add a new predicate $cleared(i)$, to indicate that the cell i is clear as result of executing the action $push(i, j)$; the adjacent cell j of i becomes not clear if there is any object in i. We add to the precondition of each action $move(i, j)$ the literal $\neg cleared(j)$, to forbid moving to cell j if it is clear. Any solution for the new problem is also a solution for the corresponding original Push-to problem, without the redundant moves to the cleared cells; a solution of the original problem may not be a solution for the new problem, as it may contain such redundant actions.

In the new version of Dispose, the goal is relaxed by accepting $n - 1$ objects disposed instead of all n objects disposed. This is reasonable as in real life we cannot always expect all the goals or criteria to be satisfied and sometimes we need to give up one of them. Thus, the goal of the new problem is described

Domain	**DNF**		**GC[LAMA]**		**CPA**		**T0**		**T1**		**CFF**		POND	
Instance	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len	Time	Len
New-Ring														
2	0.5	6	0.5	10	0.6	6	0.7	7	NA		NA		**0**	5
3	**0.6**	10	**0.6**	18	0.7	10	1.5	10	-		-		OM	
4	**0.6**	14	1.7	277	1	14	OM		-		-		-	
5	**0.6**	18	4.7	46	1.9	20	-		-		-		-	
6	**0.8**	22	24.4	48	7.6	24	-		-		-		-	
7	**1.1**	31	146	111	39	31	-		-		-		-	
8	**2.9**	30	TO		OM		-		-		-		-	
9	**9.6**	37	-		-		-		-		-		-	
10	**35**	38	-		-		-		-		-		-	
11	**125**	47	-		-		-		-		-		-	
12	**458**	56	-		-		-		-		-		-	
New-UTS-k														
2	0.5	11	OM		E		0.6	11	**0.4**	11	AB		OM	
5	0.7	29	-		-		3.6	29	**0.4**	29	-		-	
7	1.1	41	-		-		211	41	**0.5**	41	-		-	
8	1.4	47	-		-		OM		TO		-		-	
10	2.3	59	-		-		-		**0.7**	59	-		-	
20	22.6	119	-		-		-		**5.6**	119	-		-	
30	93.6	179	-		-		-		**26.6**	179	-		-	
40	286	239	-		-		-		**88.6**	239	-		-	
45	**447**	269	-		-		-		TO		-		-	
50	**672**	299	-		-		-		-		-		-	
New-UTS-cycle														
3	0.6	3	**0.5**	5	0.7	3	1	3	NA		NA		OM	
4	**0.6**	7	OM		0.7	8	1.5	7	-		-		-	
5	**0.6**	13	-		0.8	13	3.6	11	-		-		-	
6	**0.8**	18	-		1.1	19	14.4	15	-		-		-	
7	**1.2**	22	-		1.7	22	OM		-		-		-	
8	**2.5**	27	-		3.6	30	-		-		-		-	
9	**6.9**	36	-		9	38	-		-		-		-	
10	**22.4**	48	-		**22.4**	41	-		-		-		-	
11	69.8	62	-		**69.3**	56	-		-		-		-	
12	**204**	71	-		OM		-		-		-		-	
13	**608**	90	-		-		-		-		-		-	
14	**1573**	96	-		-		-		-		-		-	
New-Push														
4_2	**1.1**	37	OM		73.5	37	2.8	32	1.5	46	1.8	38	OM	
4_3	3	37	-		682	37	4.9	32	**2.1**	46	12.7	38	-	
4_4	37.1	37	-		OM		8	32	**2.8**	46	265	38	-	
4_5	OM		-		-		11.8	32	**3.9**	46	6133	38	-	
6_2	**7.2**	73	-		-		190	96	TO		359	94	-	
6_3	**83.8**	73	-		-		419	96	-		TO		-	
8_1	30	129	-		TO		OM		**16**	207	-		-	
8_2	**50.4**	129	-		OM		-		TO		-		-	
9_1	**74.4**	177	-		TO		-		76.6	258	-		-	
9_2	**120**	177	-		OM		-		TO		-		-	
New-Dispose														
4-4	1.4	148	342	500	10.2	311	1.5	109	NA		NA		452	114
4-5	1.5	189	TO		9.8	295	2.4	122	-		-		OM	
7-5	28.6	683	-		715	2645	877	504	-		-		-	
7-7	53.7	904	-		TO		OM		-		-		-	
10-5	277	1265	-		-		-		-		-		-	
10-10	1114	2476	-		-		-		-		-		-	

Table 4: Results on new hard problems, our extensions of several problems from the literature.

by a set (conjunction) of or-clauses of the form $(or(disposed(o_i), (disposed(o_j))$ for every pair of objects (o_i, o_j).

Table 4 highlights the superior performance of DNF in all these domains, compared to all the other planners; the other planners can solve none or only a few small instances of each domain in this test suite. Observe that the performance of DNF on New-UTS-k, New-UTS-cycle, and New-Dispose is comparable with that of the original domains, while most of other planners perform significantly worse on these new domains. It is worth noting that DNF scales up better on New-Ring than it does on Ring, due to the fact that the initial μDNF-state in the new domain contains much less partial states than the initial μDNF-state of the original domain.

The new constraints incorporated in the New-UTS-k, New-UTS-cycle, and New-Push domains are obstacles to GC[LAMA], as they prohibit multiple executions of the same actions, while the approach of GC[LAMA] tends to repeat the execution of the same actions when extending the solution of one sub-problem. In the New-Dispose and New-Ring domains, GC[LAMA] performs significantly worse than in the corresponding original domains, due to the disjunctions in the goal of the new problems. We hypothesize that the presence of disjunctive goals leads to a large number of solutions for each sub-problem explored by GC[LAMA], and many of such solutions are far from a solution to the main planning problem. As a result, GC[LAMA] spends a significant amount of time trying to (unnecessarily) repair such solutions. As mentioned earlier, CFF and `T1` do not handle problems with disjunctive goals or conjunctions in one-of clauses. CPA returns no solutions for any instance of New-UTS-k. We suspect that this is because of the incompleteness of the goal-splitting technique employed by this planner.

8.5. Problems with High Uncertainty in The Initial State

In order to test the effectiveness of different representations in conformant planning, we introduce another set of domains obtained by modifying several domains from the previously described test suites. Unlike the extensions in the previous test suite, this modification is simply obtained by replacing the one-of clauses in the original problems with a set of or-clauses. By doing this, the uncertainty in the initial state, i.e., the number of possible states in the initial belief state, increases dramatically. Interestingly, while this change makes the domains fall beyond the capabilities of DNF and several other planners, CNF and PIP perform exceptionally well on this class of problems, thanks to their use of conjunctive representations. The modification is applied to the Coins, Dispose, 1-Dispose, Push, and New-Push domains, resulting in a set of new problems with "or-" prefix in their name, as shown in Table 5.

Let us consider Dispose, 1-Dispose, Push, and New-Push first. In these problems, each instance is given by two numbers: n denotes the size of the grid of $n \times n$ cells (locations) and m is the number of objects given in the problem. For example, in the instance Dispose-2-3, there are $2 \times 2 = 4$ cells in the grid and 3 objects given in this problem instance. In these problems, predicates of the form $at(p_{i,j})$ indicate that the robot is at the cell (i, j) of the grid (location $p_{i,j}$) and predicates of the form $obj\_at(o_k, p_{i,j})$ indicate that the object o_k is at the location $p_{i,j}$, for $i, j \leq n$ and $k \leq m$. Initially, the location of each object is unknown among the $n \times n$ locations. Hence, in the description of the initial world, there is a set of m one-of clauses of the form one-of$(obj_at(o_i, p_{1,1}), \ldots, obj_at(o_i, p_{n,n}))$, for $i = 1, \ldots, m$.

In the new problems, each o_i represents an object type, e.g., pens, books, etc. The predicate $obj\_at(o_i, p_{k,j})$ now means "there exists an object of type o_i at location $p_{k,j}$." Likewise, the predicate $holding(o_i)$ means "holding some object(s) of type o_i." The action $pickup(o_i, p_{k,j})$ in the Dispose problem allows the robot to pick up all the objects of type o_i at location $p_{k,j}$ and hold them. Similarly, in 1-Dispose, Push, and New-Push, the action $pickup(p_{k,j})$ allows the robot to pick up every object at location $p_{k,j}$ if certain conditions

are satisfied.[7] Let us assume, initially, that all we know about the objects is that for each object type o_i there exists at least an instance of o_i somewhere among the n^2 given locations. Thus, the set of one-of clauses one-of$(obj_at(o_i, p_{1,1}), \ldots, obj_at(o_i, p_{n,n}))$ in the description of the initial world should now be replaced by the set of or-clauses or$(obj_at(o_i, p_{1,1}), \ldots, obj_at(o_i, p_{n,n}))$. Observe that, with regard to each object type o_i, there are 2^{n^2} different possible situations about the initial world in the new domains (an object of type o_i may or may not exist in any of the n^2 locations), while this number is n^2 in the original problems. Thus, while the number of possible states in the initial belief state in the original problems is $\Omega((n^2)^m)$, the size of the initial belief state in the new problems is $\Omega((2^{n^2})^m)$, i.e., exponential with an exponential base.

The modification in the Coins problem is similar, where a coin now is interpreted as a coin type, e.g., dim, nickel, quarter, etc. The action $collect(C, F, P)$ allows the robot to collect all the coins of type C at location P on floor F. The goal is to have some coin(s) of each type. In an instance with c coins (fourth dimension) and p locations on each floor (third dimension), the size of the initial belief state in the original problem is $\Omega(p^c)$, while in this new domain such size is $\Omega((2^p)^c)$, i.e., exponential with an exponential base too.

The results in Table 5 demonstrate the superior performance of CNF and PIP for this set of domains. Both planners scale up very well on all dimensions, and no other planner has comparable performance on any of these domains—with the only exception of `T1` on or-Coins. Observe that, while CNF and PIP offer superior performance for these new domains, DNF performs much worse compared to itself on the corresponding unmodified domains. The reason is that the size of the initial μDNF-state increases drastically like the size of the initial belief state in the new domains. Moreover, the one-of combination technique used in DNF does not help in these problems. In contrast, the number of clauses in the initial μCNF-state or in the initial PI-state about the location of objects is only m and even smaller than that in the original problems, which is $m \times n^2(n^2 + 1)/2$. PIP is faster than CNF for the or-Coins, or-Dispose, and or-Push domains. Let us note that, in each problem instance of these domains, the set of generated (resp. explored) PI-states is identical to the corresponding set of μCNF-states, as observed from the experiments. On the other hand, CNF scales better than PIP in the or-1-Dispose and or-New-Dispose domains. This is because, in these domains, the size of the PI-states is significantly larger than the size of the μCNF-states. For example, for the problem instance or-1-Dispose-2-5, the average numbers of clauses in a μCNF-state and in a PI-state are 19.317 and 373.907, respectively. This explains why CNF is much faster than PIP on this problem.

CPA performs poorly in this test suite for the same reason as for DNF, since they both rely on a disjunctive representation. However, DNF is still significantly better. GC[LAMA] is not able to solve any domain instance in this test suite. In the original domains, the one-of combination technique helps in reducing the number of states in the initial belief state to a number that GC[LAMA] can manage. In these new domains, the number of states in the initial belief state is extremely large—beyond the capabilities of GC[LAMA].

Let us observe that there is no noticeable difference between the performance of `T0`, `T1`, and POND on these new domains and their performance on the original domains. CFF performs better on these new domains in comparison with its performance on the original ones. This is because the approach in CFF relies on reasoning about the CNF formulae partly constructed from the initial information. The size of these formulae in the new domains should be smaller than that in the original ones.

[7] In 1-Dispose, the e-condition includes $handempty$, i.e., the robot hand is empty. On the other hand, in the Push and New-Push domains, $pickup(p_{k,j})$ is executable only if $p_{k,j}$ is a "pickup" location

Domain	**CNF**	**PIP**	**DNF**	**GC[LAMA]**	**CPA**	**T0**	**T1**	**CFF**	POND
Instance	Time (Len)	Time (Len)	Time (Len)	Time (Len)	Time (Len)	Time (Len)	Time (Len)	Time (Len)	Time (Len)
or-Coins									
10 (2-2-4-4)	0.6 (35)	0.6 (39)	2.4 (44)	OM	5.5 (59)	1.6 (26)	0.4 (34)	**0.2** (38)	0.5 (46)
15 (2-2-8-6)	0.9 (83)	0.9 (91)	OM	-	OM	0.7 (79)	**0.5** (78)	2.6 (88)	10.2 (124)
20 (2-3-8-6)	1.2 (115)	1 (134)	-	-	-	**0.8** (107)	2 (197)	16.6 (142)	83.6 (121)
22 (5-10-10-12)	415 (2755)	151 (2732)	-	-	-	OM	**92.2** (888)	TO	OM
24 (5-10-10-14)	369 (2771)	**148** (2795)	-	-	-	-	443 (1148)	-	-
25 (5-10-10-15)	218 (1864)	**92** (2061)	-	-	-	-	1616 (2003)	-	-
29 (5-10-10-19)	635 (5399)	**226** (5854)	-	-	-	-	648 (1430)	-	-
or-Dispose									
3-3	0.6 (58)	0.6 (58)	4.6 (66)	OM	28.5 (111)	0.6 (67)	0.4 (60)	**0.1** (44)	0.8 (44)
3-4	0.7 (74)	0.7 (74)	87 (97)	-	OM	0.7 (60)	0.5 (100)	**0.2** (54)	2.1 (60)
3-5	0.8 (102)	0.7 (106)	OM	-	-	0.7 (7)	0.7 (110)	**0.3** (64)	28.5 (74)
5-2	1.5 (123)	1.4 (121)	12.3 (25)	-	-	**1** (179)	3.1 (189)	7.2 (124)	OM
5-3	1.6 (170)	**1.5** (166)	TO	-	-	1.9 (201)	5.8 (229)	27.4 (162)	-
5-5	2.3 (265)	**2** (277)	-	-	-	12.7 (239)	20.2 (402)	208 (238)	-
10-1	145 (215)	139(221)	146 (229)	-	-	**9.3** (716)	163 (512)	6234 (449)	-
10-3	168 (549)	**157** (531)	OM	-	-	OM	1756 (1115)	-	-
10-5	175 (866)	**171** (850)	-	-	-	OM	4202 (1535)	-	-
10-10	251 (1820)	**229** (1824)	-	-	-	-	-	-	-
15-15	5947 (4477)	**5659** (4493)	-	-	-	-	-	-	-
or-1-Dispose									
2-2	0.6 (12)	0.6 (16)	0.6 (10)	OM	0.7 (10)	1.7 (12)	TO	2310 (169)	**0.4** (14)
2-3	**0.6** (12)	0.7 (24)	0.7 (14)	-	2.3 (10)	OM	-	TO	0.7 (14)
2-5	**0.9** (12)	51.4 (16)	26.8 (14)	-	OM	-	-	-	54.6 (14)
5-1	3.2 (410)	2.1 (478)	**1.8** (100)	-	-	-	3.7 (161)	-	OM
5-2	**4.4** (434)	5.8 (460)	35.5 (88)	-	-	-	TO	-	-
5-5	**21.7** (606)	707 (1762)	OM	-	-	-	-	-	-
5-7	**144** (756)	TO	-	-	-	-	-	-	-
5-10	**3140** (676)	-	-	-	-	-	-	-	-
10-1	278 (8380)	**190** (9716)	345 (336)	-	-	-	1870 (1151)	-	-
10-2	**346** (7852)	640 (10.3k)	OM	-	-	-	OM	-	-
10-5	**768** (12.2k)	4272 (14.8k)	-	-	-	-	-	-	-
10-7	**3119** (15.7k)	TO	-	-	-	-	-	-	-
or-Push									
4-3	**0.9** (105)	**0.9** (105)	94.4 (147)	OM	OM	12.7 (189)	**0.9** (74)	1 (48)	OM
4-4	**1** (132)	**1** (132)	TO	-	-	17.2 (210)	**1** (107)	1.2 (49)	-
7-1	3.8 (138)	**3.7** (156)	9.9 (130)	-	-	210 (252)	TO	1505 (185)	-
7-2	4.4 (146)	**4.2** (146)	355 (251)	-	-	OM	-	2968 (186)	-
7-20	295 (2699)	**217** (2905)	OM	-	-	-	-	TO	-
7-40	3578 (5569)	**2541** (5851)	-	-	-	-	-	-	-
12-1	200 (986)	**190** (1262)	686 (466)	-	-	AB	-	-	-
12-10	1086 (7130)	**694** (8066)	OM	-	-	-	-	-	-
15-1	1318 (2123)	**1244** (2493)	4428 (926)	-	-	-	-	-	-
15-14	6562 (15.4k)	**4099** (18k)	OM	-	-	-	-	-	-
15-15	TO	**5210** (22.9k)	-	-	-	-	-	-	-
or-New-Push									
4-1	0.7 (37)	0.7 (41)	50 (40)	OM	75.6 (37)	1.2 (32)	**0.5** (48)	0.6 (38)	OM
4-3	**0.8** (37)	0.9 (41)	TO	-	OM	5.4 (32)	2.6 (46)	2 (38)	-
4-5	**1.1** (37)	1.3 (41)	-	-	-	13.5 (32)	5.3 (46)	4.3 (38)	-
7-1	5 (111)	**4.6** (115)	**4.6** (99)	-	-	547 (131)	6.7 (167)	1075 (139)	-
7-3	**19** (111)	29.3 (127)	TO	-	TO	4851 (131)	TO	4211 (139)	-
7-5	**98** (123)	550 (127)	-	-	-	OM	-	TO	-
9-1	21.8(185)	**21.7** (185)	-	-	-	-	38.4 (279)	-	-
9-2	**24.4** (197)	**24.4** (185)	-	-	-	-	TO	-	-
9-5	**35.7** (177)	39.5 (177)	-	-	-	-	-	-	-
9-8	**90** (193)	2178 (189)	-	-	-	-	-	-	-
9-10	**282** (189)	TO	-	-	-	-	-	-	-

Table 5: Results on new problems of high uncertainty about the initial state, obtained by modifying those from previous test suites.

8.6. On Selection of Belief State Representation

The empirical study in the previous section shows that DNF, CNF, and PIP offer very different performances across various problems even though they employ the same (between CNF and PIP) or very similar (DNF v.s. CNF and PIP) heuristic schemes, that are mostly based on the number of satisfied subgoal and the number of known literals in the belief state. This suggests that there is no winner-take-all representation. Furthermore, for a conformant planner to exploit the strengths of different representations, it should be able to adapt its representation to each given concrete problem. This raises the question of when should a particular representation be selected. This subsection discusses some criteria for evaluating a representation and then provides some discussion on this issue with respect to the three concrete representations, μDNF, μCNF, and PI.

8.6.1. General Criteria for Evaluating a Belief State Representation $\mathcal{R}$

Since an implementation of a progression based planner relies on a transition function, the computational cost of $\Phi_{\mathcal{R}}$ will be an important criterion for evaluating a representation $\mathcal{R}$. As shown in the general methodology (Section 3), the computational cost of $\Phi_{\mathcal{R}}$ depends on the cost of computing $enb_{\mathcal{R}}$, $update_{\mathcal{R}}$, and $merge_{\mathcal{R}}$. The computation cost of $enb_{\mathcal{R}}$, in turn, depends on the cost of computing $conv_{\mathcal{R}}$ and the cost of satisfaction checking in $\mathcal{R}$ (Algorithms 3 and 4). Thus, to evaluate a representation $\mathcal{R}$, we need to consider the costs of computing all these components: $conv_{\mathcal{R}}$, satisfaction checking in $\mathcal{R}$, $update_{\mathcal{R}}$, and $merge_{\mathcal{R}}$. Theorems 4, 5, and 6 clearly indicate that these costs depend on the representation. For example, the satisfaction checking in the μDNF and prime implicate representations is polynomial in the size of the formula while it is NP-hard in general for the μCNF representation; the computation of $update_{\mu DNF}$ and $merge_{\mu DNF}$ is simpler than that of $update_{PI}$ and $merge_{PI}$, which is less expensive than the computation of $update_{\mu CNF}$ and $merge_{\mu CNF}$.

Although the computational cost of $\Phi_{\mathcal{R}}$ is an important criteria for evaluating a representation $\mathcal{R}$, it is not the only factor that influences the performance of a planner using $\mathcal{R}$. This is because a complete planner, in searching for a solution, will need to maintain the queue of unexplored $\mathcal{R}$-states and also the set of explored $\mathcal{R}$-states to avoid repeating the exploration of the same node. As such, the compactness of an $\mathcal{R}$ representation is another crucial criterium for evaluating $\mathcal{R}$. Since the number of possible belief states is exponential to the size of the problem, the compactness of the $\mathcal{R}$-states is essential to the performance and the scalability of the planner (e.g., for avoiding out-of-memory error).

8.6.2. Minimal-DNF, Minimal-CNF, or Prime Implicates?

The previous discussion provides two general criteria for the selection of an arbitrary representation. We will now discuss a possible way to decide between the three concrete representations developed in this paper, μDNF, μCNF, and prime implicate.

Observe that μCNF and PI representations rely on a specific type of conjunctive normal form while μDNF uses disjunctive normal form. Furthermore, the experimental results show that the planners CNF and PIP behave similarly in most instances but significantly different from that of the planner DNF. Thus, given a problem or a class of problems, we can start by determining whether or not μDNF is suitable. If μDNF is not suitable, then a decision between the μCNF and prime implicate representations needs to be made. We will next discuss how this could be done.

From the analysis of the computational cost of the transition functions, we have that $\Phi_{\mu DNF}$ depends on the size of the μDNF-states (Theorem 4), the number of unknown e-conditions of each action (k), and the number of propositions in the domain. Moreover, the general methodology (Section 3) shows that the transition function $\Phi_{\mathcal{R}}$ for any representation $\mathcal{R}$ is exponential in k. Thus, if the size of the μDNF-states

is not very large then $\Phi_{\mu DNF}$ can be computed efficiently—compared with not only $\Phi_{\mu CNF}$ and Φ_{PI} but also to the transition function for any representation—and thus, the minimal-DNF representation can be a good choice for problems whose μDNF-states has a small size. In fact, most of the earlier benchmarks exhibit this property and the experimental results in the first three test suites confirm that the size of the initial μDNF-states is a reasonable indicator for the usefulness of μDNF.

When the size of the μDNF-states is extremely large, μDNF is likely unsuitable. This is exemplified by the experimental results on the problems in the last test suite. In this test suite, the size of the μDNF-states is extremely large and the size of the μCNF-states and PI-states is exponentially smaller (polynomial). This leads to a poor performance of DNF comparing to the exceptional performance of CNF and PIP. The different performance of CNF and PIP can also be attributed to the size of the μCNF-states and the PI-states generated by the planners (Theorem 5 and 6) and can be observed in the last test suite as well. In domains where the size of the PI-states and the μCNF-states are comparable (e.g., or-Coins, or-Dispose, and or-Push), PIP is faster than CNF. On the other hand, when the size of the PI-states is significantly larger than that of the μCNF-states (e.g., or-1-Dispose, and or-New-Push), CNF performs better than PIP.

In summary, we can conclude that the minimal-DNF representation can be a good choice on problems where the size of the μDNF-states is not very large. Otherwise (i.e., for problems where the size of the μDNF-states is much larger than equivalent CNF formulae), if the size of the PI-states is comparable to the size of the μCNF-states then the prime implicate representation is a good choice. If the size of the PI-states is significantly larger the size of the μCNF-states then the minimal-CNF representation should be used.

9. Summary

In this paper, we presented a general methodology for the development of a complete transition function $\Phi_{\mathcal{R}}$ for an arbitrary belief state representation $\mathcal{R}$. We proved that the function is optimal with respect to the number of intermediate formulae necessarily generated in the $\Phi_{\mathcal{R}}$ function, a factor that impacts the complexity of this function. We showed how to use the methodology in the development of heuristic search and progression-based conformant planners. We investigated three different representations, minimal-DNF, minimal-CNF, and prime implicates, and illustrated the general methodology by instantiating the definition of $\Phi_{\mathcal{R}}$ in these three representations. We studied the computational complexity of each function. We implemented the minimal-DNF, minimal-CNF, and prime implicates representations in the conformant planners DNF, CNF, and PIP, respectively. We employed a simple heuristic scheme in these planners based on the number of satisfied subgoals and the number of known literals in the belief state. DNF also implemented the cardinality of the μDNF-state in its combined heuristic function.

We compared our approach with the state-of-the-art approaches, using a large and diverse set of benchmarks; these include most commonly used conformant planning benchmarks available in the literature. For a better understanding of the effectiveness of belief state representations, we diversified the test set, introducing a new set of conformant planning problems obtained as extensions and modifications of several conformant planning domains studied in the literature. These new problems are harder than their predecessors, due to the additional features incorporated in them. The experiments showed that our approach is very competitive compared with the others, especially on the hard problems. Moreover, while the performance of the existing conformant planners deteriorates in presence of these additional features, none of these features seems to affect the performance of our approach.

To further study the effectiveness of belief state representation, we introduced another new set of problems, obtained by replacing the set of one-of clauses in several existing conformant planning problem with the set of corresponding or-clauses. These problems are beyond the capabilities of several competitive planners, including DNF, CPA, and GC[LAMA], due to the huge number of states in the initial belief state. Yet, CNF and PIP offer a superior performance on these problems. The reason lies in the use of a conjunctive representation, that is very compact for these problems compared with other representations, e.g., those in DNF and CPA. The experiments also showed that PIP is faster than CNF on problems where the size of PI-states is comparable to that of μCNF-states, while CNF scales better on those where the size of PI-states is much larger. This confirms the importance of the study of alternative belief state representations and, hence, the significance of the general methodology.

We then discussed the major criteria for the evaluation of a representation, that are the computational complexity of the transition function and its components for the representation and its compactness. We provided some preliminary results on how to select a suitable representation among the three proposed representations with regard to problems.

In this paper, we focused on the problem of representation of belief states—we did not address the study of effective heuristics in the development of heuristic-search conformant planners. The investigation of more effective heuristics for conformant planners developed using the methodology proposed in this paper will be our focus in the immediate future.

Acknowledgments

We would like to thank the developers of the planner CPA, who kindly provided us with the source code that was used as the starting point for the implementation of our planners. A special thank goes to Alexandre Albore, the author of `T1`, for helping us understand the planner `T1` as well as perform some experiments.

The research has been partially supported by NSF grant IIS-0812267.

References

[1] Alexandre Albore, Miquel Ramírez, and Hector Geffner. Effective heuristics and belief tracking for planning with incomplete information. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*. AAAI Press, 2011.

[2] C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122:241–267, 2000.

[3] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. pages 52–61. AAAI Press, 2000.

[4] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[5] D. Bryce, S. Kambhampati, and D. Smith. Planning Graph Heuristics for Belief Space Search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[6] A. Cimatti and M. Roveri. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[7] A. Cimatti, M. Roveri, and P. Bertoli. Conformant Planning via Symbolic Model Checking and Heuristic Search. *Artificial Intelligence Journal*, 159:127–206, 2004.

[8] J. de Kleer. An improved incremental algorithm for computing prime implicates. In *AAAI*, pages 780–785, 1992.

[9] R. Goldman and M. Boddy. Expressive planning and explicit knowledge. In *Proceedings of 3rd International Conference on AI Planning Systems*, pages 110–117, 1996.

[10] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. of ECP-99, Lecture Notes in AI Vol 1809*. Springer, 1999.

[11] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[12] Jörg Hoffmann and Ronen I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7):507–541, 2006.

[13] Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302, 2001.

[14] A. Kean and G. Tsiknis. An incremental method for generating prime implicants/implicates. pages 185–206, 1990.

[15] H-K. Nguyen, D-V. Tran, T.C. Son, and E. Pontelli. On improving conformant planners by analyzing domain-structures. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

[16] H-K. Nguyen, D-V. Tran, T.C. Son, and E. Pontelli. On computing conformant plans using classical planners: A generate-and-complete approach. In *Proceedings of the 22st International Conference on Automated Planning and Scheduling (ICAPS-2012), Atibaia, Sao Paulo Brazil, June 25-29*. AAAI Press, 2012.

[17] H. Palacios and H. Geffner. Compiling Uncertainty Away: Solving Conformant Planning Problems Using a Classical Planner (Sometimes). In *Proceedings of the the Twenty-First National Conference on Artificial Intelligence*, pages 900–905, 2006.

[18] H. Palacios and H. Geffner. From Conformant into Classical Planning: Efficient Translations that may be Complete Too. In *Proceedings of the 17th International Conference on Planning and Scheduling*, pages 264–271, 2007.

[19] H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

[20] D.E. Smith and D.S. Weld. Conformant Graphplan. In *AAAI*, pages 889–896, 1998.

[21] Tran Cao Son and Chitta Baral. Formalizing sensing actions - a transition function based approach. *Artificial Intelligence*, 125(1-2):19–91, January 2001.

[22] Tran Cao Son and Phan Huy Tu. On the Completeness of Approximation Based Reasoning and Planning in Action Theories with Incomplete Information. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, pages 481–491, 2006.

[23] Tran Cao Son, Phan Huy Tu, Michael Gelfond, and Ricardo Morales. Conformant Planning for Domains with Constraints — A New Approach. In *Proceedings of the the Twentieth National Conference on Artificial Intelligence*, pages 1211–1216, 2005.

[24] Pierre Tison. Generalized consensus theory and application to the minimization of boolean functions. In *IEEE Transactions on Computers*, pages 446–456, 1967.

[25] S.T. To, E. Pontelli, and T.C. Son. A Conformant Planner with Explicit Disjunctive Representation of Belief States. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, pages 305–312. AAAI, 2009.

[26] S.T. To, T.C. Son, and E. Pontelli. A New Approach to Conformant Planning using CNF. In *Proceedings of the 20th International Conference on Planning and Scheduling (ICAPS)*, pages 169–176, 2010.

[27] S.T. To, T.C. Son, and E. Pontelli. On the Use of Prime Implicates in Conformant Planning. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[28] D-V. Tran, H-K. Nguyen, E. Pontelli, and T.C. Son. Improving performance of conformant planners: Static analysis of declarative planning domain specifications. In Andy Gill and Terrance Swift, editors, *Practical Aspects of Declarative Languages, 11th International Symposium, PADL 2009, Savannah, GA, USA, January 19-20, 2009. Proceedings*, volume 5418 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 2009.

[29] P.H. Tu, T.C. Son, M. Gelfond, and R. Morales. Approximation of action theories and its application to conformant planning. *Artificial Intelligence Journal*, 175(1):79–119, January 2011.

Appendix. Proofs

The proof of a proposition or theorem in the body of the paper will be specified by the proposition or theorem. The proof of a new proposition or lemma in the appendix will be provided next to the proposition or lemma.

Proof of Proposition 1.

1. The two sets are equal iff every element (state) s in the first set belongs to the second set and vice versa. Indeed,

$$\begin{array}{lll} s \in BS(\varphi_1 \vee \ldots \vee \varphi_n) & \text{iff} & s \models \varphi_1 \vee \ldots \vee \varphi_n \\ & \text{iff} & (s \models \varphi_1) \vee \ldots \vee (s \models \varphi_n) \\ & \text{iff} & s \in BS(\varphi_1) \vee \ldots \vee s \in BS(\varphi_n) \\ & \text{iff} & s \in BS(\varphi_1) \cup \ldots \cup BS(\varphi_n) \end{array}$$

2. This is a corollary of the previous result using the following fact: for every formula φ',

$$\varphi' \models \varphi \quad \text{iff} \quad \forall s \in BS(\varphi').\, s \models \varphi$$

Proof of Proposition 2. Observe that

$$\begin{aligned} BS(\varphi_1 \wedge \ldots \wedge \varphi_n) &= \{s \mid s \models (\varphi_1 \wedge \ldots \wedge \varphi_n)\} \\ &= \{s \mid s \models \varphi_1 \wedge \ldots \wedge s \models \varphi_n)\} \\ &= \{s \mid s \models \varphi_1\} \cap \ldots \cap \{s \mid s \models \varphi_n\} \\ &= BS(\varphi_1) \cap \ldots \cap BS(\varphi_n) \end{aligned}$$

Proof of Proposition 3. Since s is a model of φ, φ is evaluated to be true by the literals in s. That is, for each literal ℓ in φ, if $\ell \in s$ then ℓ is replaced with $true$ in φ. Otherwise ($\ell \notin s \wedge \bar{\ell} \in s$), ℓ is replaced with $false$ in φ. Then the formula becomes $true$ following the rules in the truth table. Observe that the literals in δ are sufficient for the evaluation of the truth value of φ and the other literals in s, which is $s \setminus \delta$, do not affect the evaluation of φ because they or their negations are not part of φ. Hence, the literals in $s \setminus \delta$ do not affect the truth value of φ. This means that if we replace any literals in $s \setminus \delta$ with their negations, the new state is also a model of φ. This implies that every state that is a superset of δ is also a model of φ. In other words, every model of δ is a model of φ. By definition, this implies that $\delta \models \varphi$.

Proof of Proposition 4. Proof by contradiction. Assume that $\varphi \models \psi$. Then $\varphi \wedge \neg\psi$ is unsatisfiable (P4).

On the other hand, since φ is satisfiable, there exists a consistent set of literals δ such that $prop(\delta) = prop(\varphi)$ and $\delta \models \varphi$ (by Proposition 3). Since ψ is non-tautological, $\neg\psi$ is satisfiable. By Proposition 3, there exists a consistent set of literals δ' such that $prop(\delta') = prop(\neg\psi) = prop(\psi)$ and $\delta' \models \neg\psi'$. Because φ and ψ are independent, $prop(\delta') \cap prop(\delta) = \emptyset$. This implies that $\delta \cup \delta'$ is a consistent set of literals. Thus, $\delta \cup \delta' \models \varphi$ and $\delta \cup \delta' \models \neg\psi$. This means that $\delta \cup \delta' \models \varphi \wedge \neg\psi$. Let s be a state which is a superset of $\delta \cup \delta'$, clearly s is a model of $\varphi \wedge \neg\psi$. This means that $\varphi \wedge \neg\psi$ is satisfiable. This contradicts (P4). As a consequence, $\varphi \not\models \psi$.

Proof of Lemma 1. Let $\alpha = (t_1, \ldots, t_m)$ be an arbitrary sequence of m elements in $\mathcal{D}'$. By α_k, $0 < k \leq m$, we denote the sequence

$$\alpha_k = (t_1, \ldots, t_k)$$

which is the prefix of the first k elements of α. By $f(x, \alpha_k)$ we denote

$$f(x, \alpha_k) = f(\ldots(f(x, t_1), \ldots), t_k)$$

Thus, we need to prove

$$f(x, Y_n) = f(x, Z_n)$$

The proof is by induction on the length n of Y (and Z).

- Base case $n = 1$: it is trivial
- Base case $n = 2$: it is given

- Inductive step: Suppose that the theorem is true for every $k < n$, for some $n \geq 3$, we will prove that it is also true for $k = n$. We consider the following two cases

  – Case $y_n = z_n$: then $Z_{n-1} = (z_1, \ldots, z_{n-1})$ is a permutation of $Y_{n-1} = (y_1, \ldots, y_{n-1})$. Hence,

$$\begin{aligned} f(x, Y_n) &= f(f(x, Y_{n-1}), y_n) \\ &= f(f(x, Z_{n-1}), y_n) && \text{(inductive hypothesis)} \\ &= f(f(x, Z_{n-1}), z_n) && (z_n = y_n) \\ &= f(x, Z_n) && \text{(proof)} \end{aligned}$$

  – Case $y_n \neq z_n$: let Y' be the sequence obtained from Y by swapping the position of z_n with y_{n-1} in Y:

$$Y' = (y'_1, \ldots, y'_{n-2}, z_n, y_n)$$

  Similarly, let

$$Z' = (z'_1, \ldots, z'_{n-2}, y_n, z_n)$$

  be the sequence obtained from Z by swapping the two elements z_{n-1} and y_n in Z. Observe that Y'_{n-1} is a permutation of Y_{n-1}, Z'_{n-1} is a permutation of Z_{n-1}, and Z'_{n-2} is a permutation of Y'_{n-2}. We have

$$\begin{aligned} f(x, Y_n) &= f(f(x, Y_{n-1}), y_n) \\ &= f(f(x, Y'_{n-1}), y_n) && \text{(inductive hypothesis on } n-1) \\ &= f(f(f(x, Y'_{n-2}), z_n), y_n) && (y'_{n-1} = z_n) \\ &= f(f(f(x, Z'_{n-2}), z_n), y_n) && \text{(inductive hypothesis on } n-2) \\ &= f(f(f(x, Z'_{n-2}), y_n), z_n) && \text{(given)} \\ &= f(f(x, Z'_{n-1}), z_n) && (z'_{n-1} = y_n) \\ &= f(f(x, Z_{n-1}), z_n) && \text{(inductive hypothesis on } n-1) \\ &= f(x, Z_n) && \text{(proof)} \end{aligned}$$

Proof of Proposition 5. Let s be an arbitrary state in $BS(\varphi)$. It suffices to prove that $e(a, s) = e(a, \varphi)$.

(1) Let ℓ be an arbitrary literal in $e(a, \varphi)$, we will show that $\ell \in e(a, s)$: by Definition 4, there exists an effect $a : \psi \to \ell$ such that $\varphi \models \psi$. This implies $s \models \psi$ since s is a state in $BS(\varphi)$, i.e., a model of φ. By the definition of $e(a, s)$, we also have $\ell \in e(a, s)$

(2) Now let ℓ' be an arbitrary literal in $e(a, s)$, we prove that $\ell' \in e(a, \varphi)$: Assume that $\ell' \notin e(a, \varphi)$. Since $\ell' \in e(a, s)$, there exists an effect $a : \psi' \to \ell'$ such that $s \models \psi'$. On the other hand, $\ell' \notin e(a, \varphi)$ so $\varphi \not\models \psi'$. This means that $\varphi \models \neg\psi'$ (because φ is enabling for action a, either $\varphi \models \psi'$ or $\varphi \models \neg\psi'$ holds). Consequently, $s \models \neg\psi'$ (a contradiction).

The proof has been obtained by (1) and (2).

Proof of Proposition 6.

1. Since $\varphi \not\models \neg\psi$ and φ is satisfiable, we have that $\varphi \wedge \psi$ is satisfiable. Let s be an arbitrary state in $BS(\varphi)$ that satisfies ψ, clearly $s \models \varphi \wedge \psi$ or $s \in BS(\varphi \wedge \psi)$. On the other hand, every state in $BS(\varphi \wedge \psi)$ satisfies both φ and ψ, i.e., it belongs to $BS(\varphi)$ and satisfies ψ. Thus, $BS(\varphi \wedge \psi)$ is the set of states in $BS(\varphi)$ that satisfy ψ.

2. Similarly, we have $\varphi \wedge \neg\psi$ is satisfiable and $BS(\varphi \wedge \neg\psi)$ is the set of states in $BS(\varphi)$ that satisfy $\neg\psi$. In addition, since a formula is always known in a state, it is easy to see that every state s in $BS(\varphi) \setminus BS(\varphi \wedge \psi)$ satisfies $\neg\psi$, i.e., $s \in BS(\varphi \wedge \neg\psi)$. In the other words, $BS(\varphi \wedge \neg\psi) = BS(\varphi) \setminus BS(\varphi \wedge \psi)$.
3. The second result implies that $BS(\varphi \wedge \psi) \cup BS(\varphi \wedge \neg\psi) = BS(\varphi)$. Hence, $(\varphi \wedge \psi) \vee (\varphi \wedge \neg\psi) \equiv \varphi$.

Proof of Proposition 7. Let $\Psi = \langle \psi_1, \ldots, \psi_n \rangle$ be an enumeration of $\Psi(a)$. By Definition 5, to prove that $\varphi \oplus_{\mathcal{R}} \Psi$ is an enabling form of φ for a, we prove that $\varphi \oplus_{\mathcal{R}} \Psi$ is a set of $\mathcal{R}$-states enabling for a, i.e., every ψ_i in $\Psi(a)$ is known in every $\mathcal{R}$-state in $\varphi \oplus_{\mathcal{R}} \Psi$, and $\bigcup_{\gamma \in \varphi \oplus_{\mathcal{R}} \Psi} BS(\gamma) = BS(\varphi)$. We also need to prove that every $\mathcal{R}$-state in $\varphi \oplus_{\mathcal{R}} \Psi$ is satisfiable. The proof is by induction on $|\Psi(a)|$ as follows.

- Base case $|\Psi(a)| = 0$: The proof is trivial as $\varphi \oplus_{\mathcal{R}} \Psi = \{\varphi\}$.
- Base case $|\Psi(a)| = 1$: Let $\Psi(a) = \{\psi\}$. Then $\Psi = \langle \psi \rangle$ and $\varphi \oplus_{\mathcal{R}} \Psi = \varphi \oplus_{\mathcal{R}} \psi$. By Definition 6 and Proposition 6, clearly this is an enabling form of φ for a and every $\mathcal{R}$-state in $\varphi \oplus_{\mathcal{R}} \Psi$ is satisfiable.
- Inductive step: Suppose that the proposition holds for $0 \leq |\Psi(a)| \leq n$, for some $n \geq 1$. We will prove that it also holds for $|\Psi(a)| = n + 1$. Consider an enumeration $\Psi = \langle \psi_1, \ldots, \psi_n, \psi_{n+1} \rangle$ of $\Psi(a)$. We have

$$\varphi \oplus_{\mathcal{R}} \Psi = \varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n, \psi_{n+1} \rangle = \bigcup_{\gamma \in (\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle)} (\gamma \oplus_{\mathcal{R}} \psi_{n+1}) \qquad \text{(by Definition 7)}$$

  By inductive hypothesis, $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle$ is a set of satisfiable $\mathcal{R}$-states and for every $\mathcal{R}$-state γ in $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle$ either $\gamma \models \psi_i$ or $\gamma \models \neg\psi_i$ holds for $i = 1, \ldots, n$. Let γ be an arbitrary $\mathcal{R}$-state in $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle$. By Definition 6 and Proposition 6, we have that $\gamma \oplus_{\mathcal{R}} \psi_{n+1}$ is a set of satisfiable $\mathcal{R}$-states satisfying γ. This implies that, for every η in $\gamma \oplus_{\mathcal{R}} \psi_{n+1}$ either $\eta \models \psi_i$ or $\eta \models \neg\psi_i$ holds for $i = 1, \ldots, n$. Furthermore, for every η in $\gamma \oplus_{\mathcal{R}} \psi_{n+1}$ either $\eta \models \psi_{n+1}$ or $\eta \models \neg\psi_{n+1}$ holds. This implies that every ψ_i in $\Psi(a)$ is known in every $\mathcal{R}$-state in $\gamma \oplus_{\mathcal{R}} \psi_{n+1}$. In other words, $\gamma \oplus_{\mathcal{R}} \psi_{n+1}$ is a set of satisfiable $\mathcal{R}$-states enabling for a. Since γ is an arbitrary $\mathcal{R}$-state in $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle$, by definition we have that $\varphi \oplus_{\mathcal{R}} \Psi$ is a set of satisfiable $\mathcal{R}$-states enabling for a.

  To prove $\bigcup_{\varphi' \in \varphi \oplus_{\mathcal{R}} \Psi} BS(\varphi') = BS(\varphi)$, we need to prove that for every $\mathcal{R}$-state γ and every consistent set of literals ψ, $\bigcup_{\gamma' \in \gamma \oplus_{\mathcal{R}} \psi} BS(\gamma') = BS(\gamma)$. We consider the following two cases.

  - $\gamma \models \psi$ or $\gamma \models \neg\psi$. By definition, $\gamma \oplus_{\mathcal{R}} \psi = \{\gamma\}$. Hence, $\bigcup_{\gamma' \in \gamma \oplus_{\mathcal{R}} \psi} BS(\gamma') = BS(\gamma)$
  - ψ is unknown in γ. Then, by definition we have $\gamma \oplus_{\mathcal{R}} \psi = \{conv_{\mathcal{R}}(\gamma \wedge \psi), conv_{\mathcal{R}}(\gamma \wedge \neg\psi)\}$. Hence,

$$\begin{aligned} \bigcup_{\gamma' \in \gamma \oplus_{\mathcal{R}} \psi} BS(\gamma') &= BS(conv_{\mathcal{R}}(\gamma \wedge \psi)) \cup BS(conv_{\mathcal{R}}(\gamma \wedge \neg\psi)) \\ &= BS(\gamma \wedge \psi) \cup BS(\gamma \wedge \neg\psi) \\ &= BS(\gamma) \qquad \text{(by Proposition 6)} \end{aligned}$$

  In both cases, we have $\bigcup_{\gamma' \in \gamma \oplus_{\mathcal{R}} \psi} BS(\gamma') = BS(\gamma)$. Applying this result in the following derivation, we have

$$\bigcup_{\varphi' \in \varphi \oplus_{\mathcal{R}} \Psi} BS(\varphi') = \bigcup_{\gamma \in (\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle)} \; \bigcup_{\gamma' \in \gamma \oplus_{\mathcal{R}} \psi_{n+1}} BS(\gamma'))$$
$$= \bigcup_{\gamma \in (\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle)} BS(\gamma)$$
$$= BS(\varphi) \qquad \text{(inductive hypothesis)}$$

The proposition has been proved.

Proof of Lemma 2. The proof is by induction on n as follows.

- Base case $n = 0$: the proof is trivial.
- Inductive step: Suppose that the proposition holds for some $n \geq 0$, we will prove that the proposition also holds for $n + 1$ by contradiction. Consider a sequence of consistent sets of literals $\Psi = \langle \psi_1, \ldots, \psi_n, \psi_{n+1} \rangle$. Assume that there exist two different formulae α and β in $\varphi \oplus_{\mathcal{R}} \Psi$ such that α and β agree on very ψ_i in $\{\psi_1, \ldots, \psi_{n+1}\}$. By definition, we have

  $$\varphi \oplus_{\mathcal{R}} \Psi = \bigcup_{\gamma \in (\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n \rangle)} \gamma \oplus_{\mathcal{R}} \psi_{n+1}$$

  Thus, there exist α_n and β_n in $\varphi \oplus \langle \psi_1, \ldots, \psi_n \rangle$ such that $\alpha \in \alpha_n \oplus_{\mathcal{R}} \psi_{n+1}$ and $\beta \in \beta_n \oplus_{\mathcal{R}} \psi_{n+1}$.

  We will prove that $\alpha_n = \beta_n$ by contradiction: Assume that $\alpha_n \neq \beta_n$. By induction, there exists ψ_i in $\{\psi_1, \ldots, \psi_n\}$ such that α_n and β_n disagree on ψ_i. By Definition 9, it is easy to see that $\alpha \models \alpha_n$ and $\beta \models \beta_n$. This implies that α and β disagree on ψ_i too, a contradiction.

  Thus $\alpha_n = \beta_n$. Hence, $\alpha, \beta \in \alpha_n \oplus_{\mathcal{R}} \psi_{n+1}$. Because $\alpha \neq \beta$, this implies that $\{\alpha, \beta\} = \{conv_{\mathcal{R}}(\alpha_n \wedge \psi_{n+1}), conv_{\mathcal{R}}(\alpha_n \wedge \neg\psi_{n+1})\}$. This means that α and β disagree on ψ_{n+1}, a contradiction.

Proof of Theorem 1. Consider an arbitrary enabling form Γ of φ for a. First, we will prove that $|\Gamma| \geq \varphi \oplus_{\mathcal{R}} \Psi$. Let $\{\varphi_1, \ldots, \varphi_n\} = \varphi \oplus_{\mathcal{R}} \Psi$. Since φ is satisfiable, every formula in $\varphi \oplus_{\mathcal{R}} \Psi$ is satisfiable. For each φ_i in $\{\varphi_1, \ldots, \varphi_n\}$, we consider a model of φ_i, denoted by s_i (i.e., $s_i \in BS(\varphi_i)$ and hence $s_i \in BS(\varphi)$). Since $\bigcup_{\gamma \in \Gamma} BS(\gamma) = BS(\varphi_1) \cup \ldots \cup BS(\varphi_n) = BS(\varphi)$, each s_i must be a model of some formula in Γ. Moreover, since each pair of different $\mathcal{R}$-states (φ_i, φ_j) in $\{\varphi_1, \ldots, \varphi_n\}$ disagree on some ψ in $\Psi(a)$ (Lemma 2), their models s_i and s_j also disagree on ψ. This implies that s_i and s_j can not be the models of a same formula in Γ. Thus, Γ contains at least n different formulae, each of which shares (at least) a model with a formula in $\{\varphi_1, \ldots, \varphi_n\}$. Hence $|\Gamma| \geq |\varphi \oplus_{\mathcal{R}} \Psi|$. This implies that $\varphi \oplus_{\mathcal{R}} \Psi$ contains the minimum number of $\mathcal{R}$-states among all the enabling form of φ for a.

Now we consider the case that $|\Gamma| = \varphi \oplus_{\mathcal{R}} \Psi = n$. Let $\{\gamma_1, \ldots, \gamma_n\} = \Gamma$. Without lost of generality, assume that γ_i and φ_i share a same model s_i, for $i = 1, \ldots, n$. We prove that $BS(\gamma_i) = BS(\varphi_i)$. We assume the contrary, i.e., $BS(\gamma_i) \neq BS(\varphi_i)$ for some $1 \leq i \leq n$. We consider the following two cases:

- Case $\exists s \in BS(\gamma_i).s \notin BS(\varphi_i)$: Since $BS(\varphi_1) \cup \ldots \cup BS(\varphi_n) = BS(\gamma_1) \cup \ldots \cup BS(\gamma_n) = BS(\varphi)$, there exists φ_j, $1 \leq j \leq n$ and $j \neq i$, such that $s \in BS(\varphi_j)$. This means that γ_i is not enabling for a as $BS(\gamma_i)$ contains two states that disagree on some e-condition of a, a contradiction.

- Case $\exists s \in BS(\varphi_i).\ s \notin BS(\gamma_i)$: Then there exists γ_j, $1 \leq j \leq n$ and $j \neq i$, such that $s \in BS(\gamma_j)$. Thus γ_j has two models s and s_j that disagree on some e-condition of a and hence γ_j is not enabling for a, a contradiction.

Both cases lead to a contradiction. Hence $BS(\gamma_i) = BS(\varphi_i)$ for $i = 1, \ldots, n$. In other words, $\varphi \oplus_{\mathcal{R}} \Psi$ and Γ represent the same set of belief states.

Proof of Theorem 2. Let a be an action and φ be a $\mathcal{R}$-state. By Definition 3, to prove that $\Phi_{\mathcal{R}}$ is a transition function for $\mathcal{R}$, we need to prove that $BS(\Phi_{\mathcal{R}}(a, \varphi)) = \Phi(a, BS(\varphi))$. We have two cases as follows.

1. $\varphi \models pre(a)$: by definition,

$$\Phi_{\mathcal{R}}(a, \varphi) = merge_{\mathcal{R}}(\{update_{\mathcal{R}}(\gamma, e(a, \gamma)) \mid \gamma \in enb_{\mathcal{R}}(a, \varphi)\})$$

Hence,

$$\begin{aligned}
BS(\Phi_{\mathcal{R}}(a, \varphi)) &= BS(merge_{\mathcal{R}}(\{update_{\mathcal{R}}(\gamma, e(a, \gamma)) \mid \gamma \in enb_{\mathcal{R}}(a, \varphi)\})) \\
&= \bigcup_{\gamma \in enb_{\mathcal{R}}(a,\varphi)} BS(update_{\mathcal{R}}(\gamma, e(a, \gamma))) && \text{(by Equation 7)} \\
&= \bigcup_{\gamma \in enb_{\mathcal{R}}(a,\varphi)} \{s \setminus \overline{e(a, \gamma)} \cup e(a, \gamma) \mid s \in BS(\gamma)\} && \text{(by Equation 6)} \\
&= \bigcup_{\gamma \in enb_{\mathcal{R}}(a,\varphi)} \{s \setminus \overline{e(a, s)} \cup e(a, s) \mid s \in BS(\gamma)\} && \text{(by Proposition 5)} \\
&= \{s \setminus \overline{e(a, s)} \cup e(a, s) \mid s \in \bigcup_{\gamma \in enb_{\mathcal{R}}(a,\varphi)} BS(\gamma)\} \\
&= \{s \setminus \overline{e(a, s)} \cup e(a, s) \mid s \in BS(\varphi)\} && \text{(by Definition 5)}
\end{aligned}$$

Because $\varphi \models pre(a)$ iff $BS(\varphi) \models pre(a)$, by definition we have $\Phi(a, BS(\varphi)) = \{s \setminus \overline{e(a, s)} \cup e(a, s) \mid s \in BS(\varphi)\}$. Thus, $BS(\Phi_{\mathcal{R}}(a, \varphi)) = \Phi(a, BS(\varphi))$.

2. $\varphi \not\models pre(a)$: by definition, $\Phi_{\mathcal{R}}(a, \varphi)$ is undefined. Hence, $BS(\Phi_{\mathcal{R}}(a, \varphi))$ is undefined. Since $\varphi \not\models pre(a)$ iff $BS(\varphi) \not\models pre(a)$, we have that $\Phi(a, BS(\varphi))$ is undefined. Thus, $BS(\Phi_{\mathcal{R}}(a, \varphi)) = \Phi(a, BS(\varphi))$.

In both cases, we have $BS(\Phi_{\mathcal{R}}(a, \varphi)) = \Phi(a, BS(\varphi))$ (proof).

Proof of Theorem 3. The proof is by induction on the length n of α_n.

- Base case $n = 0$: by definition, $BS(\widehat{\Phi_{\mathcal{R}}}([\,], \varphi)) = \widehat{\Phi}([\,], BS(\varphi)) = BS(\varphi)$.
- Inductive step: suppose that the proposition holds for every sequence of length less or equal to $n - 1$, for some $n > 0$. We will show that it holds for every sequence of length n. We have

$$\begin{aligned}
BS(\widehat{\Phi_{\mathcal{R}}}(\alpha_n, \varphi)) &= BS(\Phi_{\mathcal{R}}(a_n, \widehat{\Phi_{\mathcal{R}}}(\alpha_{n-1}, \varphi))) && \text{(by Definition 9)} \\
&= BS(\Phi_{\mathcal{R}}(a_n, \widehat{\Phi}(\alpha_{n-1}, BS(\varphi)))) && \text{(inductive hypothesis)} \\
&= \Phi(a_n, \widehat{\Phi}(\alpha_{n-1}, BS(\varphi))) && \text{(by Definition 3)} \\
&= \widehat{\Phi}(\alpha_n, BS(\varphi)) && \text{(by definition of } \widehat{\Phi} \text{)}
\end{aligned}$$

Proof of Proposition 9. Let $\{\psi_i \to \eta_i \mid i = 1, \ldots, n\}$ be the set of combined conditional effects of a. Clearly, $\Psi(a) = \{\psi_1, \ldots, \psi_n\}$ and $\Psi = \langle \psi_1, \ldots, \psi_n \rangle$ is an enumeration of $\Psi(a)$. To prove that the proposition holds, we use the following loop invariant for the outer **for** loop (lines 6-12 of Algorithm 4):

At the beginning of each iteration of the **for** loop, we have $Result = \{\langle \gamma, e_i(\gamma) \rangle \mid \gamma \in \varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_{i-1} \rangle\}$, where $e_i(\gamma) = \{\ell \mid a : \psi_j \to \ell, \gamma \models \psi_j, j < i\}$.

We need to show that this invariant is true prior to the first (outer) loop iteration, that each iteration of the loop maintains the invariant, and that the invariant provides useful properties to show correctness when the loop terminates.

- **Initialization:** Prior to the first iteration of the loop, $i = 1$ and $Result = \{\langle \varphi, \emptyset \rangle\}$ (lines 3-4). The sequence $\langle \psi_1, \ldots, \psi_{i-1} \rangle$ is empty so, by definition, $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_{i-1} \rangle = \{\varphi\}$. Furthermore, $e_1(\gamma) = \{\ell \mid a : \psi_j \to \ell, j < 1\} = \emptyset$ since $1 \leq j \leq n$ for every $a : \psi_j \to \ell$. Thus, the invariant holds.

- **Maintenance:** For each $\langle \gamma, \mathit{effect} \rangle \in Result$, by Proposition 8, $\texttt{extending}_{\mathcal{R}}(\gamma, \mathit{effect}, \psi_i \to \eta_i) = \{\langle \gamma', \mathit{effect} \cup \{\ell \mid a : \psi_i \to \ell \wedge \gamma' \models \psi_i\} \rangle \mid \gamma' \in \gamma \oplus_{\mathcal{R}} \psi_i\}$ (line 9). It is easy to see that at the end of the inner **for** loop (lines 8-10), we have

$$
\begin{aligned}
X &= \bigcup_{\langle \gamma, \mathit{effect} \rangle \in Result} \texttt{extending}_{\mathcal{R}}(\gamma, \mathit{effect}, \psi_i \to \eta_i) && \text{(Lines 7-10)} \\
&= \bigcup_{\langle \gamma, \mathit{effect} \rangle \in Result} \{\langle \gamma', \mathit{effect} \cup \{\ell \mid a : \psi_i \to \ell \wedge \gamma' \models \psi_i\} \rangle \mid \gamma' \in \gamma \oplus_{\mathcal{R}} \psi_i\} && \text{(Proposition 8)}
\end{aligned}
$$

By the loop invariant, we have that for each $\langle \gamma, \mathit{effect} \rangle \in Result$ (line 8), $\mathit{effect} = e_i(\gamma) = \{\ell \mid a : \psi_j \to \ell, \gamma \models \psi_j, j < i\}$. Consider $\gamma' \in \gamma \oplus_{\mathcal{R}} \psi_i$. Since $\gamma' \models \gamma$, we have $\gamma' \models \psi_j$ if $\gamma \models \psi_j$ and $\gamma' \models \neg\psi_j$ if $\gamma \models \neg\psi_j$ for $j < i$. This implies $\mathit{effect} = \{\ell \mid a : \psi_j \to \ell, \gamma' \models \psi_j, j < i\} = e_i(\gamma')$. Hence, $\mathit{effect} \cup \{\ell \mid a : \psi_i \to \ell \wedge \gamma' \models \psi_i\} = e_i(\gamma') \cup \{\ell \mid a : \psi_i \to \ell \wedge \gamma' \models \psi_i\} = \{\ell \mid a : \psi_j \to \ell, \gamma' \models \psi_j, j < i + 1\} = e_{i+1}(\gamma')$. Hence,

$$X = \bigcup_{\langle \gamma, e_i(\gamma) \rangle \in Result} \{\langle \gamma', e_{i+1}(\gamma') \rangle \mid \gamma' \in \gamma \oplus_{\mathcal{R}} \psi_i\}$$

Since $e_i(\gamma)$ does not appear in $\{\langle \gamma', e_{i+1}(\gamma') \rangle \mid \gamma' \in \gamma \oplus_{\mathcal{R}} \psi_i\}$, we have

$$X = \bigcup_{\gamma \in first(Result)} \{\langle \gamma', e_{i+1}(\gamma') \rangle \mid \gamma' \in \gamma \oplus_{\mathcal{R}} \psi_i\}$$

Where $first(Result) = \{\gamma \mid \langle \gamma, e_i(\gamma) \rangle \in Result\}$. By the loop invariant, we have $Result = \{\langle \gamma, e_i(\gamma) \rangle \mid \gamma \in \varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_{i-1} \rangle\}$. Hence, $first(Result) = \varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_{i-1} \rangle$. This implies that

$$X = \bigcup_{\gamma \in \varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_{i-1} \rangle} \{\langle \gamma', e_{i+1}(\gamma') \rangle \mid \gamma' \in \gamma \oplus_{\mathcal{R}} \psi_i\}$$

By Definition 7, it is easy to see that

$$X = \{\langle \gamma', e_{i+1}(\gamma') \rangle \mid \gamma' \in \varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_i \rangle\}$$

Thus, the next iteration maintains the loop invariant.

- **Termination:** At termination of the loop, $i = n + 1$. We have $Result = \{\langle \gamma, e_{n+1}(\gamma)\rangle \mid \gamma \in \varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n\rangle\}$. Since $\langle \psi_1, \ldots, \psi_n\rangle$ is an enumeration of $\Psi(a)$, by definition, we have $\varphi \oplus_{\mathcal{R}} \langle \psi_1, \ldots, \psi_n\rangle = enb_{\mathcal{R}}(a, \varphi)$. Moreover, $e_{n+1}(\gamma) = \{\ell \mid a : \psi_j \rightarrow \ell, \gamma \models \psi_j, j < n + 1\} = \{\ell \mid a : \psi_j \rightarrow \ell, \gamma \models \psi_j\} = e(a, \gamma)$. Hence, $Result = \{\langle \gamma, e(a, \gamma)\rangle \mid \gamma \in enb_{\mathcal{R}}(a, \varphi)\}$ that the procedure returns (proof).

Proof of Proposition 11. Observe that $refine(\Delta)$ is obtained by removing every inconsistent set of literals from Δ, by definition, clearly $refine(\Delta)$ is a set of partial states. Moreover, by Proposition 1, $BS(\Delta) = \bigcup_{\delta\in\Delta} BS(\delta) = \bigcup_{\delta\in\Delta\wedge\delta \text{ is consistent}} BS(\delta) \cup \bigcup_{\delta\in\Delta\wedge\delta \text{ is inconsistent}} BS(\delta) = BS(refine(\Delta)) \cup \bigcup_{\delta\in\Delta\wedge\delta \text{ is inconsistent}} BS(\delta)$. Since every inconsistent set of literals δ is unsatisfiable and $BS(\delta) = \emptyset$, we have $\bigcup_{\delta\in\Delta\wedge\delta \text{ is inconsistent}} BS(\delta) = \emptyset$. This implies that $BS(\Delta) = BS(refine(\Delta)$ or $\Delta \equiv refine(\Delta)$.

On the other hand, by definition, clearly $\mu(\Delta) = min(refine(\Delta))$ is a μDNF-state. By definition, $min(refine(\Delta)) = \{\delta \mid \delta \in refine(\Delta) \wedge \nexists \delta' \in refine(\Delta).\ \delta' \subsetneq \delta\}$ and hence $min(refine(\Delta))$ is a subset of $refine(\Delta)$. Let $\Delta_{sup} = refine(\Delta) \backslash min(refine(\Delta))$. Then Δ_{sup} is a subset of $refine(\Delta)$ and, by proposition 1, $BS(refine(\Delta)) = BS(min(refine(\Delta))) \cup BS(\Delta_{sup})$. Consider an arbitrary $\delta \in \Delta_{sup}$. Then there exists $\delta' \in min(refine(\Delta))$ such that $\delta' \subsetneq \delta$ (otherwise, $\delta \in min(refine(\Delta))$ and hence $\delta \notin \Delta_{sup}$). Hence, $\delta \models \delta'$ or $BS(\delta) \subseteq BS(\delta')$. This implies that $BS(\Delta_{sup}) \subseteq BS(min(refine(\Delta)))$, since δ is an arbitrary element in Δ_{sup}. Thus, $BS(min(refine(\Delta))) \cup BS(\Delta_{sup}) = BS(min(refine(\Delta)))$ and hence $BS(refine(\Delta)) = BS(min(refine(\Delta)))$, i.e., $refine(\Delta) \equiv min(refine(\Delta)) = \mu(\Delta)$. Thus, $\mu(\Delta) \equiv \Delta$ (proof).

Proof of Proposition 12.

1. By definition, the belief state of δ is the set of states that satisfy δ. Since δ is a consistent set of literals, a state s that satisfies δ iff s satisfies every literal in δ, i.e., $\delta \subseteq s$. Thus, $BS(\delta) = \{s \mid s \text{ is a state, and } \delta \subseteq s\}$
2. Let φ be an arbitrary formula that represents $BS(\delta)$. Then, $\varphi \equiv \delta$. This implies that φ entails every literal in δ. Thus, φ must also contain every literal in δ. As a consequence, $|\varphi| \geq |\delta|$.

Proof of Proposition 13.

1. First, we will prove that $conv_{\mu DNF}(\Delta \wedge \psi) = min(\{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset\})$ is a $\mathcal{R}$-state. Since δ and ψ are consistent set of literals and one does not contain the negation of a literal in the other set ($\delta \cap \overline{\psi} = \emptyset$), $\delta \cup \psi$ is a consistent set of literals. By, Definition 12 and Proposition 11, clearly $conv_{\mu DNF}(\Delta \wedge \psi)$ is a μDNF-state. Now we need to prove $conv_{\mu DNF}(\Delta \wedge \psi) \equiv \Delta \wedge \psi$. We have

$$
\begin{aligned}
\Delta \wedge \psi &\equiv \{\delta \wedge \psi \mid \delta \in \Delta\} && \text{(distributivity of } \wedge \text{ over } \vee) \\
&= \{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset\} \cup \\
&\quad \{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} \neq \emptyset\} \\
&\equiv \mu(\{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset\} \cup \\
&\quad \{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} \neq \emptyset\}) && \text{(Proposition 11)} \\
&= min(refine(\{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset\} \cup \\
&\quad \{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} \neq \emptyset\})) \\
&= min(\{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset\}) && (\delta \cup \psi \text{ is inconsistent when } \delta \cap \overline{\psi} \neq \emptyset) \\
&= conv_{\mu DNF}(\Delta \wedge \psi) && \text{(by definition)}
\end{aligned}
$$

2. Let $\Delta_1 = \{\delta \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} \neq \emptyset\}$ and $\Delta_2 = \bigcup_{\delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\}$. By Definition 13, we have $conv_{\mu DNF}(\Delta \wedge \neg\psi) = min(\Delta_1 \cup \Delta_2)$. Since every δ in Δ is a consistent set of literals, every $\delta \cup \{\overline{\ell}\}$ in Δ_2 is a consistent set of literals because $\ell \notin \delta$. Hence, $\Delta_1 \cup \Delta_2$ is a DNF formula that contains only consistent sets of literals. Similar to the previous proof, $min(\Delta_1 \cup \Delta_2)$ is a μDNF-state. The proof for equivalency between $conv_{\mu DNF}(\Delta \wedge \neg\psi)$ and $\Delta \wedge \neg\psi$ is as follows.

$$
\begin{aligned}
\Delta \wedge \neg\psi &\equiv \{\delta \wedge \neg\psi \mid \delta \in \Delta\} && \text{(distributivity of } \wedge \text{ over } \vee) \\
&= \{\delta \wedge \neg\psi \mid \delta \in \Delta \ \wedge \delta \models \neg\psi\} \cup \\
&\quad \{\delta \wedge \neg\psi \mid \delta \in \Delta \ \wedge \delta \not\models \neg\psi\} \\
&\equiv \{\quad \delta \quad \mid \delta \in \Delta \ \wedge \delta \models \neg\psi\} \cup \\
&\quad \{\delta \wedge \neg\psi \mid \delta \in \Delta \ \wedge \delta \not\models \neg\psi\} \\
&= \Delta_1 \cup \{\delta \wedge \neg\psi \mid \delta \in \Delta \ \wedge \delta \cap \overline{\psi} = \emptyset\} && (\delta \models \neg\psi \text{ iff } \delta \cap \overline{\psi} \neq \emptyset) \\
&\equiv \Delta_1 \cup \textstyle\bigcup_{\delta \in \Delta \ \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi\} \\
&\equiv \Delta_1 \cup \textstyle\bigcup_{\delta \in \Delta \ \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\} && (\delta \cup \{\overline{\ell}\} \text{ is inconsistent if } \ell \in \delta) \\
&= \Delta_1 \cup \Delta_2 \\
&\equiv min(\Delta_1 \cup \Delta_2) \\
&= conv_{\mu DNF}(\Delta \wedge \neg\psi)
\end{aligned}
$$

Proof of Proposition 14. First we will prove that, for every partial state δ,

$$\delta \setminus \bar{e} \cup e \equiv \{s \setminus \bar{e} \cup e \mid s \in BS(\delta)\} \qquad \text{(14a)}$$

For every state s in $BS(\delta)$, s is a superset of δ (by Proposition 12). Hence, $\delta \setminus \bar{e} \cup e \subseteq s \setminus \bar{e} \cup e$, i.e., $s \setminus \bar{e} \cup e \models \delta \setminus \bar{e} \cup e$ or $s \setminus \bar{e} \cup e \in BS(\delta \setminus \bar{e} \cup e)$ (it is easy to see that $s \setminus \bar{e} \cup e$ is a state). This means that

$$\{s \setminus \bar{e} \cup e \mid s \in BS(\delta)\} \subseteq BS(\delta \setminus \bar{e} \cup e) \qquad \text{(14b)}$$

Now we need to prove that $BS(\delta \setminus \bar{e} \cup e) \subseteq \{s \setminus \bar{e} \cup e \mid s \in BS(\delta)\}$.

Let s_1 be an arbitrary state in $BS(\delta \setminus \bar{e} \cup e)$, we will prove that s_1 also belongs to $\{s \setminus \bar{e} \cup e \mid s \in BS(\delta)\}$. Since $s_1 \in BS(\delta \setminus \bar{e} \cup e)$, $s_1 \models (\delta \setminus \bar{e} \cup e)$ or $(\delta \setminus \bar{e} \cup e) \subseteq s_1$. Let $\gamma = s_1 \setminus (\delta \setminus \bar{e} \cup e)$. Observe that γ is a consistent set of literals that are independent from $\delta \setminus \bar{e} \cup e$. That means $prop(\gamma) = F \setminus prop(\delta \setminus \bar{e} \cup e)$, where F is the set of propositions in the domain. Let $e^- = \delta \cap \bar{e}$, $e^+ = e \setminus \overline{e^-}$, and $\delta_0 = \delta \setminus e^-$. Observe that, $s_1 = \delta \setminus \bar{e} \cup e \cup \gamma = (\delta_0 \cup e^-) \setminus (e^- \cup \overline{e^+}) \cup (\overline{e^-} \cup e^+) \cup \gamma = \delta_0 \cup \overline{e^-} \cup e^+ \cup \gamma$. Now consider $s_2 = \delta_0 \cup e^- \cup e + \cup \gamma = \delta \cup e^+ \cup \gamma$. Furthermore, every partition set in s_2: δ_0, e^-, e^+, and γ are consistent; e^- and γ are independent from all the other sets while $\delta_0 \cup e^+$ is consistent. Therefore, s_2 is consistent and thereby it is a state (because $prop(s_2) = prop(s_1) = F$). Since $\delta \subseteq s_2$ so $s_2 \in BS(\delta)$. Observe that $s_2 \setminus \bar{e} \cup e = \delta \cup e^+ \cup \gamma \setminus \bar{e} \cup e = \delta \setminus \bar{e} \cup e \cup \gamma = s_1$ (because γ is independent from $\bar{e}$ and $e^+ \subseteq e$). This means that $s_1 \in \{s \setminus \bar{e} \cup e \mid s \in BS(\delta)\}$. This implies that

$$BS(\delta \setminus \bar{e} \cup e) \subseteq \{s \setminus \bar{e} \cup e \mid s \in BS(\delta)\} \qquad \text{(14c)}$$

Together (14b) and (14c) we have the proof for (14a). Now we have

$$\begin{aligned} update_{\mu DNF}(\Delta, e) &= min(\{\delta \setminus \overline{e} \cup e \mid \delta \in \Delta\}) \\ &\equiv \{\delta \setminus \overline{e} \cup e \mid \delta \in \Delta\} \\ &\equiv \bigcup_{\delta \in \Delta} \{s \setminus \overline{e} \cup e \mid s \in \delta\} && \text{(by (14a))} \\ &= \{s \setminus \overline{e} \cup e \mid s \in \bigcup_{\delta \in \Delta} BS(\delta)\} \\ &= \{s \setminus \overline{e} \cup e \mid s \in BS(\Delta)\} && \text{(by Proposition 1)} \end{aligned}$$

Proof of Proposition 15. Obviously, $(\bigcup_{\Delta \in \Gamma} \Delta)$ is a set of partial states (consistent sets of literals). Hence, $merge_{\mu DNF}(\Gamma) = min(\bigcup_{\Delta \in \Gamma} \Delta) = \mu(\bigcup_{\Delta \in \Gamma} \Delta)$. By Proposition 11, this is a μDNF-state equivalent to $(\bigcup_{\Delta \in \Gamma} \Delta)$. Moreover, due to associativity of $\vee$, we have $\bigcup_{\Delta \in \Gamma} \Delta \equiv \bigvee_{\Delta \in \Gamma} \Delta$. Thus, $merge_{\mu DNF}(\Gamma)$ is a μDNF-state equivalent to $\bigvee_{\Delta \in \Gamma} \Delta$. Hence, $BS(merge_{\mu DNF}(\Gamma)) = BS(\bigvee_{\Delta \in \Gamma} \Delta)$. By Proposition 1, we have $BS(\bigvee_{\Delta \in \Gamma} \Delta) = \bigcup_{\Delta \in \Gamma} BS(\Delta)$. Thus, $BS(merge_{\mu DNF}(\Gamma)) = \bigcup_{\Delta \in \Gamma} BS(\Delta)$ (proof).

Proof of Proposition 16. First we prove that there does not exist a pair of different μDNF-states $\Delta_1, \Delta_2 \in enb_{\mu DNF}(a, \Delta)$ such that $\exists \delta_1 \in \Delta_1 \exists \delta_2 \in \Delta_2.\ \delta_1 \subseteq \delta 2$ (Proposition 16*). Assume the contrary, that there exists a pair of different μDNF-states $\Delta_1, \Delta_2 \in enb_{\mu DNF}(a, \Delta)$ such that $\exists \delta_1 \in \Delta_1 \exists \delta_2 \in \Delta_2.\ \delta_1 \subseteq \delta 2$. By Lemma 2, there exists an e-condition ψ of a such that Δ_1 and Δ_2 disagree on ψ. Without lost of generality, assume that $\Delta_1 \models \psi$ and $\Delta_2 \models \neg\psi$. This means that $\delta_1 \models \psi$ and $\delta_2 \models \neg\psi$ (by Proposition 1). On the other hand, since $\delta_1 \subseteq \delta_2$, we have $\delta_2 \models \delta_1$ and, hence, $\delta_2 \models \psi$, a contradiction. Since a μDNF-state does not contains a pair of partial states such that one is a proper subset of the other, (16*) implies that $\bigcup_{\Delta_i \in enb_{\mu DNF}(a,\Delta)} \Delta_i$ is a μDNF-state and every δ in $\bigcup_{\Delta_i \in enb_{\mu DNF}(a,\Delta)} \Delta_i$ belongs to one and only one μDNF-state in $enb_{\mu DNF}(a, \Delta)$ (proof).

Proof of Proposition 17. First we prove that for $\delta \in \Delta$, if $\psi \subseteq \delta$ then $\delta \cap \overline{\psi} = \emptyset$ (17a). Assume the contrary, that $\exists \delta \in \Delta$ such that $\psi \subseteq \delta$ and $\delta \cap \overline{\psi} \neq \emptyset$. Consider $\ell \in \delta \cap \overline{\psi}$. Then $\overline{\ell} \in \psi$ and hence $\overline{\ell} \in \delta$ (because $\psi \subseteq \delta$). Thus, δ is not consistent since it contains $\{\ell, \overline{\ell}\}$, a contradiction. To prove the proposition, we consider the following three cases.

1. $\Delta \models \psi$: By Proposition 1, we have that $\forall \delta \in \Delta.\ \delta \models \psi$, i.e., $\forall \delta \in \Delta.\ \psi \subseteq \delta$. This implies that, after the execution of Line 4 and before the execution of Line 8, $\Delta_1 = \Delta$. By (17a), after the execution of Line 5, we have $\Delta_2 = \emptyset$. After the execution Line 6, we have $\Delta_0 = \emptyset$. Hence, Δ_1 and Δ_2 do not change after the **for** loop (Lines 8-10). It is easy to see that $Result = \{\langle \Delta_1, \mathit{effect} \cup \eta \rangle\} = \{\langle \Delta, \mathit{effect} \cup \{\ell \mid a : \psi \to \ell \wedge \Delta \models \psi\} \rangle\}$. By definition, we have that $\Delta \oplus_{\mu DNF} \psi = \{\Delta\}$ if $\Delta \models \psi$. This implies that $Result = \{\langle \Delta_i, \mathit{effect} \cup \{\ell \mid a : \psi \to \ell \wedge \Delta_i \models \psi\} \rangle \mid \Delta_i \in \Delta \oplus_{\mu DNF} \psi\}$ (proof).
2. $\Delta \models \psi$: this case can be proved similarly to the first case.
3. ψ is unknown in δ: This implies that $\Delta_0 \neq \emptyset$ and hence $\Delta_1 \neq \emptyset$, $\Delta_2 \neq \emptyset$. We need to prove that

$\Delta_1 = conv_{\mu DNF}(\Delta \wedge \psi)$ and $\Delta_2 = conv_{\mu DNF}(\Delta \wedge \neg\psi)$. From Lines 4,6,8,9,11 we have

$$\begin{aligned}
\Delta_1 &= min(\{\delta \mid \delta \in \Delta, \psi \subseteq \delta\} \cup \{\delta \cup \psi \mid \delta \in \Delta_0\}) \\
&= min(\{\delta \mid \delta \in \Delta, \psi \subseteq \delta\} \cup \{\delta \cup \psi \mid \delta \in \Delta \wedge \psi \not\subseteq \delta \wedge \delta \cap \overline{\psi} = \emptyset\}) && \text{(Lines 4-6)} \\
&= min(\{\delta \cup \psi \mid \delta \in \Delta, \psi \subseteq \delta\} \cup \{\delta \cup \psi \mid \delta \in \Delta \wedge \psi \not\subseteq \delta \wedge \delta \cap \overline{\psi} = \emptyset\}) && (\delta \cup \psi = \delta \text{ if } \psi \subseteq \delta) \\
&= min(\{\delta \cup \psi \mid \delta \in \Delta, \psi \subseteq \delta \wedge \delta \cap \overline{\psi} = \emptyset\} \cup \\
&\qquad \{\delta \cup \psi \mid \delta \in \Delta \wedge \psi \not\subseteq \delta \wedge \delta \cap \overline{\psi} = \emptyset\}) && \text{(by (17a))} \\
&= min(\{\delta \cup \psi \mid \delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset\} \\
&= conv_{\mu DNF}(\delta \wedge \psi) && \text{(by definition)}
\end{aligned}$$

Thus $\Delta_1 = conv_{\mu DNF}(\delta \wedge \psi)$. Now we will prove that $\Delta_2 = conv_{\mu DNF}(\Delta \wedge \neg\psi)$. For brevity, let $\Delta_{add} = \bigcup_{\delta \in \Delta_0} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\}$. It is easy to see that $\Delta_2 = min(\{\delta \mid \delta \in \Delta, \delta \cap \overline{\psi} \neq \emptyset\} \cup \Delta_{add})$ (Lines 5,6,8,10,11). Consider Δ_{add}, we have

$$\begin{aligned}
\Delta_{add} &= \bigcup_{\delta \in \Delta_0} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\} \\
&= \bigcup_{\delta \in \Delta \wedge \psi \not\subseteq \delta \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\} \\
&= \bigcup_{\delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\} \setminus \bigcup_{\delta \in \Delta \wedge \psi \subseteq \delta \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\} \\
&= \bigcup_{\delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\} \setminus \emptyset && (\psi \setminus \delta = \emptyset \text{ if } \psi \subseteq \delta) \\
&= \bigcup_{\delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\}
\end{aligned}$$

Thus,

$$\Delta_2 = min(\{\delta \mid \delta \in \Delta, \delta \cap \overline{\psi} \neq \emptyset\} \cup \bigcup_{\delta \in \Delta \wedge \delta \cap \overline{\psi} = \emptyset} \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\})$$

By definition, we have $\Delta_2 = conv_{\mu DNF}(\Delta \wedge \neg\psi)$. It is easy to see that $Result = \{\langle \Delta_1, \mathit{effect} \cup \eta \rangle, \langle \Delta_2, \mathit{effect} \rangle\} = \{\langle \Delta, \mathit{effect} \cup \{\ell \mid a : \psi \to \ell \wedge \Delta \models \psi\} \rangle, \langle \Delta_2, \mathit{effect} \rangle\}$. By definition, we have that $\Delta \oplus_{\mu DNF} \psi = \{conv_{\mu DNF}(\Delta \wedge \psi), conv_{\mu DNF}(\Delta \wedge \neg\psi)\} = \{\Delta_1, \Delta_2\}$ if ψ is unknown in Δ. This implies that $Result = \{\langle \Delta_i, \mathit{effect} \cup \{\ell \mid a : \psi \to \ell \wedge \Delta_i \models \psi\} \rangle \mid \Delta_i \in \Delta \oplus_{\mu DNF} \psi\}$ (proof).

In conclusion, the proposition has been proved in all cases.

Proof of Proposition 48. Following Algorithm 6 for the computation of $\Delta \oplus_{\mu DNF} \psi$, we have

- Checking whether ψ is true or false in δ (Lines 4 and 5) is $O(|\delta| + |\psi|) = O(n)$. Hence, the total cost of computation of Δ_1, Δ_2, and Δ_0 from Line 4 to line 6 is $O(n|\Delta|)$.
- Computing $min(\Delta_1 \cup \{\delta \cup \psi\})$ (Line 9) is $O(n + n) + O(n|\Delta_1|) = O(n|\Delta|)$ ($|\Delta_1| \leq |\Delta|$).
- Computing $min(\Delta_2 \cup \{\delta \cup \{\overline{\ell}\} \mid \ell \in \psi \setminus \delta\})$ (Line 10) is $O(nr + nr|\Delta_2|) = O(nr^2|\Delta|)$ ($|\Delta_2| \leq r|\Delta|$).

- Computing Δ_1 and Δ_2 from Line 8 to Line 11 is, hence, $O(nr^2|\Delta||\Delta_0|) = O(nr^2|\Delta|^2)$ (only when ψ is unknown in Δ).

- The computation from Line 12 to Line 19 is $O(n)$.

In summary, computing $\Delta \oplus_{\mu DNF} \psi$ is $T(\Delta \oplus_{\mu DNF} \psi) = O(n|\Delta|)$ if ψ is known in Δ, and $T(\Delta \oplus_{\mu DNF} \psi) = O(nr^2|\Delta|^2)$ otherwise (proof).

Proof of Theorem 4. The proof of this theorem is a consequence of a number of intermediate propositions.

Let n be the number of propositions in the domain, p be the number of combined conditional effects of a, and k be the number of e-conditions of action a that are unknown in Δ. Let r be the maximum number of unknown literals in an e-condition of a and m be the maximum number of literals in an effect of a. By $|\Delta|$ we denote the number of partial states in μDNF-state Δ. For convenience, we say that the size of Δ is $\vartheta(N)$ if $|\Delta|$ is exactly N or it can be proved to be at most N. Furthermore, for a function $f(.)$, by $T(f(.))$ we denote the running time for computing $f(.)$. We will investigate the features that affect $T(\Phi_{\mu DNF}(a, \Delta))$, the cost of computation of $\Phi_{\mu DNF}$, in the *worst case* as follows.

Proposition 48. *Let Δ be a μDNF-state and ψ be a consistent set of literals. Computing $\Delta \oplus_{\mu DNF} \psi$ (Algorithm 6) is $T(\Delta \oplus_{\mu DNF} \psi) = O(n|\Delta|)$ if ψ is known in Δ, and $T(\Delta \oplus_{\mu DNF} \psi) = O(nr^2|\Delta|^2)$ otherwise.*

Proof. The proof is by induction on k.

- Base case $k = 0$: the proposition is trivial.

- Base case $k = 1$: this means that the outer **for** loop (lines 6-10) in Algorithm 4 executes only the first iteration. It starts with the singleton set $Result = \{\langle \Delta, e(\Delta)\}$ and at the end of this iteration (Line 9), $Result$ contains at most two μDNF-states Δ_1 and/or Δ_2 of the size $\vartheta(|\Delta|)$ and $\vartheta(r|\Delta|)$ respectively. The cost of computing $Result$ in this iteration is $O(nr^2|\Delta|^2) = O(n|\Delta|^2(1 + r^2))$.

- Inductive step: Assume that the proposition is true for some $k = i$, $i \geq 1$. We will show that it is also true for $k = i+1$. Let a_i be the action whose set of combined conditional effects is the first i combined conditional effects of a that are fed to the computation of $enb_{\mu DNF}(a_i, \Delta)$ (the first i iterations of the outer for loop of Algorithm 4). Let X_i be the set of μDNF-states obtained after performing the first i iterations of the outer for loop of Algorithm 4. Observe that each of these iterations is to extend the set X (initialized with $\{\Delta\}$) on a combined conditional effect of a_i and a. Clearly, $X_i = enb_{\mu DNF}(a_i, \Delta)$ and $enb_{\mu DNF}(a, \Delta)$ is obtained by extending the set $X_i = enb_{\mu DNF}(a, \Delta)$ on the last conditional effect of a in the last iteration (iteration $i + 1$). Hence, $T(enb_{\mu DNF}(a, \Delta)) = T(enb_{\mu DNF}(a_i, \Delta)) + T_{i+1}$, where T_{i+1} is the running time for the last iteration. Since $\vartheta(r^j|\Delta|) \leq r^j|\Delta|$, for $j = 0, \ldots, i$, we have,

$$\begin{aligned}
\Sigma_{\Delta' \in X_i}|\Delta'|^2 &\leq \binom{i}{0}(r^0|\Delta|)^2 + \binom{i}{1}(r^1|\Delta|)^2 + \ldots + \binom{i}{i}(r^i|\Delta|)^2 && \text{(inductive hypothesis)} \\
&= |\Delta|^2(\binom{i}{0}(r^2)^0 + \binom{i}{1}(r^2)^1 + \binom{i}{2}(r^2)^2 + \ldots + \binom{i}{j}(r^2)^i) \\
&= |\Delta|^2(1 + r^2)^i && (1)
\end{aligned}$$

Hence, the running time for the iteration $i+1$, as presented in the body of the paper, is

$$\begin{aligned} T_{i+1} &= O(nr^2(\Sigma_{\Delta' \in X_i}|\Delta'|^2)) \\ &= O(nr^2|\Delta|^2(1+r^2)^i) \qquad (2) \end{aligned}$$

Thus, computing $enb_{\mu DNF}(a, \Delta)$ in the worst case is

$$\begin{aligned} T(enb_{\mu DNF}(a,\Delta)) &= T(enb_{\mu DNF}(a_i,\Delta)) + T_{i+1} \\ &= O(n|\Delta|^2(1+r^2)^i) + T_{i+1} && \text{(inductive hypothesis)} \\ &= O(n|\Delta|^2(1+r^2)^i) + O(nr^2|\Delta|^2(1+r^2)^i) && \text{(by (2))} \\ &= O(n|\Delta|^2(1+r^2)^{i+1}) \\ &= O(n|\Delta|^2(1+r^2)^k) \qquad (3) && (k = i+1) \end{aligned}$$

Let Δ' be a μDNF-state of the size $\vartheta(m)$. We know that extending Δ' on a conditional effect of a produces at most two μDNF-states of the size $\vartheta(|\Delta'|) = \vartheta(\vartheta(m)) = \vartheta(m)$ and $\vartheta(r|\Delta'|) = \vartheta(r\vartheta(m)) = \vartheta(rm)$ respectively. Hence, extending X_i on the last e-condition of the conditional effect in the iteration $i+1$ produces at most $\binom{i}{j}$ μDNF-states of the size $\vartheta(r^j|\Delta|)$ and $\binom{i}{j}$ μDNF-states of the size $\vartheta(r^{j+1}|\Delta|)$ for each $j = 0, \ldots i$. This means that, the number of μDNF-states of the size $\vartheta(r^j|\Delta|)$ is at most

$$\binom{i}{j-1} + \binom{i}{j} = \frac{i!}{(i-j+1)!(j-1)!} + \frac{i!}{(i-j)!(j)!} = \frac{(i+1)!}{(i+1-j)!j!} = \binom{i+1}{j}$$

for $j = 1, \ldots, i$ and it is at most $\binom{i}{0}=\binom{i+1}{0}=1$ for $j = 0$ and $\binom{i}{i}=\binom{i+1}{i+1}=1$ for $j = i+1$ (4)

Together (3) and (4) we have the proof for $k = i+1$ and, hence, the proof for the proposition.

The computation of $enb_{\mu DNF}(a, \Delta)$ and effects of a (Algorithm 4) requires:

- Computing $X = X \cup \texttt{extending}_{\mu DNF}(\Delta', \mathit{effects}, \psi_i \rightarrow \eta_i)$ (Line 9): this includes computing $\Delta' \oplus_{\mu DNF} \psi_i$, which is $O(n|\Delta'|)$ if ψ_i is known in Δ' and $O(nr^2|\Delta'|^2)$ otherwise, and adding the new set of (at most two) elements to X which is constant time.
- Computing X for each combined conditional effect $\psi_i \rightarrow \eta_i$ (lines 7-10) is

$$\begin{cases} O(n(\Sigma_{\langle \Delta', \mathit{effect}\rangle \in Result}|\Delta'|)) & \text{if } \forall \langle \Delta', \mathit{effect}\rangle \in Result, \psi_i \text{ is known in } \Delta' \\ O(nr^2(\Sigma_{\langle \Delta', \mathit{effect}\rangle \in Result}|\Delta'|^2)) & \text{otherwise} \end{cases}$$

Observe that this computation depends on both the number of μDNF-states in $Result$ and the number of partial states in each of the μDNF-states.

To evaluate the cost of computing $Result$ from Line 4 to Line 12—which is also $T(enb_{\mu DNF}(a, \Delta))$, since the other computation is negligible—we consider the outer **for** loop (lines 6-10) and obtain the following proposition.

Proposition 49.

1. *Computing* $enb_{\mu DNF}(a, \Delta)$ *(Algorithm 4) is*

$$T(enb_{\mu DNF}(a, \Delta)) = O(n|\Delta|^2(1 + r^2)^k)$$

2. $enb_{\mu DNF}(a, \Delta)$ *contains at most* $\binom{k}{i}$ μDNF*-states of the size* $\vartheta(r^i|\Delta|)$ *for* $i = 0, \ldots, k$.

For a better evaluation of $T(enb_{\mu DNF}(a, \Delta))$, we observe that for each iteration, if ψ_i is known in Δ' for every $\langle \Delta', e(\Delta') \rangle \in Result$, then the $\texttt{extending}_{\mu DNF}$ procedure does not change any μDNF-state Δ' but it might only update $e(\Delta')$, that does not affect the computational cost of the next iterations. It is easy to see that if ψ_i is known in Δ then it is known in Δ' for every $\langle \Delta', e(\Delta') \rangle \in Result$, since $BS(\Delta') \subseteq BS(\Delta)$. This implies that the introduction of a combined conditional effect, whose e-condition is known in Δ, to the **for** loop does not affect the the computational cost of the next iterations. However, because the number of μDNF-states stored in $Results$ and their size may increase after an iteration if ψ_i is unknown in some of them, the sooner a combined conditional effect, whose e-condition is known in Δ, is introduced to the **for** loop, the less computational cost for the iteration. Since we evaluate the computational cost of $\Phi_{\mu DNF}$ in the worst case, we need to consider the (worst) case where the set of combined conditional effects of a, whose e-conditions are known in Δ, will be fed to the **for** loop in its last iterations. This means that in the first k iterations, each ψ_i ($i \leq k$) is unknown in Δ and it might be (but it is not necessarily) unknown in some μDNF-state(s) stored in $Result$. In the last $p - k$ iterations, the set if μDNF-states stored in $Result$ does not change. We have the following proposition.

Proposition 50. *Let* Δ *be a* μDNF*-state,* a *be an action,* p *be the number of combined conditional effects of* a*, and* k *be the number of e-conditions of action* a *that are unknown in* Δ*. Let* r *be the maximum number of literals in an e-condition of* a *and* n *be the number of propositions in the domain. Then,*

1. *For* $enb_{\mu DNF}(a, \Delta)$ *we have*

$$T(enb_{\mu DNF}(a, \Delta)) = O(n|\Delta|^2(1 + r^2)^k + n(p - k)|\Delta|(1 + r)^k)$$

2. $enb_{\mu DNF}(a, \Delta)$ *contains at most* $\binom{k}{i}$ μDNF*-states of the size* $\vartheta(r^i|\Delta|)$ *for* $i = 0, \ldots, k$.

Proof.

1. Let a_k be the action that has the k combined conditional effects of a, whose e-conditions are unknown in Δ. We have $T(enb_{\mu DNF}(a, \Delta)) = \Sigma_{i=1}^{p} T_i$, where T_i denotes the computational cost for iteration i. As discussed earlier, we have $T(enb_{\mu DNF}(a, \Delta)) \leq \Sigma_{i=1}^{k} T_i + \Sigma_{i=k+1}^{p} T_i$, where ψ_i is unknown in Δ for $1 \leq i \leq k$ and it is known in Δ for $k + 1 \leq i \leq p$. This implies that $T(enb_{\mu DNF}(a, \Delta)) = O(T(enb_{\mu DNF}(a_k, \Delta)) + \Sigma_{i=k+1}^{p} T_i)$, where ψ_i is known in Δ for $k + 1 \leq i \leq p$. By Proposition 49, we have $T(enb_{\mu DNF}(a_k, \Delta)) = O(n|\Delta|^2(1 + r^2)^k)$. On the other hand, for $k + 1 \leq i \leq p$, ψ_i is known in every μDNF-state in $enb_{\mu DNF}(a_k, \Delta)$. This is the set of μDNF-states stored in $Result$ after the first k iterations and they remains unchanged in the last $p - k$ iterations. Hence, $T_i = O(n(\Sigma_{\Delta' \in enb_{\mu DNF}(a_k, \Delta)} |\Delta'|))$, where

$$\begin{aligned}
\Sigma_{\Delta' \in enb_{\mu DNF}(a_k, \Delta)} |\Delta'| &\leq \binom{k}{0}(r^0|\Delta|) + \binom{k}{1}(r^1|\Delta|) + \ldots + \binom{k}{k}(r^k|\Delta|) \quad \text{(by Proposition 49)} \\
&= |\Delta|(\binom{k}{0}r^0 + \binom{k}{1}r^1 + \binom{k}{2}r^2 + \ldots + \binom{k}{k}r^k) \\
&= |\Delta|(1 + r)^k
\end{aligned}$$

The computation cost for the last $p-k$ iterations, hence, is $O(n(p-k)|\Delta|(1+r)^k)$. Thus,

$$\begin{aligned} T(enb_{\mu DNF}(a,\Delta)) &= O(n|\Delta|^2(1+r^2)^k) + O(n(p-k)|\Delta|(1+r)^k) \\ &= O(n|\Delta|^2(1+r^2)^k + n(p-k)|\Delta|(1+r)^k). \end{aligned}$$

2. After the first k iterations, the set of μDNF-states stored in $Result$ is $enb_{\mu DNF}(a_k,\Delta)$ and it remains unchanged in the last $p-k$ iterations. This implies that $enb_{\mu DNF}(a,\Delta) = enb_{\mu DNF}(a_k,\Delta)$. The proof is then trivial by Proposition 49.

Finally, one can observe that for $update_{\mu DNF}$ and $merge_{\mu DNF}$ (Algorithm 8):

- Updating each partial state $\delta' = \delta \setminus \overline{e(a,\Delta')} \cup e(a,\Delta')$ (Line 10) requires time $O(n+m) = O(n)$.
- Computing $min(\Delta_{succ} \cup \{\delta'\})$ (Line 11) has cost $O(n|\Delta_{succ}|)$, where $|\Delta_{succ}|$ increases from 0 to the total number of partial states in all the μDNF-state(s) of $enb_{\mu DNF}(a,\Delta)$.
- Let N be the total number of partial states in all the μDNF-state(s) of $enb_{\mu DNF}(a,\Delta)$: $N = \Sigma_{\Delta' \in enb_{\mu DNF}(a,\Delta)}|\Delta'|$. The running time for both updating and merging is $O(Nn) + O(\Sigma_{i=1}^{N} ni) = O(nN^2)$, where

$$\begin{aligned} N = \Sigma_{\Delta' \in enb_{\mu DNF}(a,\Delta)}|\Delta'| &\leq \Sigma_{i=0}^{k} \binom{k}{i} r^i|\Delta| \qquad \text{(by Proposition 50)} \\ &= |\Delta|(1+r)^k \end{aligned}$$

Hence, the running time for Algorithm 8 to compute both $update_{\mu DNF}$ and $merge_{\mu DNF}$ functions in the computation of $\Phi_{\mu DNF}(a,\Delta)$ is $O(n|\Delta|^2(1+r)^{2k})$.

Altogether, the running time of computing $\Phi_{\mu DNF}(a,\Delta)$ is the total of running time for computing $enb_{\mu DNF}(a,\Delta)$ and updating and merging the μDNF-states:

$$T(\Phi_{\mu DNF}(a,\Delta)) = O(n|\Delta|^2(1+r^2)^k + n(p-k)|\Delta|(1+r)^k) + O(n|\Delta|^2(1+r)^{2k}) = O(n|\Delta|^2(1+r)^{2k})$$

Proof of Proposition 19. Two sets are equal iff every element in one set belongs to the other set and vice versa (iff an arbitrary element in one set is also an element in the other set and vice versa).

1. Consider an arbitrary clause γ in $r(\varphi \cup \{\alpha\})$. Then γ is nontrivial and there does not exists in $\varphi \cup \{\alpha\}$ a clause that subsumes γ. Since $\varphi \subseteq \varphi \cup \{\alpha\}$, φ does not subsumes γ. If $\gamma \in \varphi$ then $\gamma \in r(\varphi)$, otherwise $\gamma = \alpha$. This implies that $\gamma \in (r(\varphi) \cup \{\alpha\})$. Observe that $(r(\varphi) \cup \{\alpha\}) \subseteq (\varphi \cup \{\alpha\})$ so $(r(\varphi) \cup \{\alpha\})$ does not subsume γ. Hence, $\gamma \in r(r(\varphi) \cup \{\alpha\})$. Now we consider an arbitrary clause ρ in $r(r(\varphi) \cup \{\alpha\})$. Then ρ is nontrivial and there does not exists in $(r(\varphi) \cup \{\alpha\})$ a clause that subsumes ρ. Assume that $\rho \notin r(\varphi \cup \{\alpha\})$. Then there exists in $\varphi \cup \{\alpha\}$ a clause λ that subsumes ρ and $\lambda \notin (r(\varphi) \cup \{\alpha\})$. This implies that $\lambda \in (\varphi \cup \{\alpha\}) \setminus (r(\varphi) \cup \{\alpha\})$ or $\lambda \in \varphi \setminus r(\varphi)$. This means that λ is subsumed by a clause in $r(\varphi)$ and hence ρ is subsumed by the same clause in $r(\varphi)$. Thus, ρ is subsumed by a clause in $r(\varphi) \cup \{\alpha\}$, a contradiction. Hence, $\rho \in r(\varphi \cup \{\alpha\})$ (proof).
2. Using the first result of this proposition, one can easily prove that

$$r(r(r(\varphi) \cup \{\alpha\}) \cup \{\beta\}) = r(r(r(\varphi) \cup \{\beta\}) \cup \{\alpha\}) = r(\varphi \cup \{\alpha,\beta\})$$

The proof is similar to the above proof.

Proof of Proposition 20. To prove that $(\varphi \otimes \psi_1) \otimes \psi_2 = (\varphi \otimes \psi_2) \otimes \psi_1$ we first prove that

$$(\varphi \otimes \psi_1) \otimes \psi_2 = r(\varphi \times \psi_1 \times \psi_2) \qquad (20A)$$

By definition, we have that $(\varphi \otimes \psi_1) \otimes \psi_2 = r(r(\varphi \times \psi_1) \times \psi_2)$. Since $r(\varphi \times \psi_1) \subseteq \varphi \times \psi_1$, by definition we have $r(\varphi \times \psi_1) \times \psi_2 \subseteq \varphi \times \psi_1 \times \psi_2$. Hence, $r(r(\varphi \times \psi_1) \times \psi_2) \subseteq r(\varphi \times \psi_1 \times \psi_2)$, i.e., $(\varphi \otimes \psi_1) \otimes \psi_2 \subseteq r(\varphi \times \psi_1 \times \psi_2)$ (20a).

Now consider an arbitrary clause α in $r(\varphi \times \psi_1 \times \psi_2)$. Then α is a nontrivial clause in $\varphi \times \psi_1 \times \psi_2$ and there does not exist in $\varphi \times \psi_1 \times \psi_2$ a clause that subsumes α. We will prove that $\alpha \in r(r(\varphi \times \psi_1) \times \psi_2)$. By definition, we have that $\exists \beta \in \varphi \exists \gamma_1 \in \psi_1 \exists \gamma_2 \in \psi_2.\ \alpha = \beta \cup \gamma_1 \cup \gamma_2$. We prove that $\beta \cup \gamma_1$ is not subsumed by $\varphi \times \psi_1$. Assume the contrary that $\exists \beta' \in \varphi \exists \gamma_3 \in \psi_1$ such that $\beta' \cup \gamma_3$ subsumes $\beta \cup \gamma_1$. This implies that $\beta' \cup \gamma_3 \cup \gamma_2$ subsumes $\beta \cup \gamma_1 \cup \gamma_2 = \alpha$, a contradiction. Hence, $\beta \cup \gamma_1$ is not subsumed by $\varphi \times \psi_1$ and thereby $\beta \cup \gamma_1 \in r(\varphi \times \psi_1)$. By definition, we have $\beta \cup \gamma_1 \cup \gamma_2 \in r(\varphi \times \psi_1) \times \psi_2$, i.e., $\alpha \in r(\varphi \times \psi_1) \times \psi_2$. Moreover, since $r(\varphi \times \psi_1) \times \psi_2 \subseteq \varphi \times \psi_1 \times \psi_2$ and $\varphi \times \psi_1 \times \psi_2$ does not subsume α so $r(\varphi \times \psi_1) \times \psi_2$ does not subsume α either. In addition, α is nontrivial so $\alpha \in r(r(\varphi \times \psi_1) \times \psi_2)$, i.e., $\alpha \in (\varphi \otimes \psi_1) \otimes \psi_2$. Since α is an arbitrary clause in $r(\varphi \times \psi_1 \times \psi_2)$, we have that $r(\varphi \times \psi_1 \times \psi_2) \subseteq (\varphi \otimes \psi_1) \otimes \psi_2$ (20b).

(20a) and (20b) yields the proof for (20A).

Similarly, one can prove that

$$(\varphi \otimes \psi_2) \otimes \psi_1 = r(\varphi \times \psi_1 \times \psi_2) \qquad (20B)$$

(20A) and (20B) results in

$$(\varphi \otimes \psi_1) \otimes \psi_2 = (\varphi \otimes \psi_2) \otimes \psi_1 \qquad (proof)$$

Proof of Proposition 24. We will prove for the case that $\varphi_\ell \neq \emptyset$, $\varphi_{\bar{\ell}} \neq \emptyset$, and neither of ℓ or $\bar{\ell}$ is a unit literal of φ. For the other cases, e.g., either $\varphi_\ell = \emptyset$ or $\varphi_{\bar{\ell}} = \emptyset$, the proof can be obtained similarly but simpler so we omit it here.

Let $\varphi_0 = \varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}})$. Then $\varphi = \varphi_0 \cup \varphi_\ell \cup \varphi_{\bar{\ell}}$. Observe that $\varphi_\ell \equiv (\varphi_\ell - \ell) \vee \ell$. Let $\theta = \varphi_\ell - \ell$, then $\varphi_\ell \equiv \theta \vee \ell$. Similarly, $\varphi_{\bar{\ell}} \equiv \vartheta \vee \bar{\ell}$, where $\vartheta = \varphi_{\bar{\ell}} - \bar{\ell}$. Thus,

$$\begin{aligned}\varphi &\equiv \varphi_0 \wedge (\theta \vee \ell) \wedge (\vartheta \vee \bar{\ell}) \\ &\equiv (\varphi_0 \wedge \theta \wedge \vartheta) \vee (\varphi_0 \wedge \theta \wedge \bar{\ell}) \vee (\varphi_0 \wedge \vartheta \wedge \ell)\end{aligned}$$

Hence,

$$\begin{aligned}BS(\varphi) &= BS((\varphi_0 \wedge \theta \wedge \vartheta) \vee (\varphi_0 \wedge \theta \wedge \bar{\ell}) \vee (\varphi_0 \wedge \vartheta \wedge \ell)) \\ &= BS(\varphi_0 \wedge \theta \wedge \vartheta) \cup BS(\varphi_0 \wedge \theta \wedge \bar{\ell}) \cup BS(\varphi_0 \wedge \vartheta \wedge \ell)\end{aligned}$$

This implies the following

$$\begin{aligned}\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\ s \in BS(\varphi)\} = &\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\ s \in BS(\varphi_0 \wedge \theta \wedge \vartheta)\} \cup \\ &\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\ s \in BS(\varphi_0 \wedge \theta \wedge \bar{\ell})\} \cup \\ &\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\ s \in BS(\varphi_0 \wedge \vartheta \wedge \ell)\}\end{aligned} \qquad (9)$$

Observe that, φ_0, θ, and ϑ do not contain either ℓ or $\bar{\ell}$. Hence, $\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\ s \in BS(\varphi_0 \wedge \theta \wedge \vartheta)\}$ is a set of states that entail both $\varphi_0 \wedge \theta \wedge \vartheta$ and ℓ, i.e. they entail $\varphi_0 \wedge \theta \wedge \vartheta \wedge \ell$. In the other words, $\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\ s \in BS(\varphi_0 \wedge \theta \wedge \vartheta)\} \subseteq BS(\varphi_0 \wedge \theta \wedge \vartheta \wedge \ell)$. On the other hand, let s' be an arbitrary state in $BS(\varphi_0 \wedge \theta \wedge \vartheta \wedge \ell)$. Then $s' \models \varphi_0 \wedge \theta \wedge \vartheta$ and $s' \models \ell$, i.e., $s' \in BS(\varphi_0 \wedge \theta \wedge \vartheta)$ and $s' \setminus \{\bar{\ell}\} \cup \{\ell\} = s'$.

This implies that $s' \in \{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\, s \in BS(\varphi_0 \wedge \theta \wedge \vartheta)\}$. Thus, $\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\, s \in BS(\varphi_0 \wedge \theta \wedge \vartheta)\} = BS(\varphi_0 \wedge \theta \wedge \vartheta \wedge \ell)$.

Now we consider the second set in the right side of Equation 9. Since $\varphi_0 \wedge \theta$ does not contain either ℓ or $\bar{\ell}$, it is easy to see that $\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\, s \in BS(\varphi_0 \wedge \theta \wedge \bar{\ell})\}$ is a set of states that entail both $\varphi_0 \wedge \theta$ and ℓ. Similarly, we have $\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\, s \in BS(\varphi_0 \wedge \theta \wedge \bar{\ell})\} = BS(\varphi_0 \wedge \theta \wedge \ell)\}$. Finally, the third set in the right side of Equation 9 is obviously equal to $BS(\varphi_0 \wedge \vartheta \wedge \ell)$. By Equation 9, we obtain the following

$$\begin{aligned}\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\, s \in BS(\varphi)\} = & BS(\varphi_0 \wedge \theta \wedge \vartheta \wedge \ell) \cup \\ & BS(\varphi_0 \wedge \theta \wedge \ell) \cup \\ & BS(\varphi_0 \wedge \vartheta \wedge \ell)\end{aligned}$$

Observe that, the first set in the right side of the above equation is a subset of the second set (and also a subset of the third set). Hence,

$$\begin{aligned}\{s \setminus \{\bar{\ell}\} \cup \{\ell\} |\, s \in BS(\varphi)\} &= BS(\varphi_0 \wedge \theta \wedge \ell) \cup BS(\varphi_0 \wedge \vartheta \wedge \ell) \\ &\equiv (\varphi_0 \wedge \theta \wedge \ell) \vee (\varphi_0 \wedge \vartheta \wedge \ell) \\ &\equiv \varphi_0 \wedge \ell \wedge (\theta \vee \vartheta) \\ &= \varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \wedge \ell \wedge ((\varphi_\ell - \ell) \vee (\varphi_{\bar{\ell}} - \bar{\ell})) \\ &\equiv \mu_{CNF}(\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \cup \{\ell\} \cup (\varphi_\ell - \ell) \times (\varphi_{\bar{\ell}} - \bar{\ell})) \\ &= update_r(\varphi, \ell) \qquad \text{(by Definition 20)}\end{aligned}$$

Proof of Proposition 25. To prove this proposition, we use the following two lemmas.

Lemma 3. *Given two CNF formulae φ and ψ, then*

1. $r(r(\varphi)) = r(\varphi)$
2. $r(\varphi \cup r(\psi)) = r(\varphi \cup \psi)$
3. $r(\varphi \times \varphi) = r(\varphi)$
4. $r(\varphi \cup (\varphi \times \psi)) = r(\varphi)$

Proof 1. 1. *trivial*

2. *Observe that $(\varphi \cup r(\psi)) \subseteq r(\varphi \cup \psi)$ (because $r(\psi) \subseteq \psi$). Let α be an arbitrary clause in $r(\varphi \cup \psi)$. Then α is nontrivial and there exists no clause in $\varphi \cup \psi$ that subsumes α. Hence, there exists no clause in $\varphi \cup r(\psi)$ that subsumes α. Thus, $\alpha \in \varphi \cup r(\psi)$ and thereby $(\varphi \cup \psi) \subseteq r(\varphi \cup r(\psi))$. As a consequence, $r(\varphi \cup r(\psi)) = r(\varphi \cup \psi)$.*
3. *We have*

$$\begin{aligned}\varphi \times \varphi &= \{\alpha \cup \beta \mid \alpha, \beta \in \varphi\} \qquad \text{(by definition)} \\ &= \{\alpha \cup \alpha \mid \alpha \in \varphi\} \cup \{\alpha \cup \beta \mid \alpha, \beta \in \varphi \wedge \alpha \neq \beta\} \\ &= \{\alpha \mid \alpha \in \varphi\} \cup \{\alpha \cup \beta \mid \alpha, \beta \in \varphi \wedge \alpha \neq \beta\}\end{aligned}$$

   Observe that each clause in the later subset of $\varphi \times \varphi$ is subsumed by a clause in the former subset which is φ. Hence, $r(\varphi \times \varphi) = r(\varphi)$
4. *This is trivial as every clause in $\varphi \times \psi$ is a superset of a clause in φ.*

Lemma 4. *If φ and ψ are two CNF formulae and ℓ and ℓ' are not a unit literal in φ or ψ, then*

1. *If* $\ell \notin lit(\varphi)$ *then* $\varphi \times (\psi - \ell) = \varphi \times \psi - \ell$
2. *If* $\ell \notin lit(\psi)$ *and* $\ell' \notin lit(\varphi)$ *then* $(\varphi - \ell) \times (\psi - \ell') = \varphi \times \psi - \ell - \ell'$

Proof 2. 1. *We have*

$$\begin{aligned}
\varphi \times (\psi - \ell) &= \{\alpha \cup \beta \mid \alpha \in \varphi, \beta \in (\psi - \ell)\} \\
&= \{\alpha \cup (\beta - \ell) \mid \alpha \in \varphi, \beta \in \psi\} && \text{(note that } \{\ell\} \notin \psi) \\
&= \{\alpha \cup \beta - \ell \mid \alpha \in \varphi, \beta \in \psi\} && \text{(note that } \forall \alpha \in \varphi.\ \ell \notin \alpha) \\
&= \varphi \times \psi - \ell
\end{aligned}$$

2. *Using the previous result, we have*

$$(\varphi - \ell) \times (\psi - \ell') = (\varphi - \ell) \times \psi - \ell' = \varphi \times \psi - \ell - \ell'$$

To prove Proposition 25, we partition φ as follows

$$\varphi = \varphi_0 \cup \varphi_{\ell_1} \cup \varphi_{\bar{\ell_1}} \cup \varphi_{\ell_2} \cup \varphi_{\bar{\ell_2}} \cup \varphi_{\ell_1\ell_2} \cup \varphi_{\ell_1\bar{\ell_2}} \cup \varphi_{\bar{\ell_1}\ell_2} \cup \varphi_{\bar{\ell_1}\bar{\ell_2}}$$

where $\varphi_0 = \{\alpha \mid \alpha \in \varphi \wedge \ell_1, \bar{\ell_1}, \ell_2, \bar{\ell_2} \notin \alpha\}$, $\varphi_{\ell_1} = \{\alpha \mid \alpha \in \varphi \wedge \ell_1 \in \alpha \wedge \ell_2, \bar{\ell_2} \notin \alpha\}$, and $\varphi_{\ell_1\ell_2} = \{\alpha \mid \alpha \in \varphi \wedge \ell_1 \in \alpha \wedge \ell_2 \in \alpha\}$.

The other components are defined similarly. Using Lemmas 3 and 4, one can verifies that the following holds.

$$\begin{aligned}
&update_r(update_r(\varphi, \ell_1), \ell_2) = update_r(update_r(\varphi, \ell_2), \ell_1) \\
&\quad = r(\varphi_0 \cup \{\{\ell_1\}\} \cup \{\{\ell_2\}\} \cup (\varphi_{\ell_1} \times \varphi_{\bar{\ell_1}} - \ell_1 - \bar{\ell_1}) \cup (\varphi_{\ell_2} \times \varphi_{\bar{\ell_2}} - \ell_2 - \bar{\ell_2}) \cup \\
&\quad (\varphi_{\ell_1} \times \varphi_{\ell_2} \times \varphi_{\bar{\ell_1}\bar{\ell_2}} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}) \cup (\varphi_{\ell_1} \times \varphi_{\bar{\ell_2}} \times \varphi_{\bar{\ell_1}\ell_2} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}) \cup \\
&\quad (\varphi_{\bar{\ell_1}} \times \varphi_{\ell_2} \times \varphi_{\ell_1\bar{\ell_2}} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}) \cup (\varphi_{\bar{\ell_1}} \times \varphi_{\bar{\ell_2}} \times \varphi_{\ell_1\ell_2} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}) \cup \\
&\quad (\varphi_{\ell_1} \times \varphi_{\bar{\ell_1}\ell_2} \times \varphi_{\bar{\ell_1}\bar{\ell_2}} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}) \cup (\varphi_{\ell_2} \times \varphi_{\ell_1\bar{\ell_2}} \times \varphi_{\bar{\ell_1}\bar{\ell_2}} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}) \cup \\
&\quad (\varphi_{\bar{\ell_1}} \times \varphi_{\ell_1\ell_2} \times \varphi_{\ell_1\bar{\ell_2}} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}) \cup (\varphi_{\bar{\ell_2}} \times \varphi_{\ell_1\ell_2} \times \varphi_{\bar{\ell_1}\ell_2} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}) \cup \\
&\quad (\varphi_{\ell_1\ell_2} \times \varphi_{\ell_1\bar{\ell_2}} \times \varphi_{\bar{\ell_1}\ell_2} \times \varphi_{\bar{\ell_1}\bar{\ell_2}} - \ell_1 - \bar{\ell_1} - \ell_2 - \bar{\ell_2}))
\end{aligned}$$

Proof of Proposition 26. Observe that $update_{\mu CNF}(\varphi, \eta) \equiv update_r(\varphi, \eta)$. Hence, to prove the proposition, it suffices to prove the following

$$BS(update_r(\varphi, \eta)) = \{s \setminus \overline{\eta} \cup \eta \mid s \in BS(\varphi)\} \qquad \text{(P26)}$$

The proof is by induction on $|\eta|$.

- Base case $|\eta| = 0$ iff $\eta = \emptyset$: (P26) becomes $BS(\varphi) = \{s \mid s \in BS(\varphi)\}$. This is obviously true.
- Inductive step: suppose (P26) holds for every consistent set of n literals, $n \geq 0$. We will prove that it also holds for every consistent set of $n + 1$ literals. Indeed,

$$\begin{aligned}
BS(update_r(\varphi, \eta)) &= BS(update_r(update_r(\varphi, \eta \setminus \{\ell\}), \ell)) && \text{for some } \ell \in \eta \\
&= \{s \setminus \{\bar{\ell}\} \cup \{\ell\} \mid s \in BS(update_r(\varphi, \eta \setminus \{\ell\}))\} && \text{(Proposition 24)} \\
&= \{s \setminus \overline{(\eta \setminus \{\ell\})} \cup (\eta \setminus \{\ell\}) \setminus \{\bar{\ell}\} \cup \{\ell\} \mid s \in BS(\varphi)\} && \text{(inductive hypothesis)} \\
&= \{s \setminus \overline{\eta} \cup \eta \mid s \in BS(\varphi)\} && (\ell \in \eta)
\end{aligned}$$

Thus, (P26) holds for every consistent set of literals (proof).

Proof of Proposition 28.

1. Observe that $Rel(\varphi, \psi)$ is a subset of the μCNF-state φ. We have $\varphi \models Rel(\varphi, \psi)$, hence if $Rel(\varphi, \psi) \models \psi$ then $\varphi \models \psi$.
   Now we need to prove that if $\varphi \models \psi$ then $Rel(\varphi, \psi) \models \psi$. Assume the contrary that $\varphi \models \psi$ and $Rel(\varphi, \psi) \not\models \psi$. This implies that $Rel(\varphi, \psi) \wedge \neg\psi$ is satisfiable. By Proposition 3, there exists a consistent set of literals δ such that $\delta \models Rel(\varphi, \psi) \wedge \neg\psi$ and $prop(\delta) = prop(Rel(\varphi, \psi) \wedge \neg\psi)$.
   On the other hand, because $Rel(\varphi, \psi) \not\models \psi$ and $\varphi \models \psi$, $Rel(\varphi, \psi) \neq \varphi$. This means that $Rel(\varphi, \psi) \subset \varphi$, hence $\varphi \setminus Rel(\varphi, \psi) \neq \emptyset$. Therefore, $\varphi \setminus Rel(\varphi, \psi)$ is satisfiable (otherwise φ is unsatisfiable, a contradiction). By Proposition 3, there exists a consistent set of literals δ' such that $\delta' \models \varphi \setminus Rel(\varphi, \psi)$ and $prop(\delta') = prop(\varphi \setminus Rel(\varphi, \psi))$. Since $\varphi \setminus Rel(\varphi, \psi)$ is independent from ψ and $Rel(\varphi, \psi)$, clearly $\varphi \setminus Rel(\varphi, \psi)$ is independent from $Rel(\varphi, \psi) \wedge \neg\psi$. Hence, $prop(Rel(\varphi, \psi) \wedge \neg\psi) \cap prop(\varphi \setminus Rel(\varphi, \psi)) = \emptyset$, i.e., $prop(\delta) \cap prop(\delta') = \emptyset$. Hence, $\delta \cup \delta'$ is a consistent set of literals. This implies that $\delta \cup \delta' \models Rel(\varphi, \psi) \wedge \neg\psi$ and $\delta \cup \delta' \models \varphi \setminus Rel(\varphi, \psi)$. This implies that $\delta \cup \delta' \models (Rel(\varphi, \psi) \wedge \neg\psi) \wedge (\varphi \setminus Rel(\varphi, \psi))$ or $\delta \cup \delta' \models \varphi \wedge \neg\psi$. Let s be a state such that $\delta \cup \delta' \subseteq s$ then s is a model of $\varphi \wedge \neg\psi$. This means that $\varphi \wedge \neg\psi$ is satisfiable or $\varphi \not\models \psi$, a contradiction.
2. $\varphi \wedge \psi$ is satisfiable iff $\varphi \not\models \neg\psi$ iff $Rel(\varphi, \psi) \not\models \neg\psi$ (note that $Rel(\varphi, \psi) = Rel(\varphi, \neg\psi)$) iff $Rel(\varphi, \psi) \wedge \psi$ is satisfiable.

Proof of Proposition 29. We will investigate the running time of Algorithm 9 for computing $Rel(\varphi, \psi)$ in the worst case as follows.

- Computing the set $\pi = prop(\psi) \cup \overline{prop(\psi)}$ (line 3) is linear in the size of ψ. Since ψ is usually very small so this is negligible.
- Computing the set of clauses in φ that depend on ψ (lines 4-9) is $O(nm)$
- Computing Y, the set of clauses in V that depend on a clause newly added to R, in the **while** loop (line 14) is $O(n|V|) = O(nm)$. The computation in line 16 is $O(|Y|) = O(m)$. Hence, the running time for each iteration of the **while** loop is $O(nm)$. Since each clause α is taken once to compute the set Y in an iteration when V is not empty, the maximum number of iterations of the **while** loop is $m - 1$. Hence, the running time for the **while** loop is $O(nm^2)$.
- The running time of other computations is negligible.

In summary, using Algorithm 9, $Rel(\varphi, \psi)$ can be computed in $O(nm^2)$ time (in the worst case).

Proof of Proposition 30. We will analyze the cost of computing μ_{CNF} by Algorithm 10 in the worst case as follows.

- Computing $r(\varphi)$ (Line 3): $r(\varphi)$ is computed by
  - removing trivial clauses: Checking whether a clause is trivial is linear in the size of the clause, i.e., $O(n)$. Removing a clause takes $O(\lg m)$ time. Thus, this computation (for m clauses in φ) is $O(mn + m \lg m)$.

– removing subsumed clauses: checking whether a clause is subsumed by a given clause is $O(n + n) = O(n)$. Checking subsumption for all m clauses in φ takes $O(m^2 n)$. Hence, this computation is $O(nm^2 + m \lg m) = O(nm^2)$.

The overall running time for $r(\varphi)$, hence, is $O(nm^2)$.

- Computing the resolvents of a clause α with other clauses in φ (Line 7): checking whether α is resolvable with a clause β is $O(n)$ and computing a resolvent of two clauses is $O(n)$. At the beginning, φ contains m clauses. Thereafter, each time a new clause is added to φ, at least one clause is removed from φ (Lines 10-12). Thus, the number of clauses in φ is at most m during the computation of the algorithm. Hence, computing every possible resolvents of α with other clauses in φ is $O(nm)$ and φ_0 contains at most $m - 1$ clauses (resolvents).

- Computation of **for** loop (Lines 8-13): Computing $\varphi' = \{\beta' \mid \beta' \in \varphi \wedge \beta \subset \beta'\}$ is $O(nm)$. The computation of the **if** command (Lines 10-12) in each iteration is $O(m)$. There are $O(m)$ clauses in φ_0 so the computation of the **for** loop is $O(nm^2)$.

- Computing the **while** loop (Lines 5-15): The computation in one iteration is $O(nm) + O(nm^2) = O(nm^2)$. The number of iterations is at most the number of clauses in φ before entering the **while** loop plus the number of clauses added to φ during the computation in this loop. When a new clause is added to φ, the size of φ (initially it is N) reduces at least by 1 because at least a clause of larger size is removed from φ.

  Let M be the largest possible number of clauses that can be added to φ with the only condition that each time a clause is added to φ, at least one clause in φ of larger size is removed from φ and the size of each clause is at least 1. We will prove that $M \leq N - m$. Observe that M is obtained if each time a new clause α is added to φ, exactly one clause of the size greater $|\alpha|$ is removed from φ. Otherwise, assume that there exists k clauses, $k > 1$, that are removed at once when adding a new clause α to φ. Since every removed clause is larger than the new clause that replaces it, there exists a scheme that can add more than M clauses to φ as follows: remove only one among k clauses and add α to φ. Then the process of adding clauses to and removing clauses from φ is exactly same as the scheme that obtains M. When the process is done, there are still $k - 1$ clause(s) ($k > 0$) whose size is greater than some clause in φ then we can replace them with at least one more possible new clause in φ. This implies that the number of clauses added to φ is at least $M + 1$, a contradiction. Observe that each time a clause is added, the size of φ reduces at least by 1. At the end, φ still contain m clauses so its size is at least m. Thus, $M \leq N - m$.

  Obviously, the number of new clauses added to φ during the **while** loop cannot exceed M. Hence, the total number of clauses that ever exist in φ is at most $m + M \leq m + N - m = N$. Therefore, the number of iterations of the **while** loop is at most N. As a consequence, the computation cost for the **while** loop is $O(nm^2 N)$.

In summary, the overall running time for μ_{CNF} is $O(nm^2 + nm^2 N) = O(nm^2 N)$.

Proof of Proposition 31. Recall that $\texttt{extending}_{\mathcal{R}}(\varphi, \varphi \oplus_{\mu CNF} \psi$ is computed by Algorithm 3. We consider the following:

- Checking satisfaction (Lines 3 and 4): If a quick checking case is detected, e.g., there exists a literal ℓ in ψ such that $\{\bar{\ell}\} \in \varphi$ then $\varphi \models \neg\psi$ and $\varphi \not\models \psi$, checking satisfaction of ψ (or $\neg\psi$) in φ is easy

and there is no need a call to the SAT-solver. In the other case, usually ψ contains a small number of possibly unknown literals, mostly 1 or 2, and thereby the set of clauses in φ that are relating to the literal(s) is rather small. Let v be the size of $Rel(\varphi, \psi') \wedge \neg\psi$. Then checking the satisfiability of $Rel(\varphi, \psi')) \wedge \neg\psi$ in the worst case incurs 2^u different combinations of assignments of the truth value (either $true$ or $false$) to u variables in the formula. The time for checking each such assignment combination is linear in the size v of the formula. Hence, in the worst case, this computation is $O(2^u v) = O(2^u N)$ (observe that $v \leq N$).

- Since ψ is a set of few literals and $\neg\psi$ is a clause of few literals, the size of $(\varphi \wedge \psi)$ and $(\varphi \wedge \neg\psi)$ is almost the same as the size of φ. Hence, by Proposition 30, computing $conv_{\mu CNF}(\varphi \wedge \psi)$ (Line 9) and $conv_{\mu CNF}(\varphi \wedge \neg\psi)$ (Line 11) is $O(nm^2N)$.
- The computational cost in Lines 5, 10, and 12 is $O(n)$.

In summary, computing $\texttt{extending}_{\mathcal{R}}(\varphi, \varphi \oplus_{\mu CNF} \psi$ is $O(2^u N + nm^2N)$, if ψ is unknown in φ, and it is $O(2^u N + n)$, otherwise.

Proof of Proposition 32. Consider an arbitrary μCNF-state φ' in the set X and an arbitrary e-condition ψ of a. Let $u' = |prop(Rel(\varphi', \psi''))|$, where $\psi'' = \{\ell \in \psi \mid \ell \text{ is possibly unknown in } \varphi'\}$.

First, we provide a sketch proof for the fact $u' \leq u$. Observe that each μCNF-state in X is generated by adding clauses to another μCNF-state in X, initialized with φ, and simplifying by μ_{CNF}. Thus, the set of unit clauses in φ' is a superset of unit clauses in φ as an unit clause is never removed by a simplification. Hence, $\psi'' \subseteq \psi'$, where $\psi' = \{\ell \in \psi \mid \ell \text{ is possibly unknown in } \varphi\}$. This implies that $u' = |prop(Rel(\varphi', \psi''))| \leq |prop(Rel(\varphi', \psi'))|$. Observe that φ' is obtained by simplifying a subset of $\varphi \cup \Psi'(a)$ possibly with some added unit clauses and $Rel(\varphi', \psi')$ does not contain unit clauses. One can prove that, for each set of clauses φ_1, $|prop(Rel(\mu_{CNF}(\varphi_1), \psi'))| \leq |prop(Rel(\varphi_1, \psi'))|$ and if $\varphi_2 \subseteq \varphi_1$ then $|prop(Rel(\varphi_2, \psi')) \leq |prop(Rel(\varphi_1, \psi'))|$. This implies that $|prop(Rel(\varphi', \psi'))| \leq |prop(Rel(\varphi \cup \Psi'(a), \psi'))| \leq u$. Thus, we have

$$u' = |prop(Rel(\varphi', \psi''))| \leq u \tag{32a}$$

Similar to the analysis of $enb_{\mu DNF}$, the cost of computing $enb_{\mu CNF}(a, \varphi)$ is maximum when the k combined conditional effects of these k e-conditions are introduced to the outer **for** loop (Lines 6-12 of Algorithm 4) first. By Proposition 31, (32a), and the size limit assumption, the running time of the inner **for** loop (Lines 8-10) is $O((2^u N + nm^2N)|X|)$ for each of the first k iterations (ψ is unknown in φ) and it is $O((2^u N + n)|X|)$ for each of the last $p - k$ iterations (ψ is known in φ). Consider the first k iterations of the outer **for** loop. In the beginning of first iteration, X contains one formula which is φ. After each iteration, the number of formulae in X increases at most twice. Hence, in the beginning of iteration i, $i \leq k$, X contains at most 2^{i-1} formulae. The running time of the first k iterations is

$$O((2^u N + nm^2N)(1 + 2 + \ldots + 2^{k-1})) = O(2^k(2^u N + nm^2N)).$$

The running time of the last $p - k$ iterations, where X stores at most 2^k μCNF-states and this set does not change, is

$$O(2^k(2^u N + n)(p - k)).$$

The overall running time of Algorithm 4 for computing $enb_{\mu CNF}(a, \varphi)$ is then

$$T(enb_{\mu CNF}(a, \varphi)) = O(2^k(2^u N + nm^2N) + 2^k(2^u N + n)(p - k))$$

Simplifying this formula by ignoring the terms of lower complexity, we obtain

$$T(enb_{\mu CNF}(a,\varphi)) = O(2^{k+u}Np + 2^k Nnm^2)$$

Proof of Proposition 34. We again make the size limit assumption in this computation. First, we consider the computation of each iteration of the **for** loop in Algorithm 11 for each ℓ in $e(a,\varphi)$. It is easy to see that the computation in the first two cases (when either ℓ or $\bar{\ell}$ is a unit clause of φ') is simpler than that in the third case. Since we want to evaluate the running time in the worst case, we will evaluate the running time of the computation in the third case (lines 7-21).

- One can see that computing φ_0, φ^+, and φ^- all together (Lines 10-12) is $O(nm)$.
- Computing the updated formula w.r.t. ℓ in Line 13 includes:
  - Computing $(\varphi^+ \times \varphi^-)$ is $O(n|\varphi^+||\varphi^-|)$. Observe that

    $$|\varphi^+||\varphi^-| \leq \left(\frac{|\varphi^+| + |\varphi^-|}{2}\right)^2 \leq (|\varphi|/2)^2 \leq m^2/4$$

    Hence, the running time for this computation is $O(nm^2)$.
  - Computing the reduced CNF formula $\varphi = r(\varphi_0 \cup \{\{\ell\}\} \cup (\varphi^+ \times \varphi^-))$: Usually the set of resolvents on a non-unit literal is very small and they may be subsumed by a clause in φ_0. Furthermore, the new unit clause $\{\ell\}$ can subsume a great number of clauses in the formula (The experiments show that the size of the updated formula is usually smaller than the formula before updating). For this reason, we assume that the size of the reduced CNF formula $r(\varphi_0 \cup \{\{\ell\}\} \cup (\varphi^+ \times \varphi^-))$ does not exceed the size of φ before the update. To compute this formula, first we remove every clause in φ_0 that is subsumed by $\{\ell\}$ and add $\{\ell\}$ to it. Then, for every clause α in $(\varphi^+ \times \varphi^-)$, before adding α to the formula, we check subsumption between α and the other clauses in the formula. This checking is $O(nm)$. Since the number of clauses in $(\varphi^+ \times \varphi^-)$ is bounded by $m^2/4$, adding every clause in this set to the formula is $O(nm^3)$. Hence, the time for updating the formula on each literal ℓ is $O(nm^3)$.

There are at most n literals in the set $e(a,\varphi')$, hence, computing $update_{\mu CNF}(a,\varphi')$ is $O(n^2m^3)$. Since $enb_{\mu CNF}(a,\varphi)$ contains at most 2^k μCNF-states, the total running time for updating every μCNF-states in this set is $O(2^k n^2 m^3)$.

Proof of Theorem 5. The cost of computing $\Phi_{\mu CNF}(a,\varphi)$ is the summation of the costs of computing $enb_{\mu CNF}(a,\varphi)$ ($O(2^{k+u}Np + 2^k Nnm^2)$), updating the μCNF-states in $enb_{\mu CNF}(a,\varphi)$ ($O(2^k n^2 m^3)$), and merging the set of updated μCNF-states ($O(2^k nm^2 N)$). Observe that $N \leq nm$. Thus,

$$T(\Phi_{\mu CNF}(a,\varphi)) = O((2^{k+u}Np + 2^k Nnm^2) + (2^k n^2 m^3) + (2^k nm^2 N)) = O(2^{k+u}Np + 2^k n^2 m^3)$$

Proof of Proposition 37.

1. Since φ is a PI-state, a literal ℓ is satisfied in φ iff $\{\ell\} \in \varphi$. Checking whether ℓ is satisfied in φ is equivalent to checking whether $\{\ell\}$ belongs to φ that is linear in the number of clauses in φ, i.e., linear in m.

2. The nontrivial clause α is satisfied in φ iff α is an implicate of φ. If α is a prime implicate of φ then $\alpha \in \varphi$. Otherwise, there exists a clause β in φ that subsumes α. Thus, checking whether α is satisfied in φ is equivalent to check whether there exists a clause β in φ such that $\beta \subseteq \alpha$, which is linear in nm (assuming that the literals in each clause are sorted).

Proof of Proposition 38. Let φ be a PI-state. Since every clause in φ is a prime implicate of φ so it is a nontrivial clause and no clauses in φ subsumes another. Hence, $r(\varphi) = \varphi$. Furthermore, no clause in φ is subsumed by a resolvent of φ, which is an implicate of φ. Therefore, $\mu_{CNF}(\varphi) = \varphi$. Thus, φ is a μCNF-state.

Proof of Proposition 40.

- **Prove $update_{PI}(\varphi, \ell)$ is a PI-state:** If $\{\ell\} \in \varphi$ then, by definition, $update_{PI}(\varphi, \ell) = \varphi$, a PI-state. We now consider the case $\{\ell\} \notin \varphi$, i.e., $update_{PI}(\varphi, \ell) = \varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \cup \{\{\ell\}\}$. We need to prove that every clause in $update_{PI}(\varphi, \ell)$ is a prime implicate of this formula and every prime implicate of $update_{PI}(\varphi, \ell)$ already exists in it as follows.
  - Let α be an arbitrary clause in $update_{PI}(\varphi, \ell)$. If α is the unit clause $\{\ell\}$ then obviously α is a prime implicate of $update_{PI}(\varphi, \ell)$. Otherwise, α is a clause of $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}$, and thereby, a prime implicate of φ. Hence there does not exist an implicate of φ that subsumes α. Since the set of implicates of $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}$ is a subset of the set of implicates of φ, there does not exists an implicate of $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}$ that subsumes α. Moreover, since ℓ is independent from $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}$, no clauses in this set is subsumed by or resolvable with $\{\ell\}$. This implies that α is a prime implicate of $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \cup \{\{\ell\}\} = update_{PI}(\varphi, \ell)$.
  - Assume that there exists a prime implicate α of $update_{PI}(\varphi, \ell)$ such that $\alpha \notin update_{PI}(\varphi, \ell)$. Since ℓ is independent from $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}})$, α is a prime implicate of this set and $\ell, \bar{\ell} \notin \alpha$. This also implies that α is an implicate of φ and it is subsumed by a clause β in φ. Because α cannot be subsumed by a clause in $\varphi_\ell \cup \varphi_{\bar{\ell}}$ ($\ell, \bar{\ell} \notin \alpha$), $\beta \in \varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}})$. Thus, α is not a prime implicate of $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}})$, a contradiction. Thus $update_{PI}(\varphi, \ell)$ contains every prime implicate of it.
- **Prove $BS(update_{PI}(\varphi, \ell)) = \{s \setminus \{\bar{\ell}\} \cup \{\ell\} \mid s \in BS(\varphi)\}$:** Since $BS(update_r(\varphi, \ell)) = \{s \setminus \{\bar{\ell}\} \cup \{\ell\} \mid s \in BS(\varphi)\}$ (Proposition 24), it suffices to prove $update_{PI}(\varphi, \ell) \equiv update_r(\varphi, \ell)$. We consider the following three cases:
  - $\{\ell\} \in \varphi$: the proof is trivial as $update_{PI}(\varphi, \ell) = update_r(\varphi, \ell) = \varphi$
  - $\{\bar{\ell}\} \in \varphi$: Since φ is a PI-state, no clauses in φ subsumes $\bar{\ell}$, i.e., $\varphi_{\bar{\ell}} = \{\{\bar{\ell}\}\}$. Now we prove that $\varphi_\ell = \emptyset$. Assume that there exists $\alpha \in \varphi_\ell$. Then α is resolvable with $\bar{\ell}$ and their resolvent is $\alpha \setminus \{\ell\}$, that subsumes α, a contradiction. Thus, $\varphi_\ell = \emptyset$ and $\varphi_{\bar{\ell}} = \{\{\bar{\ell}\}\}$. This implies that $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \cup \{\{\ell\}\} = \varphi \setminus \{\{\bar{\ell}\}\} \cup \{\{\ell\}\}$. Hence, $update_{PI}(\varphi, \ell) \equiv update_r(\varphi, \ell)$ in this case.
  - $\{\ell\}, \{\bar{\ell}\} \notin \varphi$: As discussed earlier, every clause in the cross-product $(\varphi_\ell - \ell) \times (\varphi_{\bar{\ell}} - \bar{\ell})$ is trivial, subsumed by a clause in $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}})$, or already exists in this set. Hence $\varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \cup \{\{\ell\}\} \cup (\varphi_\ell - \ell) \times (\varphi_{\bar{\ell}} - \bar{\ell}) \equiv \varphi \setminus (\varphi_\ell \cup \varphi_{\bar{\ell}}) \cup \{\{\ell\}\}$. This means $update_{PI}(\varphi, \ell) \equiv update_r(\varphi, \ell)$.

  In all cases, we have $update_{PI}(\varphi, \ell) \equiv update_r(\varphi, \ell)$ (proof).

Proof of Proposition 41. By Proposition 40, we have

$$\begin{aligned} BS(update_{PI}(update_{PI}(\varphi,\ell_1),\ell_2)) &= \{s' \setminus \{\overline{\ell_2}\} \cup \{\ell_2\} \mid s' \in BS(update_{PI}(\varphi,\ell_1))\} \\ &= \{s' \setminus \{\overline{\ell_2}\} \cup \{\ell_2\} \mid s' \in \{s \setminus \{\overline{\ell_1}\} \cup \{\ell_1\} \mid s \in BS(\varphi)\} \\ &= \{s \setminus \{\overline{\ell_1},\overline{\ell_2}\} \cup \{\ell_1,\ell_2\} \mid s \in BS(\varphi)\} \qquad (\ell_1 \neq \overline{\ell_2}) \end{aligned}$$

Similarly, we have

$$BS(update_{PI}(update_{PI}(\varphi,\ell_2),\ell_1)) = \{s \setminus \{\overline{\ell_1},\overline{\ell_2}\} \cup \{\ell_1,\ell_2\} \mid s \in BS(\varphi)\}$$

Thus,

$$BS(update_{PI}(update_{PI}(\varphi,\ell_1),\ell_2)) = BS(update_{PI}(update_{PI}(\varphi,\ell_2),\ell_1)).$$

Two PI-states $update_{PI}(update_{PI}(\varphi,\ell_1),\ell_2)$ and $update_{PI}(update_{PI}(\varphi,\ell_2),\ell_1)$ are equivalent so they are identical, i.e., $update_{PI}(update_{PI}(\varphi,\ell_1),\ell_2) = update_{PI}(update_{PI}(\varphi,\ell_2),\ell_1)$ (proof).

Proof of Proposition 42. The proof is similar to that for Proposition 26 (by induction on $|\eta|$) based on Proposition 40.

Proof of Proposition 45.

1. We will evaluate the cost of computing $PI(\varphi \cup \{\alpha\})$ and the size of the obtained formula in the worst case based on Algorithm 13. It is easy to see that the worst case occurs in the **for** loop (lines 8-13). First, we will prove by induction on the size of α that the number of clauses in X at the end of the **for** loop is $O(m^r)$ (1).
   - Base case $n = 1$: the set X_ℓ contains only one clause which is α. The set $\varphi_{\bar{\ell}}$ is a subset of φ so it contains at most m clauses. Hence, there are at most m new resolvents generated in Y. At the end of the **for** loop, therefore, X will contain at most $m + 1 = O(m)$ clauses.
   - Inductive step: suppose that (1) is true for every such clause of the size $r-1$, for some $r > 1$. Let β be the clause that contains the first $r-1$ literals in α in the order that the literals are introduced to the **for** loop. Clearly, computing $PI(\varphi \cup \{\alpha\})$ (by Algorithm 13) is done by computing $PI(\varphi \cup \{\beta\})$ and executing one more iteration of the **for** loop for the last literal ℓ in α and $\ell \notin \beta$. By induction, at the beginning of the last iteration, X contains $O(m^{r-1})$ clauses and so does X_ℓ. If X contains one clause then clearly at the end of the iteration, the number of clauses in X is $O(m)$ and thereby $O(m^r)$. Otherwise, there is at least a clause β in φ that already resolved with α in an earlier iteration on some literal ℓ'. That clause cannot be resolvable with α on ℓ (otherwise, the resolvent computed earlier $\alpha|\beta$ is trivial as it contain both ℓ and $\bar{\ell}$). Hence, in this iteration, at most $m-1$ resolvents of α and other clauses in X_ℓ can be generated. Observe that, every clause in X is a resolvent of a clause in φ but α and a resolvent of a clause cannot be resolvable with that clause again (otherwise, the resolvent contains a pair of complementary literals and thereby it is trivial). Hence, at most $O(m^{r-1}) \times (m-1) = O(m^{r-1}(m-1))$ new resolvents are generated in Y. Thus, at the end of the last iteration, X will contain at most $O(m^{r-1}) + O(m^{r-1}(m-1)) = O(m^r)$ clauses.

   Now we will consider the computational cost of each iteration i of the **for** loop:

   - Computing Y (line 9) is $O(nm^{i-1}m) = O(nm^i)$.

- The computation for subsumption (lines 10-11): X and Y contains at most $O(m^{i-1})$ and $O(m^i)$ clauses respectively. Hence, this computation is $O(n(m + m^{i-1})m^i) = O(nm^{2i-1})$[8].
- Computing X in line 12 is $O(m^i)$.

In summary, the computational cost of each iteration i is $O(nm^{2i-1})$. The computation cost for the *for* loop, hence, is

$$O(\Sigma_{i=1}^{r} nm^{2i-1}) = O(\Sigma_{i=1}^{r} \frac{n}{m}(m^2)^i) = O(\frac{n}{m}\frac{(m^2)^{r+1}-1}{m^2-1}) = O(nm^{2r})$$

The number of clauses in the resulting formula is $O(m^r + m) = O(m^r)$.

2. Let ℓ be a literal in α that is known in φ. Since φ is a prime implicate formula, there are two possibilities:
   - ℓ is a unit literal in φ. In this case, α is subsumed by the unit clause $\{\ell\}$ of φ so $PI(\varphi \cup \{\alpha\}) = \varphi$ and the algorithm will returns φ immediately in line 4. The proof in this case, hence, is trivial.
   - $\bar{\ell}$ is a unit literal in φ. In this case, α can be replaced with the clause $\alpha \setminus \{\ell\}$.

   In the second case, every literal in α that is known in φ can be removed from it before computing the prime implicate form. This operation is easy in prime implicate formula and it does not affect the order of the computational cost of computing the prime implicate form. Let α' be the resulting clause after removing from α every literal that is known in φ. It is easy to see that $\varphi \cup \{\alpha\} \equiv \varphi \cup \{\alpha'\}$. Hence, computing $PI(\varphi \cup \{\alpha\})$ can be replaced with computing $PI(\varphi \cup \{\alpha'\})$. Observe that the number of literals in α' is u so we obtain the proof.

Proof of Theorem 6. Let φ be a PI-state and a be an action. Let m be the number of clauses in φ, p be the number of combined conditional effects of a, k be the number of e-conditions of a that are unknown in φ, and n be the number of propositions in the domain. Let r be the maximum number of literals in an e-condition of a and u be the maximum number of literals in an e-condition of a that are unknown in φ. Let $e = max\{e(a, s) \mid s \in BS(\varphi)\}$. We will discuss the cost of computing the components of Φ_{PI} and then the cost of computing this function in the worst case, with the assumption that the number of clauses in each intermediate formula generated during the computation of Φ_{PI} does not exceed m, the number of clauses in φ.

Proposition 51. *Let φ be a PI-state, m be the number of clauses in φ, ψ be a consistent set of literals, ψ' be the set of literals in ψ that are possibly unknown in φ, and $u = |\psi'|$. Computing $\varphi \oplus_{PI} \psi$ (Algorithm 3) is $O(nm)$ if ψ is known in φ and $O(nm^2r + nm^{2u})$ otherwise.*

Proof. Recall that $\varphi \oplus_{PI} \psi$ is computed by the procedure $\texttt{extending}_{\texttt{PI}}(\varphi, e(\varphi), \psi \rightarrow \eta)$ (Algorithm 3). We have:

- Checking satisfaction of ψ and $\neg\psi$ in φ (Lines 3-4 in Algorithm 3) is $O(|\psi|m) + O(nm) = O(nm)$ (Proposition 37)
- Updating the effects of executing a (Lines 5,10, and 12) is $O(n)$

[8]There is an exception for the case $i = 1$ where this should be $O(nm^2)$. However, it is easy to see that this simplification does not affect the order of the analysis result.

- Computing $conv_{PI}(\varphi \wedge \psi)$ by the `convPI`(φ, ψ) procedure (Algorithm 15) is $O(|\psi| nm^2) = O(nm^2 r)$ (Proposition 45)
- Computing $conv_{PI}(\varphi \wedge \neg\psi)$ by the `convPI`$(\varphi, \overline{\psi})$ procedure (Algorithm 13) is $O(nm^{2u})$ (Proposition 45)

In summary, the total cost of computing $\varphi \oplus_{PI} \psi$ is then $O(nm)$ if ψ is known in φ and $O(nm^2 r + nm^{2u})$ otherwise.

Proposition 52. *Let φ be a PI-state and ψ be a consistent set of literals.*

1. *If the number of literals in ψ is bounded by a constant, then $\varphi \oplus_{PI} \psi$ can be computed in polynomial time.*
2. *If the number of literals in ψ that are unknown in φ is bounded by a constant, then $\varphi \oplus_{PI} \psi$ can be computed in polynomial time.*

As an e-condition of an action of most problems contains a few literals (r is usually smaller than 3), and among them the number of literals unknown in φ (u) is even smaller, then this computation is polynomial time in practice.

As seen for the μDNF and μCNF representations, the running time of `enabling`(a, φ) is maximal when the k combined conditional effects whose e-conditions are unknown in φ are introduced in the first k iterations of the outer **for** loop (Lines 6-12). Similar to the analysis for μCNF, in the worst case, the running time of the first k iterations is

$$O((nm^2 r + nm^{2u})(1 + 2 + \ldots + 2^{k-1})) = O(2^k (nm^2 r + nm^{2u}))$$

and the running time of the last $p - k$ iterations is

$$O((nm)2^k (p - k))) = O(2^k nmp)$$

The overall running time of computing $enb_{PI}(a, \varphi)$ is shown in the following proposition.

Proposition 53. *Let φ be a PI-state and a be an action. Let m be the number of clauses in φ, p be the number of combined conditional effects of a, k be the number of e-conditions of a that are unknown in φ, and n be the number of propositions in the domain. Let r be the maximum number of literals in an e-condition of a and u be the maximum number of literals in an e-condition of a that are unknown in φ. If the number of clauses in each intermediate formula φ' generated during the computation of $enb_{PI}(a, \varphi)$ does not exceed the number of clauses m in φ, then*

$$T(enb_{PI}(a, \varphi)) = O(2^k (rnm^2 + nm^{2u} + nmp))$$

Updating a PI-state in $enb_{PI}(a, \varphi)$ by a literal (Line 7) takes $O(nm)$ time. There are at most e literals in η, so updating a PI-state takes $O(nme)$ time. Hence, updating all the PI-states in $enb_{PI}(a, \varphi)$, that contains at most 2^k PI-states, takes $O(2^k nme)$.

Computing $\oplus$ of two PI-states in $enb_{PI}(a, \varphi)$ includes:

- computing their cross-product and checking for the trivial clauses costs $O(nm^2)$.
- checking for subsumed clauses in the set costs $O(nm^3)$.

Hence, computing the PI-form of the cross-product of two PI-states in $enb_{PI}(a, \varphi)$ requires $O(nm^3)$ time. Since there are at most 2^k PI-states in $enb_{PI}(a, \varphi)$, then the computation of $merge_{PI}(enb_{PI}(a, \varphi))$ will need to apply $\oplus$ $2^k - 1$ times and, hence, it takes $O(2^k nm^3)$ time.